(19)日本国特許庁(JP)　(12)公表特許公報(A)　(11)特許出願公表番号
特表2004-518185
(P2004-518185A)
(43)公表日　平成16年6月17日(2004.6.17)

(51)Int.Cl.[7]	F I		テーマコード（参考）
G06K 17/00	G06K 17/00	B	5B058

審査請求 有　予備審査請求 有　（全 512 頁）

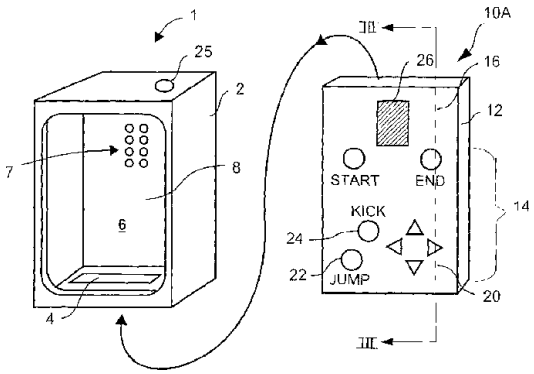
(21)出願番号	特願2002-527905(P2002-527905)
(86)(22)出願日	平成13年9月12日(2001.9.12)
(85)翻訳文提出日	平成15年3月12日(2003.3.12)
(86)国際出願番号	PCT/AU2001/001142
(87)国際公開番号	WO2002/023320
(87)国際公開日	平成14年3月21日(2002.3.21)
(31)優先権主張番号	PR 0073
(32)優先日	平成12年9月12日(2000.9.12)
(33)優先権主張国	オーストラリア(AU)
(31)優先権主張番号	PR 5593
(32)優先日	平成13年6月8日(2001.6.8)
(33)優先権主張国	オーストラリア(AU)

(71)出願人　000001007
キヤノン株式会社
東京都大田区下丸子３丁目３０番２号
(74)代理人　100076428
弁理士　大塚　康徳
(74)代理人　100112508
弁理士　高柳　司郎
(74)代理人　100115071
弁理士　大塚　康弘
(74)代理人　100116894
弁理士　木村　秀二

最終頁に続く

(54)【発明の名称】サービスへのアクセスのためのカード読取り装置

(57)【要約】
インタフェースカード（１０）を読む読取り装置（１）が開示される。カード（１０）は読取り装置（１）へと挿入されるように構成される。カード（１０）は、表面に形成される印（１４）と外部装置と通信するためにデータを格納するメモリ（１９）とを具備する。読取り装置（１）は、読取り装置（１）にカード（１０）を受け入れるとインターフェースカード（１０）に重なるように配置されるほぼ透明な感圧膜（８）を備える。また、読取り装置（１）は、サービス識別子とユーザにより選択される印に関連するデータの特定の部分とを前記外部装置に送信する中央処理装置を具備する。特定データはユーザにより選択された印（１４）に関連し、外部装置はデータを受信するとサービス識別子により識別されるサービスを提供する。



【特許請求の範囲】
【請求項１】
読取り装置へと挿入されるように構成されるインタフェースカードを読む読取り装置であって、前記カードが表面に形成される印と外部装置と通信するためにデータを格納するメモリとを具備し、
前記読取り装置に前記インタフェースカードを受け入れると前記カードに重なるように配置されるほぼ透明な感圧膜と、
サービス識別子とユーザにより選択される印に関連するデータの特定の部分とを前記外部装置に送信する中央処理装置とを具備し、前記外部装置は前記データを受信すると前記サービス識別子により識別されるサービスを提供する読取り装置。
【請求項２】
アプリケーションにより使用されるようにサービス識別子がベンダにより設定される請求項１記載の読取り装置。
【請求項３】
前記サービス識別子は中央機関により前記ベンダに割り当てられる請求項２記載の読取り装置。
【請求項４】
前記中央処理装置は前記読取り装置から読まれる読取り装置識別子を前記外部装置に送信する請求項１記載の読取り装置。
【請求項５】
前記中央処理装置は、前記カードが前記読取り装置へと挿入されると挿入メッセージを前記外部装置に送信する請求項１記載の読取り装置。
【請求項６】
前記中央処理装置は、前記カードが前記読取り装置から抜き取られると抜取りメッセージを前記外部装置に送信する請求項１記載の読取り装置。
【請求項７】
前記中央処理装置は、前記印のうちの少なくとも１つが前記感圧膜を介して押下されると押下メッセージを前記外部装置に送信する請求項１記載の読取り装置。
【請求項８】
前記中央処理装置は、前記押下される印のうちの少なくとも１つの押下が解除されると押下解除メッセージを前記外部装置に送信する請求項１記載の読取り装置。
【請求項９】
前記中央処理装置は、押下が解除されることなく押下位置が移動すると移動メッセージを送信する請求項１記載の読取り装置。
【請求項１０】
前記中央処理装置は、前記不正なカードが前記読取り装置へと挿入されると不良カードメッセージを送信する請求項１記載の読取り装置。
【請求項１１】
前記中央処理装置は、前記読取り装置に電力を供給するバッテリの充電レベルが所定の閾値に到達するとバッテリ低下メッセージを送信する請求項１記載の読取り装置。
【請求項１２】
前記感圧膜上のユーザ押下位置が前記読取り装置に送信される請求項１記載の読取り装置。
【請求項１３】
ベンダにより提供されたアプリケーションにより座標が計算される請求項１２記載の読取り装置。
【請求項１４】
前記カードは前記サービス識別子及び前記データをメモリチップに格納する請求項１記載の読取り装置。
【請求項１５】

前記外部装置はセットトップボックスである請求項１記載の読取り装置。
【請求項１６】
前記外部装置はパーソナルコンピュータである請求項１記載の読取り装置。
【請求項１７】
基板とその上に形成される印とを具備するインタフェースカードを受け入れるレセプタクルを有する読取り装置において実行されるプログラムにおいて、
サービス識別子と、前記カードのメモリ内に格納され、ユーザにより選択される印に関連するデータの特定の部分とを読み出すコードと、
前記サービス識別子と前記特定のデータとを外部装置に送信し、それにより、前記サービス識別子により識別されるサービスを前記外部装置により提供するコードとを具備するプログラム。
【請求項１８】
前記プログラムは、前記読取り装置中のコンピュータ可読な媒体上に格納される請求項１７記載のプログラム。
【請求項１９】
印が形成された基板を具備するインタフェースカードを受け入れるレセプタクルを有する読取り装置のためのデータ処理方法において、
サービス識別子と前記カードのメモリ内に格納され、ユーザにより選択される印に関連するデータの特定の部分とを読み出す過程と、
前記サービス識別子と前記特定のデータとを外部装置に送信し、それにより、前記サービス識別子により識別されるサービスを前記外部装置により提供する過程とから成るデータ処理方法。
【請求項２０】
表面に形成される印とデータを格納するメモリとを具備するインタフェースカードを読む読取り装置であって、前記カードが前記読取り装置へと挿入されるように構成され、
サービス識別子と前記カードに格納され、ユーザにより選択される印に関連するデータの特定の部分とを読み出す読出し手段と、
前記サービス識別子と、前記データとを外部装置に送信する送信手段とを具備し、前記外部装置は前記サービス識別子により識別されるサービスを提供する読取り装置。
【請求項２１】
挿入されるように構成され、少なくとも印が形成された基板を具備するインタフェースカードと前記インタフェースカードを受け入れるレセプタクルを具備する読取り装置とを具備するカードインタフェースシステムにおいて、
サービス識別子とユーザにより選択される印に関連するデータとを格納する前記カード上のメモリと、
前記カードのユーザが前記サービス識別子により識別されるサービスを外部装置から受けるために、前記サービス識別子と前記データとを前記外部装置に送信する、前記読取り装置内に一体的に形成される中央処理装置とを具備するカードインタフェースシステム。
【請求項２２】
データ制御機器を制御するために表面に形成された少なくとも１つの印とデータを格納する電子メモリとを有する電子カードを読む電子カードリーダにおいて、
前記リーダのユーザにより押下可能に構成される上面を有するほぼ透明な感圧膜と、
前記電子カードを受け入れるように形作られるレセプタクルであって、受け入れられる前記電子カード及び前記印が前記感圧膜を介して見ることができるレセプタクルと、
前記印のうちの特定の１つと関連付けられる前記膜の押下に従って前記メモリから前記データの特定の部分を読み出すために前記膜に結合された電子回路であって、前記特定のデータの受信により制御される機能を提供する前記データ制御機器に、前記特定のデータをサービス識別子と共に送信する電子回路とを具備し、前記機能は前記サービス識別子と関連付けられる電子カードリーダ。
【請求項２３】

インタフェースカードのための読取り装置であって、前記データが少なくとも１つの印が形成された基板を具備し、
前記インタフェースカードを受け入れるように形作られるレセプタクルと、
前記レセプタクルに前記インタフェースカードを受け入れたときに前記インタフェースカードに重なるように配置され、これを介して前記印を見ることができるほぼ透明な感圧膜と、
前記インタフェースカードのメモリコンポーネントに結合し、前記膜を介して前記読取り装置のユーザにより選択される前記印のうちの１つに関連するデータを読み出す電子回路とを具備し、前記電子回路は前記データをサービス識別子と共に外部装置に送信するように構成され、それにより、前記外部装置は前記読み出されたデータを受信すると前記サービス識別子と関連付けられるサービスを提供する読取り装置。

【請求項２４】
基板とその上に形成される印とを具備するインタフェースカードを受け入れるレセプタクルを有する読取り装置において、
カードユーザが前記押下された印に従って前記サービス識別子により識別されるサービスを受けるために、前記読取り装置へと挿入された前記カードの有効性を判定し、サービス識別子とユーザにより押下される印に関連するデータとを外部装置に送信する中央処理装置を具備する読取り装置。

【請求項２５】
前記サービスは前記外部装置において実行されるアプリケーションにより識別される請求項２４記載の読取り装置。

【請求項２６】
基板とその上に形成される印とを具備する取外し可能なインタフェースカードに重なるように配置されるほぼ透明な感圧膜を有する読取り装置において、
カードユーザが前記押下された印に従って前記サービス識別子により識別されるサービスを受けるために、前記読取り装置へと挿入された前記カードの有効性を判定し、サービス識別子と、前記印を選択するために前記感圧膜上で押下されるユーザ押下座標とを外部装置に送信する中央処理装置を具備する読取り装置。

【請求項２７】
前記サービスは前記外部装置において実行されるアプリケーションにより識別される請求項２６記載の読取り装置。

【請求項２８】
基板とその上に形成される印とを具備する取外し可能なインタフェースカードに重なるように配置されるほぼ透明な感圧膜を有する読取り装置において、
カードユーザが前記押下された印に従って前記サービス識別子により識別されるサービスを受信するために、前記カードに格納される識別子を区別し、前記区別の結果に基づいて、サービス識別子及び前記印を選択するために前記感圧膜上で押下されるユーザ押下座標、あるいは、サービス識別子及び前記印を選択するために前記感圧膜上で押下されるユーザ押下座標を外部装置に送信する中央処理装置を具備する読取り装置。

【請求項２９】
前記サービスは前記外部装置において実行されるアプリケーションにより識別される請求項２８記載の読取り装置。

【請求項３０】
基板とその上に形成される印とを具備する取外し可能なインタフェースカードに重なるように配置されるほぼ透明な感圧膜を有する読取り装置において、
前記感圧膜上のユーザ押下接触を検知し、前記ユーザ押下接触の接触座標を外部装置に送信する中央処理装置を具備し、前記接触座標はカーソル情報として使用される読取り装置。

【請求項３１】
インタフェースカードを受け入れるレセプタクルを有する読取り装置において、

前記カードが前記読取り装置において実行可能な機能に影響する情報を読み出し、前記情報に基づいて前記機能を実行する中央処理装置を具備する読取り装置。
【請求項３２】
インタフェースカードを受け入れるレセプタクルを有する読取り装置において、
外部装置に挿入されるカードの有効性を判定するために、カード挿入の現在のセッションを識別し、前記カードが前記読取り装置へと挿入されるたびに増分される数値であるセッション識別子を生成し、前記セッション識別子を前記外部装置に送信する中央処理装置を具備する読取り装置。
【請求項３３】
挿入されるように構成されるユーザインタフェースカードを読む読取り装置であって、前記カードが表面に形成される複数の印と識別されるサービスを提供するために外部装置と通信するサービス識別子を格納するメモリとを具備し、
前記カードを受け入れ、前記読取り装置に前記カードを受け入れると前記複数の印の可視性を提供するレセプタクルと、
ユーザが前記複数の可視の印のうちの少なくとも１つを選択できるようにする選択メカニズムと、
前記選択メカニズムに結合され、前記ユーザにより選択される印に関連するデータを生成し、前記サービス識別子を前記メモリから読み出す中央処理装置と、
前記データに関連する前記印と前記サービス識別子とを前記外部装置に送信する送信装置とを具備し、前記外部装置は前記データ及びサービス識別子を受信すると前記サービス識別子及び前記データにより識別されるサービスを提供する読取り装置。
【請求項３４】
前記ユーザにより選択される印に関連するデータの生成は、前記メモリに予め格納される前記選択される印と関連付けられるデータを読み出すことを含む請求項３３記載の読取り装置。
【請求項３５】
挿入されるように構成されるユーザインタフェースカードを読む読取り装置であって、前記カードが表面に形成される複数の印を具備し、
前記カードを受け入れ、前記読取り装置に前記カードを受け入れると前記複数の印の可視性を提供するレセプタクルと、
ユーザが前記複数の可視の印のうちの少なくとも１つを選択できるようにする選択メカニズムと、
前記選択メカニズムに結合され、前記ユーザにより選択される印に関連するデータを生成し、サービスを提供するために外部装置と通信するサービス識別子を前記カードのメモリから読み出す中央処理装置と、
前記印に関連するデータと前記サービス識別子とを前記外部装置に送信する送信装置とを具備し、前記外部装置は前記データ及びサービス識別子を受信すると前記サービス識別子及び前記データにより識別されるサービスを提供する読取り装置。
【請求項３６】
前記ユーザにより選択される印に関連するデータの生成は、前記メモリに予め格納される前記選択される印と関連付けられるデータを読み出すことを含む請求項３５記載の読取り装置。
【発明の詳細な説明】
【０００１】
本発明は、リモートリーダ装置と共に使用される制御テンプレート又はスマートカードに関し、特に、サービスを提供するカードインタフェースシステムに関する。また、本発明はカードインタフェースシステムに対するコンピュータプログラムを記録したコンピュータ可読な媒体を含むコンピュータプログラム製品に関する。
【０００２】
様々な種類の制御パッドが知られており、かなり多岐に渡る分野で使用されている。通常

、このようなパッドは、ユーザが適切な圧力を印加すると信号を発生して関連する制御回路に供給する１つ以上のキー、ボタン、又は感圧領域を含む。

【０００３】

しかし、従来の制御パッドは、キー、ボタン、又は感圧領域の単一構成のみを考慮しているという点で多少制限がある。どの分野でも標準のレイアウトが存在することはまれであり、ユーザは使用する各制御パッドに関して新規のレイアウトについて学ぶことを強いられることが多い。例えば、多くの現金自動預払機（「ＡＴＭ」）及び販売時点電子資金移動（「ＥＦＴＰＯＳ」）装置は、データ入力での要求事項が比較的類似しているにも関わらず、異なるレイアウトを使用している。このため、ユーザは制御パッドごとに押下する必要のあるボタンの位置を判定しなければならず、混乱を招く可能性がある。このような制御パッドは、ユーザの関心、更には、ユーザの使用能力を超える多くのオプションを提供することが多いので問題は悪化することになる。

【０００４】

コンピュータのキーボードなどのためのオーバレイテンプレートが知られている。しかし、これらのテンプレートは設計上柔軟性に乏しく、オーバレイを使用するたびにキーボードが関連付けられているシステムを正確に構成するをユーザに要求している。

【０００５】

既知のシステムの１つは、機器のリモート制御のために設計されたスマートカード読取り装置を伴う。この装置により、例えば、テレビメーカはカードを製造し、このカードをリモート制御筐体及びテレビ受像機と共に供給することができる。顧客は、カードと共に筐体をテレビ受像機用のリモート制御装置として使用することができるようになる。テレビメーカ又はラジオメーカは、自社の製品のために固有のリモート制御装置を製造する必要はないが、リモート制御筐体を固有のカードと共に使用することができる。しかし、上述の概念には、カード上に格納され、関連する装置を制御するのに使用される制御データ（ＰＬＡＹコマンド、ＲＥＣＯＲＤコマンド、ＲＥＷＩＮＤコマンドなど）が装置のメーカによるものであり、そのため、適用が制限されるという欠点がある。

【０００６】

別の既知のシステムは、ビデオ装置、オーディオ装置などをリモート制御するのに使用される「リモートコマンダ（ｒｅｍｏｔｅ　ｃｏｍｍａｎｄｅｒ）」として知られる操作カード読取り装置を伴う。この既知のシステムの操作カードは、リモートコマンダがどのモードで操作しているか及びリモートコマンダからどの制御データが送信されるかを識別するためのカード識別メカニズムを含む。操作カード識別メカニズムは、カードの側面に形成される電極／ノッチ及び操作カード内に格納される識別情報のいずれかの形態をとることができる。操作カード識別メカニズムは、リモートコマンダが識別メカニズムの構成によってビデオテープレコーダ及びテレビ受像機のいずれかのためのコマンドを送信できるように構成することができる。この既知のシステムにも、ビデオテープレコーダ又はテレビを制御するのに使用される制御データ（ＰＬＡＹコマンド、ＲＥＣＯＲＤコマンド、ＲＥＷＩＮＤコマンドなど）が装置のメーカによるものであり、そのため、適用が制限されるという欠点がある。更に、操作カード識別メカニズムは、ユーザが制御対象装置の変更を希望するたびに構成される必要があり、制御対象のビデオ装置、オーディオ装置などとのインタラクションに使用できないように操作カードに限定される。

【０００７】

更に別の既知のスマートカードシステムは、テレビのチャンネルから情報を受信するための光学部品及びリモートサービスプロバイダ上で実行されているアプリケーションとのリアルタイム双方向通信を提供するためのモデムを伴う。この既知のスマートカードシステムは、既存のホームショッピングアプリケーションなどのリモートサービストランザクションに使用される。この既知のシステムによると、ホームショッピングプログラム情報、商品名、商品の明細、商品価格、商品ＣＭ、及びプログラミングの再実行回数を含む情報をスマートカードへとダウンロードすることができる。スマートカードは、スマートカードのモデムと共にアクセス情報を使用してホームショッピングプログラムの自動サービス

コンピュータに自動的にダイヤルし、注文を行なう。しかし、商品購入にスマートカードを使用するたびにアクセス情報をダウンロードしなければならず、商品名及び商品明細により指定される商品を購入する際にしか使用することができないため、このシステムの適用も制限される。

【０００８】

上述のシステムは全てが柔軟性に欠けており、それぞれの適用に制限がある。これらのシステムは、全てが事前に実行されるアプリケーションと共に使用され、メーカによる指定のもの以外のアプリケーションとの間にはインタラクションがない。

【０００９】

本発明の目的は、既存の構成の１つ以上の欠点をほぼ克服するか、あるいは、少なくとも改善することである。

【００１０】

本発明の一面によると、読取り装置へと挿入されるように構成されるインタフェースカードを読む読取り装置であって、前記カードが表面に形成される印と外部装置と通信するためにデータを格納するメモリとを具備し、
前記読取り装置に前記インタフェースカードを受け入れると前記カードに重なるように配置されるほぼ透明な感圧膜と、
サービス識別子とユーザにより選択される印に関連するデータの特定の部分とを前記外部装置に送信する中央処理装置とを具備し、前記外部装置は前記データを受信すると前記サービス識別子により識別されるサービスを提供する読取り装置が提供される。

【００１１】

本発明の別の面によると、基板とその上に形成される印とを具備するインタフェースカードを受け入れるレセプタクルを有する読取り装置において実行されるプログラムにおいて、
サービス識別子と、前記カードのメモリ内に格納され、ユーザにより選択される印に関連するデータの特定の部分とを読み出すコードと、
前記サービス識別子と前記特定のデータとを外部装置に送信し、それにより、前記サービス識別子により識別されるサービスを前記外部装置により提供するコードとを具備するプログラムが提供される。

【００１２】

本発明の更に別の面によると、印が形成された基板を具備するインタフェースカードを受け入れるレセプタクルを有する読取り装置のためのデータ処理方法において、
サービス識別子と前記カードのメモリ内に格納され、ユーザにより選択される印に関連するデータの特定の部分とを読み出す過程と、
前記サービス識別子と前記特定のデータとを外部装置に送信し、それにより、前記サービス識別子により識別されるサービスを前記外部装置により提供する過程とから成るデータ処理方法が提供される。

【００１３】

本発明の更に別の面によると、表面に形成される印とデータを格納するメモリとを具備するインタフェースカードを読む読取り装置であって、前記カードが前記読取り装置へと挿入されるように構成され、
サービス識別子と前記カードに格納され、ユーザにより選択される印に関連するデータの特定の部分とを読み出す読出し手段と、
前記サービス識別子と、前記データとを外部装置に送信する送信手段とを具備し、前記外部装置は前記サービス識別子により識別されるサービスを提供する読取り装置が提供される。

【００１４】

本発明の更に別の面によると、挿入されるように構成され、少なくとも印が形成された基板を具備するインタフェースカードと前記インタフェースカードを受け入れるレセプタクルを具備する読取り装置とを具備するカードインタフェースシステムにおいて、

サービス識別子とユーザにより選択される印に関連するデータとを格納する前記カード上のメモリと、
前記カードのユーザが前記サービス識別子により識別されるサービスを外部装置から受けるために、前記サービス識別子と前記データとを前記外部装置に送信する、前記読取り装置内に一体的に形成される中央処理装置とを具備するカードインタフェースシステムが提供される。

【００１５】

本発明の更に別の面によると、データ制御機器を制御するために表面に形成された少なくとも１つの印とデータを格納する電子メモリとを有する電子カードを読む電子カードリーダにおいて、
前記リーダのユーザにより押下可能に構成される上面を有するほぼ透明な感圧膜と、
前記電子カードを受け入れるように形作られるレセプタクルであって、受け入れられる前記電子カード及び前記印が前記感圧膜を介して見ることができるレセプタクルと、
前記印のうちの特定の１つと関連付けられる前記膜の押下に従って前記メモリから前記データの特定の部分を読み出すために前記膜に結合された電子回路であって、前記特定のデータの受信により制御される機能を提供する前記データ制御機器に、前記特定のデータをサービス識別子と共に送信する電子回路とを具備し、前記機能は前記サービス識別子と関連付けられる電子カードリーダが提供される。

【００１６】

本発明の更に別の面によると、インタフェースカードのための読取り装置であって、前記データが少なくとも１つの印が形成された基板を具備し、
前記インタフェースカードを受け入れるように形作られるレセプタクルと、
前記レセプタクルに前記インタフェースカードを受け入れたときに前記インタフェースカードに重なるように配置され、これを介して前記印を見ることができるほぼ透明な感圧膜と、
前記インタフェースカードのメモリコンポーネントに結合し、前記膜を介して前記読取り装置のユーザにより選択される前記印のうちの１つに関連するデータを読み出す電子回路とを具備し、前記電子回路は前記データをサービス識別子と共に外部装置に送信するように構成され、それにより、前記外部装置は前記読み出されたデータを受信すると前記サービス識別子と関連付けられるサービスを提供する読取り装置が提供される。

【００１７】

本発明の更に別の面によると、基板とその上に形成される印とを具備するインタフェースカードを受け入れるレセプタクルを有する読取り装置において、
カードユーザが前記押下された印に従って前記サービス識別子により識別されるサービスを受けるために、前記読取り装置へと挿入された前記カードの有効性を判定し、サービス識別子とユーザにより押下される印に関連するデータとを外部装置に送信する中央処理装置を具備する読取り装置が提供される。

【００１８】

本発明の更に別の面によると、基板とその上に形成される印とを具備する取外し可能なインタフェースカードに重なるように配置されるほぼ透明な感圧膜を有する読取り装置において、
カードユーザが前記押下された印に従って前記サービス識別子により識別されるサービスを受けるために、前記読取り装置へと挿入された前記カードの有効性を判定し、サービス識別子と、前記印を選択するために前記感圧膜上で押下されるユーザ押下座標とを外部装置に送信する中央処理装置を具備する読取り装置が提供される。

【００１９】

本発明の更に別の面によると、基板とその上に形成される印とを具備する取外し可能なインタフェースカードに重なるように配置されるほぼ透明な感圧膜を有する読取り装置において、
カードユーザが前記押下された印に従って前記サービス識別子により識別されるサービス

を受信するために、前記カードに格納される識別子を区別し、前記区別の結果に基づいて、サービス識別子及び前記印を選択するために前記感圧膜上で押下されるユーザ押下座標、あるいは、サービス識別子及び前記印を選択するために前記感圧膜上で押下されるユーザ押下座標を外部装置に送信する中央処理装置を具備する読取り装置が提供される。

【００２０】

本発明の更に別の面によると、基板とその上に形成される印とを具備する取外し可能なインタフェースカードに重なるように配置されるほぼ透明な感圧膜を有する読取り装置において、

前記感圧膜上のユーザ押下接触を検知し、前記ユーザ押下接触の接触座標を外部装置に送信する中央処理装置を具備し、前記接触座標はカーソル情報として使用される読取り装置が提供される。

【００２１】

本発明の更に別の面によると、インタフェースカードを受け入れるレセプタクルを有する読取り装置において、

前記カードが前記読取り装置において実行可能な機能に影響する情報を読み出し、前記情報に基づいて前記機能を実行する中央処理装置を具備する読取り装置が提供される。

【００２２】

本発明の更に別の面によると、インタフェースカードを受け入れるレセプタクルを有する読取り装置において、

外部装置に挿入されるカードの有効性を判定するために、カード挿入の現在のセッションを識別し、前記カードが前記読取り装置へと挿入されるたびに増分される数値であるセッション識別子を生成し、前記セッション識別子を前記外部装置に送信する中央処理装置を具備する読取り装置が提供される。

【００２３】

本発明のその他の面も開示される。

【００２４】

【本発明の実施の形態】

図面を参照しながら本発明の１つ以上の実施形態を説明する。

【００２５】

添付のいずれか１つ以上の図面において、同じ符号を有するステップ及び／又は特徴に言及する場合、特に明記されていない限り、以下の記述においては、これらのステップ及び／又は特徴は、同じ機能又は動作を有するものとする。

【００２６】

以下に開示される各実施形態は、主に、リモート制御システム、現金自動預払機、ビデオゲームコントローラ、及びネットワークアクセスと共に使用されるべく開発されたものであり、これらの適用例及びその他の適用例を参照して説明する。しかし、本発明がこれらの使用分野に限定されないことは明らかであろう。

【００２７】

説明を簡単にする目的で、以下の説明では、第１．０節（ｓｅｃｔｉｏｎ）から第１３．０節に分割する。各節は関連する項（ｓｕｂｓｅｃｔｉｏｉｎ）を有する。

【００２８】

１．０　カードインタフェースシステムの概要

図１は、カードレセプタクル４及び表示領域６を定義する筐体２を有するリモートリーダ１を示す。データ読取り手段が、露出した電気接点７及び関連する制御回路（不図示）の形態で設けられる。また、リモートリーダ１は、表示領域６を覆うタッチパネル８を形成するほぼ透明な感圧膜の形態のセンサ手段を含む。ここで開示されるリモートリーダ１は、タッチパネル８を形成するほぼ透明な感圧膜を用いて説明されるが、代替の技術をほぼ透明なタッチパネルとして使用できることが当業者には明らかであろう。例えば、タッチパネルは電気抵抗の高いものにすることも、感温性のものにすることもできる。リモートリーダ１は、ユーザインタフェースカードと共に使用するように構成される。図１から図

３に示すカードにおいて、ユーザインタフェースカードは電子スマートカード１０Ａの形態をとる。スマートカード１０Ａは、４方向コントローラ２０、上面１６に印刷された「ジャンプボタン」２２、「キックボタン」２４、「開始ボタン」、及び「終了ボタン」の形態の種々の制御印１４を有する積層基板１２を含む。プロモーション素材又は教示的な素材などの他の非制御印を制御印の横に印刷することができる。例えば、図２に示すように、広告素材２６をスマートカード１０Ａの前面又は裏面２７に印刷することができる。

【００２９】

図３に示すように、スマートカード１０Ａは制御印と関連付けられるデータに対してオンボードメモリチップ１９の形態の記憶手段を含む。また、スマートカード１０Ａは、オンボードメモリチップ１９に接続されると共にリモートリーダ１上の露出した接点７に対応する電気データ接点１８を含む。

【００３０】

図３に示すように、上面１６は、制御印１４が印刷された粘着ラベル６０により形成されても良い。ここで、制御印１４は「終了ボタン」及び方向コントローラ２０の右矢印「ボタン」に対応する。ラベル６０は積層基板１２に貼付される。一般家庭のユーザは、Ｃａｎｏｎ，　Ｉｎｃ製のカラーＢＵＢＢＬＥＪＥＴ（登録商標）プリンタなどのプリンタを使用することによって、特定のスマートカード１０Ａと共に使用するための適切なラベルを印刷することができる。また、制御印１４を積層基板に直接印刷することも、各制御印に別々の粘着ラベルを使用することもできる。

【００３１】

使用の際、スマートカード１０Ａはカードレセプタクル４へと挿入されるので、感圧タッチパネル８はスマートカード１０Ａの上面１６を覆うようになる。このとき、制御印は透明な感圧タッチパネル８を通して表示領域６内に見えることになる。

【００３２】

リーダ１の露出した接点７及び関連回路は、スマートカード１０Ａの制御テンプレートレセプタクル４への挿入時に自動的に、あるいは、リモートリーダ１からの信号に応答して選択的に、メモリチップ１９から制御印１４と関連付けられた格納データを読むように構成される。信号は、例えば、露出した接点７及びデータ接点１８を介してスマートカード１０Ａに送信することができる。

【００３３】

制御印１４と関連付けられたデータが一度読み取られると、ユーザは感圧タッチパネル８の制御印１４に重なる各領域を押下することができる。感圧タッチパネル８にかかる圧力を検知し、格納データを参照することによって、リモートリーダ１はユーザが選択したのがどの制御印１４であるかを推定することができる。例えば、ユーザが感圧タッチパネル８の「キックボタン」２４の付近に圧力をかける場合、リモートリーダ１は圧力が印加された位置を推定し、格納データを参照し、「キック」ボタン２４が選択されたと判定するように構成される。この情報を使用して、関連するビデオゲームコンソール（従来の構成のもの、不図示）などの外部装置を制御することができる。以上の説明から、制御印１４は実際にはボタンでないことが認識されるであろう。制御印１４は、タッチパネル８のマッピングデータ及び機能との対応する結び付きによって、リモート制御装置と従来から関連付けられてきたボタンをエミュレートするように動作するユーザ選択可能な機能である。

【００３４】

１つの有利な実現例において、リモートリーダ１は、ユーザにより選択される印に関連する情報を送信するための赤外線（ＩＲ）送信機又は無線周波数（ＲＦ）送信機などの送信機（従来の構成のもの、不図示）を含む。図１に示すように、リモートリーダ１はＩＲ発光ダイオード（ＬＥＤ）２５を有するＩＲ送信機を組み込む。制御印１４のうちの１つを選択すると、リモートリーダ１は選択された印に関連する情報をリモートコンソール（図１では示されていない）に送信させる。リモートコンソールでは、対応するＩＲ受信機又はＲＦ受信機により、リーダ１のユーザがプレーするゲームなどの何らかの機能を制御す

る際に使用される情報を検知・復号化することができる。
【００３５】
直接ハード配線を含む任意の適切な送信方法を使用してリモートリーダ１からリモートコンソールへと情報を送信することができる。更に、先に説明したのと逆の方向に送信するために、リモートコンソール自体が送信機を組み込み、リモートリーダ１が受信機を組み込むこともできる。リモートコンソールからリモートリーダ１への送信は、例えば、ハンドシェーキングデータ、セットアップ情報、又はリモートコンソールからリモートリーダ１に送信することが望まれる任意の形態の情報を含むことができる。
【００３６】
図４に戻ると、制御カード１０Ｂが示される。制御カード１０Ｂは、制御印（不図示）を有する積層基板１２を含む。制御カード１０Ｂにおいて、記憶手段は制御カード１０Ｂの裏面２７の縁部２８に沿って形成される磁気ストリップ２９の形態をとる。制御印と関連付けられる格納データは、従来のように磁気ストリップ２９に格納されても良い。この構成の対応するリーダ（不図示）は、対応する制御テンプレートレセプタクルへの入口部又はその付近に位置する磁気読取りヘッドを含む。制御カード１０Ｂがカードレセプタクルへと差し込まれるとき、格納データは磁気読取りヘッドにより磁気ストリップ２９から自動的に読み取られる。リーダ１は図１のカード１０Ａに対応する方法で操作されても良い。
【００３７】
図５は制御カード１０Ｃの形態の別のカードを示す。制御カード１０Ｃにおいて、記憶手段は機械読取り可能な印の形態をとる。図５のカード１０Ｃにおいて、機械読取り可能な印は、カード１０Ｃの裏面２７の縁部３８に沿って形成されるバーコード３６の形態をとる。格納データは符号化されているのが好ましく、図示する位置に印刷される。図５のカード１０Ｃの対応するコントローラ（不図示）は、関連する制御テンプレートレセプタクルへの入口部又はその付近に位置する光学読取りヘッドを含むことになる。カード１０Ｃがカードレセプタクルへと差し込まれるとき、格納データは光学読取りヘッドによりバーコード３６から自動的に読み取られる。また、バーコードは、カード１０Ｃを挿入する直前にリーダと関連するバーコードリーダを使用して走査することも、カード１０Ｃが完全に挿入されてから内部のバーコードリーダスキャナにより走査することもできる。カード１０Ｃは、その後図１のカード１０Ａに対応する方法で操作されても良い。バーコードの位置、向き、及び符号化は、特定のアプリケーションに適合するように変更可能であることが認識されるであろう。更に、エンボス加工の機械読取り可能な図形、印刷された英数字、パンチングやその他の方法で形成された切抜き、光学的な印又は光磁気の印、２次元バーコードを含むその他の任意の形態の機械読取り可能な印を使用することができる。
【００３８】
図６（ａ）は、カードインタフェースシステム６００Ａのハードウェアアーキテクチャを示す。システム６００Ａにおいて、リモートリーダ１は通信ケーブル３を介してパーソナルコンピュータシステム１００に配線接続される。あるいは、配線接続の代わりに無線周波数送受信機又はＩＲ送受信機１０６を使用して、リモートリーダ１と通信を行なうことができる。パーソナルコンピュータシステム１００は画面１０１及びコンピュータモジュール１０２を含む。パーソナルコンピュータシステム１００については、図７を参照して以下で更に詳細に説明する。キーボード１０４及びマウス２０３も設けられる。
【００３９】
システム６００Ａは、上述のスマートカード１０Ａと同様の構成のスマートカード１０Ｄを含む。スマートカード１０Ｄはプログラム可能であり、サードパーティにより作製又はカスタマイズすることができる。この場合、サードパーティは、カード１０Ｄ及び／又はカードリーダ１のメーカ以外の業者であっても良い。サードパーティは、スマートカード１０Ｄ自体の最終ユーザであっても、メーカとユーザとの間の仲介業者であっても良い。図６（ａ）のシステム６００Ａによると、スマートカード１０Ｄは１回の接触操作でコンピュータ１００と通信を行ない、インターネットなどのネットワーク２２０を介してサー

ビスを取得するようにプログラム・カスタマイズすることができる。コンピュータ１００は、特定のプロトコルに従ってリモートリーダ１から通信ケーブル３を介して送信される信号を解釈するように動作する。このプロトコルについては詳細に後述する。コンピュータ１００は、触れられた制御印に従って選択された機能を実行し、ネットワーク２２０を介してデータを送受信するように構成することができる。このように、コンピュータ１００は、リモートサーバコンピュータ１５０、１５２上に格納されたアプリケーション及び／又はデータへのアクセスを許可し、表示装置１０１での適切な再現を可能にすることができる。

【００４０】

図６（ｂ）は、カードインタフェースシステム６００Ｂのハードウェアアーキテクチャを示す。システム６００Ｂにおいて、出力インタフェースに接続するセットトップボックス６０１においてローカルにサービスを取得するためにリモートリーダ１をプログラムすることができる。出力インタフェースは、ここではデジタルテレビ受信機などの音声映像出力装置１１６の形態をとる。セットトップボックス６０１は、リモートリーダ１から特定のプロトコル（詳細に後述）に従って受信される信号１１２を解釈するように動作する。この信号１１２は、電気信号であっても、無線周波数信号であっても、あるいは赤外線（ＩＲ）信号であっても良い。セットトップボックス６０１は、触れられた制御印に従って選択された機能を実行し、出力装置１１６での適切な再現を可能にするように構成することができる。また、セットトップボックス６０１は、信号１１２を通信に適した形態に変換し、コンピュータ１００への適切な送信を行なうように構成することができる。このため、コンピュータ１００は、制御印に従って選択された機能を実行し、セットトップボックス６０１にデータを提供することで出力装置１１６での適切な再現を可能にすることができる。セットトップボックス６０１については、図４２を参照して以下で詳細に説明する。

【００４１】

システム６００Ｂの１つの適用例において、スマートカード１０Ｄはサービスをリモートで又はローカルに取得するためにプログラムすることができる。例えば、スマートカード１０Ｄは、リモートサーバコンピュータ１５０、１５２上に格納されたアプリケーション及び／又はデータをネットワーク２２０を介して取得し、これをセットトップボックス６０１へとロードするようにプログラムすることができる。後者の場合、カードはセットトップボックス６０１上のロードされたアプリケーションからサービスを取得するようにプログラムすることもできる。

【００４２】

特に明記する場合を除き、以降、システム６００Ａ及び６００Ｂをシステム６００と総称的に呼ぶ。更に、特に明記する場合を除き、以降、スマートカード１０Ａ、１０Ｂ、１０Ｃ、及び１０Ｄを総称的にスマートカード１０と呼ぶ。

【００４３】

図７は、システム６００の汎用コンピュータシステム１００を示す。このシステムを使用してカードインタフェースシステムを実行させると共に、スマートカード１０をプログラムするためのソフトウェアアプリケーションを実行させることができる。コンピュータシステム１００は、コンピュータモジュール１０２、キーボード１０４及びマウス２０３などの入力装置、並びにプリンタ（不図示）及び表示装置１０１を含む出力装置を具備する。変復調（モデム）送受信装置２１６は、コンピュータモジュール１０２が、例えば、電話回線２２１又はその他の機能媒体を介して接続可能な通信ネットワーク２２０と通信を行なうのに使用される。モデム２１６は、インターネット及び構内通信網（ＬＡＮ）又は広域通信網（ＷＡＮ）などのその他のネットワークシステムへのアクセス権を取得するのに使用することができる。

【００４４】

コンピュータモジュール１０２は、通常、少なくとも１つの中央処理装置（ＣＰＵ）２０５、例えば、半導体ランダムアクセスメモリ（ＲＡＭ）及び読出し専用メモリ（ＲＯＭ）

から形成されるメモリユニット２０６、ビデオインタフェース２０７とキーボード１０４及びマウス２０３用のＩ／Ｏインタフェース２１３とを含む入出力（Ｉ／Ｏ）インタフェース、書込み装置２１５、並びにモデム２１６用のインタフェース２０８を含む。記憶装置２０９が設けられるが、この記憶装置２０９は、通常、ハードディスクドライブ２１０及びフロッピーディスクドライブ２１１を含む。磁気テープドライブ（不図示）を使用しても良い。ＣＤ－ＲＯＭドライブ２１２は、通常、不揮発性のデータソースとして設けられる。コンピュータモジュール２０１の構成要素２０５から２１３は、通常、相互接続バス２０４を介して、コンピュータシステム１０２の動作モードが、当業者には既知の従来の動作モードになるように通信を行なう。上述の構成を実施可能なコンピュータの例としては、ＩＢＭのコンピュータ及びその互換機、Ｓｕｎ Ｓｐａｒｃｓｔａｔｉｏｎ、又はそれを進化させた同様のコンピュータシステムがある。

【００４５】

通常、システム６００のソフトウェアプログラムはハードディスクドライブ２１０に常駐し、ＣＰＵ２０５による実行の際に読み出されて制御される。ソフトウェアアプリケーションプログラムとネットワーク２２０から取り込まれるデータとの中間記憶装置は、半導体メモリ２０６を使用し、場合によってはハードディスクドライブ２１０と協働させるように使用して達成されても良い。アプリケーションプログラムは、ＣＤ－ＲＯＭ又はフロッピーディスク上に符号化された形でユーザに供給され、対応するドライブ２１２又は２１１を介して読み取られる場合もあれば、ネットワーク２２０からモデム装置２１６を介してユーザが読み取る場合もある。更に、ソフトウェアは、磁気テープ、ＲＯＭ又は集積回路、光磁気ディスク、コンピュータモジュール１０２と別の装置との間の無線又は赤外線の伝送チャネル、スマートカード及びコンピュータのＰＣＭＣＩＡカードなどのコンピュータ可読なカード、並びに電子メール送受信及びウェブサイトなどに記録された情報などを含むインターネット又はイントラネットを含む他のコンピュータ可読な媒体からコンピュータシステム１０２へとロードすることもできる。以上の例は、関連するコンピュータ可読な媒体の一例に過ぎない。添付の請求の範囲により定義される本発明の趣旨から逸脱することなく、その他のコンピュータ可読な媒体も実施することができる。

【００４６】

スマートカード１０は、コンピュータモジュール１０２のＩ／Ｏインタフェース２１３に接続される書込み装置２１５を用いてプログラムすることができる。書込み装置２１５は、スマートカード１０上のメモリにデータを書き込む機能をもつことができる。また、書込み装置２１５は、スマートカード１０の上面に図形を印刷する機能を有するのが好ましい。書込み装置２１５は、スマートカード１０上のメモリからデータを読み取る機能ももつことができる。まず、ユーザはスマートカード１０を書込み装置２１５へと挿入する。ユーザが汎用コンピュータ１０２のキーボード１０４を介して所要のデータを入力すると、ソフトウェアアプリケーションが書込み装置２１５を介してこのデータをスマートカードのメモリに書き込む。格納データがバーコードの使用などの光学的な復号化用に符号化されている場合、書込み装置は符号化されたデータをスマートカード１０に印刷することができる。

【００４７】

図４２は、システム６００のセットトップボックス６０１を示す。このセットトップボックス６０１を使用してリモートリーダ１から受信された信号１１２を解釈することができる。一部の実現例では、セットトップボックス６０１は実質的にコンピュータモジュール１０２の規模を変更したものとなる。セットトップボックス６０１は、通常、少なくとも１つのＣＰＵユニット４３０５、例えば、半導体ランダムアクセスメモリ（ＲＡＭ）及び読出し専用メモリ（ＲＯＭ）から形成されるメモリユニット４３０６、並びにデジタルテレビ１１６用のＩ／Ｏインタフェース４３１３、信号１１２を送受信するためのＩＲ送受信機を有するＩ／Ｏインタフェース４３０８、ネットワーク２２０に接続するためのＩ／Ｏインタフェース４３１７とを少なくとも含む入出力（Ｉ／Ｏ）インタフェースを含む。セットトップボックス６０１の構成要素４３０５、４３０６、４３１３、４３１５、及び

４３１７は、通常、相互接続バス４３０４を介して、動作モードが従来の動作モードになるように通信を行なう。リモートリーダ１又はネットワーク２２０から受信されるデータの中間記憶装置は、半導体メモリ４３０６を使用して達成されても良い。また、セットトップボックスは記憶装置２０９と同様の記憶装置（不図示）を含むことができる。

【００４８】

カードインタフェースシステム６００については、以下の段落で詳細に説明する。

【００４９】

２．０　カードインタフェースシステムソフトウェアアーキテクチャ

２．１　ソフトウェアアーキテクチャのレイアウト

システム６００により表されるハードウェアアーキテクチャに対するソフトウェアアーキテクチャ２００がほぼ図８に示される。アーキテクチャ２００は、幾つかの別個のプロセスコンポーネント及び１つのプロセスクラスへと分割することができる。別個のプロセスには、口語で「Ｉ／Ｏデーモン」３００とも呼ばれるＩ／Ｏインタフェース３００、イベントマネージャ３０１、ディスプレイマネージャ３０６、（アプリケーション）ランチャ３０３、及びディレクトリサービス３１１が含まれる。プロセスクラスは１つ以上のアプリケーション３０４により形成される。ここで説明するアーキテクチャ２００には、セットトップボックス６０１により通常形成されるスマートカードリモート接続ごとに、１つのＩ／Ｏデーモン３００、１つのイベントマネージャ３０１、１つのディスプレイマネージャ３０６、及び１つのランチャ３０３と、各ランチャ３０３を稼動させているコンピュータ１００（例えば、サーバコンピュータ１５０、１５２）ごとに１つのマスタランチャ（不図示）と、全てのシステムに対して少なくとも１つのディレクトリサービス３１１とが存在する。ランチャ３０３は、サービスの名前又は位置あるいはそのサービスに対して使用されるアプリケーション３０４の名前又は位置を示すリソースロケータ（ＵＲＬなど）へとサービスデータを変換するためにディレクトリサービス３１１に問合せを行なう。

【００５０】

アーキテクチャ２００は、図８の破線により示されるように、６つの別個の部分１０１、３０７、３０９、３１２、３１３、及び６０１へと物理的に分離することができる。各部分は物理的に別々の計算装置上で実行させることができる。システム６００の各部分間の通信は、伝送制御プロトコル／インターネットプロトコル（ＴＣＰ／ＩＰ）ストリームを使用して実行される。また、各部分１０１、３０７、３０９、３１２、３１３、及び６０１は、同じマシン上で実行させることもできる。

【００５１】

図６（ａ）のシステム６００Ａにおいて、プロセスコンポーネント３００、３０１、３０３、３０４、及び３０６の全てはコンピュータ１００上で実行させることができる。イベントマネージャ３０１、ランチャ３０３、及びディスプレイマネージャ３０６は、コンピュータ１００のハードディスク２０９に格納され、ＣＰＵ２０５による実行時に読み出して制御することができる１つの実行可能なプログラムへと統合されるのが好ましい。ディレクトリサービス３１１は、同じコンピュータ１００又はネットワーク２２０を介してコンピュータ１００に接続される別のコンピュータ（サーバ１５０など）上で実行される。

【００５２】

図６（ｂ）のシステム６００Ｂにおいて、コンポーネント３００から３０４及び３０６の全てはセットトップボックス６０１から実行させることができる。この例において、コンポーネント３００から３０４及び３０６はセットトップボックス６０１のメモリ４３０６に格納することができ、ＣＰＵ４３０５による実行時に読み出して制御することができる。ディレクトリサービス３１１は、コンピュータ１００上で実行させることができると共にコンピュータ１００のメモリ２０６に格納することができる。ディレクトリサービス３１１はＣＰＵ２０５による実行時に読み出して制御することができる。また、ディレクトリサービス３１１をセットトップボックス６０１上で実行させたり、あるいは、その機能をランチャ３０３により実行させたりすることもできる。

【００５３】

セットトップボックス６０１がシステム６００をローカルに実行させるのに十分な能力をもたない場合、Ｉ／Ｏデーモン３００のみをセットトップボックス６０１上で実行させれば良く、アーキテクチャ２００のその他の部分（すなわち、プロセスコンポーネント３０１、３０３、３０４、３０６、及び３１１）は他のサービス（１５０、１５２）上においてリモートで実行させることができる。他のサービスには、ネットワーク２２０を介してアクセスすることができる。この例では、Ｉ／Ｏデーモン３００はセットトップボックス６０１のメモリ４３０６に格納することができ、ＣＰＵ４３０５による実行の際に読み出して制御することができる。このようなシステムの機能部分も図８に示すように分割することができる。

【００５４】

２．１．１　Ｉ／Ｏデーモン

Ｉ／Ｏデーモン３００は、リモートリーダ１から受信されたデータグラムをＴＣＰ／ＩＰストリームへと変換するプロセスコンポーネントである。このＴＣＰ／ＩＰストリームは、イベントマネージャ３０１へと送信することができると同時にイベントマネージャ３０１から受信することもできる（例えば、双方向プロトコルを使用するとき）。リモートリーダ１は任意の適切なデータ形式を使用することができる。Ｉ／Ｏデーモン３００はリモートリーダ１のデータ形式の変更に影響されないのが好ましく、リモートリーダ１の複数の構成に対処することができる。システム６００の１つの有利な実現例では、Ｉ／Ｏデーモン３００はイベントマネージャ３０１へと統合される。

【００５５】

システム６００Ａにおいて、ユーザがコンピュータ１００を起動することによってスマートカードシステム６００を開始し、イベントマネージャ３０１が開始されているとき、Ｉ／Ｏデーモン３００が開始される。また、ユーザがセットトップボックス６０１の電源を入れることによってシステム６００を開始するとき、Ｉ／Ｏデーモン３００が開始される。

【００５６】

Ｉ／Ｏデーモン３００については第９．０節を参照して以下で更に詳細に説明する。

【００５７】

２．１．２　イベントマネージャ

イベントマネージャ３０１は、全ての通信がイベントマネージャ３０１を経由するという点でアーキテクチャ２００の中心を形成する。イベントマネージャ３０１は、リモートリーダ１により生成され、Ｉ／Ｏデーモン３００により中継される全てのイベントを収集するように構成される。これらのイベントは、種々のプロセスコンポーネント３００から３０４及び３０６並びに実行中の各アプリケーションへと再分配される。また、イベントマネージャ３０１は、イベントが有効なヘッダ、正しいデータ長を有することをチェックするように構成されるが、通常、イベントが正しい形式であることをチェックするようには構成されていない。ここでの「イベント」は、Ｉ／Ｏデーモン３００、ランチャ３０３、又はアプリケーション３０４からの単一のデータトランザクションを表す。

【００５８】

異なるシステム間でのプロトコルの変更にはイベントマネージャ３０１が対処する。可能な場合には、現在実行中のアプリケーション３０４により理解されるデータ形式に適合するようにイベントを書き換えることができる。書き換えが可能でない場合、イベントマネージャ３０１は送信元のアプリケーション３０４にエラーを報告する。例えば、複数枚のスマートカードを稼働させるシステムにおいて、異なるデータ形式が使用されている場合、イベントマネージャ３０１は混乱の発生を最小限に抑えることを保証するのが好ましい。

【００５９】

イベントマネージャ３０１は、表示画面又はその他の出力装置１１６にはその存在を示さない。しかし、イベントマネージャ３０１は、どのアプリケーションが現在必要とされており（すなわち、「フロント」アプリケーション）、現在画面１０１に表示されるべきで

あるかについてディスプレイマネージャ３０６に指示するように構成することができる。イベントマネージャ３０１は、ランチャ３０３からアプリケーション３０４へと渡されるメッセージからこの情報を推定する。これについては、第１０．０節を参照して以下で詳細に説明する。

【００６０】

イベントマネージャ３０１は、Ｉ／Ｏデーモン接続の受信の有無を常に聞き取るように構成することも、あるいは、システム６００を開始させることもできる。使用される方法はシステム６００の全体的な構成によって決まる。イベントマネージャ３０１がシステム６００を開始させることができるか、あるいは、セットトップボックス６０１がＩ／Ｏデーモン３００の接続の受信を使用してシステム６００を開始させることができる。イベントマネージャ３０１については、第７．０節を参照して以下で更に詳細に説明する。

【００６１】

２．１．３　マスタランチャ

シン（ｔｈｉｎ）クライアントコンピュータが利用され、各々が１つのセットトップボックスを担当する複数のランチャ３０３が実行されている場合、イベントマネージャ３０１と直接通信を行なうマスタランチャ（不図示）を使用することができる。マスタランチャは、２つ以上のイベントマネージャ３０１がシステム６００上で実行されている場合に、各イベントマネージャ３０１に対応するランチャ３０３を開始するのに使用される。まず、Ｉ／Ｏデーモン３００がイベントマネージャ３０１に接続するとき、マスタランチャがイベントマネージャ３０１に対して第１のプロセスを開始するよう、イベントマネージャ３０１が要求する。この第１のプロセスは、通常、任意のスマートカードアプリケーション３０４に対するランチャ３０３である。また、マスタランチャは、イベントマネージャ３０１が要求するときにはアプリケーション３０４のランチャ３０３を停止し、イベントマネージャ３０１にランチャ３０３が終了したことを通知するように構成することができる。

【００６２】

関連するスマートカードアプリケーション３０４を実行中の物理的に別々のサーバ（例えば、１５０、１５２）ごとに１つのマスタランチャが実行されるのが好ましい。この１つのマスタランチャは特定のサーバ上でランチャを要求する全てのイベントマネージャに対する要求を扱う。図７に示すようにコンピュータ１００上で実行されるときには、マスタランチャはシステム６００のその他の部分より前又はそれと同時に動作を開始する。この例では、マスタランチャは最初に開始される。

【００６３】

マスタランチャは、例えば、関連するランチャがイベントマネージャ３０１と同じコンピュータ上で実行されているときには、イベントマネージャ３０１へと統合することができる。

【００６４】

２．１．４　ランチャ／第１のアプリケーション

システム６００の１つの有利な実現例では、スマートカード１０のリモートリーダ１への挿入により開始される第１のプロセスはランチャ３０３である。特定のシステムでは特定のアプリケーションが開始されるであろう。例えば、自動現金預払機はバンキングアプリケーションを開始することができる。別の例としては、特定の１組のアプリケーションのみを開始する限定されたランチャの使用が含まれる。ランチャ３０３は、特定のイベントマネージャ３０１に対して他のアプリケーションを開始するアプリケーションである。ランチャ３０３はアプリケーションを開始／終了させることができると共に、各セッションを開始／終了させることができる。また、ランチャ３０３はイベントマネージャ３０１にいつアプリケーションが開始／終了するかを通知し、アプリケーション３０４にいつフォーカスを受け取る／失うか、あるいは、いつ終了する必要があるかを通知する。これに関して、複数のアプリケーション３０４が同時に動作している場合、現在画面に表示されているアプリケーション３０４がフォーカスを有し、「フロントアプリケーション」として

知られるアプリケーションである。別のアプリケーションが優先権を得ようとすると、ランチャ３０３はフロントアプリケーションにフォーカスが失われることを通知するので、現在のアプリケーションは当座のタスクを完了することができる。また、ランチャ３０３は新規のアプリケーション３０４にフォーカスが得られ、まもなくアプリケーション３０４が変更状態に入ることを通知する。ランチャ３０３はアプリケーションを強制的に終了するようにも構成される。

【００６５】

ランチャ３０３は、リモートリーダ１が生成する「カードなし」、「バッテリの電力低下」、及び「不良カード」などのイベントを受信しても良い。また、ランチャ３０３は、現在フロントアプリケーションでない各アプリケーション用のイベントも受信し、これらのイベントを正確に解釈するように動作する。

【００６６】

ランチャ３０３は、イベントマネージャ３０１によりその開始を要求する要求が生成されるときにのみ開始されるのが好ましい。また、ランチャ３０３は、イベントマネージャ３０１により終了するように指示したり、強制的に終了したりすることもできる。

【００６７】

ランチャ３０３はディレクトリサービス３１１と通信を行なう必要がある唯一のプロセスコンポーネントであるのが好ましい。新規のアプリケーション３０４を開始する必要があるとき、ランチャ３０３はサービスデータを有するディレクトリサービス３１１に問合せを行なう。ディレクトリサービス３１１は、アプリケーション３０４の位置及び新規のアプリケーション３０４と関連付けられるサービスデータを戻す。サービスデータは、ここでは、ＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントと呼ばれるイベント中の初期化データとして新規のアプリケーション３０４に送信される。アプリケーション位置は実行対象のアプリケーション３０４の位置を指定する。これは、ローカルコンピュータを用いた実現例ではローカルなものであっても良く、あるいは、ネットワーク化されていても良い。アプリケーション位置が空である場合、ランチャ３０３はサービスデータに基づいてどのアプリケーションを開始すべきかを決定しなければならない。

【００６８】

ランチャ３０３は、システム６００の動作中はほぼ常に実行されているであろうブラウザコントローラなどの任意のアプリケーションを開始するように構成することもできる。このようなアプリケーションは永続的アプリケーションと呼ばれる。アプリケーションは、対応するスマートカード１０上での最初のユーザ選択に応答して、あるいは、アプリケーション３０４のうちの別の１つの要求に応じてランチャ３０３により開始することができる。

【００６９】

システム６００の一部の実現例では、ランチャ３０３はイベントマネージャ３０１へと統合することができる。

【００７０】

ランチャ３０３については、第１０．０節を参照して以下で更に詳細に説明する。

【００７１】

２．１．５　ディスプレイマネージャ

ディスプレイマネージャ３０６は、どのスマートカードアプリケーション３０４が現在表示画面１０１上に出力を表示することができるかを選択する。ディスプレイマネージャ３０６には、ランチャ３０３が送信元であるＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントにより表示できるのがどのアプリケーション３０４であるのかが通知される。このイベントはディスプレイマネージャ３０６に直接送信することもできるが、イベントマネージャ３０１がディスプレイマネージャ３０６及び予定される受信者にイベントのコピーを送信することもできる。

【００７２】

一般的に、出力を表示するように試みるべき唯一のアプリケーション３０４はフロントア

プリケーションである。ディスプレイマネージャ３０６は、表示の制御権を有する各アプリケーション間での切り替え中に安定した出力を提供することができる。ディスプレイマネージャ３０６は、フロントアプリケーションとしての各アプリケーションの切り替え中には推定のデータを使用する必要があるかもしれない。

【００７３】

アーキテクチャ２００は、ディスプレイマネージャ３０６が必要とされない、あるいは、ディスプレイマネージャ３０６の役割がアーキテクチャ２００の別の部分３０１又は３０３により想定されても良いように構成することができる。

【００７４】

２．１．６　ディレクトリサービス

ディレクトリサービス３１１は、スマートカード１０上に格納されるサービス識別子をサービスの位置又はサービスと関連付けられるアプリケーションの位置を示すリソースロケータ（ＵＲＬなど）へと変換するように構成される。また、ディレクトリサービス３１１はオプションのサービスデータを変換するようにも構成される。ディレクトリサービス３１１は、特定のカード１０と関連付けられたランチャ３０３がリソースロケータを用いて何をすべきか、例えば、関連するアプリケーション３０４をダウンロード・実行するか、あるいは、リソースロケータをブラウザアプリケーションへとロードするかを決定できるようにする。

【００７５】

２．１．７　アプリケーション

特定のスマートカード１０と関連付けられたアプリケーション３０４は、対応するカード上の最初のボタン押下に応じて、スマートカード１０と関連付けられたランチャ３０３により開始することができる。各アプリケーション３０４は１つ以上のサービスグループのメンバであることもできる。このサービスグループについては本明細書で後述する。アプリケーション３０４は、任意のサービスグループの一部とならないように指定することができ、この場合、アプリケーションは他のアプリケーションと共に実行されることはない。アプリケーションは、実行中になればサービスグループの一部となることができ、そのアプリケーションが現在フロントアプリケーションであるときにはサービスグループから外れることができる。

【００７６】

一部のアプリケーションは、システム６００が始動されると開始することができ、これらのアプリケーション、例えば、ブラウザ制御アプリケーション又はメディアプレーイングアプリケーションは常に実行させることができる。これらの永続的アプリケーションはシステム固有であっても、より汎用的に適用可能なものであっても良い。

【００７７】

図９は、上述のプロセスコンポーネント３０１から３０６を含むカードインタフェースシステムの概略ブロック図表現である。図９のシステムにおいて、リモートリーダ１はＩＲリンク及びそれを制御するためのＩ／Ｏデーモン３００を介してコンピュータ１００と通信を行なう。更に、コンピュータ１００は、ここではウェブサーバ４１０に対するインターネット４００により表される通信ネットワークと通信を行なうように構成される。この例において、スマートカード１０及びリモートリーダ１を利用してアクセス可能なアプリケーション３０４の一部は、様々なスマートカード１０と関連付けられるウェブページ４０６であることもできる。ウェブライブラリ４０７は、スマートカード１０と共に使用されるウェブページに含まれる可能性がある関数（例えば、Ｊａｖａｓｃｒｉｐｔ関数）及びクラス（Ｊａｖａクラス）を含む。ウェブページ４０６は、ウェブブラウザ４０３と呼ばれる実行中のアプリケーションを用いてアクセスすることができる。図９のシステムにおいて、イベントマネージャ３０１はリモートリーダ１からイベントを受信するように構成される。イベントはランチャ３０３に送信される。ランチャ３０３はウェブブラウザ４０３を制御するブラウザコントローラ４０２にメッセージを送信するように構成することができる。アプリケーションセッション又はブラウザセッションを開始するプロセスにつ

いては以下で詳細に説明する。ランチャ３０３は、ファイルサーバ４１１から実行中のアプリケーションと共にアプリケーション４０８もダウンロードするように構成することができる。ファイルサーバ４１１はインターネット４００を介してコンピュータ１００に接続される。

【００７８】

３．０　リーダ

リモートリーダ１は、スマートカード１０とインタフェースをとってカスタマイズ可能なユーザインタフェースを提供するハンドヘルド・バッテリ駆動のユニットであるのが好ましい。上述のように、リモートリーダ１は、デジタルＴＶ、セットトップボックス、コンピュータ、又はケーブルＴＶ機器と共に使用されることを意図しており、家庭環境におけるオンライン顧客サービスに対する簡単で直感的なインタフェースを提供する。

【００７９】

図４３及び４４は上述のリーダ１と同様のリーダ４４０１を示す。リーダ４４０１はカード１０の読取りのために構成される。リーダ４４０１はカードレセプタクル４４０４及び表示領域４４０６を組み込む筐体４４０２により形成される。カードレセプタクル４４０４は、図１に示されるスマートカード１０を挿入可能なアクセス開口部４４１０を含む。

【００８０】

表示領域４４０６の上側境界は、上述の膜８と同様のほぼ透明な感圧膜４４０８の形態のセンサ手段により定義される。膜４４０８の下方に配置されるのは、スマートカード１０の相補的な接点に接触するように構成される露出した電気接点４４０７の構成の形態で設けられるデータ読取り手段である。

【００８１】

図４５に示すように、カード１０はアクセス開口部４４１０を介してリーダ４４０１へと挿入される。リーダ４４０１の構成により、ユーザはリーダ４４０１を一方の手に保持し、他方の手でスマートカード１０をリーダ４４０１へと簡単に挿入することができる。スマートカード１０がリーダ４４０１に完全に挿入されると、感圧膜４４０８はスマートカード１０の上面１６を完全に覆うようになる。表示領域４４０６は、上面１６が透明な感圧膜４４０８を通して表示領域４４０６内に事実上完全に見えるようにカード１０の上面１６とほぼ同じ寸法を有するのが好ましい。

【００８２】

図４６は、カードが完全に挿入された後にリーダ４４０１を操作するユーザを示す。

【００８３】

図４７（ａ）から図４７（ｃ）において、筐体４４０２は、膜４４０８を包囲する最上部４８２７と、最上部４８２７との接続部４８２９から膜４４０８の横方向の中心の下方にあり且つこれに近接する位置４８１１まで延出するベース部４８０５とにより定義される実質的に二部構成の外部ケーシングから形成される。ベース部４８０５は、赤外線（ＩＲ）透過材から形成される対向端面４８１５を含むので、ＩＲ通信情報をリーダ４４０１により送ることができる。

【００８４】

位置４８１１は、ベース部４８０５とカード支持面４８０７との接続点を定義する。カード支持面４８０７は、接点４４０７が面４８０７と最上部４８２７との間で膜４４０８を挟持する内部接合部４８３５の方に位置する平面を通って延出する。アクセス開口部４４１０は、図４７（ａ）に示すように、位置４８１１と筐体４４０２の周縁部４８３６との間の空間により実質的に定義される。

【００８５】

接点４４０７は、プリント回路基板（ＰＣＢ）４８０１上に設置されるコネクタブロック４８３７から延出する。ＰＣＢ４８０１は、２つのマウンティング４８１７及び４８１９によりベース部４８０５と支持面４８０７との間に配置される。ＰＣＢ４８０１のコネクタブロック４８３７に対向する側には、コネクタ４４０７及びタッチセンシティブ膜４４０８に電気的に接続され、膜４４０８の押下に応じてカード１０からデータを読み取るよ

うに構成された電子回路（不図示）が配置される。ＰＣＢ４８０１からは、制御対象の装置（セットトップボックス６０１など）との通信用のＩＲウィンドウとして機能する端部４８１５に隣接して配置される赤外線発光ダイオード（ＬＥＤ）４８００も取り付けられている。

【００８６】

図４７（ｂ）は、スマートカード１０がアクセス開口部４４１０を介してレセプタクル４４０４に部分的に挿入された図４７（ａ）に類似する図を示す。図４７（ｂ）において明らかなように、支持面４８０７は、接点４４０７の平面から接合部４８１１に向かって下方に延出する一体的に形成された曲線状の輪郭４８４０を有する。この構成により、図４７（ｂ）に示すように、スマートカード１０が最初はレセプタクル４４０４の平面に対して傾斜するように、リーダ４４０１はスマートカード１０を受け入れる。支持面４８０７の曲線状の輪郭部４８４０の構成により、ユーザの手の力を受けるとスマートカード１０は完全な挿入位置にまで導かれる。具体的には、カード１０が更に挿入されると、支持面４８０７の湾曲がカード１０を接点４４０７及びレセプタクル４４０４の平面へと導く。

【００８７】

図４７（ｃ）は、スマートカード１０がレセプタクル４４０４に完全に挿入された図４７（ａ）に類似する図を示す。このとき、カード１０は、レセプタクル４４０４及びスマートカード１０のデータ接点（不図示）のうちの関連する１つに接触する接点４４０７の平面に位置する。スマートカード１０は感圧膜４４０８により覆われる。更に、接点４４０７はカード１０に対向する力を提供するように動作するばね接点であって、膜４４０８と関連付けられるのが好ましい。この構成は、適切な衝突収納（ｎｅａｔ　ｉｎｔｅｒｆｅｒｎｃｅ　ｆｉｔ）によりカード１０をレセプタクル内に保持するのに十分である。

【００８８】

以下の説明では、リーダ１への言及は、図１のリーダ１又は図４３から図４７（ｃ）のリーダ４４０１として実現されるリーダへの言及として解釈される。

【００８９】

図１０は、リモートリーダ１の内部構成を更に詳細に示す概略ブロック図である。リモートリーダ１は、リモートリーダ１を制御し、リモートリーダ１と、例えば、セットトップボックス６０１との間の通信を調整し、マッピング情報を格納するためのマイクロコントローラ４４を含む。マイクロコントローラ４４は、ランダムアクセスメモリ（ＲＡＭ）４７及びフラッシュ（ＲＯＭ）メモリ４６を含む。また、マイクロコントローラ４４は中央処理装置（ＣＰＵ）４５を含む。マイクロコントローラ４４はクロックソース４８とマイクロコントローラ４４内のイベントのタイミングを調整するためのクロックコントローラ４３とに接続される。ＣＰＵ４５には５Ｖバッテリ５３からの電力が供給され、ＣＰＵ４５の動作は電源制御装置５０により制御される。マイクロコントローラ４４は、カードエントリ状態について及び「ボタン」押下に対して可聴のフィードバックを与えるためのビーパ５１にも接続される。

【００９０】

赤外線（ＩＲ）通信は、マイクロコントローラ４４に接続される２つの回路、すなわち、ＩＲ送信用のＩＲ送信機（送信機）４９及びＩＲ受信用のＩＲ受信機（受信機）４０を使用して実現されるのが好ましい。

【００９１】

リモートリーダ１の感圧タッチパネル８は、タッチパネルインタフェース４１を介してマイクロコントローラ４４と通信を行なう。スマートカードインタフェース４２は電気接点７に接続する。

【００９２】

システム内プログラミングインタフェース５２もマイクロコントローラ４４に接続される。これにより、マイクロコントローラフラッシュメモリ４６を介してファームウェアを用いてマイクロコントローラ４４をプログラミングできるようになる。ファームウェアについては、第６．０節を参照して本文書で詳細に後述する。

【００９３】
ここでは、リモートリーダ１の内部構成を更に詳細に説明する。
【００９４】
３．１　低電力モードの寿命
電源制御装置５０は、低電力「スリープ」モード及びアクティブモードの２つの電力モードを提供するように動作可能である。低電力モードの寿命は、バッテリ５３の寿命を年単位で表したものである。リモートリーダ１が機能しておらず、低電力モードにあるとき、寿命は２年間よりも長くなる可能性がある。
【００９５】
リーダ１がスリープモードにあるときにユーザがタッチパネル８を押下する場合、リモートリーダ１はスリープモードから抜け出て、ＣＰＵ４５は接触座標を計算すると共に赤外線送信によりシリアルメッセージを送信する。バッテリ５３は１００，０００回を超えるボタン押下の電流供給要求に対して使用可能状態を維持するのが好ましい。
【００９６】
３．２　耐用寿命
耐用寿命は、リモートリーダ１が使用可能状態を維持することが予想される期間として定義され、バッテリの交換は含まない。耐用寿命は平均故障間隔（ＭＴＢＦ）の数値に関連し、通常、加速寿命試験を使用することで統計的に得られる。従って、リモートリーダ１の耐用寿命は５年よりも長くなる可能性がある。
【００９７】
３．３　マイクロコントローラ
リモートリーダ１のマイクロコントローラ４４は、４０９６バイトのフラッシュメモリ４６及び１２８バイトのランダムアクセスメモリ４７と共に８ビット中央ＣＰＵを有する。マイクロコントローラ４４は３から５ボルトの供給電力で動作するのが好ましく、フレキシブルオンボードタイマ、割込み源、８ビットＡ／Ｄコンバータ（ＡＤＣ）、クロックウオッチドッグ、及び低電圧リセット回路を有する。また、マイクロコントローラ４４は大電流出力ピンを有し、わずかな外部接続のみで回路中にプログラムできるのが好ましい。
【００９８】
３．４　クロックソース
リモートリーダ１用のメインクロックソース４８は、内蔵の平衡コンデンサを有する３ピン４．９１ＭＨｚセラミック共振子であるのが好ましい。周波数許容範囲は０．３％である。この許容範囲は水晶ほど良くはないが、シリアル通信に適しており、水晶よりも小型で安価である。
【００９９】
３．５　ビーパ（Ｂｅｅｐｅｒ）
ビーパ５１はリモートリーダ１の中に含まれており、カードエントリ状態について及びボタン押下に対して可聴のフィードバックを与える。ビーパ５１は圧電セラミックディスク型であるのが好ましい。
【０１００】
３．６　赤外線通信
上述のように、赤外線（ＩＲ）通信は、ＩＲ送信用のＩＲ送信機４９及びＩＲ受信用のＩＲ受信機４０の２つの回路を使用して実現されるのが好ましい。この２つの回路４０及び４９は、リモートリーダ１内のプリント回路基板（例えば、図４７のＰＣＢ４８０１）上で組み合わされるのが好ましい。プリント回路基板４８０１は、４方向フラットプリントケーブルによりマイクロコントローラ４４に接続することができる。送信時に必要なサージ電流を提供するために、ＰＣＢ４８０１には大型のバルク減結合コンデンサが必要とされる。
【０１０１】
３．７．１　赤外線送信
ＩＲ送信は、ＩＲ送信機４９の一部を形成する赤外線発光ダイオード（ＬＥＤ）（例えば

、図４７（ａ）のＬＥＤ４８００）により行なわれるのが好ましい。
【０１０２】
３．７．２　赤外線受信
ＩＲ受信機４０は、赤外線フィルタ、ＰＩＮダイオード、増幅器、及び弁別回路と共に単一のデバイスへと統合されるのが好ましい。受信されるシリアル情報は、このデバイスからマイクロコントローラ４４の入力ポートへと直接進む。このポートは、データの受信時に割込みを発生し、迅速な格納及び入力信号の処理が可能になるようにプログラムすることができる。
【０１０３】
３．８　ＣＰＵ／メモリカードインタフェース
リモートリーダ１は、国際標準化機構（ＩＳＯ）標準７８１６－３及びＩＳＯ７８１０により定義されるスマートカード１０をサポートできるのが好ましい。３Ｖメモリカード及び５Ｖメモリカードと同様に、Ｔ＝０プロトコル及びＴ＝１プロトコルを有する３ＶＣＰＵカード及び５ＶＣＰＵカード（すなわち、内蔵のマイクロプロセッサを有するカード）もサポート可能である。
【０１０４】
カード１０とマイクロコントローラ４４とを接触させるのに使用される電気接点７は、８つのワイピング接点及び１つの「カードイン」スイッチを有する表面実装型コネクタであるのが好ましい。ＩＳＯの基準によると、以下の信号が提供されなければならない：
・ピン１－ＶＣＣ－供給電圧、
・ピン２－ＲＳＴ－リセット信号、カードへの２進出力、
・ピン３－ＣＬＫ－クロック信号、カードへの２進出力、
・ピン４－ＲＦＵ－未使用、常時未接続、
・ピン５－ＧＮＤ－グランド、
・ピン６－ＶＰＰ－プログラミング電圧、不要、ＧＮＤ、ＶＣＣへの連結又は開放、
・ピン７－Ｉ／Ｏ－データＩ／Ｏ、双方向信号、及び
・ピン８－ＲＦＵ－未使用、常時未接続
ＲＳＴピン及びＩ／Ｏピンはマイクロコントローラ４４に直接接続されるのが好ましい。電源を除く全てのピンには、静電気放電の問題を回避するために、シリーズターミネーションダイオード及び過度電圧抑制器ダイオードが備わっている。
【０１０５】
３．９　ＣＰＵカード電源
上述のように、マイクロコントローラ４４が動作するには３Ｖ～５Ｖの電源が必要である。５Ｖ電源は、調整５ＶチャージポンプＤＣ－ＤＣコンバータチップの形態の電源制御装置５０を用いてバッテリ５３として動作する３Ｖコイン型リチウム電池から生成することができる。
【０１０６】
３．１０　タッチセンシティブインタフェース
上述のように、リモートリーダ１の感圧タッチパネル８は、タッチパネルインタフェース４１を介してマイクロコントローラ４４と通信を行なう。タッチパネルインタフェース４１はタッチパネル８上の接触の位置に従ってアナログ信号を提供する。このアナログ信号はマイクロコントローラ４４に送信される。
【０１０７】
接触座標の計算には、下側及び左側のタッチパネル８の接点（不図示）がマイクロコントローラ４４上のＡ／Ｄコンバータの入力へと接続されることが必要である。
【０１０８】
タッチパネル８への接触は、好ましくは、リモートリーダ１をスリープモードからウェイクアップするのに使用することができる。左側の画面接点から図示されるスリープ中のウェイクアップポートまで抵抗接続することでこの機能が提供される。尚、システム内プログラミング中に最大８ボルトが割込み要求ピン（ＩＲＱ）と呼ばれるマイクロコントロー

ラ４４上のピンに印加される可能性があるので、デバイスの損傷を防止するためにクランピングダイオードをこのピンに取り付ける必要がある。この例では、タッチパネル８の押下を検知するのに必要なバイアスを実際に提供するのはＩＲＱピン上の内部プルアップである。

【０１０９】

３．１１　バッテリ

上述のように、リモートリーダ１はバッテリ５３を使用する。５Ｖコイン型リチウム電池をバッテリ５３として使用し、リモートリーダ１の全回路に電力を供給することができる。

【０１１０】

３．１２　システム内プログラミング

マイクロコントローラはシステム内プログラミング（ＩＳＰ）オプションをサポートする。リモートリーダ１においては、システム内プログラミングインタフェース５２が使用され、ファームウェアを用いたマイクロコントローラフラッシュＲＯＭメモリ４６のプログラミングなどのマイクロコントローラ４４のプログラミングが実行される。

【０１１１】

３．１３　プリント回路基板及び相互接続

リモートリーダ１は、リーダ４４０１の１つのＰＣＢ４８０１の代わりに、以下に示すような２つのプリント回路基板（ＰＣＢ）を含むことができる：

（ｉ）赤外線ダイオードを保持し、ＦＥＴ及び受信機を駆動する赤外線（ＩＲ）ＰＣＢ、及び

（ｉｉ）上述の他の全ての構成要素４０から５３を保持するメインＰＣＢ（例えば、図４７（ａ）のＰＣＢ４８０１）

上述の双方のＰＣＢボードは、標準のＦＲ４、１．６ｍｍＰＣＢ材料を使用する両面設計であるのが好ましい。完成品のＰＣＢの厚さは重要であるので、メインＰＣＢは表面実装型部品を利用するのが好ましく、部品は最大約３ｍｍの高さに制限されるのが好ましい。

【０１１２】

ＩＲ　ＰＣＢはスルーホール部分を使用することができるが、部品の高さには厳しい制限が課せられるのが好ましい。２つのＰＣＢの相互接続は、カスタムデザイン４方向プリントケーブル（ＦＣＡ）を介して行なわれる。これは、タッチパネル８とインタフェースをとるのに使用される同一部品の表面実装型ＦＣＡコネクタを介して２つのＰＣＢとインタフェースをとる。

【０１１３】

３．１４　低電力モード

リモートリーダ１が短時間の間使用されない場合、バッテリの寿命を浪費しないために事前にプログラムされたファームウェアが装置を低電力モードにするのが好ましい。低電力モードでは、全ての電流を消費する構成要素への供給電力が停止され、マイクロコントローラ４４の各ポートは安全なスリープ状態に設定され、クロック４８は停止される。この状態で、リモートリーダ１の電流消費は５μＡ未満である。ＰチャネルＦＥＴを使用して電流を消費する構成要素への電力供給を制御することができる。

【０１１４】

低電力モードからリモートリーダ１をウェイクアップするには以下に示すような３つの代替の好適な方法がある：

・タッチパネル８への接触、

・カードレセプタクル４へのカードの挿入、及び

・バッテリ５３の取外し及び再挿入

カード挿入ウェイクアップにより、装置が低電力モードにあるか否かに関わらず、リモートリーダ１はカードが挿入されるときには常にビープ音を発することができる。接触／カード挿入ウェイクアップは、マイクロコントローラ４４上に図示されるようにＩＲＱピンにより処理される。新規の接触又はカード挿入のみがマイクロコントローラをウェイクア

ップするように、このピンは「エッジトリガ」に設定されることが重要である。ＩＲＱの感度が「レベル」トリガに設定される場合、例えば、リモートリーダ１が荷物の中に詰められているときにタッチパネル８が偶然押下されたままの状態になると、リモートリーダ１は低電力モードに入ることができない。

【０１１５】

３．１５　割込み及びリセット

リモートリーダ１用のマイクロコントローラ４４のファームウェアは、２つの外部割込み原因及び１つの内部割込み原因を使用する。低電力モードのウェイクアップの場合、外部割込みはＩＲＱピンから発生する。内部割込みはタイマのオーバーフローにより引き起こされ、種々の外部インタフェースの時間を調整するのに使用される。これらの割込みは事前にプログラムされたファームウェア手順により行なわれる。

【０１１６】

マイクロコントローラには以下に示すように４つの可能なリセット原因が存在する：

・２．４Ｖでの低供給電力リセット、

・不正なファームウェア演算コードリセット、

・ファームウェアがループに入り込んだ場合のコンピュータ正常動作（ＣＯＰ）リセット、及び

・システム内プログラミング（ＩＳＰ）が開始されるときにリセットピンに対して強制されるＩＳＰリセット

【０１１７】

４．０　カードデータ形式

上述のカード１０に対するデータ形式を以下の段落で説明する。図４に関連して説明した制御カード１０Ｂのようなメモリカードの場合、後述する形式に従ったデータはカードに直接コピーされることになる。上述のＣＰＵカードの場合、後述する形式に従ったデータは、ファイルとしてカードのＣＰＵのファイルシステムへとロードすることができる。

【０１１８】

上述のカード１０は、種々のカード特性及びカード上に印刷された任意のユーザインタフェース印を記述するデータ構造を格納するのが好ましい。また、カード１０はカード、ベンダ、及びサービスについての情報などの属性を指定するグローバル特性を含むことができる。ユーザインタフェースオブジェクトが存在する場合には、カード１０の表面の領域と関連付けるべきデータを指定する。

【０１１９】

ここで説明するユーザインタフェースは、所定の領域又はカード１０の表面に直接転写されたアイコン表現をコマンド又はアドレス（例えば、Ｕｎｉｆｏｒｍ　Ｒｅｓｏｕｒｃｅ　Ｌｏｃａｔｏｒ（ＵＲＬ））と関連付けるマッピングデータを表す。マッピングデータは、一般的に、カード１０上のユーザインタフェース要素（例えば、所定の領域）のサイズ及び位置を定義する座標を含む。ここで、ユーザインタフェース要素という用語は、通常、カード１０上の印を指すが、ユーザインタフェースオブジェクトという用語は、通常、特定の印に関連するデータを指す。しかし、これらの用語は以下の説明において区別なく使用される。

【０１２０】

ユーザインタフェースオブジェクトはカード１０に直接格納されるのが好ましい。また、ユーザインタフェースオブジェクトはカード１０自体ではなく、システム６００に格納することもできる。例えば、カード１０はオンカードメモリを介して固有の識別子であるバーコード又は磁気ストリップを格納することができる。この固有の識別子は、ほぼ同様のユーザインタフェース要素及びレイアウトを有するカード１０に固有のものである。固有の識別子は、ユーザの押下の結果タッチパネル８から判定された座標と共にリーダ１によりシステム６００のコンピュータ１００又はセットトップボックス６０１に送信することができる。コンピュータ１００、セットトップボックス６０１、又はサーバ１５０上に格納されたユーザインタフェースオブジェクトを有するシステム６００は、カード１０上の

ユーザインタフェース要素により表される所望の機能を提供するために、判定された座標からカード１０及びユーザ押下と関連付けられたサービスに関連する対応するコマンド、アドレス、又はデータへのマッピングを実行することができる。この例では、上述のユーザにより選択された印に関連するデータは、所望の印に重なるタッチパネル８の部分をユーザが押下した結果リーダ１により判定される座標の形態をとる。

【０１２１】

上述のカード（例えば、１０）において、カード１０により格納されるデータはカードヘッダを含み、このカードヘッダには以下の各節で説明する０個以上のオブジェクトが続く。

【０１２２】

４．１　カードヘッダ

図１１は、スマートカード１０に格納されたカードヘッダ１１００のデータ構造を示す。ヘッダ１１００は、各々が４バイトのデータを表す複数の列１１０１を含む。データは「ビッグエンディアン」形式であるのが好ましい。完全なヘッダは２０バイト長であり、以下のフィールドを含む（詳細は図１２に記載）：

（ｉ）カードを有効なメモリカードとして指定する定数を含むマジックナンバーフィールド１１０２（例えば、このマジックナンバーフィールド１１０２は、特定の製品に帰属するプロプライエタリカードが使用されていることを確認又は検証するのに使用することができる）、

（ｉｉ）カードレイアウトの下位のバージョンと互換性のあるリーダにより読むことができないレイアウトの変更を指定する各バージョンの増加分を含むバージョンフィールド１１０３、

（ｉｉｉ）予約フィールド１１０４（このフィールドは今後の使用のために予約されている）、

（ｉｖ）カード用のフラグを含むフラグフィールド１１０５（図１３参照）、

（ｖ）サービス１１０６及びサービス固有１１０７フィールドの２つのフィールドを含む区別用識別子フィールド１１１０（サービスフィールド１１０６は対応するカードのサービスを識別し、サービス固有フィールド１１０７はオプションとしてサービス固有値を含む）、

（ｖｉ）ヘッダに続くオブジェクトの数を表す数値を含むオブジェクト数フィールド１１０８（このフィールドは０に設定することもできる）、及び

（ｖｉｉ）チェックサム自体を除くカード上の全てのデータのカードチェックサムを含むチェックサムフィールド１１０９

【０１２３】

図１２は、図１１を参照して説明した種々の（数値）フィールドの内容の記述を提供する。特に、区別用ＩＤ番号フィールド１１１０は８バイトの区別用識別子を具備する。区別用識別子は、データの単位である２つの部分、すなわち、サービス識別子及びサービス固有識別子を含む。区別用識別子は、サービス識別子が区別用識別子の値全体のうちの５バイトを占め、サービス固有識別子が３バイトを占めるように構成されるのが好ましい。

【０１２４】

フィールド１１０６に含まれるサービス識別子は、サービス又はベンダを区別する。すなわち、例えば、サービスはカードユーザにそのサービスを提供するアプリケーションと関連付けることができる。これは、複数のアプリケーションを提供することによってカードユーザに複数のサービスを提供することができるベンダとは異なる。

【０１２５】

サービス識別子は使用するアプリケーション又はアプリケーションの位置（例えば、ＵＲＬ）を識別するための識別子となることができる。また、総称カードがシステム６００Ａ又は６００Ｂに追加されても良く、これらのカードはサービス識別子の特殊な使用例である。総称カードは、既に実行中の現在のアプリケーションへの入力を提供するのに使用可能な特殊なサービス識別子を有するカードである。サービスの特殊値０ｘ００００００００

００１は「総称サービス識別子」として知られ、「総称カード」上で使用される。総称カードは既に実行中のフロントアプリケーションにデータを送信するのに使用することができる。これらのカードは、例えば、テキスト入力を任意のアプリケーションに送信するのに使用可能なキーパッド又は個人的な詳細情報を任意のアプリケーションに送信するのに使用されても良い個人情報付きのカードの代わりに使用される。

【０１２６】

フィールド１１０７に含まれるサービス固有識別子は、特定のサービスのベンダが、その特定サービスと関連付けられる所定の機能を提供するのに任意で使用することができる。サービス固有識別子の使用は、システム６００上で実行中のアプリケーション３０４によってほぼ決まる。例えば、サービス識別子及びサービス固有識別子は、カード１０に対する固有の識別子として使用することができる。この固有の識別子は、特定のサービスと関連付けられる特定の機能へのアクセスを獲得又は拒絶し、ログファイル中に固有サービス識別子を再現してその値を有する特定のカード１０がサービスにアクセスするのに使用されたことを確認又は検証できるようにし、データベース中の対応する値に合わせることができる固有の識別子を提供してサービスのユーザに関する情報（名前、住所、クレジットカード番号など）を取得するために使用することができる。

【０１２７】

サービス固有識別子の使用の他の例として、カード１０の配布のメカニズム又は様式（郵便、バスターミナルの売店、列車内での配布など）に関する情報の提供を含むことができる。更に、サービス固有識別子は、サービスがアクセスされたときにシステム６００にどのデータがロードされるべきかを識別することができる。

【０１２８】

以上の説明はサービス固有識別子の可能な使用法又は適用例を網羅したリストではなく、可能な適用例の少数のサンプルに過ぎない。フィールド１１０７のサービス固有識別子にはその他にも多くの適用例がある。

【０１２９】

４．１．１　カードフラグ

図１１のヘッダ１１００のフラグフィールド１１０５は以下のような３つのフラグを含んでも良い：

（ｉ）ビープ停止、

（ｉｉ）移動イベントなし、及び

（ｉｉｉ）イベント座標なし

図１３は上述の各フラグの記述を示す。上述のフラグは、各フラグの記述により定義されるように、リモートリーダ１においてスマートカード１０が実行可能な機能に影響を及ぼす。図１３で参照されたユーザインタフェース要素の一例はカード１０上の「ボタン」である。ユーザインタフェース要素は本文書で更に詳細に後述する。

【０１３０】

４．２　オブジェクト

図５７に示すように、図１１のカードヘッダ１１００の直後に、特定のカード１０の各オブジェクトを定義し且つカード１０上に格納されたデータの一部を形成する０個以上のオブジェクト構造５７１３を続けることができる。各オブジェクト構造５７１３は以下のような４つのフィールドを具備する：

（ｉ）種別フィールド５７０１、

（ｉｉ）オブジェクトフラグフィールド５７０３、

（ｉｉｉ）長さフィールド５７０５、及び

（ｉｖ）データフィールド５７０７

データフィールド５７０７の構造は後述するオブジェクト種別によって決まる。

【０１３１】

図１４は、オブジェクト構造５７１３の各フィールド５７０１、５７０３、５７０５、及び５７０７の記述を示す。オブジェクト構造５７１３のフラグフィールド５７０３は、非

アクティブフラグを含むのが好ましい。図１５は非アクティブフラグの記述を示す。
【０１３２】
上述のカード１０Ａ、１０Ｂ、１０Ｃ、及び１０Ｄには以下に示すように５つのオブジェクト種別が提供されるのが好ましい：
（ｉ）ユーザインタフェースオブジェクト（すなわち、カード１０上のボタンを定義するデータ）、
（ｉｉ）カードデータ、
（ｉｉｉ）固定長データ、
（ｉｖ）リーダ挿入、
（ｖ）操作なし、及び
（ｖｉ）操作なし（１バイト）
図１６は、上述のオブジェクト種別（ｉ）から（ｖｉ）の各々の記述を示している。
【０１３３】
４．２．１　ユーザインタフェースオブジェクト
各ユーザインタフェースオブジェクトは、カード１０上の矩形領域とユーザがパネル８のカード１０の対応する矩形領域の上にある領域に接触するときに送信される幾つかの関連データとを定義する。座標マッピングシステムの起点は、スマートカード１０がＩＳＯ標準メモリスマートカードであり、チップ接点１８がビューアの反対方向且つカード１０の底部の方向を向くように縦向きに保持されたときのカード１０の左上である。このカードの向きを使用しないリーダ１の場合、正しい「ボタン」押下を通知するように四隅の点の値がリーダ１によって調整されなければならない。
【０１３４】
ユーザインタフェース（要素）オブジェクト構造は、以下に示すように６つのフィールドを有するのが好ましい：
（ｉ）フラグフィールド、
（ｉｉ）Ｘ１フィールド、
（ｉｉｉ）Ｙ１フィールド、
（ｉｖ）Ｘ２フィールド、
（ｖ）Ｙ２フィールド、及び
（ｖｉ）例えば、ＵＲＬ、コマンド、キャラクタ又は名前に対するユーザインタフェース要素と関連付けられるデータを通常含むデータフィールド
【０１３５】
図１７は、先に説明したユーザインタフェースオブジェクト構造に対する上述の各フィールドの記述を示す。感圧タッチパネル８の押下は、以下のような場合に特定のユーザインタフェースオブジェクトの内側であると定義される：
（ｉ）押下位置のＸ値が関連するユーザインタフェースオブジェクトのＸ１値以上であり、その特定のユーザインタフェースオブジェクトのＸ２値未満である場合、及び
（ｉｉ）押下位置の押下Ｙ値が特定のユーザインタフェース要素のＹ１値以上であり、Ｙ２値未満である場合
【０１３６】
ユーザインタフェース要素は重複しても良い。押下が２つ以上のユーザインタフェース要素の範囲内である場合、送信されるオブジェクトはＺ順により判定される。カード上のユーザインタフェース要素の順序は、その特定のカード上の全てのユーザインタフェース要素に対するＺ配列を定義する。先頭のユーザインタフェース要素は特定のカード１０に対する第１のユーザインタフェース要素である。最後尾のユーザインタフェース要素はそのカード１０に対する最終のユーザインタフェース要素である。これにより、非矩形の領域の定義も考慮される。例えば、「Ｌ」型のユーザインタフェース要素を定義するには、第１のユーザインタフェースオブジェクトがデータフィールドに０バイトを伴って定義され、第２のユーザインタフェースオブジェクトが第１のユーザインタフェースオブジェクトの左下にユーザインタフェースオブジェクトに重なるように定義されるであろう。

【０１３７】
押下の位置は「フィンゲル（ｆｉｎｇｅｌ）」において通知される。この「フィンゲル」はフィンガー要素（画素を表す「ピクセル」に類似）を表す。フィンゲルの高さはＩＳＯメモリスマートカードの長さの１／２５６であり、幅はＩＳＯメモリスマートカードの幅の１／１２８であると定義される。各要素と関連付けられる挙動は１つ以上のフラグを用いて変更されても良い。各ユーザインタフェース要素は以下に示すように４つの関連するフラグを有するのが好ましい：
（ｉ）ビープイネーブル反転
（ｉｉ）オートリピート
（ｉｉｉ）押下時データ送信禁止
（ｉｖ）リリース時データ送信禁止
図１８は、各ユーザインタフェース要素フラグの記述を示す。
【０１３８】
４．２．２　カードデータ
カードデータオブジェクトは、特定のカードに固有のデータを格納するのに使用される。このオブジェクトに対するデータレイアウトは固定の形式をもたない。カード１０がリーダ１へと挿入されるとき、カードデータオブジェクトの内容がＩＮＳＥＲＴメッセージの一部としてリーダ１から送信される。
【０１３９】
４．２．３　固定長データ
固定長データオブジェクトは、例えば、コンピュータ１００により書き込むことが可能なカード上の固定長ブロックを定義するのに使用される。
【０１４０】
４．２．４　リーダ挿入
リーダ挿入オブジェクトは、特定のカードが挿入されたときにリモートリーダ１に対する命令を格納するのに使用される。これは、例えば、ＩＲコマンドの特定の構成を使用して特定のセットトップボックス又はＴＶなどと送受信できるようにするべくリーダ１に指示するのに使用することができる。
【０１４１】
４．２．５　操作なし
操作なしオブジェクトは、特定のカード上の他のオブジェクト間の未使用のセクションを埋めるのに使用される。操作なしオブジェクトに格納されるあらゆるデータはリモートリーダ１によって無視される。カード１０の端の未使用の空間は操作なしオブジェクトを用いて埋める必要がない。
【０１４２】
４．２．６　操作なし（１バイト）
操作なし（１バイト）オブジェクトは、完全なオブジェクト構造には小さすぎるオブジェクト間の隙間を埋めるのに使用される。これらのオブジェクトは合計しても１バイト長である。
【０１４３】
５．０　リーダプロトコル
リモートリーダ１は、例えば、リモートリーダ１とセットトップボックス６０１又はコンピュータ１００との間の単方向通信及び双方向通信の双方をサポートするデータグラムプロトコルを使用する。リモートリーダ１とユーザとのインタラクションの結果、リモートリーダ１から受信するメッセージに使用される形式は、リモートリーダ１に送信される形式とは異なる形式である。
【０１４４】
５．１　メッセージ種別
リモートリーダ１により送信することが可能なメッセージイベント種別は少なくとも７つある。これらのイベントは以下の通りである：

・ＩＮＳＥＲＴ：カード１０がリモートリーダ１へと挿入され、カード１０が検証されると、リモートリーダ１によりＩＮＳＥＲＴイベントが生成され、関連するメッセージが送信される。このメッセージはカード１０の存在を受信機（例えば、セットトップボックス６０１）に通知するものである。ＩＮＳＥＲＴメッセージは特定の区別用識別子を含むのが好ましく、最初のインタラクションが起こるまで待たずにカード１０の挿入直後にアプリケーションを開始又はフェッチできるようにする。ＩＮＳＥＲＴメッセージは、この種のオブジェクトがカード１０上に存在する場合、リーダ１へと挿入されるカード１０からのカードデータオブジェクトの内容を含むのが好ましい。

【０１４５】

・ＲＥＭＯＶＥ：カード１０がリモートリーダ１から抜き取られると、対応するＲＥＭＯＶＥイベントが生成され、ＲＥＭＯＶＥメッセージがリモートリーダ１と関連付けられた特定の受信機に送信される。ＩＮＳＥＲＴメッセージと同様に、関連する区別用識別子がメッセージと共に送信される。抜き取られたカード１０からは区別用識別子を読むことができないので、区別用識別子はリモートリーダ１のメモリ４７に格納される。他の全てのメッセージが区別用識別子を必要とするが、区別用識別子が必要となるたびにカード１０から区別用識別子を読むのでは遅すぎる可能性があるので、これは有用な最適化である。システム６００は処理を制御する際にＩＮＳＥＲＴメッセージ及びＲＥＭＯＶＥメッセージに依存しない。受信したメッセージが予期されるものでない場合、システム６００は欠落したメッセージがあることを推定するように構成される。例えば、アプリケーションが２つの連続するＩＮＳＥＲＴメッセージを検知した場合、現在の構成では一度に２枚のカードを挿入することはできないので、第１のＩＮＳＥＲＴメッセージのカードと関連付けられたＲＥＭＯＶＥメッセージが欠落したと想定することができる。アプリケーションは、第２のＩＮＳＥＲＴメッセージを処理する前に必要なあらゆる動作を行なうことができる。

【０１４６】

・メッセージの欠落が起こりうる別の例は、有線式のリーダに対してハンドヘルドの赤外線接続のリーダ１が使用される場合である。ユーザはカードの挿入又は抜き取り時にリーダを受信機に直接向けないことが多い。この問題は、連続するＰＲＥＳＳ／ＲＥＬＥＡＳＥの対において区別用識別子が異なることに基づいてＩＮＳＥＲＴ操作又はＲＥＭＯＶＥ操作を推定するシステム６００により解消することができる。

【０１４７】

・ＢＡＤ　ＣＡＲＤ：無効なカードが挿入される場合、リモートリーダ１はＢＡＤ　ＣＡＲＤイベントを生成し、ＢＡＤ　ＣＡＲＤメッセージを送信するように構成されるのが好ましい。このメッセージにより、関連する受信機はユーザに対して無効なカードであることを警告すべく何らかの動作を行なうことができる。

【０１４８】

・ＰＲＥＳＳ：リモートリーダ１により接触が検知されるとき、ＰＲＥＳＳイベントが生成され、ＰＲＥＳＳメッセージが関連する受信機に送信される。ＰＲＥＳＳメッセージは、関連するカードの詳細、押下の位置、及びその特定の位置においてユーザインタフェース要素と関連付けられたデータを含む。その位置に対して定義されるユーザインタフェース要素が存在しない場合（カード１０上に定義されるユーザインタフェース要素が全く存在しない場合を含む）、関連するカードの詳細及び押下の位置を含むＰＲＥＳＳメッセージが送信される。ＰＲＥＳＳイベントが生成されたときにリモートリーダ１内にカードが存在しない場合、特殊な「ＮＯ＿ＣＡＲＤ」識別子（すなわち、８バイトの０－０ｘ００）及び押下の位置を含むＰＲＥＳＳメッセージが送信される。

【０１４９】

・ＲＥＬＥＡＳＥ：ＲＥＬＥＡＳＥイベントはＰＲＥＳＳイベントを補うものであり、システム６００のアプリケーションプログラムにＰＲＥＳＳが解除されたことを通知するためにＲＥＬＥＡＳＥメッセージを送信することができる。全てのＰＲＥＳＳイベントは対応するＲＥＬＥＡＳＥイベントを有するのが好ましい。リーダは複数回の押下を登録した

り、あるいは、ＰＲＥＳＳメッセージとＲＥＬＥＡＳＥメッセージとの間に発生する可能性があるその他のイベントを提供したりすることができる。

【０１５０】

・ＭＯＶＥ：ＰＲＥＳＳイベントの処理後に接触位置がある量だけ変化する場合、指（又はカードに接触するのに使用されるもの）が移動中であると想定される。ＭＯＶＥ ＥＶＥＮＴが生成され、接触が解除されるまでＭＯＶＥメッセージが送信される。接触位置が静止したままであるときには、ＭＯＶＥイベントは直前のＭＯＶＥメッセージを再送信することによってオートリピートする。接触が解除され、対応するＲＥＬＥＡＳＥメッセージが送信されると、送信の繰り返しは終了する。ＰＲＥＳＳイベント及びＲＥＬＥＡＳＥイベントとは異なり、ＭＯＶＥイベントと関係するユーザインタフェースオブジェクトは存在しない。

【０１５１】

・ＬＯＷ ＢＡＴＴ：リモートリーダ１中のバッテリ５３の電力が低下しているとき、ＬＯＷ ＢＡＴＴイベントが生成され、ＬＯＷ ＢＡＴＴメッセージが送信される。このメッセージは、ユーザとのインタラクションの後にメッセージがシステム６００のその他の部分により受信される可能性を高めるために送信される。ＬＯＷ ＢＡＴＴメッセージの送信は、リモートリーダ１が低電力状態に入ることを妨害しない。

【０１５２】

５．２　データ形式

システム６００において使用されるリーダプロトコルの好適なデータ形式は、固定サイズのヘッダの後に０バイト以上の長さを有することが可能な可変長のデータフィールドが続き、更にその後に８ビットのチェックサム及び補数が続く形式である。

【０１５３】

５．２．１　メッセージヘッダ

メッセージヘッダは固定長であることが好ましく、リモートリーダ１から送信される全てのメッセージの前に付加される。帯域幅に制約がある可能性があるので、メッセージヘッダは可能な限り小さくする必要がある。図１９は、リモートリーダ１から送信されるメッセージヘッダの形式を示す。

【０１５４】

ベンダが特定のサービスを登録するとき、スマートカード識別機関によりサービス識別子及びサービス固有識別子をそのベンダに割り当てることができる。サービス及びサービス固有識別子は任意のカードからの全てのメッセージに対して同じである。サービス固有識別子はアプリケーションと共に使用するためにベンダにより設定されるのが好ましい。リーダ識別子も各メッセージのヘッダ中に存在する。この識別子は、例えば、マルチプレーヤゲームにおいてユーザを区別するためにアプリケーション３０４により使用することができる。

【０１５５】

図２０は、以上説明したメッセージイベント種別を列挙するテーブルを示している。

【０１５６】

５．２．２　単純なメッセージ

幾つかのメッセージ種別は、上述のメッセージヘッダ並びにそれに続くメッセージチェックサムバイト及びその補数のみから成るという点で単純であると考えられる。例えば、ＢＡＤＣＡＲＤメッセージ、ＬＯＷ＿ＢＡＴＴメッセージ、及びＲＥＭＯＶＥメッセージは単純なメッセージである。

【０１５７】

図２１は、単純なメッセージの形式を示す。

【０１５８】

５．２．３　ＭＯＶＥメッセージ

ＭＯＶＥメッセージは、上述のメッセージヘッダ及びそれに続くリモートリーダ１のタッチパネル８上の接触位置の座標を定義する２つのフィールドから形成される。図２２はＭ

ＯＶＥメッセージの形式を示す。
【０１５９】
５．２．４　ＰＲＥＳＳメッセージ及びＲＥＬＥＡＳＥメッセージ
図２３は、ＰＲＥＳＳメッセージ及びＲＥＬＥＡＳＥメッセージの形式を示す。ＰＲＥＳＳメッセージ及びＲＥＬＥＡＳＥメッセージは、ＭＯＶＥメッセージと同様にメッセージヘッダ及び接触座標を含む。加えて、接触位置がカード上で定義されるユーザインタフェース要素と一致する場合、ＰＲＥＳＳメッセージ及びＲＥＬＥＡＳＥメッセージはユーザインタフェース要素と関連付けられるデータを送信する。このデータは可変長であり、実際のサイズは対応するカード１０により定義される。接触位置がカード上で定義されるユーザインタフェース要素と一致しない場合（カード上で定義されるユーザインタフェース要素がない場合を含む）、ユーザインタフェース要素と関連付けられる０バイトのデータが送信される。リーダ１内にカード１０がない場合、サービス識別子は全てが０（すなわち、０ｘ００）に設定され、ユーザインタフェース要素と関連付けられる０バイトのデータが送信される。ユーザインタフェース要素と関連付けられたデータは、通常、カード上で定義されるユーザインタフェース要素と関連付けられたデータに対応するが、カード１０又はリーダ１での処理により変更又は生成されても良い。
【０１６０】
図２４は、システム６００内の上述のメッセージの流れを示すデータフロー図である。図２４に示すように、カードヘッダ１１００及びオブジェクト構造５７１３は、リモートリーダ１のＣＰＵ４５により読み取られる。ＣＰＵ４５は対応するＩＮＳＥＲＴメッセージ、ＲＥＭＯＶＥメッセージ、ＰＲＥＳＳメッセージ、ＲＥＬＥＡＳＥメッセージ、ＭＯＶＥメッセージ、ＢＡＤＣＡＲＤメッセージ、又はＬＯＷ　ＢＡＴメッセージをＩ／Ｏデーモン３００を介してイベントマネージャ３０１に送信する。以下で更に詳細に後述するが、イベントマネージャ３０１は２１種類のコアメッセージを有し、これらのメッセージはＭＬ３０２、ランチャ３０３、及びアプリケーション３０４との間で送受信される。
【０１６１】
５．２．５　ＩＮＳＥＲＴメッセージ
ＩＮＳＥＲＴメッセージは上述のメッセージヘッダ及び挿入されたカード１０からのカードデータオブジェクトの内容から形成される。図２１ＡはＩＮＳＥＲＴメッセージの形式を示す。
【０１６２】
６．０　リーダファームウェア
６．１　概要
マイクロコントローラ４４は内蔵の不揮発性メモリ４６を有し、この不揮発性メモリ４６は詳細に後述するファームウェアを用いてプログラムすることができる。マイクロコントローラ４４及び周辺ハードウェア（コンピュータ１００など）と協働するファームウェアは、リモートリーダ１の機能的要求を指図することができる。
【０１６３】
６．２　コード種別
リモートリーダ１のユーザに対する費用を最小限に抑えるためには、リモートリーダ１上のメモリを最小限にするのが好ましい。従って、リモートリーダ１用に作成されるアプリケーションプログラム（すなわち、ファームウェア）は可能な限り小規模且つ高速にしなければならない。
【０１６４】
６．３　リソース制約条件
マイクロコントローラ４４は以下のような特徴を有する：
６．３．１　不揮発性メモリ
フラッシュメモリ４６は４０９６バイトのフラッシュＲＯＭを有するように構成され、ファームウェア格納用に利用することができる。フラッシュＲＯＭは再プログラム可能であるが、大量生産の場合、マスクＲＯＭ部品を利用することができる。

【０１６５】
６．３．２　ランダムアクセスメモリ（ＲＡＭ）
ＲＡＭ４７はファームウェアにより使用される１２８バイトのＲＡＭとして構成される。
【０１６６】
６．４　割込み
リモートリーダ１は、マイクロコントローラ４４がサポートする数多くの割込み原因のうちの２つを使用する。これらの割込みは以下に示すように説明することができる：
６．４．１　受信データ割込み
赤外線（ＩＲ）シリアルデータ受信機は、一般的に、入力データを受信すると立下り端を生成する。このデータは可能な限り迅速にサンプリング・バッファリングされる必要がある。マイクロコントローラ４４の１つのポートは、立下り端で割込みを開始することができる入力タイミングキャプチャピンの役割を兼ねる。
【０１６７】
６．４．２　タイマオーバフロー割込み
マイクロコントローラ４４は、オーバーフローしたときに割込みを発生するようにプログラム可能な自走１６ビットタイマを有する。４．９１ＭＨｚクロックソース及びプリスケール係数６４の場合、３．４１秒ごとの割込みになる。割込みサービスルーチンは、好ましくは、約１分の休止の後に低電力モードでのサスペンド状態を引き起こすカウンタを増分する。
【０１６８】
６．５　リセット
マイクロコントローラ４４は５つのリセットソースをサポートするが、リモートリーダ１は全てのリセットリソースを使用するように構成されるのが好ましい。これらのリセットリソースは以下のように説明することができる：
６．５．１　パワーオンリセット（ＰＯＲ）
ＰＯＲリセットは新規のバッテリがリモートリーダ１に取り付けられると開始される。マイクロコントローラ４４はパワーオン状態を検知しリセットを生成する回路を含む。
【０１６９】
６．５．２　低電圧禁止（ＬＶＩ）リセット
ＬＶＩリセットは、供給電圧が２．４ボルトを割ったことをマイクロコントローラ４４内の回路（不図示）が検知すると開始される。この種のリセットが発生するとリセット状態レジスタ（ＲＳＲ）中のフラグが設定され、初期化ルーチンはバッテリ５３が切れかけていると推定することができる。例えば、赤外線データの送信中、データがパルス状に送信されると共に赤外線ＬＥＤは大電流を消費する。バッテリ５３が切れると、送信中に供給電圧は２．４ボルトの閾値を割ってＬＶＩリセットを引き起こす可能性がある。リセット後、バッテリ５３の電圧は回復し、ＬＶＩリセットは次の大電流ドレインまで発生しない。これにより、ユーザによるバッテリ５３の交換を促すことができるように、リモートリーダ１にはバッテリ５３の消耗を関連するセットトップボックス又はリモート機器にフラグを立てて通知する機会が与えられる。
【０１７０】
６．５．３　コンピュータ正常動作（ＣＯＰ）リセット
ＣＯＰリセットはマイクロコントローラ４４が特定の動作を異常に長い時間実行したままでいる場合、マイクロコントローラ４４をリセットするように構成される。ＣＯＰ回路はオーバーフローする許可が得られる場合にリセットを生成するカウンタの形態をとる。ＣＯＰレジスタはＣＯＰリセットを回避するために所定の時間間隔で書き込まれなければならない。
【０１７１】
６．５．４　不正アドレス／演算コードリセット
不正アドレス／演算コードリセットは、マイクロコントローラ４４が所定の範囲外のアドレス及び所定の条件に合致しない演算コードに遭遇する場合に生成される。このリセット

は停止することができず、コードデバッギング中にのみ出現するべきである。
【０１７２】
６．５．５　ハードウェアリセット
ハードウェアリセットは通常動作中にマイクロコントローラ４４の「リセット」ピンをローに駆動することによって生成される。また、マイクロコントローラ４４が低電力モードにある場合、割込み要求（ＩＲＱ）ピンの立下り端もハードウェアリセットを生成する。このリセットはファームウェアにおいてマイクロコントローラ４４を低電力モードからウェイクアップするのに使用されるメカニズムである。ＩＲＱピンがこの機能に適しているが、それは、リセットピンのようにレベルを感知するのではなく、端のみを感知するように構成することができるからである。
【０１７３】
６．６　メモリカード／ＣＰＵカードインタフェース
ファームウェアは、集積回路プロトコル（例えば、Ｉ２Ｃプロトコル）を使用するメモリカード周辺装置のみをサポートするのが好ましい。また、ファームウェアはＣＰＵカード形式をサポートすることができる。
【０１７４】
６．７　電力消費
ファームウェアはバッテリ５３の寿命を伸ばすのに重要な役割を果たす。マイクロコントローラ４４が実行する全ての動作は、可能な限り迅速にして電力の消費を可能な限り抑えるように最適化される。リモートリーダ１がある時間（例えば、１分）非アクティブ状態であると、マイクロコントローラ４４は低電力モードでサスペンドし、バッテリの寿命を更に伸ばす。低電力モードは通常動作モードの約１／１０００の電流を消費するので、このモードでの効率的なサスペンドが非常に望ましい。ファームウェアは、低電力モード中マイクロコントローラ４４のポートの状態を制御する。
【０１７５】
６．８　デバイスプログラミング
マイクロコントローラ４４は、マイクロコントローラ４４内の内蔵モニタによりサポートされるシステム内プログラム（ＩＳＰ）機能を使用してプログラムすることができる。モニタコードは、通常、メーカにより工場で設定されており、変更することができない。
【０１７６】
特定のハードウェアに対するマイクロコントローラ４４のプログラミングは、インサーキットシミュレータ（ＩＣＳ）キット及びモニタモードダウンロードケーブルを使用して実行することができる。このケーブルは、マイクロコントローラ４４のＶＣＣピン、ＧＮＤピン、ＲＳＴピン、ＩＲＱピン、及びＰＲＢ０ピンを使用する。プログラムするソースコードは、例えば、ＩＣＳハードウェアに対するコンピュータシリアルポート及びマイクロコントローラ４４のピンに対するダウンロードケーブルを介してＷｉｎｄｏｗｓ（登録商標）９５開発環境から供給することができる。このプログラミング方法はファームウェア開発／検査には理想的であるが、大量生産の場合は変更しても良い。マイクロコンピュータにおいてはモニタモードプログラミングモデルが好ましく、生産用の埋込み型プログラミングジグを使用することができる。生産ＩＳＰを考慮して信号をプログラムするための検査ポイントを提供することができる。ファームウェアがマイクロコントローラ４４へとマスクプログラムされる場合、デバイスプログラミングは必要とされない。
【０１７７】
６．９　ファームウェアプログラミングシーケンス
ローカルコンピュータ１００に接続されて動作中のリモートリーダ１を参照して、ファームウェアのプログラミングを説明する。
【０１７８】
６．９．１　メインループ
図２５は、ソフトウェアアーキテクチャ２００を組み込むシステム６００のリモートリーダ１により実行される読取り方法２５００を示すフローチャートである。方法２５００は

、上述のように、リセットイベントの発生後に開始される。方法２５００はＣＰＵ４５により実行される。図２５の方法は「一定ペースのループ」状に構成される。すなわち、方法２５００は１０ミリ秒の遅延を生成するルーチンによりペース設定される。この遅延は、割込みの処理に十分な待ち時間を提供しながら必要なルーチンに適切なサービスを与える。

【０１７９】

第１のステップ２６００において、初期化ルーチンがＣＰＵ４５により実行される。初期化ルーチンは構成レジスタを初期化するために実行されるものであり、図２６のフローチャートを参照して後述する。方法２５００は次のステップ２５０１へと続く。ステップ２５０１において、コンピュータ正常動作（ＣＯＰ）レジスタがクリアされるが、これは、ファームウェアが循環ループに入り込んでいないことを示す。次のステップ２７００において、特定のスマートカード１０の存在及び有効性の変化をチェックするためにＣＰＵ４５によりチェックカードプロセスが実行される。チェックカードプロセスについては、図２７を参照して以下で詳細に説明する。方法２５００は次のステップ２８００へと続く。ステップ２８００において、ユーザによるタッチパネル８の接触をチェックするためにＣＰＵ４５により走査タッチパネルプロセスが実行される。次のステップ２９００において、ＣＰＵ４５により１０ミリ秒待機ルーチンが実行され、方法２５００はステップ２５０１へと戻る。

【０１８０】

６．９．１　初期化プロセス

上述の５つのソースのうちのいずれか１つによるリセット後、全ての構成レジスタは正しい初期化を必要とする。ＬＶＩリセットが受信された場合、「バッテリ切れ可能性」フラグが設定される。図２６は、ソフトウェアアーキテクチャ２００を組み込むシステム６００を初期化する方法２６００を示すフローチャートである。方法２６００はＣＰＵ４５により実行されるものであり、ステップ２６０１で開始される。ステップ２６０１において、全てのレジスタが所定のデフォルト状態へと初期化される。次のステップ２６０２において、ＣＰＵ４５によりチェックが実行され、リセットがＬＶＩリセットであるか否かが判定される。ステップ２６０２においてＬＶＩリセットでない場合、方法２６００は終了する。ＬＶＩリセットの場合、方法２６００はステップ２６０３へと進む。ステップ２６０３において、バッテリ切れ可能性フラグが設定され、方法２６００は終了する。

【０１８１】

６．９．２　チェックカードプロセス

図２７は、ソフトウェアアーキテクチャ２００を組み込むシステム６００のカード１０をチェックする方法２７００を示すフローチャートである。上述のように、方法２７００はリモートリーダ１においてスマートカード１０の存在及び有効性の変化をチェックし、それに従って応答する。方法２７００はＣＰＵ４５により実行されるものであり、ステップ７０１で開始される。ステップ７０１において、スマートカード１０がリモートリーダ１へと挿入される場合、方法２７００はステップ７０２へと進む。ステップ７０２において、カード１０が新規のカードである（すなわち、それ以前の状態ではリーダ１にカードが存在しなかった）場合、方法２７００はステップ７０３へと進む。新規のカードでない場合、方法２７００は終了する。次のステップ７０３において、「マジックナンバー」フィールド及び「チェックサム」フィールドがカード１０のメモリ１９に格納されたカードヘッダから読み取られ、適正さがチェックされる。「マジックナンバー」及び「チェックサム」が適正である場合、方法２７００はステップ７０４へと進む。方法２７００はステップ７０４へと続く。ステップ７０４において、区別用識別子がカードヘッダから読み取られ、「移動イベントなし」フラグ及び「イベント座標なし」フラグが設定される。カードデータが存在する場合には、それもこのステップ７０４においてカードから読み取られる。次のステップ７０５において、カードデータが存在する場合はそのカードデータも含むＩＮＳＥＲＴメッセージがコンピュータ１００に送信され、ＣＰＵ２０５により処理される。ステップ７０６において、「ビープ」が鳴って方法２７００は終了する。

【０１８２】
ステップ７０３において「マジックナンバー」フィールド及び「チェックサム」フィールドが適正でない（すなわち、カード１０が有効でない）場合、方法２７００はステップ７１０へと進む。ステップ７１０において、ビープ停止フラグ、移動イベントなしフラグ、及びイベント座標フラグが設定される。次のステップ７１１において、ＢＡＤ ＣＡＲＤメッセージがコンピュータ１００に送信され、ＣＰＵ２０５により処理される。ステップ７１２において、「ブープ（ｂｏｏｐ）」が鳴って方法２７００は終了する。
【０１８３】
ステップ７０１において、スマートカード１０がリモートリーダ１に挿入されていない場合、方法２７００はステップ７０７へと進む。ステップ７０７において、これがリセット後のリーダ１の最初の動作である場合、方法２７００は終了する。最初の動作でない場合、方法２７００はステップ７０８へと進む。ステップ７０８において、「ビープ停止」フラグ、「移動イベントなし」フラグ、及び「イベント座標なし」フラグが設定され、メモリ４７に格納される区別用識別子が「ＮＯ＿ＣＡＲＤ」に設定される。次のステップ７０９において、ＲＥＭＯＶＥメッセージがコンピュータ１００に送信され、ＣＰＵ２０５により処理される。方法２７００はステップ７０９の後に終了する。
【０１８４】
６．９．３　走査タッチパネルルーチン
図２８は、ソフトウェアアーキテクチャ２００を組み込むシステム６００のリーダ１のタッチパネル８を走査する方法２８００を示すフローチャートである。上述のように、走査タッチパネルルーチンはカードボタン押下と同等のタッチパネル接触をチェックし、それに従って応答する。方法２８００はＣＰＵ４５により実行されるものであり、ステップ８０１で開始される。ステップ８０１において、パネル８が触れられている場合、方法２８００はステップ８０２へと進む。パネル８が触れられていない場合、方法２８００はステップ８１２へと進む。ステップ８１２において、パネル８が以前に触れられていた場合、方法２８００はステップ８１３へと進む。以前に触れられていなかった場合、方法２８００は終了する。
【０１８５】
ステップ８１３において、「ビープ停止」フラグ、「移動イベントなし」フラグ、及び「イベント座標なし」フラグが設定される。ステップ８１４において、メッセージ種別がＲＥＬＥＡＳＥに設定され、方法２８００はステップ８０５へと進む。
【０１８６】
方法２８００はステップ８０２へと続く。ステップ８０２において、接触がなかった状態以来の初めての接触の認知である場合、方法２８００はステップ８０３へと進む。次のステップ８０３において、ＣＰＵ４５はステップ７０３の結果をチェックすることによって、不良カードがリーダ１へと挿入されたか否かを判定する。不良カードがリーダ１へと挿入された場合、方法２８００はステップ８１５へと進む。ステップ８１５において、ＢＡＤ ＣＡＲＤメッセージがコンピュータ１００に送信され、メモリ２０６に格納され、方法２８００は終了する。ステップ８０３において、ステップ７０３の結果をチェックすることによってカードが有効である、あるいは、ステップ７０１のチェックによってリーダ１にカードが挿入されていないと判定された場合、方法２８００はステップ８０４へと進む。ステップ８０４において、図１９のメッセージヘッダ中のメッセージ種別がＰＲＥＳＳに設定される。次のステップ８０５において、ＣＰＵ４５はタッチパネルインタフェース４１を介して接触座標（すなわち、ユーザ押下位置のＸＹ座標）を判定する。次のステップ８０７において、オフセット機能及びスケール機能が座標に適用される。オフセット機能及びスケール機能は、タッチパネル８の座標空間をカード１０の座標空間へとマップする。方法２８００は次のステップ８０７へと続く。ステップ８０７において、ＣＰＵ４５がチェックステップ７０１により送信されたメッセージがＭＯＶＥである、及び／又は、カードがリーダ１に挿入されていないと判定する場合、方法２８００は直接ステップ８０９へと進む。それ以外の場合、方法２８００はステップ８０８へと進み、Ｘ１値、Ｙ１

値、Ｘ２値、及びＹ２値が接触座標の位置する範囲を形成する最初のユーザインタフェース要素を探し出すためにカード１０のメモリ１９が検索され、一致するユーザインタフェース要素と関連付けられたデータがカード１０から読み取られる。ステップ８０９において、メッセージが任意のデータと共に関連するコンピュータ１００に送信され、コンピュータ１００のＣＰＵ２０５がそのメッセージを処理する。方法２８００は次のステップ２８１１へと進む。ステップ２８１１において、ビープ音が鳴って方法２８００は終了する。

【０１８７】

ステップ８０２において、接触がなかった状態以来の初めての接触の認知でない場合、方法２８００はステップ８１６へと進む。ステップ８１６において、ステップ８０１で検知された接触が移動である場合、方法２８００はステップ８１７へと進む。移動でない場合、方法２８００は終了する。ステップ８１７において、メッセージ種別がＭＯＶＥに設定され、方法２８００はステップ８０５へと進む。例えば、ＭＯＶＥメッセージは図１９及び図２２により定義されるような接触位置のＸＹ座標と共に送信することができ、ＰＲＥＳＳメッセージ及びＲＥＬＥＡＳＥメッセージは図１９及び図２３により定義されるような接触位置のＸＹ座標及びユーザインタフェースオブジェクト（すなわち、印１４のうちの１つ）と関連付けられるデータと共に送信することができる。ステップ８０７において、メッセージがＭＯＶＥであると判定された場合、ステップ８０９において、ＣＰＵ４５はコンピュータ１００にＭＯＶＥメッセージを送信する。ＣＰＵ２０５はＸＹ座標をカーソル情報として処理し、ビデオディスプレイ１０１に表示されるカーソルを移動させる。この場合、次のＲＥＬＥＡＳＥメッセージはカーソル位置で表示されるオブジェクトを選択する（例えば、プログラムを実行する、項目を選択する、又はＵＲＬをロードする）コマンドとして解釈することができる。更に、イベント座標なし（図１３参照）がカード１０に設定されている場合、リーダ１は接触の位置のＸＹ座標を送信せずに、ユーザインタフェースオブジェクトと関連付けられるデータをコンピュータ１００又はＳＴＢ６０１中のイベントマネージャ３０１に送信しても良い。

【０１８８】

加えて、アプリケーション３０４が図１７に示すようなユーザインタフェースオブジェクト構造及びステップ８０８のような照合機能を有する場合、リーダ１は接触位置のＸＹ座標をアプリケーション３０４に送信しても良い。その結果、ＣＰＵ２０５は同じ照合機能を実行し、ユーザインタフェースオブジェクトと関連付けられるデータをイベントマネージャ３０１から読み、読んだデータと関連付けられるサービス識別子１１０６により識別されるサービス（ゲームなど）をカードユーザに提供する。例えば、図４１のステップ４２０５において、ＣＰＵ２０５はメッセージのデータフィールドにデータが存在するか否かを判定する。データがデータフィールドにある場合、ＣＰＵ２０５はデータを読み、図４１の次のステップでデータの処理を行なう。データがデータフィールドに存在しない場合、ＣＰＵ２０５はメッセージからＸＹ座標を読み、その座標に対して照合機能を実行し、ユーザが押下した印と関連付けられるデータを取得する。また、イベントマネージャ３０１は、それ自体にとって利用可能なユーザインタフェースオブジェクト構造を使用してこの機能を実行することができる。

【０１８９】

従って、カードユーザがタッチペイン１８上で指を動かすことによってリーダ１を（カード１０を挿入せずに）マウスとして使用する場合、ユーザはＴＶディスプレイ上に表示されたＳＴＢメニューのＳＴＢサービスのうちの１つを選択することができる。また、カードユーザが挿入されたカード１０と共にリーダ１を使用し、何らかの印１４を選択する場合、ユーザはコンピュータ１００又はＳＴＢ６０１からサービス（ゲームなど）を受信する。特に、ユーザがＳＴＡＲＴ印を選択する場合、所望のゲームをコンピュータ１００又はＳＴＢ６０１において実行することができ、ゲーム中のオブジェクトがＫＩＣＫ印１４の選択に従ってボールをキックする。

【０１９０】

カード１０に対して予めカードごとのフラグ値を定義することによって、種々のカード１０をユーザに提供することができる。例えば、「移動イベントなし」のフラグ（すなわち、情報）が予めカード１０に設定されている場合、リーダ１はフラグに基づいてマウスとして動作しないように構成することができる。これに対し、「移動イベントなし」のフラグが予めカード１０に設定されていない場合、リーダ１はフラグに基づいてマウスとして機能するように構成することができる。

【０１９１】

図１３に示すように、リーダ１はデフォルト状態を有する。このデフォルト状態において、リーダ１は音声フィードバックを提供し、マウスとして機能し、押下イベント、押下解除イベント、及びその他のイベントに対する座標を送信する。また、リーダ１は音声フィードバックを提供せず、マウスとして機能せず、座標も送信しないデフォルト状態を提供することもできる。

【０１９２】

リーダ１がカードごとのフラグ値を使用して「ビープ機能」を実行するように構成される場合、リーダ１は「ビープ」を鳴らし、図２７及び図２８に示すフローチャートに従って方法を実行する。また、リーダ１がカードごとのフラグ値を使用して「マウス機能」を実行するように構成される場合、リーダ１はマウスとして機能し、図２７及び図２８のフローチャートに従って方法を実行する。更に、リーダ１がカードごとのフラグ値を使用して「照合機能」を実行するように構成される場合、リーダ１は押下イベント、押下解除イベント、及び移動イベントに対する座標を送信し、図２７及び図２８のフローチャートに従って方法を実行する。

【０１９３】

図２８のステップ８０８のようにＥＭ３０１でも照合機能が実行される。また、カード１０はマウス機能及び／又は基本機能（例えば、ユーザが選択した印と関連付けられるデータをＥＭ３０１に送信）のみを有するカードとして構成することもできる。このため、カードごとのフラグ値をランダムに組み合わせることによって、種々のカード１０をユーザに提供することができる。

【０１９４】

ここで説明するように、サービス識別子１１０６はシステム６００にとって不可欠な識別子である。少なくとも区別用識別子１１１０中のサービス識別子１１０６をＥＭ３０１に送信することによって、サービスをユーザに提供することができる。

【０１９５】

上述のサービス固有識別子１１０７は、特定のアプリケーションと共に使用されるようにベンダにより設定されるのが好ましい。このため、ベンダが各カード１０に対して一意的なサービス固有識別子１１０７を定義する場合、カード１０も一意的になるであろう。サービス固有識別子１１０７が特定のカードが配布された手段（郵便、列車内での配布など）についての情報を提供するのに使用されている場合、サービスにアクセスするのに使用されるのがどのカードであるかの記録を提供するファイルにそのサービス固有識別子１１０７を加えることができる。この情報はどれほど効果的な種々の配布手段が使用されたかを後で判定する際に使用される。

【０１９６】

６．９．４　１０ミリ秒待機プロセス

図２９は、１０ミリ秒待機ルーチン２９００を示すフローチャートである。１０ミリ秒待機ルーチン２９００は１０ミリ秒が経過するまでＣＰＵサイクルを消費するようにループする。遅延プロセス２９００はＣＰＵ４５により実行されるものであり、ステップ９０１で開始される。ステップ９０１において、所定のプロセスカウンタがクリアされる。次のステップ９０２において、カウンタが増分される。ステップ９０３において、１０ミリ秒が経過していない場合、方法２９００はステップ９０２に戻る。経過した場合、遅延プロセス２９００は終了する。

【０１９７】

７．０　イベントマネージャ
イベントマネージャ３０１は、ソフトウェアアーキテクチャ２００のプロセスコンポーネントのうちの１つである。イベントマネージャ３０１はアーキテクチャ２００の各規則を実施し、その他のプロセスコンポーネント間の整合性のある挙動を保証する。
【０１９８】
７．１　システムにおける役割
大半の通信はイベントマネージャ３０１を介して行なわれるので、イベントマネージャ３０１は、アーキテクチャ２００の中でもディレクトリサービス３１１コンポーネントを除く全てのプロセスコンポーネントが直接送受信可能でなければならない唯一のコンポーネントである。イベントマネージャ３０１はアーキテクチャ２００の各規則の実施者として機能するが、１つの別個のプログラムとして構成される必要はない。また、イベントマネージャ３０１は、イベントマネージャの役割の一部を実行する複数の高信頼中継器又はその他の別々のプロセスコンポーネントから形成することができる。これは、例えば、効率又は安全上の理由から行なうことができる。
【０１９９】
イベントマネージャ３０１は、Ｉ／Ｏデーモン３００及びランチャ３０３などのソフトウェアアーキテクチャ２００のその他の種々の部分を組み込んでも良い。イベントマネージャ３０１はブラウザコントローラなどのアプリケーションでさえも組み込んで良い。
【０２００】
イベントマネージャ３０１は、直接に又は高信頼中継器を介してディレクトリサービス３１１を除くシステム６００の全てのプロセスコンポーネントと通信することができる。これらのコンポーネントには、Ｉ／Ｏデーモン３００、ランチャ３０３、及びアプリケーション３０４のいずれかが含まれる。イベントマネージャ３０１は他のプロセスコンポーネントと通信するのに任意の適切な通信方法を使用することができる。好適な通信方法は、ほぼ世界的に実現されている伝送制御プロトコル／インターネットプロトコル（ＴＣＰ／ＩＰ）であるが、Ｕｎｉｘ（登録商標）ソケットなどのその他のＯＳ特有の方法も使用することができる。プロセスコンポーネントが統合されるときには、通信するのに使用される方法は内部データの別々のスレッド間での送受信となる可能性がある。
【０２０１】
イベントマネージャ３０１は、イベントマネージャ３０１をキルすることが可能な、あるいは、イベントマネージャ３０１のＣＰＵ時間又はネットワーク帯域幅を枯渇させることが可能な他のプロセスを含む他のプロセスコンポーネントからの干渉に影響されないように構成されるのが好ましい。これにより、イベントマネージャ３０１は確実にシステム６００の最終的な制御を維持することができる。
【０２０２】
７．２　内部の要求事項
イベントマネージャ３０１は、ポーリング（注：ポーリングはＣＰＵ負荷の関係で推奨しない）、割込み駆動型Ｉ／Ｏ、各コンポーネントを読み書きする別々のスレッドを有する方法、又は目的を達成するその他の任意の適切な方法などの方法により、アーキテクチャ２００の他の全てのプロセスコンポーネント３００、３０３、３０４、及び３０６に対して無閉塞Ｉ／Ｏを実行する。これにより、１つのコンポーネントの寿命が別のコンポーネントにより尽きてしまうことが確実になくなり、ユーザの待ち時間も減少する。
【０２０３】
また、イベントマネージャ３０１は、全ての入力データの有効性をチェックし、可能であれば出力前にデータを修復するように構成される。これは、高信頼コンポーネントからのデータを含む。イベントマネージャ３０１もフェールセーフであるのが好ましい。コンポーネント３００、３０３、３０４、及び３０６のうちの１つから不測のデータを受信する場合、イベントマネージャ３０１はそのデータを処理し、完全に不可避でない限りは終了しないように構成される。
【０２０４】

イベントマネージャ３０１は、相当に長い時間実行されるように要求される可能性があり、時間が経過しても性能が低下しないことを保証するように構成される。イベントマネージャ３０１は、伝送メカニズムが所定のイベントマネージャプロトコル（すなわち、ＥＭプロトコル）を使用する任意のコンポーネントとの通信において信頼性が高いことを前提とするのが好ましいが、Ｉ／Ｏデーモン３００を介してリモートリーダ１と通信を行なうのに使用される伝送メカニズムは信頼性が低く、入力データの一部が正しくないか、あるいは、欠落している恐れがあることを前提としている。

【０２０５】

７．３　手順

イベントマネージャ３０１は、システム６００の動作の一部に直接関与しているが、アーキテクチャ２００のその他の動作の多くにも透過的に参加している。イベントマネージャ３０１は、データパケットの通過の際にこれを変更せずに使用する点で透過的である。特に、第８．０節を参照して手順を詳細に後述する。

【０２０６】

図３０は、ソフトウェアアーキテクチャ２００を組み込むシステム６００により実行されるイベントの概要プロセス３０１０を示すフローチャートである。プロセス３０１０は、システム６００の構成によってＣＰＵ２０５により実行される。プロセス３０１０はステップ３０００で開始される。ステップ３０００において、イベントマネージャ３０１の開始を含むシステム初期化ルーチンが実行される。ステップ３０００において、Ｉ／Ｏデーモンも通常イベントマネージャ３０１と共に開始される。

【０２０７】

次のステップ３７００において、イベントマネージャ３０１はランチャ３０３を開始させる。ステップ３３００において、イベントマネージャ３０１はメッセージをランチャ３０３に渡し、ランチャ３０３がどのアプリケーション３０４を実行すべきかを判定できるようにする。ランチャ３０３は対応するアプリケーション３０４を開始させる。プロセス３０１０は次のステップ３４００へと続く。ステップ３４００において、新規のカード１０がリーダ１に挿入されるときなどに現在実行中のアプリケーション３０４が必要とされなくなると、ランチャ３０３は実行中のアプリケーションの実行を終了するために実行中のアプリケーションに終了メッセージを供給する。システム６００の電源が落とされる（スイッチを切られる）ときには全てのアプリケーションが終了させられる。

【０２０８】

図３１は、イベントマネージャ３０１により実行されるイベントを受信する方法３０００を示すフローチャートである。方法３０００は、コンピュータで実現する場合にはＣＰＵ２０５により実行することができる。また、セットトップボックスで実現する場合にはＣＰＵ４３０５により実行することができる。方法３０００はステップ３１０１で開始される。ステップ３１０１において、ランチャ３０３が開始される。次のステップ３１０３において、イベントマネージャ３０１はイベントを受信する。次のステップ３１０５において、ステップ３１０３で受信されたイベントがリモートリーダ１からのものでない場合、方法３０００はステップ３１０７へと進む。ステップ３１０７において、コンポーネント識別子（ＸＩＤ）がチェックされ、必要に応じて訂正される。方法３０００は次のステップ３１０９へと続く。ステップ３１０９において、イベントを送信しようとする新規のアプリケーションがそのイベントの送信を許可される場合、方法３０００はステップ３１１１へと進む。ステップ３１１１において、イベントが宛先プロセスコンポーネントに送信され、方法３０００はステップ３１０３へと戻る。送信元のアプリケーションがステップ３１０９においてイベントの送信を許可されない場合、方法３０００はステップ３１１３へと進む。ステップ３１１３において、イベントは放棄され、方法３０００はステップ３１１３へと戻る。

【０２０９】

ステップ３１０５において、イベントがリモートリーダ１からのものである場合、方法３０００はステップ３１１５へと進む。ステップ３１１５において、イベントがＢＡＤＣＡ

ＲＤイベント、ＬＯＷＢＡＴイベント、ＩＮＳＥＲＴイベント、又はＲＥＭＯＶＥイベントの場合、方法３０００はステップ３１１７へと進む。それ以外のイベントの場合、方法３０００はステップ３１１９へと進む。ステップ３１１７において、イベントはランチャ３０３に渡され、方法３０００はステップ３１０３へと戻る。ステップ３１１９において、区別用識別子がＮＯ＿ＣＡＲＤ識別子である場合、ステップ３１１７において、対応するメッセージがランチャ３０３に渡される。ＮＯ＿ＣＡＲＤ識別子でない場合、方法３０００はステップ３１２１へと進む。ステップ３１２１において、区別用識別子のサービス識別子部分が現在のフロントアプリケーションを判定するのに使用されるサービス識別子と比較される。サービス識別子がフロントアプリケーションを判定するのに使用されたサービス識別子と同じでなく、区別用識別子のサービス識別子部分が特殊な総称サービス識別子でない場合、方法３０００はステップ３１１７へと進む。ステップ３１１７において、このメッセージはランチャ３０３に渡される。フロントアプリケーション判定用のサービス識別子と同じであるか、あるいは、特殊な総称サービス識別子である場合、方法３０００はステップ３１２３へと進む。ステップ３１２３において、イベントはフロントアプリケーションに送信され、方法３０００はステップ３１０３へと戻る。

【０２１０】

７．４　フォーカス変更

イベントマネージャ３０１は、現在のフロントアプリケーションに対するものでないＥＭ＿ＬＯＳＩＮＧ＿ＦＯＣＵＳイベントを問題なく無視することができる。イベントマネージャ３０１は、フロントアプリケーションになるアプリケーション及びそのアプリケーションと関連付けられるサービス識別子に対するＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳメッセージに注意する必要がある。イベントマネージャ３０１は、同じサービス識別子を有する同じアプリケーションに対する複数のＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベント及び現在フロントアプリケーションでないアプリケーションに対するＥＭ＿ＬＯＳＩＮＧ＿ＦＯＣＵＳイベントを問題なく無視することができる。無視されるメッセージは通常通りに渡される。

【０２１１】

７．５　リーダメッセージ

イベントマネージャ３０１は、メッセージを正しいコンポーネントに配布する役割も担う。イベントマネージャ３０１は所定のプロトコル規則に従うように構成されるが、これについては詳細に後述する。

【０２１２】

７．６　メッセージ送信に対する規制

イベントマネージャ３０１の更なる役割は、メッセージの送信に対する所定の規制を実施することである。

【０２１３】

８．０　イベントマネージャプロトコル

イベントマネージャプロトコル（ＥＭプロトコル）は、ディレクトリサービス３１１を除くアーキテクチャ２００の全てのコンポーネント間で通信を行なうのに使用されるプロトコルである。一般的に、全てのメッセージは指定受信者に渡される前にイベントマネージャ３０１を通過するように構成される。ＥＭプロトコルは、例えば、伝送制御プロトコル／インターネットプロトコル（ＴＣＰ／ＩＰ）などの信頼性の高い通信プロトコルの上で実現されるデータグラムベースのプロトコルである。イベントマネージャ３０１は、送信される全てのデータが変更されずに正しい順序で到着することを前提とするように構成される。イベントマネージャ３０１は、アーキテクチャ２００のプロセスコンポーネント間で同期をとる信頼性の高い方法があることを前提としない。

【０２１４】

全てのマルチバイト値がインターネットバイト順（すなわち、ビッグエンディアン（ｂｉｇ　ｅｎｄｉａｎ））で送信される。この例外は、各サービスを表す「区別用識別子」の値である。この値は、幾つかの１バイトから成るブロックとして送信され、常にそのよう

に（すなわち、区別用識別子の値はバイト配列の問題のため、通常、数値として格納されることはない）扱われる。

【０２１５】

８．１　通信方法

イベントマネージャプロトコルは、アーキテクチャ２００のコンポーネントとシステム６００のハードウェアコンポーネントとの間での通信方法のように、ＴＣＰ／ＩＰを前提とするように構成されるのが好ましい。また、信頼性の高い転送を保証する既知の通信方法を使用することもできる。例えば、「ＵＮＩＸソケット」などのオペレーティングシステム特有の方法も使用することができる。データは、マルチスレッドのアプリケーションなどにおける内部データ構造を直接介してプロセスコンポーネント３０１、３０３、３０４、及び３０６間で受け渡しすることができる。

【０２１６】

コンポーネント間の通信の代替の方法が使用されるアーキテクチャの場合、バイト配列の問題が考慮されなければならない。アプリケーションが異なるバイト配列を有するマシン上で実行することが可能であるか、あるいは、データがネットワークバイト順（全てのコンポーネントはデフォルトでこの順序を前提とする）であることを期待するコンポーネントと通信を行なうことが要求される場合、全ての対象となる通信はネットワークバイト順で行なうことができる。

【０２１７】

８．２　データ形式

８．２．１　基本的なデータ型

データ型に言及すべく以下の段落で使用される略語の一部は以下の通りである：

・ｉｎｔ８：　　８ビット符号付き値、
・ｕｎｉｔ８：　８ビット符号無し値、
・ｉｎｔ１６：　１６ビット符号付き値、
・ｕｎｉｔ１６：　１６ビット符号無し値、
・ｉｎｔ３２：　３２ビット符号付き値、
・ｕｎｉｔ３２：　３２ビット符号無し値、及び
・ｘｉｄ＿ｔ：　３２ビット符号無し値

８．２．２　コンポーネントアドレッシング

アーキテクチャ２００中の全てのアドレス指定可能なプロセスコンポーネントには、「ｘｉｄ」（又はコンポーネント識別子）と呼ばれる３２ビット符号無し値が割り当てられる。この数値は個々のシステム６００のインスタンスの範囲内では一意的である。プロセスコンポーネントの一部のｘｉｄは常に同じである。そのｘｉｄは以下の通りである：

・イベントマネージャ３０１：ＥＭ＿ＥＶＥＮＴ＿ＭＡＮＧＥＲ＿ＸＩＤ
・マスタランチャ：ＥＭ＿ＭＡＳＴＥＲ＿ＬＡＵＮＣＨＥＲ＿ＸＩＤ
・ランチャ３０３：ＥＭ＿ＦＩＲＳＴ＿ＡＰＰ＿ＸＩＤ
・ディスプレイマネージャ３０６：ＥＭ＿ＤＩＳＰＬＡＹ＿ＭＡＮＡＧＥＲ＿ＸＩＤ

ｘｉｄ値は１バイトの型フィールドと３バイトの識別子とに分割される。種々の型が以下の表１に示される。

【０２１８】

【表１】

表1

値	型
内部ｘｉｄ	これらｘｉｄ値は転送可能ではなく、全てのコンポーネントは内部で使用することができる。ＥＭにより見られる場合には放棄される
コアシステムのｘｉｄ	これらはユーザインタフェースカードシステムのコアシステムコンポーネントを識別する。これらのコンポーネントには、イベントマネージャ、ランチャ、及びマスタランチャが含まれる
標準アプリケーション	これは必要に応じてランチャにより開始／終了される標準アプリケーションである
特殊なアプリケーション	これはアプリケーションを開始／終了するための標準規則により制御されない特殊なアプリケーションを識別する。このアプリケーションは、ビデオオンデマンドプレーヤ又はブラウザコントローラのような他のアプリケーションにより制御することができる機能をユーザインタフェースカードシステムに提供するために作成されるアプリケーションである
リーダ	リーダにはイベントマネージャによりｘｉｄが割り当てられる。このｘｉｄは、イベントマネージャの存続中システムにアクセスするのに使用される各リーダに固有のものである。イベントマネージャ及びシステムが再始動される場合、リーダのｘｉｄは変更することになる

【０２１９】

８．３　メッセージ種別

ＥＭプロトコルには２２個のコアメッセージがあり、これらは以下のラベルを有するのが好ましい：

・ＥＭ＿ＮＥＷ＿ＬＡＵＮＣＨＥＲ
・ＥＭ＿ＫＩＬＬ＿ＬＡＵＮＣＨＥＲ
・ＥＭ＿ＡＰＰ＿ＲＥＧＩＳＴＥＲ
・ＥＭ＿ＥＸＩＴ＿ＮＯＷ
・ＥＭ＿ＣＬＯＳＥ
・ＥＭ＿ＡＰＰ＿ＳＴＡＲＴＩＮＧ
・ＥＭ＿ＡＰＰ＿ＤＹＩＮＧ
・ＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳ
・ＥＭ＿ＬＯＳＩＮＧ＿ＦＯＣＵＳ
・ＥＭ＿ＬＩＳＴ＿ＭＥＳＳＡＧＥＳ
・ＥＭ＿ＬＩＳＴ＿ＡＰＰＳ
・ＥＭ＿ＳＥＮＤ＿ＭＥＳＳＡＧＥ
・ＥＭ＿ＰＯＳＴ＿ＭＥＳＳＡＧＥ
・ＥＭ＿ＧＥＴ＿ＭＥＳＳＡＧＥ
・ＥＭ＿ＤＥＬＥＴＥ＿ＭＥＳＳＡＧＥ
・ＥＭ＿ＲＥＡＤＥＲ＿ＩＮＳＥＲＴ
・ＥＭ＿ＲＥＡＤＥＲ＿ＲＥＭＯＶＥ
・ＥＭ＿ＲＥＡＤＥＲ＿ＢＡＤＣＡＲＤ
・ＥＭ＿ＲＥＡＤＥＲ＿ＭＯＶＥ
・ＥＭ＿ＲＥＡＤＥＲ＿ＰＲＥＳＳ
・ＥＭ＿ＲＥＡＤＥＲ＿ＲＥＬＥＡＳＥ
・ＥＭ＿ＲＥＡＤＥＲ＿ＬＯＷ＿ＢＡＴＴ

これらのメッセージについては、以下の段落で詳細に説明する。

【０２２０】

８．３．１　メッセージヘッダ

システム６００内で送信されるメッセージは、好ましくは、以下の情報を含むヘッダ部分を有する：
ｖｅｒｓｉｏｎ：　これは、コンポーネントにより使用されるプロトコルのバージョン番号を表す。これは、常にＥＭ＿ＰＲＯＴＯＣＯＬ＿ＶＥＲＳＩＯＮに設定されるべきであり、ＥＭ＿ＰＲＯＴＯＣＯＬ＿ＶＥＲ　ＳＩＯＮはライブラリにより使用されるバージョンとしてライブラリヘッダ中に定義される。
ｔｙｐｅ：　これは、ヘッダが開始するメッセージ種別を表し、後述の列挙されたメッセージ種別のうちの１つに設定される。メッセージの長さにはラベルｄａｔａＬｅｎｇｔｈが割り当てられる
ｒｅｓｅｒｖｅｄ：　これは、この２バイトの値が予約されていることを表し、０に設定されるべきである。
ｔｉｍｅｓｔａｍｐ：　これは、データパケットのタイムスタンプを表す。
ｔｏ＿ｘｉｄ：　これは、特定のパケットの宛先ｘｉｄを表す。これは、パケットの最終宛先であり、イベントマネージャが指定の最終受信者である場合はイベントマネージャにのみ設定されるべきである。
ｆｒｏｍ＿ｘｉｄ：　これは、パケットの送信元ｘｉｄを表す。
ｄａｔａＬｅｎｇｔｈ：これは、ヘッダに続くデータの長さを表す。この値は０となる可能性がある。異なる種別のメッセージはメッセージヘッダに続くデータに対して異なる要求事項を課す。コンポーネントは種別からメッセージの長さを想定すべきでない。メッセージの適正なサイズと異なる場合であってもｄａｔａＬｅｎｇｔｈフィールドのバイト数は常に読まれ、適正でないｄａｔａＬｅｎｇｔｈによってのみストリームは破損する可能性があることが保証される。

【０２２１】

８．３．２　ＥＭ＿ＮＥＷ＿ＬＡＵＮＣＨＥＲ

ＥＭ＿ＮＥＷ＿ＬＡＵＮＣＨＥＲメッセージは、イベントマネージャ３０１が新規のランチャ３０３を要求するときに送信される。ソフトウェアアーキテクチャ２００がマスタランチャを含む場合、このメッセージはイベントマネージャ３０１とこのようなマスタランチャとの間でのみ送信される。このメッセージを含むパケットは、新規のランチャがイベントマネージャ３０１に接続する必要があるという情報も含む。ＥＭ＿ＮＥＷ＿ＬＡＵＮＣＨＥＲメッセージは以下の情報を含むのが好ましい：
ｐｏｒｔ：　これは、イベントマネージャ３０１が新規の接続の有無を聞き取っているポート番号を表す。
ｈｏｓｔ：　これは、イベントマネージャ３０１上で実行中のマシンのホスト名を表す。

【０２２２】

８．３．３　ＥＭ＿ＫＩＬＬ＿ＬＡＵＮＣＨＥＲ

ＥＭ＿ＫＩＬＬ＿ＬＡＵＮＣＨＥＲメッセージは、イベントマネージャ３０１がマスタランチャに現在のランチャ３０３をキルすることを望むときに送信される。ＥＭ＿ＫＩＬＬ＿ＬＡＵＮＣＨＥＲメッセージと関連付けられるデータはない。

【０２２３】

８．３．４　ＥＭ＿ＡＰＰ＿ＲＥＧＩＳＴＥＲ

ＥＭ＿ＡＰＰ＿ＲＥＧＩＳＴＥＲメッセージは、ランチャ３０３に対してアプリケーションが起動し、アーキテクチャ２００の残りのコンポーネントにメッセージを受信する準備ができたことを通知するときに送信される。登録される前にアプリケーション３０４が送信するメッセージはイベントマネージャ３０１により破棄されることになる。

【０２２４】

ＥＭ＿ＡＰＰ＿ＲＥＧＩＳＴＥＲメッセージは以下の情報を含むのが好ましい：
ｘｉｄ：　これは、アプリケーションランチャ３０３により割り当てられたコンポーネント識別子を表す。送信される情報の残りは、残りのフィールドが可変長であるために構造により表すことができない。ｘｉｄに続くデータは、一連のヌルで終結する

文字列であり、最大長は終端のヌルを除いて２５６文字である。この文字列は、ａからｚまでの小文字及び大文字、数字０から９、及び各文字（．，から＿）から構成される。文字列が２５６文字よりも長い場合、２５６文字で切り捨てられる。
Ａｐｐｌｉｃａｔｉｏｎ Ｎａｍｅ：これは、現在のアプリケーションを他のアプリケーションに対して識別するために使用される名前を表す。
Ｓｅｒｖｉｃｅ Ｇｒｏｕｐ：これは、アプリケーションがその一部となることを希望するサービスグループの１つ以上の名前を表す。
【０２２５】
ブラウザコントローラなどの永続的なアプリケーションは１度登録されるだけで良い。このような永続的なアプリケーションはＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントを取得するたびに登録する必要がない。
【０２２６】
８．３．５　ＥＭ＿ＥＸＩＴ＿ＮＯＷ
ＥＭ＿ＥＸＩＴ＿ＮＯＷメッセージは、アプリケーションが強制されて終了しようとするときにランチャ３０３により送信される。ＥＭ＿ＥＸＩＴ＿ＮＯＷメッセージと関連付けられるデータはない。
【０２２７】
８．３．６　ＥＭ＿ＣＬＯＳＥ
ＥＭ＿ＣＬＯＳＥメッセージは、現在のセッションが終了することを示し、アプリケーションを起動状態に戻すために永続的なアプリケーションに送信される。このメッセージを受信すると、アプリケーションは入力／出力フォーカスの変更ではなく、新規のセッションの開始として次のＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントを処理するように要求される。ＥＭ＿ＣＬＯＳＥメッセージと関連付けられるデータはない。
【０２２８】
８．３．７　ＥＭ＿ＡＰＰ＿ＳＴＡＲＴＩＮＧ
ＥＭ＿ＡＰＰ＿ＳＴＡＲＴＩＮＧメッセージは、アプリケーションが始動しようとしているときにランチャ３０３によりイベントマネージャ３０１に送信される。ＥＭ＿ＡＰＰ＿ＳＴＡＲＴＩＮＧメッセージは以下の情報を含むのが好ましい：
ｘｉｄ：　　これは、始動しようとするアプリケーションのコンポーネント識別子を表す。
【０２２９】
８．３．８　ＥＭ＿ＡＰＰ＿ＤＹＩＮＧ
ＥＭ＿ＡＰＰ＿ＤＹＩＮＧメッセージは、アプリケーションが終了したときにランチャ３０３によりイベントマネージャ３０１に送信される。ＥＭ＿ＡＰＰ＿ＤＹＩＮＧメッセージはランチャ３０３がアプリケーションの終了を確信した後で初めて送信される。ＥＭ＿ＡＰＰ＿ＤＹＩＮＧメッセージは以下の情報を含むのが好ましい：
ｘｉｄ：　　これは、終了したアプリケーションのコンポーネント識別子を表す。
【０２３０】
８．３．９　ＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳ
ＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳメッセージは、アプリケーション３０４がリモートリーダ１からの入力の受信を開始しようとするときにランチャ３０３によりアプリケーションに送信される。ＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳメッセージは以下の情報を含むのが好ましい：
ｉｄ：　　これは、アプリケーションに送信されるリモートリーダ１のメッセージの区別用識別子を表す。
Ｄａｔａ：　　これは、フォーカスを受け取ろうとするときにアプリケーションに送信される追加のデータを表す。これは各サービス特有のものであり、データを解釈するのはアプリケーションの責任である。追加のデータは、バイト配列の問題に関してチェックを受けない。これはアプリケーションが対処すべきである。マルチバイトデータはア

プリケーションによりネットワークバイト順で送信され、受信側アプリケーションはこの順序であることを前提とする。受信側アプリケーションがブラウザコントローラであるとき、このデータの一例はブラウザコントローラがロードするように指示されるＵＲＬである。

【０２３１】

８．３．１０　ＥＭ＿ＬＯＳＩＮＧ＿ＦＯＣＵＳ

ＥＭ＿ＬＯＳＩＮＧ＿ＦＯＣＵＳメッセージは、アプリケーション３０４がリモートリーダ１及びディスプレイ１０１からの入力／出力フォーカスを手放そうとするときに送信される。ＥＭ＿ＬＯＳＩＮＧ＿ＦＯＣＵＳメッセージには追加のデータはない。

【０２３２】

８．３．１１　ＥＭ＿ＬＩＳＴ＿ＡＰＰＳ

ＥＭ＿ＬＩＳＴ＿ＡＰＰＳメッセージは、アプリケーションがある時点において他のどのアプリケーションが実行中であるか知ることを希望するときに送信される。ＥＭ＿ＬＩＳＴ＿ＡＰＰＳメッセージは、アプリケーションリストを含むデータフィールドを伴ってアプリケーションに戻される。このメッセージはプロセスコンポーネント３０１から３０６のうちのいずれかに宛てて送信される必要はない。イベントマネージャ３０１は、ヘッダのｔｏ＿ｘｉｄフィールドに関係なく、ＥＭ＿ＬＩＳＴ＿ＡＰＰＳメッセージが正しいコンポーネント（通常、ランチャ３０３）に送信されることを確認する。どのアプリケーションを列挙すべきか決定するのが受信側コンポーネントの役割である。

【０２３３】

応答として使用される場合、ＥＭ＿ＬＩＳＴ＿ＡＰＰＳメッセージは２つの形式を有する。第１の形式はＥＭ＿ＬＩＳＴ＿ＡＰＰＳが要求として送信されるときに使用される形式であり、第２の形式は応答として送信されるときの形式である。要求と関連付けられる追加のデータはない。

【０２３４】

ＥＭ＿ＬＩＳＴ＿ＡＰＰＳメッセージは以下の情報を含むのが好ましい：

ａｐｐ＿ｘｉｄ：　これは、記述されるアプリケーションのｘｉｄを表す。

ａｐｐ＿ｄｅｓｃ：　これは、アプリケーションが最初に登録されるときにランチャ３０３に与えられる名前の文字列を表す。

【０２３５】

８．３．１２　ＥＭ＿ＳＥＮＤ＿ＭＥＳＳＡＧＥ

ＥＭ＿ＳＥＮＤ＿ＭＥＳＳＡＧＥメッセージは、システム６００中の任意の２つの同時に実行されているアプリケーション間で送信することができる。アーキテクチャ２００によりこのメッセージに強制される構造はないが、通信アプリケーションは共通のデータ構造について同意する必要がある。

【０２３６】

８．３．１３　ＥＭ＿ＬＩＳＴ＿ＭＥＳＳＡＧＥＳ

ＥＭ＿ＬＩＳＴ＿ＭＥＳＳＡＧＥＳメッセージは、現在の掲示板（ｍｅｓｓａｇｅ ｂｏａｒｄ）にある全てのメッセージのリストを取得するのに使用される。この掲示板はアーキテクチャ２００において使用される。掲示板の詳細については、第８．４．７．１節を参照して後述することとする。ＥＭ＿ＬＩＳＴ＿ＭＥＳＳＡＧＥＳメッセージは要求形式及び応答形式を有する。要求形式と関連付けられるデータはない。応答は以下の情報を含むのが好ましい：

ｍｅｓｓａｇｅ＿ｃｏｕｎｔ：これは、現在掲示板にあるメッセージの数を表し、０に等しくなる可能性がある。

Ｍｅｓｓａｇｅｓ：　これは、以下の構造を有する可変サイズの構造の可変数（すなわち、ｍｅｓｓａｇｅ＿ｃｏｕｎｔに等しい）を表す。

【０２３７】

各メッセージは以下の情報を含むのが好ましい：

ｍｅｓｓａｇｅ＿ｉｄ：これは、このメッセージのメッセージ識別子を表す。

ｐｏｓｔｅｒ＿ｉｄ：　これは、このメッセージを提示するコンポーネントのｘｉｄ（コンポーネント識別子）を表す。
ｍｉｍｅ＿ｔｙｐｅ：　これは、このメッセージと関連付けられるデータのＭｕｌｔｉｐｕｒｐｏｓｅ　ｉｎｔｅｒｎｅｔ　Ｍａｉｌ　Ｅｘｔｅｎｓｉｏｎ－ｔｙｐｅ（ＭＩＭＥタイプ）を表し、ゼロ長となり得るヌル終結文字列である。この場合でも、終端の０は存在する。
ｍｅｓｓａｇｅ＿ｄｅｓｃ：これは、メッセージが提示アプリケーションにより提示されたときに割り当てられたこのメッセージの記述を表す。これは、終端の０を含まない最大２５５文字長のヌル終結文字列である。この文字列の長さは０の可能性がある。この場合でも、終端の０は存在する。

【０２３８】

８．３．１４　ＥＭ＿ＰＯＳＴ＿ＭＥＳＳＡＧＥ

ＥＭ＿ＰＯＳＴ＿ＭＥＳＳＡＧＥメッセージは、アーキテクチャ２００において使用される掲示板に何らかのデータを提示するのに使用される。このメッセージは、サービスグループの変更があるまで続き、任意の実行中のアプリケーションによりアクセスすることができる。また、ＥＭ＿ＰＯＳＴ＿ＭＥＳＳＡＧＥメッセージは任意の現在実行中のアプリケーションにより削除することができ、完全に信頼できるものであるとは想定されない。メッセージが提示されると、提示したアプリケーションにそのメッセージのメッセージ識別子を通知するために、アプリケーションにメッセージが戻される。このメッセージはアプリケーションによりランチャ３０３に送信される。アプリケーション（すなわち、メッセージを提示したアプリケーション）からのメッセージは以下の情報を含む：
ｍｅｓｓａｇｅ＿ｄｅｓｃ：これはメッセージの記述を表す。これは、終端の０を含まない最大２５５文字長となり得るヌル終結文字列である。記述は０バイトとなる可能性があるが、終端の０をもたなければならない。
ｍｉｍｅ＿ｔｙｐｅ：　これは、提示されるメッセージデータのＭＩＭＥタイプを表す。ＭＩＭＥタイプは必要でないが、終端の０はなければならない。
ｍｅｓｓａｇｅ＿ｄａｔａ：これは、掲示板に提示されるデータを表す。

【０２３９】

また、アプリケーションに戻されるメッセージは以下の情報を含むのが好ましい：
ｍｅｓｓａｇｅ＿ｉｄ：これはメッセージ識別子を表し、このメッセージ識別子によりメッセージの取得又は削除を行なうことができる。

【０２４０】

８．３．１５　ＥＭ＿ＧＥＴ＿ＭＥＳＳＡＧＥ

ＥＭ＿ＧＥＴ＿ＭＥＳＳＡＧＥメッセージは、掲示板からメッセージを取得するのに使用される。ＥＭ＿ＧＥＴ＿ＭＥＳＳＡＧＥメッセージは、コンポーネントが取得を希望するメッセージのメッセージ識別子を伴って送信され、その識別子を有するメッセージが存在しないというメッセージ又はエラーを伴ってコンポーネントに戻される。このメッセージはアプリケーション３０４によりランチャ３０３に送信される。

【０２４１】

メッセージの要求時に含まれる情報は以下の通りである：
ｍｅｓｓａｇｅ＿ｉｄ：これは、アプリケーションが取得を希望するメッセージのメッセージ識別子を表す。
ｆｌａｇｓ：　これはフラグワードである。全ての未使用ビットは０に設定されるべきである。フラグの記述は以下の表２に示される：

【０２４２】

【表２】

表2

フラグ	記述	値
EM_GM_DELETE	メッセージの送信後、それを掲示板から削除する	０ｘ０１

【０２４３】
応答は以下の情報を有する：
ｅｒｒｏｒ：　　エラーが発生すると、これは以下の表３中の値のうちの１つに設定される。
【０２４４】
【表３】
表3

値	記述
EM_GM_NO_ERROR	エラー発生なし。メッセージはメッセージフィールドにある
EM_GM_NO_SUCH_MESSAGE	掲示板にはそのメッセージ識別子を有するメッセージはない

【０２４５】
ｍｅｓｓａｇｅ＿ｉｄ：これは取得されたメッセージのメッセージ識別子を表す。
ｍｉｍｅ＿ｔｙｐｅ：　これは、取得されたメッセージのＭＩＭＥタイプを表す。これはヌル終結文字列である。このメッセージと関連付けられたＭＩＭＥタイプが存在しない場合、文字列はゼロ長であるが、終端の０は存在する。
ｍｅｓｓａｇｅ：　　エラーの発生がない場合、このフィールドは掲示板に提示されるデータを含むことになる。長さはヘッダ中のｄａｔａＬｅｎｇｔｈの値－エラーフィールドのサイズにより判定される。
【０２４６】
８．３．１６　ＥＭ＿ＤＥＬＥＴＥ＿ＭＥＳＳＡＧＥ
ＥＭ＿ＤＥＬＥＴＥ＿ＭＥＳＳＡＧＥメッセージは、掲示板からメッセージを削除するのに使用される。存在しないメッセージを削除するのはエラーではない。このメッセージはフロントアプリケーションによりランチャ３０３に送信される。ＥＭ＿ＤＥＬＥＴＥ＿ＭＥＳＳＡＧＥメッセージは以下の情報を含むのが好ましい：
ｍｅｓｓａｇｅ＿ｉｄ：これは、削除対象のメッセージのメッセージ識別子を表す。
【０２４７】
８．３．１７　ユーザインタフェースカードリーダメッセージ
ユーザインタフェースカードリーダメッセージはリモートリーダ１により生成され、イベントマネージャプロトコルに従うように、イベントマネージャ３０１によりカプセル化される。リモートリーダ１により生成されるメッセージは３種類ある。「単純な」メッセージ、「移動」メッセージ、及び「押下／押下解除」メッセージである。移動メッセージは座標が付加された単純なメッセージであり、押下／押下解除メッセージはデータ及び座標が付加された単純なメッセージである。
【０２４８】
８．３．１７．１　単純なメッセージ
以下のメッセージが単純なメッセージである：
・ＥＭ＿ＲＥＡＤＥＲ＿ＩＮＳＥＲＴ
・ＥＭ＿ＲＥＡＤＥＲ＿ＲＥＭＯＶＥ
・ＥＭ＿ＲＥＡＤＥＲ＿ＢＡＤＣＡＲＤ
・ＥＭ＿ＲＥＡＤＥＲ＿ＬＯＷ＿ＢＡＴＴ
これらの単純なメッセージは以下の情報を含むのが好ましい。
ｉｄ：　　これは、リモートリーダ１により送信された区別用識別子を表し、ＢＡＤＣＡ

ＲＤメッセージの場合は意味をもたない。
【０２４９】
８．３．１７．２　移動メッセージ
ＥＭ＿ＲＥＡＤＥＲ＿ＭＯＶＥメッセージは以下の情報を含むのが好ましい：
ｉｄ：　これは、リモートリーダ１により送信された区別用識別子を表し、カードなしメッセージの場合、全てが０に設定される。
Ｘ：　これは、ｘ値を表す。
Ｙ：　これは、ｙ値を表す。
【０２５０】
８．３．１７．３　押下／押下解除メッセージ
ＥＭ＿ＲＥＡＤＥＲ＿ＰＲＥＳＳメッセージ及びＥＭ＿ＲＥＡＤＥＲ＿ＲＥＬＥＡＳＥメッセージは以下の情報を含むのが好ましい：
ｉｄ：　これは、リモートリーダ１により送信された区別用識別子を表す。
Ｘ：　これは、ｘ値を表す。
Ｙ：　これは、ｙ値を表す。
ｄａｔａ：これは、押下又は押下解除と関連付けられる（ユーザインタフェース要素データと関連付けられる）任意のデータを表す。
【０２５１】
８．４　手順
以下の段落は、アーキテクチャ２００の各プロセスコンポーネントが後続するメインの手順を説明する。
【０２５２】
８．４．１　新規のアプリケーションの開始
図３２は、新規のアプリケーションを開始する方法３３００の詳細を示すフローチャートであり、ランチャ３０３が新規のアプリケーションを開始するたびに実行される。方法３３００は、コンピュータで実現する場合にはＣＰＵ２０５により実行することができる。また、セットトップボックスで実現する場合にはＣＰＵ４３０５により実行することができる。方法３３００は第１のステップ３３０１で開始される。ステップ３３０１において、ランチャ３０３はサービス識別子をＵＲＬへと変換するためのマッピングを実行する。次のステップ３３０３において、ランチャ３０３はイベントマネージャのホスト名及びポート番号をランチャ３０３に通知するアプリケーションを取り込み、開始させる。方法３３００は次のステップ３３０５へと続く。ステップ３３０５において、ランチャ３０３はイベントマネージャ３０１に開始するアプリケーションのｘｉｄを通知するＥＭ＿ＡＰＰ＿ＳＴＡＲＴＩＮＧメッセージをイベントマネージャ３０１に送信する。次のステップ３３０７において、新規のアプリケーションはイベントマネージャ３０１に接続し、ランチャ３０３にＥＭ＿ＡＰＰ＿ＲＥＧＩＳＴＥＲメッセージを送信する。更に、通常は、新規のアプリケーションへのフォーカス変更が存在する。
【０２５３】
８．４．２　アプリケーションの終了
図３３は、ソフトウェアアーキテクチャ２００を組み込むシステム６００においてアプリケーションを終了する方法３４００を示すフローチャートである。方法３４００は、コンピュータで実現する場合にはＣＰＵ２０５により実行することができる。また、セットトップボックスで実現する場合にはＣＰＵ４３０５により実行することができる。この方法はランチャ３０３が実行中のアプリケーションを終了させるたびに使用される。方法３４００はステップ３４０１で開始される。ステップ３４０１において、ランチャ３０３は実行中のアプリケーションにＥＭ＿ＥＸＩＴ＿ＮＯＷメッセージを送信する。ランチャ３０３はこの時点でタイムアウトを設定し、作業を片付けて終了する機会をアプリケーションに与える。次のステップ３４０３において、実行中のアプリケーションは片付け作業を行ない、終了する。これに対し、アプリケーションがＥＭ＿ＥＸＩＴ＿ＮＯＷメッセージを無視すると、ランチャ３０３がタイムアウトになり、アプリケーションを強制的に終了さ

せる。ステップ３４０５において、ランチャ３０３はイベントマネージャ３０１にＥＭ＿ＡＰＰ＿ＤＹＩＮＧメッセージを送信してアプリケーションの終了を通知すると共に、あらゆる待ち状態のデータを破棄し、アプリケーションへの接続がオープンのままである場合には接続を終了すべきであると通知する。方法３４００は終了する。

【０２５４】

８．４．３　永続的なアプリケーションのセッションの終了

図３４は、ソフトウェアアーキテクチャ２００を組み込むシステム６００上で永続的なアプリケーションの現在のセッションを終了する方法３５００を示すフローチャートである。方法３５００は、コンピュータで実現する場合にはＣＰＵ２０５により実行することができる。また、セットトップボックスで実現する場合にはＣＰＵ４３０５により実行することができる。方法３５００はアプリケーションの終了と類似しているが、アプリケーションが実際に終了する訳ではない。方法３５００はステップ３５０１で開始される。ステップ３５０１において、ランチャ３０３は永続的なアプリケーションにＥＭ＿ＣＬＯＳＥメッセージを送信する。次のステップ３５０３において、永続的なアプリケーションは初期状態にリセットし、方法３５００は終了する。これは、外部のサーバへの接続の終了、デフォルトのウェブページのロードなどを伴っても良い。永続的なアプリケーションが受信する次のＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントは、新規のセッションの開始であると想定される。

【０２５５】

８．４．４　フォーカス変更

図３５は、ソフトウェアアーキテクチャ２００を組み込むシステム６００上でフォーカス変更を実行する方法３６００を示すフローチャートである。方法３６００は、コンピュータで実現する場合にはＣＰＵ２０５により実行することができる。また、セットトップボックスで実現する場合にはＣＰＵ４３０５により実行することができる。方法３６００は、アプリケーションに入力／出力フォーカスを受け取る又は失うことを通知するのに使用される。この通知はアプリケーションが終了する信号ではない。第１のステップ３６０１において、ランチャ３０３は現在入力／出力フォーカスを有するアプリケーションを変更する決定を行ない、通常はカード変更に基づいて入力フォーカスを受け取る予定のアプリケーションにＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントを送信する。このイベントの送信はイベントマネージャ３０１により使用され、所定の条件に基づいてどのアプリケーションが入力／出力フォーカスを受け取るべきかが決定される。ステップ３６０３において、ランチャ３０３は前のフロントアプリケーションにＥＭ＿ＬＯＳＩＮＧ＿ＦＯＣＵＳイベントを送信し、方法３６００は終了する。このメッセージの重要度は低く、現在のフロントアプリケーションが変わらないときには送信されないが、ＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳは必要である（すなわち、ＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントがブラウザコントローラ４０２にベースＵＲＬを通知するのに使用されるブラウザコントローラの場合）。

【０２５６】

８．４．５　メッセージの受け渡し（Ｍｅｓｓａｇｅ　Ｐａｓｓｉｎｇ）

アーキテクチャ２００によりサポートされるアプリケーション間のメッセージの受け渡しには２種類ある。現在のサービスグループと同程度に永続的な掲示板を介する方法と２つのコンポーネントが後述するように直接相互に通信を行なう直接メッセージ方式である。

【０２５７】

８．４．５．１　掲示板（Ｍｅｓｓａｇｅ　Ｂｏａｒｄ）

アーキテクチャ２００の１つのコンポーネント（通常は、ランチャ３０３）が掲示板を維持するが、イベントマネージャ３０１にはどのコンポーネントがこの維持を行なうかが分かっている。掲示板は、掲示板を管理するプロセスコンポーネントにより識別子として３２ビット符号無し数値を割り当てられたメッセージのリストから形成される。各メッセージは、テキスト記述、メッセージデータに対するオプションのＭＩＭＥタイプ、及びメッセージ自体から形成される。アプリケーションは、ＥＭ＿ＬＩＳＴ＿ＭＥＳＳＡＧＥＳメ

ッセージを送信することによって、現在掲示板にある全てのメッセージのリストを要求することができる。この要求に対して、関連するメッセージ識別子を伴って現在掲示板にある全てのメッセージのテキスト記述が戻されることになる。アプリケーションは、必要とするメッセージのメッセージ識別子と共にＥＭ＿ＧＥＴ＿ＭＥＳＳＡＧＥを送信することによって特定のメッセージを要求することができる。掲示板のリストの取得から実際のメッセージの要求までの間にメッセージが削除される可能性もある。ＥＭ＿ＧＥＴ＿ＭＥＳＳＡＧＥメッセージ応答のエラーフィールドはこれを示すように構成される。

【０２５８】

８．４．５．２　直接通信

２つのアプリケーションは、直接通信を使用することによって相互に任意のデータを直接送信することができる。これは、ＥＭ＿ＳＥＮＤ＿ＭＥＳＳＡＧＥメッセージを使用することによって一方のアプリケーションが他方のアプリケーションにデータを送信することにより実行される。２つのアプリケーションでは、このメッセージに対するデータ形式が一致する必要があり、バイト配列の問題を考慮しなければならない。相手側のアプリケーションのコンポーネント識別子を取得するために、アプリケーションはＥＭ＿ＬＩＳＴ＿ＡＰＰＳメッセージを送信することによって全ての実行中のアプリケーションのリストが送信されるように要求することができる。このメッセージは、現在実行中の全ての公開されているアプリケーションのリストを戻す。

【０２５９】

８．５　リーダメッセージ

本節では、各ＥＭ＿ＲＥＡＤＥＲ＿＊ メッセージを転送するためにイベントマネージャ３０１により使用される規則の概要を説明する。以下のメッセージは、どのアプリケーションが現在フォーカスをもっているかに関わらず、常にランチャ３０３に送信される。

・ＥＭ＿ＲＥＡＤＥＲ＿ＩＮＳＥＲＴ
・ＥＭ＿ＲＥＡＤＥＲ＿ＲＥＭＯＶＥ
・ＥＭ＿ＲＥＡＤＥＲ＿ＢＡＤＣＡＲＤ
・ＥＭ＿ＲＥＡＤＥＲ＿ＬＯＷ－ＢＡＴＴ

対応するフィールド１１０６中に現在のフロントアプリケーションと同じサービス識別子を有するカード１０からのものである場合、以下のメッセージは現在のフロントアプリケーションに送信される。サービス固有識別子はこの比較では考慮されない。サービス識別子が現在のフロントアプリケーションと異なる場合、あるいは、区別用識別子がＮＯ＿ＣＡＲＤの現在値（すなわち、全て０）である場合、メッセージは前述のようにランチャ３０３に送信される。

・ＥＭ＿ＲＥＡＤＥＲ＿ＰＲＥＳＳ
・ＥＭ＿ＲＥＡＤＥＲ＿ＲＥＬＥＡＳＥ
・ＥＭ＿ＲＥＡＤＥＲ＿ＭＯＶＥ

【０２６０】

８．６　メッセージ送信に対する規制

システム６００の安全性及び安定性を改善するためには、メッセージの送信に対して規制が課されるのが好ましい。この規則に違反するメッセージはイベントマネージャ３０１により破棄されることになる。

【０２６１】

８．６．１　全てのコンポーネントに対する規制

リモートリーダ１以外のコンポーネントにはＥＭ＿ＲＥＡＤＥＲ＿＊メッセージを送信する許可が与えられない。

【０２６２】

８．６．２　イベントマネージャに対する規制

イベントマネージャ３０１は規則の実施者であり、そのように任意の必要なメッセージを送信することができる。イベントマネージャ３０１は、ＥＭ＿ＫＩＬＬ＿ＬＡＵＮＣＨＥＲメッセージ及びＥＭ＿ＮＥＷ＿ＬＡＵＮＣＨＥＲメッセージのみを生成するだけで良い

ように構成されるが、メッセージをコピーし、そのコピーを目的のコンポーネントでないプロセスコンポーネントに送信することができる。また、イベントマネージャ３０１はコンポーネント間の全ての送信を扱う。

【０２６３】

８．６．３　ランチャに対する規制

ランチャ３０３は、アーキテクチャ２００の全てのコンポーネント３０１から３０６にメッセージを送信する。ランチャ３０３が送信することができないメッセージは以下の通りである：

・ＥＭ＿ＫＩＬＬ＿ＬＡＵＮＣＨＥＲ
・ＥＭ＿ＮＥＷ＿ＬＡＵＮＣＨＥＲ

【０２６４】

８．６．４　アプリケーションに対する規制

アプリケーションが他のアプリケーション（ランチャ３０３を含む）に送信するのは以下のメッセージのみである：

・ＥＭ＿ＡＰＰ＿ＲＥＧＩＳＴＥＲ
・ＥＭ＿ＳＥＮＤ＿ＭＥＳＳＡＧＥ
・ＥＭ＿ＬＩＳＴ＿ＡＰＰＳ
・ＥＭ＿ＰＯＳＴ＿ＭＥＳＳＡＧＥ
・ＥＭ＿ＧＥＴ＿ＭＥＳＳＡＧＥ
・ＥＭ＿ＤＥＬＥＴＥ＿ＭＥＳＳＡＧＥ
・ＥＭ＿ＬＩＳＴ＿ＭＥＳＳＡＧＥＳ

【０２６５】

８．７　コンポーネント手順リスト

本節では、アーキテクチャ２００の各コンポーネントが関係する機能を列挙する。

【０２６６】

８．７．１　イベントマネージャ

イベントマネージャ３０１は以下の手順に直接関与する：

・システム初期化
・システム起動
・新規のアプリケーションの開始
・アプリケーションの終了
・フォーカス変更
・メッセージの受け渡し
・リーダメッセージ

【０２６７】

８．７．２　ランチャ

ランチャ３０３は以下の手順に関与する：

・システム初期化
・システム起動
・新規のアプリケーションの開始
・アプリケーションの終了
・フォーカス変更
・メッセージの受け渡し（場合による）
・リーダメッセージ（場合による）

【０２６８】

８．７．３　アプリケーション

アプリケーション３０４は以下の手順に関与する：

・新規のアプリケーションの開始
・アプリケーションの終了
・アプリケーションが永続的な場合のセッションの終了

・フォーカス変更
・メッセージの受け渡し
・リーダメッセージ（場合による）
【０２６９】
９．０　Ｉ／Ｏデーモン
Ｉ／Ｏデーモン３００は、リモートリーダ１からイベントマネージャ３０１に送信され、双方向プロトコルの場合は逆方向にも送信されるデータを転送する役割を担う。Ｉ／Ｏデーモン３００は、システム６００のハードウェアから直接に、あるいは、ＩＲリンク又は標準シリアルハードウェア接続などのリモートリーダ１とのインタフェースであるオペレーティングシステムドライバを介して読むことができるように構成される。また、Ｉ／Ｏデーモン３００には、ＴＣＰ／ＩＰポートで聞き取りながらイベントマネージャ３０１が接続するのを待つことが要求される。ここで、Ｉ／Ｏデーモン３００はＴＣＰ／ＩＰストリーム中にカプセル化した状態でデータをリモートリーダ１からイベントマネージャ３０１に送信する。
【０２７０】
Ｉ／Ｏデーモン３００は、リモートリーダ１のデータをイベントマネージャ３０１に送信する場合及びＩ／Ｏデーモン３００とリモートリーダ１との間でのオプションの双方向プロトコル構成において逆方向の送信を行なう場合を除き、システム６００のその他の部分とは通信を行なわない。
【０２７１】
Ｉ／Ｏデーモン３００の機能はシステム６００に存在しなければならないが、Ｉ／Ｏデーモン３００は独立したコンポーネントである必要はない。例えば、イベントマネージャ３０１がリモートリーダ１とインタフェースをとるのに使用されるハードウェアと同じマシン上で実行されている場合、Ｉ／Ｏデーモン３００はイベントマネージャ３０１へと統合することができる。
【０２７２】
Ｉ／Ｏデーモン３００は、システム６００のその他の部分がリモートで実行されている場合、最小限のハードウェア上で稼働するように構成される。
【０２７３】
９．１　要求事項
９．１．１　一般的な要求事項
Ｉ／Ｏデーモン３００が実現されるプラットフォームは、リモートリーダ１から信号を受信（オプションとして、信号を送信）できるように構成されなければならない。また、プラットフォームは、システムのその他の部分（すなわち、イベントマネージャ（ＥＭ）３０１）と通信を行なうために、ＴＣＰ／ＩＰスタック又はその他の信頼性の高い通信方法を実現するのが好ましい。Ｉ／Ｏデーモン３００は、多重化Ｉ／Ｏを行なうように要求される可能性があり、アーキテクチャ２００のＩ／Ｏシステムは多重化Ｉ／Ｏをサポートするように構成されるのが好ましい。アーキテクチャ２００は、Ｉ／Ｏデーモン３００が聞き取りを行なうであろうポートを、例えば、コマンドライン引数として割り当てるように構成されるのが好ましい。
【０２７４】
９．１．２　内部の要求事項
Ｉ／Ｏデーモン３００は、リモートリーダ１により使用されるプロトコルを理解する必要はない。Ｉ／Ｏデーモン３００に要求されるのは、受信する全てのデータを任意の聞き取り中のＥＭ（イベントマネージャ）に転送することのみである。Ｉ／Ｏデーモン３００は、通信リンクのトランスポートプロトコルにより（すなわち、エラー訂正コード又はそれに類するものを介して）サポートされない限り、リモートリーダ１からの送信のエラーを訂正する必要がない。使用中のトランスポートプロトコルがエラー検出をサポートするが、エラー訂正はサポートしない場合、エラーチェックに合格しないデータをイベントマネージャ３０１に渡すことができる。

【０２７５】
９．１．３　外部インタフェースの要求事項
Ｉ／Ｏデーモン３００は、１つ以上のＴＣＰ／ＩＰ接続を受け入れることができるのが好ましい。イベントマネージャ３０１に送信されるデータストリームは、リモートリーダ１により送信されるデータの内容である。使用される通信プロトコルの一部として送信される全てのヘッダ情報及びフッタ情報は除去されるのが好ましい。バイト配列はビッグエンディアンである。アーキテクチャ２００の通信方法が使用不可能になる（例えば、エラー発生が原因）場合、Ｉ／Ｏデーモン３００はエラー状態が起こると即座に全ての接続を終了する。
【０２７６】
９．２　外部インタフェース
Ｉ／Ｏデーモン３００の外部インタフェース（不図示）は、最小限のハードウェア上で実行可能であるように意図的に簡単なものとなっている。Ｉ／Ｏデーモン３００は以下のように構成されるのが好ましい。
【０２７７】
９．２．１　起動手順
Ｉ／Ｏデーモン３００は、何らかの方法で、例えば、コマンドライン引数により指定されるＴＣＰ／ＩＰポートで聞き取りを行なう。Ｉ／Ｏデーモン３００にＴＣＰ／ＩＰポートを通知する厳密な方法は実現例に依存する。リモートリーダ１と通信するために使用される通信ハードウェアは必要に応じて初期化され、リモートリーダ１から送信されるデータを読む方法はデータを受信できる状態にあるように構成される。接続を待つ間、Ｉ／Ｏデーモン３００はリモートリーダ１により送信されるデータを消費するので、接続されたときには新規のデータのみが送信される。この新規のデータはメッセージ境界で開始する必要がない。
【０２７８】
９．２．２　イベントマネージャからの接続
ＴＣＰ／ＩＰポートで接続が成功する場合、Ｉ／Ｏデーモン３００はその接続を受け入れ、接続を下ってリモートリーダ１から受信されるデータの送信を開始するように構成される。Ｉ／Ｏデーモン３００が既にイベントマネージャ（ＥＭ）３０１に接続されている場合、Ｉ／Ｏデーモン３００には２つの選択肢がある。まず、Ｉ／Ｏデーモンは接続を受け入れ、全ての現在接続されているイベントマネージャを下って全てのデータを送信することができる。この選択肢はシステムデバッギングのために提供される。第２の方法は第２の接続を拒否し、既に接続されているＥＭへのデータの送信を継続する方法である。ストリームの暗号化は、ポートトンネリングなどの他の何らかの方法によって外部で処理することができる。
【０２７９】
９．２．３　イベントマネージャからの接続の終了
いつでもイベントマネージャ３０１への接続が終了する場合には、Ｉ／Ｏデーモン３００はイベントマネージャ３０１に送信されるのを待っているリモートリーダ１からのデータを破棄するように構成される。これが接続されている唯一のイベントマネージャである場合、Ｉ／Ｏデーモン３００は初期の起動状態に戻るように構成される。これにより、Ｉ／Ｏデーモン３００はリモートリーダ１により送信されるデータを消費しながら接続を待つ。
【０２８０】
９．２．４　回復不能エラーの出現
対処不可能であり、Ｉ／Ｏデーモン３００の終了を引き起こすであろうエラーをＩ／Ｏデーモン３００が検出する場合、Ｉ／Ｏデーモン３００はＥＭへの全ての接続を終了し、ＥＭにＩ／Ｏデーモン３００がエラーを検出したことを通知するように構成される。エラーの例としては、リモートリーダ１と通信するのに使用されるハードウェアが使用不可能になる場合、あるいは、Ｉ／Ｏデーモン３００が終了を引き起こすであろう信号を受信する

場合が含まれる。Ｉ／Ｏデーモン３００はエラーに遭遇すると即座に全ての接続を終了するように構成される。

【０２８１】

１０．０　ランチャ

ランチャ３０３は、許可されたアプリケーション及び基本アプリケーションの構成規則などのサイト特有の規則を実施するプロセスコンポーネントである。ランチャ３０３は、システムアーキテクチャ２００の他のコンポーネントプロセス３００、３０１、３０４、３０５、及び３０６が、一般家庭のセットトップボックス６０１から非常に特殊なアプリケーション（例えば、現金自動預払機）に至る広範囲のアプリケーションにおいて使用されるのを許可する。ランチャ３０３は、ネットワークごと又は施設ごとに個別に作成することができる。

【０２８２】

ランチャ３０３は特権を有するように構成される。例えば、ランチャ３０３は、システム６００が起動する際にイベントマネージャ３０１に接続する最初のコンポーネントになるように構成することができる。更に、ランチャ３０３は、リモートリーダ１により送信される全ての「ＬＯＷ＿ＢＡＴＴ」メッセージ、「ＢＡＤＣＡＲＤ」メッセージ、「ＩＮＳＥＲＴ」メッセージ、及び「ＲＥＭＯＶＥ」メッセージを受信すると共に、任意の時点においてフロントアプリケーションが関連付けられるスマートカード１０以外のカードから送信される全ての「ＰＲＥＳＳ」メッセージ、「ＲＥＬＥＡＳＥ」メッセージ、及び「ＭＯＶＥ」メッセージを受信する。また、ランチャ３０３は特殊な「ＮＯ＿ＣＡＲＤ」区別用識別子を有するＰＲＥＳＳメッセージ、ＲＥＬＥＡＳＥメッセージ、及びＭＯＶＥメッセージも受信する。ランチャ３０３は、更に、ＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベント及びＥＭ＿ＬＯＳＩＮＧ＿ＦＯＣＵＳイベントを介してどのアプリケーションをフロントアプリケーションにするかに関する制御権を有する。

【０２８３】

ランチャ３０３は、アプリケーションがいつ開始され、いつ終了させられる必要があるかを決定するように構成される。また、ランチャ３０３は、常にそうである訳ではないが、アプリケーションを開始／終了するのに使用される。この役割は、例えば、アプリケーション３０４がアーキテクチャ２００のその他のコンポーネントとは別のマシンで実行される場合に、ランチャ３０３の指示に応じて別のアプリケーションが引き受けることができる。

【０２８４】

イベントが現在のフロントアプリケーションではなく、ランチャ３０３に送信されることにより、ランチャ３０３はどのアプリケーションが任意の時点において実行されるべきかを決定することができる。アプリケーションを強制的に終了するように構成されることは、ランチャ３０３がどのアプリケーションが現在実行中であるべきかを強制できることを意味する。また、ランチャ３０３には、アプリケーションをいつ開始／終了するかをイベントマネージャ３０１に通知することも要求される。

【０２８５】

図３６は、ランチャ３０３により実行される方法３７００の概要を示すフローチャートである。方法３７００は、コンピュータで実現する場合にはＣＰＵ２０５により実行することができる。また、セットトップボックスで実現する場合にはＣＰＵ４３０５により実行することができ、リモートサーバのＣＰＵによっても実行することができる。方法３７００は第１のステップ３７０１で開始される。ステップ３７０１において、ランチャ３０３はイベントマネージャ３０１に接続し、次のステップ３７０２へと続く。ステップ３７０２において、永続的なアプリケーションが開始される。次のステップ３７０３において、ランチャ３０３はイベントを待ち、イベントが受信されるとステップ３７０５へと進む。ステップ３７０５において、イベントがＮＯ＿ＣＡＲＤ識別子である場合、プロセスはステップ３７０７へと進む。ＮＯ＿ＣＡＲＤ識別子でない場合、プロセスはステップ３７０９へと進む。ステップ３７０７において、ランチャ３０３はＮＯ＿ＣＡＲＤ識別子に応じ

て所定のシステム特有の機能（例えば、ディスプレイ１０１にメッセージを表示）を実行し、方法３７００はステップ３７０３へと戻る。

【０２８６】

決定ステップ３７０５におけるイベントがＮＯ＿ＣＡＲＤ識別子でないと判定されると、別の決定ステップ３７０９へと入り、イベントがＰＲＥＳＳ、ＲＥＬＥＡＳＥ、ＲＥＭＯＶＥ、又はＭＯＶＥであるか否かが判定される。この決定ステップ３７０９が「ｙｅｓ」を戻す場合、すなわち、イベントが前述のイベントのうちの１つである場合、方法３７００はステップ３８００へと進む。イベントが前述のイベントのうちの１つでない場合、方法３７００は更なる決定ステップ３７１３へと進む。ステップ３８００において、ランチャ３０３はフローチャートの図３７を参照して説明したプロセスステップに従ってアプリケーションを変更する。

【０２８７】

ステップ３７０９におけるイベントがＰＲＥＳＳイベント、ＲＥＬＥＡＳＥイベント、ＲＥＭＯＶＥイベント、及びＭＯＶＥイベントのうちの１つでない場合、決定ステップ３７１３に入る。この決定ステップ３７１３はＢＡＤＣＡＲＤイベント又はＬＯＷ＿ＢＡＴＴイベントに関する判定を行なう。ステップ３７１３において、イベントがＢＡＤＣＡＲＤイベント又はＬＯＷ＿ＢＡＴＴイベントである場合、方法３７００はステップ３７１５へと進み、いずれのイベントでもない場合、方法３７００はステップ３７１７へと進む。ステップ３７１５において、ランチャ３０３はユーザに対して発生したイベントに関するフィードバック（例えば、ＬＯＷ＿ＢＡＴＴイベントが判定される場合はディスプレイ１０１上に「バッテリ低下」メッセージを表示し、ＢＡＤＣＡＲＤイベントの判定の際には「不正なカード」を表示）を行ない、方法３７００はステップ３７０３へと戻る。決定ステップ３７１３におけるイベントがＢＡＤＣＡＲＤイベントでもＬＯＷ＿ＢＡＴＴイベントでもない場合、ステップ３７１７に入る。

【０２８８】

ステップ３７１７において、イベントがＡＰＰ＿ＲＥＧＩＳＴＥＲである場合、方法３７００はステップ３９００の「アプリケーション登録」へと進む。ＡＰＰ＿ＲＥＧＩＳＴＥＲでない場合、方法３７００はステップ３７２５へと進む。ステップ３９００において、図３８を参照してここで説明するように（すなわち、第８．３．４節を参照して先に説明したように、アプリケーションが他のコンポーネント３０１、３０２、及び３０６にメッセージの受信が可能な状態になったことを通知する）アプリケーションが登録され、方法３７００はステップ３７０３へと戻る。ステップ３９００に従ってアプリケーションを登録する方法については、図３８のフローチャートを参照して詳細に後述する。ステップ３７２５においてイベントは破棄され、方法３７００はステップ３７０３へと戻る。

【０２８９】

図３７は、アプリケーションを変更する方法３８００を示すフローチャートであり、この方法はランチャ３０３により実行される。方法３８００は、コンピュータで実現する場合にはＣＰＵ２０５により実行することができる。また、セットトップボックスで実現する場合にはＣＰＵ４３０５により実行することができ、リモートサーバのＣＰＵによっても実行することができる。方法３８００はステップ３８１７で開始される。ステップ３８１７において、ランチャ３０３によりＲＥＭＯＶＥメッセージが受信されると、プロセスは直接ステップ３８１３へと進む。ＲＥＭＯＶＥメッセージが受信されない場合、方法３８００は決定ステップ３８０１へと続く。決定ステップ３８０１において、イベントにより表されるサービスが登録されているアプリケーションと関連付けられている場合、方法３８００は直接ステップ３８１９へと進む。アプリケーションと関連付けられていない場合、方法３８００はステップ３８０３へと続く。ステップ３８０３において、新規のアプリケーションの位置及び／又は名前と新規のアプリケーションと関連付けられる初期データとを判定するために、サービス識別子参照が実行される。例えば、初期データはブラウザ４０３へとロードされるＵＲＬであっても、あるいは、メディアプレーヤアプリケーションへとロードされるメディアファイルであっても良い。次のステップ３８０５において、

アプリケーションが既に実行中の場合、方法３８００はステップ３８１９へと進む。アプリケーションが実行中でない場合、方法３８００はステップ３８０９へと進む。ステップ３８０９において、新規のアプリケーションがアプリケーション３０４から得られる。次のステップ３８１１において、新規のアプリケーションがフロントアプリケーションとして開始される。ステップ３８１２において、イベントマネージャ３０１にはこの新規のフロントアプリケーションのコンポーネント識別子（Ｘｉｄ）が通知される。

【０２９０】

イベントにより表されるサービスが登録されているアプリケーションと関連付けられている場合（ステップ３８０１から）、あるいは、アプリケーションが既に実行中の場合、決定ステップ３８１９に入る。ステップ３８１９において、ＩＮＳＥＲＴメッセージがランチャ３０３により受信されたと判定される場合、方法３８００は終了する。判定されない場合、方法３８００はステップ３８０７へと進む。ステップ３８０７において、新規のアプリケーションがすぐに変更状態になることを示すＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントが新規のアプリケーションに送信される。新規のアプリケーションにＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントが送信された後、あるいは、決定ステップ３８１７でＲＥＭＯＶＥイベントが検知された結果、制御が決定ステップ３８１３に渡される。ステップ３８１３において、現在のフロントアプリケーションが存在するか否かが判定される。既にフロントアプリケーションが存在しない場合、方法３８００は終了する。フロントアプリケーションが存在する場合、前のフロントアプリケーションにＬＯＳＩＮＧ＿ＦＯＣＵＳイベントが送信され、前のフロントアプリケーションは方法３８００が終了する前に当座のタスクを終了することができる。

【０２９１】

図３８は、新規のアプリケーションを登録する方法又はプロセス３９００を示すフローチャートであり、この方法はランチャ３０３により実行される。方法３９００は、コンピュータで実現する場合にはＣＰＵ２０５により実行することができる。また、セットトップボックスで実現する場合にはＣＰＵ４３０５により実行することができ、リモートサーバのＣＰＵによっても実行することができる。プロセス３９００はステップ３９０１で開始される。ステップ３９０１において、アプリケーションを含み、図３６のステップ３９００を参照して言及される新規のサービスグループリストが生成される。次のステップ３９０３において、このアプリケーションにＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントが送信される。ステップ３９０５において、いずれのアプリケーションも新規のサービスグループの一部でなく、永続的でもない場合、方法３９００はステップ３９０７へと進む。サービスグループの一部であるか、あるいは、永続的である場合、方法３９００は終了する。ステップ３９０７において、サービスグループの一部でないアプリケーションにはＥＸＩＴ＿ＮＯＷイベントが送信され、方法３９００は次のステップ３９０８へと進む。ステップ３９０８において、イベントマネージャ３０１は新規のサービスグループの一部でないアプリケーションが終了したことを通知される。その後、方法３９００は終了する。

【０２９２】

図３９は、ランチャ３０３からイベントを受信するときにアプリケーションにより実行されるプロセスステップ４０００を示すフローチャートである。方法４０００は、コンピュータで実現する場合にはＣＰＵ２０５により実行することができる。また、セットトップボックスで実現する場合にはＣＰＵ４３０５により実行することができ、リモートサーバのＣＰＵによっても実行することができる。方法４０００はステップ４００１で開始される。ステップ４００１において、ランチャ３０３はイベントマネージャ３０１に接続し、方法４０００はステップ４００２へと進む。ステップ４００２において、ランチャ３０３にＡＰＰ＿ＲＥＧＩＳＴＥＲメッセージを送信することによってアプリケーションが登録される。図３９に示すフローチャートを次のステップ４００３までたどると、アプリケーションはイベントを待ち、イベントが受信されるとプロセスはステップ４００５へと進む。ステップ４００５において、イベントがＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントである場合、方法４０００はステップ４００７へと進む。ＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントで

ない場合、方法４０００はステップ４００９へと進む。ステップ４００７において、オプションとして区別用識別子を使用すると共にオプションとしてＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントのデータフィールドを使用することによって、必要に応じてアプリケーションが初期化される。初期化に使用されるこのデータフィールドは、ロード対象のＵＲＬ、ロード対象のファイル名などを含んでも良い。制御はステップ４００３におけるイベント待ちに戻る。

【０２９３】

ステップ４００９において、イベントがＰＲＥＳＳイベント、ＲＥＬＥＡＳＥイベント、又はＭＯＶＥイベントである場合、方法４０００はステップ４０１１へと進む。いずれのイベントでもない場合、方法４０００はステップ４０１３へと進む。ステップ４０１１において、イベントに応じてアプリケーション特有の動作が実行される。アプリケーション特有の動作は、イベントからのデータ（すなわち、カード１０上の印と関連付けられるデータ（例えば、ＵＲＬ、キャラクタ又はビデオの名前））、Ｘ／Ｙ位置又は区別用識別子、あるいは、これらの任意の組み合わせを使用して実行される。

【０２９４】

アプリケーション特有の動作は、通常、カード１０上の印と関連付けられる。例えば、印は特定のＵＲＬと関連付けることが可能であり、印が押下されるときにＵＲＬがアクセスされても良い。このため、例えば、コンピュータ１００又はＳＴＢ６０１はＵＲＬにより指定されたウェブページから所望のプログラムをダウンロードすることができ、カードユーザはシステム６００からサービス（すなわち、プログラムダウンロード）を受け取ることができる。更に、印は特定のメモリアドレスと関連付けることが可能であり、印が押下されるときにそのメモリアドレスにデータを格納するのにアドレスを使用することができる。従って、例えば、コンピュータ１００又はＳＴＢ６０１はメモリアドレスにより指定されたメモリ又はネットワーク上のファイルサーバから所望の画像データをダウンロードすることができ、カード１０のユーザはシステム６００からサービス（例えば、画像データダウンロード）を受け取ることができる。ステップ４０１１の後、図３９に示すように、方法４０００はステップ４００３へと戻る。

【０２９５】

上述の図３９のフローチャートによると、対応する決定ステップ４００５又は４００９において、イベントがＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベント、ＰＲＥＳＳイベント、ＲＥＬＥＡＳＥイベント、及びＭＯＶＥイベントのうちのいずれかであると判定されない場合、プロセスステップ４０００はフィルタを通過してステップ４１０３へと至る。ステップ４０１３において、イベントがＬＯＳＩＮＧ＿ＦＯＣＵＳイベントである場合、ステップ４０１３において、方法４０００はステップ４０１５へと進む。ＬＯＳＩＮＧ＿ＦＯＣＵＳイベントでない場合、方法４０００は決定ステップ４０１７へと進む。ステップ４０１５において、アプリケーションは非アクティブ状態へと戻り、方法４０００はステップ４００３へと戻る。ステップ４０１７において、イベントがＥＸＩＴ＿ＮＯＷイベントである場合、方法４０００は終了する。ＥＸＩＴ＿ＮＯＷイベントでない場合、方法４０００はステップ４０１９へと進む。ステップ４０１９において、イベントは無視され、方法４０００はステップ４００３へと戻る。

【０２９６】

図４０は、ランチャ３０３からイベントを受信するときにブラウザコントローラ４０２アプリケーションにより実行される方法４１００を示すフローチャートである。方法４１００は、コンピュータで実現する場合にはＣＰＵ２０５により実行することができる。また、セットトップボックスで実現する場合にはＣＰＵ４３０５により実行することができ、リモートサーバのＣＰＵによっても実行することができる。方法４１００はステップ４１０１で開始される。ステップ４１０１において、ブラウザアプリケーションはランチャ３０３にＡＰＰ＿ＲＥＧＩＳＴＥＲメッセージを送信する。次のステップ４１０３において、ブラウザアプリケーションはイベントを待ち、イベントが受信されると方法４１００はステップ４１０５へと進む。ステップ４１０５において、イベントがＧＡＩＮＩＮＧ＿Ｆ

ＯＣＵＳイベントである場合、方法４１００はステップ４１０７へと進む。ＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントでない場合、方法４１００はステップ４１０９へと進む。ステップ４１０７において、アプリケーションは必要に応じて初期化される。例えば、アプリケーションはＧＡＩＮＩＮＧ＿ＦＯＣＵＳメッセージのデータフィールドを読み、データフィールドがＵＲＬを表す場合、そのＵＲＬをロードする。初期化は、初期ＵＲＬをブラウザアプリケーション４０３へとロードし、ＵＲＬのベースを格納することによってブラウザコントローラ４０２上で実行される。方法４１００は次のステップ４１２１へと続く。ステップ４１２１において、区別用識別子がイベントから判定される。次のステップ４１２３において、現在のトップレベル文書中で引数としての区別用識別子１１１０と共にＪａｖａＳｃｒｉｐｔコールバック関数（Ｎｏｔｉｆｙ＿Ｃａｒｄ＿ＩＤとして知られるのが好ましい）が呼び出され、方法４１００はステップ４１０３へと戻る。

【０２９７】

ステップ４１０９において、イベントがＰＲＥＳＳイベント、ＲＥＬＥＡＳＥイベント、又はＭＯＶＥイベントである場合、方法４１００はステップ４２００へと進む。いずれのイベントでもない場合、方法４１００はステップ４１１３へと進む。ステップ４２００において、イベントに応じてブラウザアプリケーション特有の動作が実行される。ブラウザアプリケーション特有の動作については、図４１のフローチャートを参照して詳細に後述する。ステップ４２００の後、方法４１００はステップ４１０３へと戻る。

【０２９８】

ステップ４１１３において、イベントがＬＯＳＩＮＧ＿ＦＯＣＵＳイベントである場合、方法４１００はステップ４１１５へと進む。ＬＯＳＩＮＧ＿ＦＯＣＵＳイベントでない場合、方法４１００はステップ４１１７へと進む。ステップ４１１５において、ブラウザアプリケーションは非アクティブ状態へと戻り、方法４１００はステップ４１０３へと戻る。

【０２９９】

ステップ４１１７において、イベントがＥＸＩＴ＿ＮＯＷイベントである場合、方法４１００は終了する。ＥＸＩＴ＿ＮＯＷイベントでない場合、方法４１００はステップ４１１９へと進む。ステップ４１１９において、イベントは無視され、方法４１００はステップ４１０３へと戻る。

【０３００】

図４１は、ソフトウェアアーキテクチャ２００を組み込むシステム６００上で実行されるブラウザアプリケーション方法４２００を示すフローチャートである。方法４２００は、コンピュータで実現する場合にはＣＰＵ２０５により実行することができる。また、セットトップボックスで実現する場合にはＣＰＵ４３０５により実行することができ、リモートサーバのＣＰＵによっても実行することができる。方法４２００はステップ４２０１で開始される。ステップ４２０１において、イベントがＰＲＥＳＳイベントである場合、方法４２００はステップ４２２５へと進む。ＰＲＥＳＳイベントでない場合、方法４２００はステップ４２０３へと進む。ステップ４２０３において、イベントは無視され、方法４２００は終了する。ステップ４２２５において、区別用識別子がイベントから判定される。次のステップ４２２７において、現在の区別用識別子について現在のページに通知されている場合、方法４２００はステップ４２０５へと進む。通知されていない場合、方法４２００はステップ４２２９へと進む。ステップ４２２９において、現在のトップレベル文書中で引数としての区別用識別子と共にＮｏｔｉｆｙ＿Ｃａｒｄ＿ＩＤとして知られるＪａｖａＳｃｒｉｐｔコールバック関数が呼び出され、方法４２００はステップ４２０５へと進む。

【０３０１】

ステップ４２０５において、データがイベントから得られる。次のステップ４２０７において、データが１文字である場合、方法４２００はステップ４２０９へと進む。１文字でない場合、方法４２００はステップ４２１１へと進む。ステップ４２０９において、文字がブラウザアプリケーション４０３に送信され、方法４２００は終了する。これは、ユー

ザがキーボードのキー又は従来のリモコンのボタンを押下するのと同じ効果を提供するのに使用されても良い。現在のページは、Ｈｙｐｅｒ Ｔｅｘｔ Ｍａｒｋ－ｕｐ Ｌａｎｇｕａｇｅ（ＨＴＭＬ）により提供されるような既存の方法を使用して、任意のキー押下を受信する際に実行される動作を提供しても良い。

【０３０２】

ステップ４２１１において、データが「ｊｓ：」で開始される場合、方法４２００はステップ４２１３へと進む。「ｊｓ：」で開始されない場合、方法４２００はステップ４２１５へと進む。ステップ４２１３において、現在のトップレベル文書中のＪａｖａＳｃｒｉｐｔ関数が呼び出され、方法４２００は終了する。指定のデータはオプションとしてＪａｖａＳｃｒｉｐｔ関数に対する引数を含んでも良い。例えば、データ「ｊｓ：ｈｅｌｌｏ」はブラウザコントローラがＪａｖａＳｃｒｉｐｔ関数「ｈｅｌｌｏ」を呼び出すことを示し、データ「ｊｓ：ｈｅｌｌｏ（ｗｏｒｌｄ）」はブラウザコントローラが引数「ｗｏｒｌｄ」と共にＪａｖａＳｃｒｉｐｔ関数「ｈｅｌｌｏ」を呼び出すことを示すことになる。

【０３０３】

ステップ４２１５において、データが「ｃｍｄ：」で開始される場合、方法４２００はステップ４２１７へと進む。「ｃｍｄ：」で開始されない場合、方法４２００はステップ４２１９へと進む。ステップ４２１７において、指定のブラウザ関数が呼び出され、方法４２００は終了する。例えば、データ「ｐｒｉｎｔ」により、ブラウザコントローラはデータ「ｂａｃｋ」がブラウザコントローラにより前に表示されたページに戻るようにブラウザに指示するように指示するであろう。

【０３０４】

ステップ４２１９において、データが絶対ＵＲＬである場合、方法４２００はステップ４２２１へと進む。絶対ＵＲＬでない場合、方法４２００はステップ４２２３へと進む。ステップ４２２１において、データがＵＲＬとしてブラウザアプリケーション４０３へとロードされ、方法４２００は終了する。

【０３０５】

ステップ４２２３において、ベースＵＲＬが付加された後、データはＵＲＬとしてブラウザアプリケーション４０３へとロードされ、方法４２００は終了する。

【０３０６】

図４０を参照して先に説明したブラウザコントローラアプリケーションの変形例がソフトウェアプログラムを制御するプログラムコントローラである。ソフトウェアプログラムは、１つ以上のキー押下イベント（例えば、キーボードのキー押下イベント又はゲームコントローラでのそれに類するイベント）と共に通常制御される任意のプログラムを含むことができる。プログラムコントローラは、対話式ゲームなどの既存のソフトウェアプログラムのカードベースの制御を提供するのに使用することができる。プログラムコントローラプロセスは、以下に示すような例外はあるが、図４０を参照して説明したのとほぼ同様に動作する。ステップ４１０５におけるイベントがＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントである場合、プログラムコントローラプロセスはＧＡＩＮＩＮＧ＿ＦＯＣＵＳメッセージから制御対象のソフトウェアプログラムに対するリソースロケータを取得するステップへと進む。プロセスは、リソースロケータにより指定されるソフトウェアプログラムを取得・開始するステップへと進む。プログラムコントローラプロセスはステップ４１０３へと進む。更に、ステップ４１０９において、ＰＲＥＳＳイベント、ＲＥＬＥＡＳＥイベント、又はＭＯＶＥイベントを検査する代わりに、方法４１００におけるこの変形例は実質的にＰＲＥＳＳイベントについてチェックするであろう。イベントがＰＲＥＳＳイベントである場合、プロセスはイベントからデータを取得し、データから最初の文字をとり、その文字のキー押下を実現し、結果的に、ユーザがキーボード上でその文字をタイプしたのと同じ効果が得られるステップへと進む。

【０３０７】

１０．１　ランチャに対する特殊なルーティング規則

ランチャ３０３は特殊な一連のルーティング規則を有し、以下のイベントを常時受信する：
・ＥＭ＿ＲＥＭＯＴＥ＿ＩＮＳＥＲＴ
・ＥＭ＿ＲＥＭＯＴＥ＿ＲＥＭＯＶＥ
・ＥＭ＿ＲＥＭＯＴＥ＿ＢＡＤＣＡＲＤ
また、ランチャは、サービス識別子が現在のフロントアプリケーションと一致しない場合、あるいは、区別用識別子がＮＯ＿ＣＡＲＤの現在の識別子（すなわち、全て０）を表す場合、ＥＭ＿ＲＥＭＯＴＥ＿ＰＲＥＳＳメッセージ、ＥＭ＿ＲＥＭＯＴＥ＿ ＲＥＬＥＡＳＥメッセージ、及びＥＭ＿ＲＥＭＯＴＥ＿ＭＯＶＥメッセージを受信する。メッセージが一致するか否かを判定するために、サービス固有識別子は無視される。

【０３０８】
ランチャ３０３は、ＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントをそれ自体に送信することによって明確にそれ自体をフロントアプリケーションにするように構成することができる。この例では、全てのメッセージはメッセージのサービス識別子に関わらずランチャ３０３に送信されることになる。ランチャ３０３は、プロトコルによってこれらのメッセージのいずれかに応答する必要はない。

【０３０９】
１０．２　サンプル実現例
本節はランチャ構成の幾つかの例の概要を説明する。

【０３１０】
１０．２．１　総称ランチャ
総称ランチャは、ブロードバンドインターネット接続能力を有するオープンなセットトップボックス又はコンピュータ環境で使用することができる。この構成によると、ランチャ３０３はローカルマシン又は指定のリモートマシンへとダウンロードし、実行させることができるアプリケーションが存在することを前提とする。また、総称ランチャは、ブラウザコントローラ４０２を介してブラウザ４０３を使用するアプリケーションを使用できるように構成することができる。

【０３１１】
総称ランチャは、永続的なアプリケーションをサポートすると共に、アプリケーションをダウンロードするように構成することができる。システム６００を稼働させるコンピュータ１００は、相当に高速なインターネット接続が利用可能であるのが好ましい。この例では、アプリケーション３０４の一部は、上述のように、ブラウザコントローラ４０２と呼ばれる永続的なアプリケーションにより処理されるＪａｖａＳｃｒｉｐｔを有するウェブページであっても良い。更に、アプリケーション３０４の一部は協働するように設計することができる。また、総称ランチャは、リモートリーダ１により使用される通信リンクの信頼性が低く（すなわち、ＩＲリンク）、メッセージが損失する可能性があることを前提とするのが好ましい。

【０３１２】
１０．２．２　総称ランチャに対する規則
以下の規則は、ランチャ３０３がシステム６００を定義するのに使用されるのが好ましい規則である。

【０３１３】
・ＮＯ＿ＣＡＲＤの現在の識別子（すなわち、全て０）を有するＥＭ＿ＲＥＭＯＴＥ＿ＰＲＥＳＳイベント及びＥＭ＿ＲＥＭＯＴＥ＿ＲＥＬＥＡＳＥイベントは、ユーザがフロントアプリケーションから出ることを希望する合図として使用される。システム６００はその構成により、ディスプレイ１０１上に「カードを挿入してください」メッセージを生成するか、あるいは、前のアプリケーションへと戻ることになるであろう。

【０３１４】
・ＥＭ＿ＲＥＭＯＴＥ＿ＢＡＤＣＡＲＤイベントにより、ランチャ３０３はカードが不良であることを示すフィードバックをユーザに提供する。

【０３１５】
・リモートリーダ１からイベントマネージャ３０１までの通信方法の信頼性が低いことが前提とされているため、ＥＭ＿ＲＥＭＯＴＥ＿ＩＮＳＥＲＴイベント及びＥＭ＿ＲＥＭＯＴＥ＿ＲＥＭＯＶＥイベントは、セッションの境界を提供することが期待されていない。
【０３１６】
・ＥＭ＿ＲＥＭＯＴＥ＿ＰＲＥＳＳメッセージ、ＥＭ＿ＲＥＭＯＴＥ＿ＲＥＬＥＡＳＥメッセージ、又はＥＭ＿ＲＥＭＯＴＥ＿ＭＯＶＥメッセージを受信する場合、ランチャ３０３はサービスマッピングを行ない、サービス識別子がダウンロード可能なアプリケーションへと分解される場合、対応するアプリケーションがダウンロードされ、実行される。マッピングは、カードからのサービス情報を伴ってディレクトリサーバ３０５に問合せを行なうことによって実行される。ディレクトリサーバ３０５から戻される値は、アプリケーション位置及び関連するサービスデータである。アプリケーション位置は、アプリケーションの位置又はローカルアプリケーションとしてランチャが認識する値を指定する。サービスデータは、ＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳメッセージに含まれてアプリケーションに送信される初期化データである。アプリケーション位置が空の場合、ランチャ３０３はＵＲＬとなるであろうサービスデータに基づいてどのアプリケーションを使用すべきかを決定するように構成される。
【０３１７】
・新規のアプリケーションがＥＭ＿ＡＰＰ＿ＲＥＧＩＳＴＥＲメッセージと共に登録されるとき、指定のサービスグループが現在実行中の一連のアプリケーションと比較され、重複がない場合、現在実行中の他の全てのアプリケーションは終了するように通知される。新規のアプリケーションは現在のフロントアプリケーションになり（ＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳイベントを使用）、前のフロントアプリケーションにはＥＭ＿ＬＯＳＩＮＧ＿ＦＯＣＵＳイベントが送信される。この状況が発生し、サービス識別子がウェブページへと分解される場合、ＥＭ＿ＧＡＩＮＩＮＧ＿ＦＯＣＵＳメッセージを使用することによって、データフィールドにウェブページのアドレス（位置）が入った状態でフォーカスがブラウザコントローラ４０２へと変更される。データフィールドは、ランチャ３０３にサービス識別子がウェブページへと分解されたことを通知する問合せに含まれて戻される。この状況で、ＥＭ＿ＬＯＳＩＮＧ＿ＦＯＣＵＳイベントも現在のフロントアプリケーションに送信される。他の全てのアプリケーションは終了するように通知される。
【０３１８】
１０．３　単独使用のシステムの例
アーキテクチャ２００は単一の専用アプリケーションと共に使用するように構成することができる。この例では、ユーザインタフェースの一部又は全体を印刷可能な物理的なトークン（例えば、バンクカード）を有するのが有利な場合にランチャ３０３を使用することができる。後述する例は自動現金預払機の形態をとるが、特殊なアプリケーションとして説明されているに過ぎず、自動現金預払機に限定されるものとして解釈すべきではない。このようなシステムは、１枚又は少なくとも非常に限定された枚数のカードを使用できるように構成することができる。このシステムでは、挿入されるカードに関わらず、他のアプリケーション３０４は開始されない。ランチャ３０３はシステムコントローラの役割に加え、単一のアプリケーション３０４の役割を担う。イベントマネージャ３０１には変更が加えられない。
【０３１９】
単独使用のシステムは、例えば、自動現金預払機に使用することができる。銀行は、表面に一般に用いられるオプションをつけた個人用バンクカードを作製することができる。このカードは、自動現金預払機に対する唯一又は補助のインタフェースとして使用される。この例では、自動現金預払機はイベントマネージャ３０１及びアーキテクチャ２００の他のコアプロセスコンポーネントを含むのが好ましい。この例では、リモートリーダ１とイベントマネージャ３０１との間の通信リンクは信頼性の高いものでなければならない。
【０３２０】

１０．３．１　規則
以下の規則は、ランチャ３０３が単独使用のシステムの銀行の現金自動預払機の一例を定義するのに使用することができる：
・参加銀行と関連付けられるカードによるものでないイベントは、ランチャが端末に互換性のないカードの画面を表示する原因となる。
・ＥＭ＿ＲＥＭＯＴＥ＿ＢＡＤＣＡＲＤイベントは無視される。
・ＥＭ＿ＲＥＭＯＴＥ＿ＩＮＳＥＲＴイベントはトランザクションを開始するのに使用される。
・ＥＭ＿ＲＥＭＯＴＥ＿ＲＥＭＯＶＥイベントはトランザクションを終了するのに使用される。
・ＥＭ＿ＲＥＭＯＴＥ＿ＰＲＥＳＳイベント、ＥＭ＿ＲＥＭＯＴＥ＿ＲＥＬＥＡＳＥイベント、及びＥＭ＿ＲＥＭＯＴＥ＿ＭＯＶＥイベントはユーザインタラクションとして扱われる。これらのイベントは実行中のアプリケーションのようにランチャにより直接処理されるのが好ましい。
・外部のディレクトリサーバへのサービスマッピングが行なわれることはない。カードが特定の自動現金預払機（ＡＴＭ）が認識するものでない場合、カードは拒否される。
【０３２１】
これらの規則は、ＡＴＭの形態で専用のアプリケーションを提供するように単独使用のシステムを構成することができる方法の例である。
【０３２２】
１０．４　ディレクトリサービスの動作
図５８は、ディレクトリサービス３１１により実行されるプロセス５８００の概要を示すフローチャートである。プロセス５８００はコンピュータ１００のＣＰＵ２０５により実行される。このＣＰＵ２０５がディレクトリサービス３１１の役割を果たす。図５８に示すソフトウェアプログラムは、システム６００Ａのメモリ２０６又はＣＤ－ＲＯＭ２１２あるいはシステム６００Ｂのメモリ４３０６などのメモリ媒体に格納される。プロセス５８００は第１のステップ５８０１で開始される。ステップ５８０１において、ディレクトリサービス３１１が開始される。次のステップ５８０２において、ＣＰＵはランチャ３０３からの入力イベントを待つ。イベントはイベントマネージャ３０１を介して読取り装置１からランチャ３０３に送信される。次のステップ５８０３において、ＣＰＵはランチャ３０３から区別用識別子を含む要求を受信する。この区別用識別子はディレクトリサービス３１１によりマップされる。ランチャ３０３とディレクトリサービス３１１との間の接続は図８に示される。
【０３２３】
次のステップ５８０４において、ＣＰＵはディレクトリマッピングテーブルを検索し、テーブルが区別用識別子に対応するエントリを有するか否かをチェックする。ディレクトリマッピングテーブルは、通常、サービス識別子と対応するアプリケーション位置（ＵＲＬなど）及びサービスデータとの間の関係を含み、更に、区別用識別子と対応するアプリケーション位置及びサービスデータとの間の関係を含む。通常、サービス識別子を含む関係はカード１０に対して使用され、ディレクトリサービス３１１は、そのサービスに対して使用することができる全てのカード１０に対するサービスレベル情報（例えば、カード１０に対するサービスを提供するように実行されるアプリケーション３０４の位置）を維持することを目的とする。また、区別用識別子を含む関係はカード１０に対して使用され、ディレクトリサービス３１１は、同一のサービス固有識別子を有する実際のカード１０又はカードのグループ１０に特有の情報（例えば、カード１０に対するサービスを提供するべく再生されるメディアファイルの位置）を維持することを目的とする。ディレクトリマッピングテーブルは、通常、ハードディスク２１０又はメモリ２０６に格納される。ステップ５８０４において、ディレクトリマッピングテーブル中に区別用識別子に対するエントリがある場合、次のステップ５８０５において、ＣＰＵはこのエントリからアプリケーション位置及びサービスデータを取得し、ステップ５８０６へと移る。ステップ５８０４

において、テーブル中に区別用識別子に対するエントリが存在しない場合、ステップ５８０８において、ＣＰＵはこの値の関連部分（通常は、図１１に示すような最初の５バイト）をとることによって区別用識別子からサービス識別子を抽出する。次のステップ５８０９において、ＣＰＵはサービス識別子に対応するエントリを求めてディレクトリマッピングテーブルを検索する。エントリが見つかった場合、次のステップ５８１０において、ＣＰＵはこのエントリからアプリケーション位置及びサービスデータを取得し、ステップ５８０６へと移る。ステップ５８１１において、エントリが見つからない場合、特定の区別用識別子に対する要求が行なわれたことを示すエントリがログファイルに配置される。ステップ５８１２において、供給された区別用識別子のサービス識別子部分がディレクトリサービス３１１にとって未知のものであることを示すエラーがランチャ３０３に戻される。フローはステップ５８０２へと続く。

【０３２４】

ステップ５８０６において、すなわち、区別用識別子又はサービス識別子の検索が成功した場合、区別用識別子と対応するアプリケーション位置及びサービスデータとがログファイルに書き込まれ、ＣＰＵは要求を行なったランチャ３０３にアプリケーション位置及びサービスデータを戻す。フローはステップ５８０２へと続き、別のイベントを待つ。

【０３２５】

１１　総則

通常、アプリケーション３０４はハードディスクドライブ２１０に常駐し、ＣＰＵ２０５による実行時に読み出され制御される。プログラム及びネットワーク２２０から取り込んだデータの中間記憶装置は、半導体メモリ２０６を使用し、場合によってはハードディスクドライブ２１０と協働させるように使用して達成されても良い。アプリケーション３０４は、ＣＤ－ＲＯＭ又はフロッピーディスク上に符号化された形でユーザに供給され、対応するドライブ２１２又は２１１を介して読み取られる場合もあれば、ネットワーク２２０からモデム装置２１６を介して読み取られる場合もある。その他のコンピュータ可読な媒体からコンピュータシステム１００へとソフトウェアアプリケーションをロードするその他のメカニズムには、磁気テープ、ＲＯＭ又は集積回路、光磁気ディスク、コンピュータモジュール１０２と別の装置との間の無線又は赤外線の伝送チャネル、スマートカード、コンピュータのＰＣＭＣＩＡカードなどのコンピュータ可読なカード、及び電子メール送受信並びにウェブサイトなどに記録された情報を含むインターネット並びにイントラネットが含まれる。以上の媒体は、関連するコンピュータ可読な媒体のほんの一例である。上述の例の組み合わせを含め、その他のコンピュータ可読な媒体も使用可能である。

【０３２６】

また、上述のプロセスコンポーネント３０１から３０６は、上述の関数及び下位関数を実行する１つ以上の集積回路などの専用のハードウェアで実現することもできる。このような専用のハードウェアは、グラフィックＣＰＵ、デジタル信号ＣＰＵ、又は、１つ以上のマイクロＣＰＵ及び関連メモリを含んでも良い。専用ハードウェアの例は、図６（ｂ）を参照して説明したテレビ用のセットトップボックス６０１である。

【０３２７】

１２　その他の変形例

１２．１　セッション識別子

上述のように、区別用識別子は、リーダ１からコンピュータ１００又はセットトップボックス６０１に送信される全てのＩＮＳＥＲＴメッセージ、ＲＥＭＯＶＥメッセージ、ＰＲＥＳＳメッセージ、ＲＥＬＥＡＳＥメッセージ、及びＭＯＶＥメッセージに含まれる。また、区別用識別子はＩＮＳＥＲＴメッセージのみと共に送信することもできる。この例では、新規のカード１０の挿入時に、リーダ１がセッション識別子（不図示）を生成する。セッション識別子はカード挿入の現在のセッションを識別する。セッション識別子は、例えば、擬似乱数（２バイトのデータを用いて表すことができる）とすることも、カードが挿入されるたびに増分される（所定の値に達すると０にリセット）数値とすることもできる。リーダ１はコンピュータ１００又はセットトップボックス６０１にＩＮＳＥＲＴメッ

セージを送信する。このメッセージは先に説明した区別用識別子及び新規の挿入ごとに生成されるセッション識別子を含む。後続するＰＲＥＳＳメッセージ、ＲＥＬＥＡＳＥメッセージ、及びＭＯＶＥメッセージは、全て区別用識別子を含む必要がないが、セッション識別子と先に説明したユーザインタフェースオブジェクトデータ又は押下座標とを含むであろう。

【０３２８】

セッション識別子を使用するとき、システム６００は、リーダ１からＩＮＳＥＲＴメッセージを受信する際にイベントマネージャ３０１がセッション識別子を現在のセッション識別子として格納し、区別用識別子を現在の区別用識別子として格納することを除いては図６（ａ）及び図６（ｂ）を参照して説明したのと同様の処理を実行する。ＰＲＥＳＳメッセージ、ＲＥＬＥＡＳＥメッセージ、又はＭＯＶＥメッセージを受信するとき、イベントマネージャ３０１はセッション識別子が現在のセッション識別子と等しいことを確認する。等しい場合、イベントマネージャ３０１は、全てのメッセージにおいて使用される区別用識別子を現在の区別用識別子に設定する。これに対し、セッション識別子が現在のセッション識別子と等しくない場合、イベントマネージャ３０１は、ディスプレイマネージャ３０６及び表示装置１０１を介してメッセージが対応するＩＮＳＥＲＴメッセージを伴わずに受信されたことをユーザに通知する。この場合ユーザは、例えば、カード１０を抜き取って再挿入するように要求される。

【０３２９】

１２．２　ユーザ押下のその他の特徴

上述のように、情報の送信は、リーダ１のタッチパネル８上でのオブジェクトの押下、移動、及び押下解除（通常、指又はスタイラスを用いる）に関する。しかし、リーダ１は、タッチパネル８からのインタラクションに関する追加の情報をシステム６００により使用するためにコンピュータ１００又はセットトップボックス６０１に送信することができる。例えば、追加の情報は、押下の結果タッチパネル８に対して作用する圧力の時間及び量を表すことができる。この追加の情報は、リーダ１からシステム６００に送信されるＰＲＥＳＳメッセージに組み込むことも、システム６００内で送信されるＥＭ＿ＲＥＡＤＥＲ＿ＰＲＥＳＳメッセージに組み込むこともできる。この例では、情報はリーダ１に挿入されるカードに対応するアプリケーション３０４に渡される。アプリケーションは、追加の情報を利用して、例えば、特定の動作に対して付加的な効果をもたらすことができる。例えば、アプリケーションは音量の増加を示す印の押下と関連付けられるときに、圧力情報を使用して音量の増加量を判定することができる。すなわち、選択された印に対する押下が強いと音量の増加率が高くなり、逆に、選択された印に対する押下が弱いと増加率が低くなる。

【０３３０】

タッチパネル８とのインタラクションの時間（すなわち、持続時間）に関し、追加の情報の使用の別の例を以下で説明する。押下の持続時間が非常に短い場合、押下は「タップ」であると見なすことができる。これに対し、押下の持続時間が非常に長い場合、キー押下の永続的な「押し下げ」であると見なすことができる。この例では、追加の情報はインスタントソフトウェアアプリケーションと対話するモードに追加の次元を加えることができる。例えば、タッチパネル８上での「タップ」をソフトウェアアプリケーションに現在の（画面上の）カーソル位置に表示される項目を選択するように指示する命令とすることができる。

【０３３１】

１２．３　座標なし

ＰＲＥＳＳメッセージ及びＲＥＬＥＡＳＥメッセージは、ユーザのタッチパネル８とのインタラクションの座標データを含まないように構成することができる。この例では、座標データのみが、ＭＯＶＥメッセージと共にリーダ１からシステム６００に送信される。ＰＲＥＳＳメッセージ及びＲＥＬＥＡＳＥメッセージに座標データを含まないことの利点は、アプリケーション３０４が座標からユーザインタフェース要素データへとマッピングす

るのに座標情報を必要としない場合に、リーダ１によりシステム６００に送信されるメッセージのサイズを縮小できることである。

【０３３２】

１２．４　双方向プロトコル

単方向又は双方向通信は、リーダ１とコンピュータ１００又はセットトップボックス６０１との間の通信に使用することができる。図１０を参照したリーダ１の説明及び図８及び図９を参照したＩ／Ｏデーモンの説明は、リーダ１からコンピュータ１００又はセットトップボックス６０１への情報の送信及び逆方向の送信を含むものであった。コンピュータ１００又はセットトップボックス６０１からリーダ１への情報の送信は、カード１０に格納されたデータを変更するのに使用することができる。例えば、スマートカード１０のメモリチップ上に格納されたユーザインタフェースオブジェクトデータの変更に使用することができる。

【０３３３】

また、双方プロトコルはプロトコルにおけるハンドシェーキングを可能にするために使用することができる。例えば、リーダ１とセットトップボックス６０１又はコンピュータ１００との間の双方向プロトコルは、カードがリーダ１に挿入されるときにシステム６００がＩＮＳＥＲＴメッセージの受信を確認することができるように使用することができる。双方向プロトコルをサポートするシステム６００は、アプリケーションがカード１０上の格納されたデータの一部を変更するために要求を送信することができるように、イベントマネージャ３０１を介してＩ／Ｏデーモン３００に送信されるイベントマネージャプロトコルでの追加のメッセージを提供すべきである。Ｉ／Ｏデーモン３００は、リーダ１にメッセージを送信して要求される動作を引き起こすことができる。例えば、システム６００が双方向プロトコルを使用する場合、システム６００はアプリケーションがユーザの許可又はシステム定義の特権なしにカードを変更することができないことを保証するための機密保護メカニズムを提供することができる。このようなシステムの一例において、イベントマネージャ３０１は、アプリケーションが現在挿入されているカードを変更しても良いか否かを尋ねる表示メッセージをユーザに対して提示することができる。ユーザは、タッチパネル８の第１の領域を押下することによって提案に同意することができると共に、タッチパネル８の第２の領域を押下することによって提案への不同意を示すことができる。ユーザがカード１０の変更に同意する場合、イベントマネージャ３０１は、アプリケーション３０４からの要求がＩ／Ｏデーモン３００へと渡され、更に、リーダ１へと渡されるようにすることができる。これに対し、ユーザが変更に不同意の場合、イベントマネージャ３０１はメッセージを放棄するので情報はリーダ１に送信されない。

【０３３４】

１２．５　代替の読取り装置

上述のシステム６００Ａ及び６００Ｂにおいて、読取り装置１はカード１０に重なるように構成されるほぼ透明な感圧膜を有する。読取り装置１の費用を削減するために、読取り装置１は感圧膜の代わりに複数のユーザ操作可能なスイッチをレセプタクルの周囲に配置しても良い。レセプタクルには、データ、すなわち、区別用識別子とデータを各スイッチと関連付けるための関係情報とを読むためにスマートカード１０を挿入することが可能である。各スイッチのうちの操作可能なものはスマートカード上の印と視覚的と関連付けられるため、ユーザはカード上の少なくとも１つの印に対応するスイッチのうちの少なくとも１つを選択することができる。この場合、ＣＰＵ４５はカード１０からの関係情報及び区別用識別子に基づいてユーザにより押下されるスイッチに対応するデータを読み、それをイベントマネージャ３０１に送信する。

【０３３５】

１３．０　代替のソフトウェアアーキテクチャ

システム６００により表されるハードウェアアーキテクチャに対する更なる一般的なソフトウェアアーキテクチャ４９００が図４８に示される。このアーキテクチャ４９００は、これまで各節で説明したものの代替のソフトウェアアーキテクチャを表す。代替のアーキ

テクチャ４９００は、ユーザの自宅における非常に低レベルのハードウェア要件（すなわち、簡易セットトップボックス）から、例えば、セットトップボックス６０１の機能がパーソナルコンピューティングシステム上で実現される強力なホームシステムに至るまでスケール変更されるように構成される。更に、代替の構成４９００は、ハードウェアシステム６００内で実現されるのが好ましい。

【０３３６】

１３．１　構造

アーキテクチャ４９００は、６つの別個のプロセス及び１つのプロセスクラスへと分割される。別個のプロセスは、アーキテクチャ２００と同様にＩ／Ｏデーモンと呼ばれるスマートカードインタフェース４９０２、イベントマネージャ４９０４、ディスプレイマネージャ４９０６、マスタランチャ４９０８、（アプリケーション）ランチャ４９１０、及びディレクトリサービス４９１２を含む。プロセスクラスは１つ以上のスマートカードアプリケーション４９２０により形成される。アーキテクチャ４９００には、セットトップボックス６０１により通常形成されるスマートカードリモート接続ごとに、１つのカードデーモン４９０２、１つのイベントマネージャ４９０４、１つのディスプレイマネージャ４９０６、及び１つのランチャ４９１０と、各ランチャ４９１０を稼動させているコンピュータごとに１つのマスタランチャ４９０８と、全てのシステムに対して少なくとも１つのディレクトリサービス４９１２とが存在する。

【０３３７】

この形態において、図４８の破線により示すように、アーキテクチャ４９００は３つの別個の部分４９１４、４９１５、及び４９１６へと物理的に分離することができる。各部分は物理的に別々の計算装置上で実行させることができる。システムの各部分間の通信は、アーキテクチャ２００と同様にＴＣＰ／ＩＰストリームを使用して実行される。

【０３３８】

Ｉ／Ｏデーモン４９０２は、スマートカードリモートリーダ１から受信されるデータグラムをＴＣＰ／ＩＰストリームへと変換するプロセスである。Ｉ／Ｏデーモン４９０２は、リーダ１により使用されるデータ形式を理解するのではなく、スマートカードリモートデータ形式の変更とは関係なく動作することを意図し、複数のバージョンのリーダ１と協働する機能を提供する。

【０３３９】

Ｉ／Ｏデーモン４９０２は、ユーザがシステム６００を開始するときに開始されるが、これは、セットトップボックスシステム６００Ｂの場合、セットトップボックス６０１の電源が投入されるときである。コンピュータシステム６００Ａの場合、Ｉ／Ｏデーモン４９０２は、イベントマネージャ４９０４及びマスタランチャ４９０８が開始された後、ユーザがスマートカードシステムを開始するときに開始されても良い。

【０３４０】

イベントマネージャ４９０４は、全ての通信がイベントマネージャ４９０４を経由するという点でアーキテクチャ４９００の中心を形成する。イベントマネージャ４９０４は、スマートカードリモートリーダ１により生成され、Ｉ／Ｏデーモン４９０２により中継される全てのイベントを収集する役割を担う。これらのイベントは、種々のプロセス及び実行中の各アプリケーションへと再分散される。

【０３４１】

イベントマネージャ４９０４の更なる役割は、挙動の悪いアプリケーションを他の挙動の良いアプリケーションから切り離すことである。この点に関して、イベントマネージャ４９０４を経由するイベントは、イベントマネージャ４９０４がチェック可能な範囲で正しいことが保証される。イベントマネージャ４９０４は、イベントが有効なヘッダ及び正しいデータ長を有することをチェックすることが要求されるが、通常、データが正しい形式であることをチェックするようには構成されていない。

【０３４２】

異なるバージョン間でのプロトコルの変更にも、イベントマネージャ４９０４が対処する

。可能な場合には、動作中のアプリケーション４９２０が理解するデータ形式に適合するようにイベントは書き換えられる。書き換えが可能でない場合、イベントマネージャ４９０４は送信元のアプリケーション４９２０にエラーを報告する。異なるデータ形式バージョンが使用されている場合、イベントマネージャ４９０４は混乱の発生を最小限に抑えることを保証する。

【０３４３】

ディスプレイマネージャ４９０６は、どの動作中のアプリケーション４９２０が特定の出力装置１１６、典型的にはディスプレイ（例えば、１１６）に対して優先権を有するかに関して制御するように動作するアプリケーション４９２０と協働して動作する。どのビデオストリームがイベントマネージャ４９０４を介してディスプレイ１１６に送信されるかを選択するのがディスプレイマネージャ４９０６の役割である。この情報はアプリケーション４９２０の各ランチャ４９１０から得られる。一般的に、フロント（すなわち、フォアグラウンド）アプリケーションのみがビデオディスプレイストリームを生成する。更に、ディスプレイマネージャ４９０６は、整合性のない入力ストリームから一定のストリームを出力するように維持し、推定データを用いて出力ストリームの一部を補充するように動作しても良い。

【０３４４】

イベントマネージャ４９０４は、いつアプリケーション４９２０を開始／終了しなければならないかを決定する、あるいは、実際にアプリケーション４９２０を開始／終了する責任はない。後述するように、これらの動作はランチャ４９０８及び４９１０の責任である。更に、イベントマネージャ４９０４は、ユーザ画面又はその他の出力装置１１６にはその存在を示さない。スマートカードの最初の挿入の表示などのシステム関連のフィードバックはランチャ４９１０により実行される。

【０３４５】

代替のアーキテクチャ４９００を組み込む図６（ｂ）のシステム６００Ｂの場合、通常、システム６００Ｂに接続することを許された全てのセットトップボックス６０１に対してイベントマネージャ４９０４が実行されるであろう。アーキテクチャ４９００を組み込むシステム６００Ａの場合、マスタランチャ４９０８が開始された後にスマートカードシステム６００Ａが開始されるときにイベントマネージャ４９０４が開始されることになる。

【０３４６】

マスタランチャ４９０８の役割は、イベントマネージャ４９０４のいずれかの要求に応じてランチャ４９１０を開始することである。Ｉ／Ｏデーモン４９０２がイベントマネージャ４９０４に接続するとき、イベントマネージャ４９０４はマスタランチャ４９０８にイベントマネージャ４９０４に対する第１のプロセスを開始するように要求する。この第１のプロセスは、通常、任意のスマートカードアプリケーション４９２０に対するランチャ４９１０であろう。また、マスタランチャ４９０８は、イベントマネージャ４９０４の要求時にアプリケーション４９２０のランチャ４９１０を停止すると共に、イベントマネージャ４９０４に正しいランチャ４９１０が終了したことを通知する役割を担う。

【０３４７】

代替のアーキテクチャ４９００を組み込む図６（ｂ）のシステム６００Ｂの場合、スマートカードアプリケーション４９２０を実行する物理的に別々のサーバ１５０、１５２の各々に対して、常に、１つのマスタランチャ４９０８が実行されることになる。この１つのマスタランチャ４９０８は、そのサーバ上にランチャ４９１０を要求する全てのイベントマネージャ４９０４に対する要求に対処する。システム６００Ａの場合、マスタランチャ４９０８はスマートカードシステムのその他の部分より前又はそれと同時に動作を開始する。

【０３４８】

カードディレクトリサービス４９１２は、スマートカード１０内に格納されるベンダーアプリケーション値（サービス識別子の値）を後述するベンダーアプリケーション対（サービス識別子）と関連付けられるアプリケーション４９２０を指し示すＵｎｉｆｏｒｍ Ｒ

ｅｓｏｕｒｃｅ Ｌｏｃａｔｏｒ（ＵＲＬ）などのアプリケーション位置へと変換するために設けられる。ディレクトリサービス４９１２は、アプリケーション４９２０がランチャ４９１０に対して別々のシステム上で実行可能であるようにランチャ４９１０を変更することによって複数の部分へと分割することができる。ディレクトリサービス４９１２は、分散型参照システムを使用してこの機能を実行する。このシステムでは、現在問合せを所持するディレクトリサービスがその答えを知らない場合、問合せは別のディレクトリサーバへと渡される。このような分散型システムにより、各ディレクトリサーバはベンダ－アプリケーションＩＤ対からＵＲＬへの変換について限られた知識を有することになるが、依然として全てのＩＤをＵＲＬへと変換することができる。これにより、各ディレクトリにおけるデータベースの簡易化を含め、数多くの利点が提供される。頑強性が向上し、サーバは問合せが許可されている間、動作不能になる（すなわち、クラッシュ又はサービスから外される）ことを許される。

【０３４９】

図５２を参照すると、スマートカード１０を従来のスマートカードと区別する制御テンプレートカスタマイズ情報は、ベンダ識別子、カード識別子、及びアプリケーション識別子によるデータの組を含む。ベンダ識別子／アプリケーション識別子対は、アーキテクチャ２００に関して先に説明したサービス識別子に相当する。また、カード識別子は、アーキテクチャ２００に関して先に説明したサービス固有識別子に相当する。更に、各アイコン４８０４と関連付けられるのは、ユーザがアイコン４８０４上のタッチパネルを押下するときにイベントデータとして送信される対応データである。このイベントデータは、特定のアプリケーション４９２０に渡されるとそのアプリケーション内の特定の動作を実施する。また、例えば、アイコンの押下ではなくアイコンの押下解除を検知するため、更に、ユーザがタッチパネル８を横断するように指をスクライブして特定の機能を実行する場合に押下の移動を検知するため、ユーザ動作の検知が組み込まれても良い。このような動作のたびに、カード上に格納されるイベントデータを送信することができる。このデータはその都度カード上のメモリの異なる位置から読まれても良い。この代替のアーキテクチャ４９００においてベンダ－アプリケーション識別子対として実現されるサービス識別子により、スマートカードと関連付けられるアプリケーションのベンダは相互に区別することができる。スマートカードと関連付けられるアプリケーションのベンダを区別する必要がない場合のアーキテクチャ４９００の展開において、ベンダ識別子及びアプリケーション識別子は、単一の値：サービス識別子として扱うことができる。

【０３５０】

スマートカード１０のリーダ１への挿入により開始される第１のプロセスは、汎用システム（一般家庭など）ではランチャ４９１０になるであろう。特殊なシステムでは、特殊なアプリケーションが開始されても良い。例えば、バンキングテラーはバンキングアプリケーションを開始するであろう。別の例としては、アプリケーションの特定の部分集合のみを開始する限定的なランチャの使用が含まれる。ランチャ４９１０は、特定の１つのイベントマネージャ４９０４に対して他のアプリケーション４９２０を開始するスマートカードアプリケーションである。アプリケーション４９２０を開始／終了させ、実際にアプリケーションを開始／終了するのはランチャ４９１０の決定である。ランチャ４９１０は、イベントマネージャ４９０４にいつアプリケーションが開始／終了するかを通知し、アプリケーション４９２０にいつフォーカスを受け取る／失うか、あるいは、いつ終了する必要があるかを通知する。この点に関して、複数のアプリケーション４９２０が同時に動作している場合、現在画面に表示されているアプリケーションがフォーカスを有するアプリケーションである。別のアプリケーションが優先権を得ようとすると、ランチャ４９１０は現在のアプリケーションにフォーカスが失われることを通知するので、現在のアプリケーションは当座のタスクを完了することができる。また、ランチャ４９１０は新規のアプリケーションにフォーカスが得られ、まもなく新規のアプリケーションが変更状態に入ることを通知する。ランチャ４９１０はアプリケーションを強制的に終了できるように構成されなければならない。

【０３５１】
開始される第１のアプリケーション４９２０（すなわち、通常は、ランチャ４９１０）には特権が与えられ、リモートリーダ１により生成された「ＮＯ＿ＣＡＲＤ」イベント、「Ｂａｄ＿ＣＡＲＤ」イベント、及び「ＰＯＷＥＲ＿ＯＦＦ」イベントを受信する。また、第１のアプリケーション４９２０は、現在のフロントアプリケーションでない各アプリケーション４９２０用のイベントも受信し、これらのイベントを正確に解釈するように動作する。これは上述の特定のアプリケーションに関連するので、ランチャはあらゆる変化を正確に解釈する。ランチャ４９１０は、その他のアプリケーションを開始・停止する権利を含む特殊な権利を有するアプリケーション４９２０である。

【０３５２】
ランチャ４９１０は、イベントマネージャ４９０４によりその開始が要求されるときにのみ開始されるのが好ましい。また、ランチャ４９１０は、イベントマネージャ４９０４により終了するように指示することもできる。

【０３５３】
アプリケーションは、対応するスマートカード１０上での第１のユーザ選択に対する応答として、あるいは、アプリケーション４９２０の別の１つの要求に応じてランチャ４９１０により開始される。この点に関して、アーキテクチャ４９００は、プログラミング中に１つ以上のアプリケーションサービスグループのメンバとして編成される各アプリケーション４９２０を介して、従来の構成に対する実質的な改良を提供する。

【０３５４】
１３．２　アプリケーションサービスグループ
アプリケーションサービスグループは、単に同時に動作するのではなく協力して動作し、機能の特定の集合を提供する複数のスマートカードアプリケーション４９２０から構成される。サービスグループの一部を形成するアプリケーション４９２０は、同時に実行されることを許されると共に、データの交換を行なう際に利用されるであろう通信手段（すなわち、イベントマネージャ４９０４）を共有する。このようなアプリケーション４９２０は、各々が特定のユーザインタフェース又はユーザインタフェースの集合に対応する機能の集合を提供するプロセス又は下位プロセスである。このアプリケーション４９２０は、可視のディスプレイを有しても有しなくても良い。

【０３５５】
図４９に表される例を参照すると、サービスグループは、その一部を形成するアプリケーション４９２０が開始され、そのサービスグループをイベントマネージャ４９０４に登録するときに開始される。図４９に示すように、第１のアプリケーション４９２６は２枚のスマートカード４９２４及び４９２６をそれ自体と関連付け、第２のアプリケーション４９３４はスマートカード４９２８、４９３０、及び４９３２を用いて動作可能である。従って、カード４９２２をリーダ１へと挿入する際に、カードデーモン４９０２は、ランチャ４９１０を介してアプリケーション４９２６を開始するイベントマネージャ４９０４にその発生を通知する。アプリケーション４９２６の開始により、サービスグループ４９３６が使用可能になる。このグループはアプリケーション４９３４を含む。現在確立されているサービスグループに対応するアプリケーションは、関連するスマートカードを挿入することによって開始されても良い。例えば、カード４９２２の抜取り及びカード４９３２の挿入は、アプリケーション４９３４を起動するように作用し、サービスグループ４９３６をアクティブ状態に維持する。更に、同じサービスグループの一部を形成するアプリケーションの開始は、同じサービスグループの他のアプリケーションの終了を引き起こさない。その他のアプリケーションはバックグラウンドで実行を継続する。

【０３５６】
サービスグループの終了は、空のリモートリーダ１に接触する、あるいは、サービスグループ４９４２中のアプリケーション４９４０に対応するカード４９３８などの別のサービスグループに対応するスマートカードを挿入することによって引き起こされる。サービスグループの終了により、現在そのサービスグループの一部として実行されている全てのア

プリケーションも終了させられる。
【０３５７】
同じサービスグループ下で実行されるアプリケーションは、図４９に示すように、サービスグループ定義のプロトコル４９５０によってイベントマネージャ４９０４を介して相互に通信しても良い。プロトコル４９５０において、アプリケーション（例えば、４９２６及び４９３４）間で送信されるデータパケットの形式及び内容は、同じサービスグループ内に共存するアプリケーションの作成者により定義されるべきである。
【０３５８】
図５０に示されるのは、アーキテクチャ４９００内のサービスグループの別の特徴である。ここでは、サービスグループは他のいずれかのサービスグループの一部として実行される１つ以上のアプリケーションを含んでも良い。これらのアプリケーションは、複数のサービスグループにまたがって要求される可能性があるサービスを提供する。このようなアプリケーションの一例は、ユーザの住所及びクレジットカードの詳細（ユーザが一度これらの詳細を提供することに同意した場合）を提供することができる個人識別サービスである。この点に関して、このようなサービスは、オンラインショッピング及びオンラインバンキングを含む金融取引を必要とする数多くのその他のサービス又はトランザクションのコンポーネントを形成しても良い。
【０３５９】
アーキテクチャ４９００をサポートするためのアプリケーションの設計は、開発されるアプリケーションがアーキテクチャ４９００により提供される新規の特徴を必要とするか否かによって、アプリケーション設計に対する既存のアプローチと同じであっても、あるいは、異なっていても良い。何らかの変形を施しても、既存のアプリケーションはアーキテクチャ４９００の下で機能するであろう。このような変形の例としては、既存のアーキテクチャの下で稼働する各アプリケーションが、実行中のアプリケーションと同じ名前を有するサービスグループをもつ（すなわち、各アプリケーションはメンバを１つだけ有する独自のサービスグループを形成する）ことを想定できる場合、あるいは、一意的なグループ名を選択するその他の方法がある。この方法は、他のアプリケーションと協働しない既存のアプリケーションに対してグループ名をもたないことを含む。
【０３６０】
同じサービスグループ内の各アプリケーションは、同じ物理ハードウェア上で動作する必要はなく、ＯＳ定義の方法を使用することによって相互に直接通信可能でなくても良い。２つの通信方法は、イベントマネージャ４９０４において実現され、アプリケーション間通信の標準的な方法を提供するのが好ましい。これらの方法は：
（ｉ）メッセージが１つのアプリケーションにより別のアプリケーションに送信されるデータグラムベースのプロトコルと、
（ｉｉ）アプリケーション４９２０により、同じサービスグループ中の任意のアプリケーション４９２０がメッセージを読むことができる共通の領域へとメッセージが掲示される掲示板に基づくプロトコルである。
【０３６１】
イベントマネージャ４９０４は、アプリケーション４９２０間で渡されるデータに対して構造を強制しない。全てのメッセージは既知の長さのただのデータブロックである。データに対して強制される他の任意の構造は、特定のサービスグループのアプリケーションにより理解されれば良い。データブロックには、ポスティングアプリケーションによってイベントマネージャ４９０４により格納される型（例えば、生データ、．ｗａｖ、．ｄｏｃなど）が付与されても良い。
【０３６２】
サービスグループ中の１つのアプリケーションから同じサービスグループ中の別のアプリケーションへの任意の長さのデータの送信を可能にするのには、データグラム方式が使用される。この方法では、送信側アプリケーションが受信側アプリケーションの識別（ＩＤ）番号（上述のようにＸｉｄとも呼ばれる）を知っている必要がある。ＩＤ番号は、アプ

リケーション４９２０が開始されるときに対応するランチャ４９１０により生成され、アプリケーション４９２０は一意的に識別される。ＩＤ番号はイベントマネージャ４９０４のコンテキストにおいてのみ一意的である。このように、多くの実行中のアプリケーションは同じＩＤ番号をもつことができるが、全てのＩＤ番号はそのアプリケーションが接続される同じイベントマネージャ４９０４に接続される全てのアプリケーション４９２０において一意的である。イベントマネージャ４９０４は、２重のＩＤ番号が使用されるときにそれを検知することができるが、この検知を保証すると共に新規のアプリケーションの開始を防止するのは、対応するランチャ４９１０の責任である。

【０３６３】

データグラム方式を使用してメッセージを送信するために、送信側アプリケーションはイベントマネージャ４９０４から宛先アプリケーションのＸｉｄを取得し、Ｘｉｄを使用してメッセージのアドレス指定を行ない、イベントマネージャ４９０４を介してメッセージを宛先アプリケーションに送信する。イベントマネージャ４９０４は、メッセージを含むパケットに対してデータ長及びヘッダの送信者フィールドが正しいことを確認する以外には何も行なわない。

【０３６４】

データグラム方式を利用可能にするには、イベントマネージャ４９０４はサービスグループ中で他のどのアプリケーションが実行中であるかを判定する何らかの方法をアプリケーションに提供しなければならない。この情報は、アプリケーションが他のアプリケーションの能力を判定するための何らかの方法も含む。アーキテクチャ４９００では、アプリケーションがイベントマネージャ４９０４に登録するときにアプリケーションが列挙する機能文字列のリストを使用してこれが実行される。イベントマネージャ４９０４はどんな形であろうとこれらの機能を理解する必要がなく、機能のリストはサービス固有のものである。サービス中の他のアプリケーションのみが各機能文字列が何を意味するのかを理解する必要がある。

【０３６５】

イベントマネージャ４９０４は、この方法を使用することで送信可能なメッセージのサイズに上限を課しても良い。

【０３６６】

アーキテクチャ４９００では、上述の掲示板により、データを即座にサービスグループ中の全てのアプリケーションに同報通信することができると共に、サービスグループ中のアプリケーションは中央のリポジトリにデータを格納することができる。これにより、いずれか１つのアプリケーションが常にサービスグループに存在する必要がなくなる。また、掲示板により、スマートカードは任意の順序で挿入することができ、サービスグループ中のアプリケーションは任意の順序で実行することができる。アプリケーションはサービスグループに提供するデータを掲示板に掲示し、アプリケーションがアクションをとる必要があるときには、必要なデータを求めて掲示板を検査することができる。

【０３６７】

データを掲示板に掲示するために、ポスティングアプリケーションは、掲示することが望まれるデータ、すなわち、記述文字列と、場合によっては、何らかの形態のタイピング情報（ＭＩＭＥタイプなど）とをイベントマネージャ４９０４に送信する。アプリケーションがタイプ情報を供給しない場合、イベントマネージャ４９０４はデータにデフォルトのタイプ（例えば、デフォルトのバイナリデータ、ＭＩＭＥタイプアプリケーション／オクテットストリーム）を割り当てることになる。イベントマネージャ４９０４は、掲示板のメッセージを識別するのに使用されるメッセージ識別子をこのメッセージに割り当てる。このメッセージ識別子は、他のアプリケーションにより掲示板からメッセージを取得するのに使用される。また、メッセージ識別子はポスティングアプリケーションが掲示板からメッセージを削除するのにも使用される。掲示板とサービスグループに対応する掲示板に残っているメッセージとは、サービスグループが終了させられると破壊される。

【０３６８】

掲示板からメッセージを取得するには、アプリケーションは必要なメッセージのメッセージ識別子を探さなければならない。アプリケーションは掲示板上のメッセージのリストを取得することができるが、このリストはメッセージ識別子、ポスタ識別子、及び掲示板上の各メッセージのメッセージ記述を含むであろう。第２の方法も、イベントマネージャ４９０４から実行中のアプリケーションのリストを取得することを含む。これにより、アプリケーションには、各アプリケーションがサービスに対して提供する機能が提供される。掲示板にメッセージを要求するアプリケーションは、情報を必要とするアプリケーションのアプリケーション識別子（Ｘｉｄ）を相互参照し、そのアプリケーションにより掲示される全てのメッセージを取得することができる。

【０３６９】

掲示板上のメッセージ及びメッセージ記述の双方の形式はサービスグループにより決定されるものであり、完全に任意のものであっても良い。イベントマネージャ４９０４はデータに対していかなる構造も強制しない。

【０３７０】

このような通信方法をサポートするために、イベントマネージャ４９０４には掲示板を維持することが要求される。イベントマネージャ４９０４にとっては、掲示板は単なる既知の長さのデータブロックのリストとして見える。アプリケーションが掲示板にメッセージを掲示するとき、イベントマネージャ４９０４はそのデータ及び長さを格納する。アプリケーションが掲示板からメッセージを読むとき、イベントマネージャ４９０４はそのデータをアプリケーションに送信する。また、イベントマネージャ４９０４は、掲示板の内容のリストを要求するアプリケーションに対してこれを作成する。

【０３７１】

イベントマネージャ４９０４は、サービスグループ中の全てのアプリケーションにより掲示することが可能な全てのメッセージの合計サイズのみならず、各アプリケーションが掲示することが可能なメッセージの合計サイズを制限しても良く、その結果、各アプリケーション及び各サービスグループはメッセージサイズ制限を有するようになる。また、アプリケーション及びサービスグループが掲示するメッセージの数を制限しても良い。メッセージの記述のサイズも最大長にまで制限されても良い。

【０３７２】

１３．３　システム初期化

本節では、図４８のソフトウェアアーキテクチャ４９００を組み込むシステム６００を最初に開始するプロセスを説明する。これは、分散型セットトップボックスシステム６００Ｂのみならず、コンピュータシステム６００Ａにも関連する。

【０３７３】

最初に、マスタランチャ４９０８が開始され、ネットワーク２２０を介して通信ポートで応答の有無を聞き取る。イベントマネージャ４９０４が開始され、イベントマネージャ４９０４はマスタランチャ４９０８に接続する。

【０３７４】

アーキテクチャ４９００のこれらの２つのコア部分を開始する順序は、システム６００Ａの場合には任意であるが、セットトップボックスシステム６００Ｂにおいて使用されるときには明白な利点を有する。システム６００Ｂでは、イベントマネージャ４９０４が開始されるときにはマスタランチャ４９０８は既に実行中であり、より多くのユーザがサービスを申し込むときにはより多くのイベントマネージャを開始し、ユーザがサービスを終了するときには実行中のイベントマネージャの数を減少させることができる。

【０３７５】

１３．４　システム起動

本節では、図６Ａ又は図６Ｂのハードウェアアーキテクチャ及び図４８の代替のソフトウェアアーキテクチャを組み込むスマートカードシステムを開始するプロセスを説明する。この説明は、イベントマネージャ４９０４及びマスタランチャが既に実行中であり、オープンな接続を有することを前提とする。

（ｉ）Ｉ／Ｏデーモン４９０２が開始され、イベントマネージャ４９０４への接続を開始する。
（ｉｉ）イベントマネージャ４９０４はＩ／Ｏデーモン４９０２からの接続を受け入れる。サービス課金を実行することができるのはこの段階である。例えば、ユーザが料金の支払いを済ませていない場合、接続を拒否することができる。
（ｉｉｉ）イベントマネージャ４９０４は、マスタランチャ４９０８にイベントマネージャ４９０４がどのポートで聞き取りを行なっているかを通知してマスタランチャ４９０８に新規のランチャ４９１０を要求し、入力接続を待つ。
（ｉｖ）マスタランチャ４９０８は新規のランチャ４９１０を開始し、新規のランチャ４９１０にイベントマネージャ４９０４のアドレス及びポート番号を教える。
（ｖ）新規のランチャ４９１０はイベントマネージャ４９０４との接続を開始する。
（ｖｉ）イベントマネージャ４９０４は接続を受け入れる。

【０３７６】

システム６００は、ユーザがスマートカードをリーダ１へと挿入し、最初のボタン押下を開始するときには、アプリケーション４９２０を開始する準備ができている。

【０３７７】

１３．５　第１のスマートカードサービスの開始

本節では、図４８のソフトウェアアーキテクチャを組み込むシステム６００上で他のサービスが実行中でない場合にスマートカードサービスを開始するプロセスを説明する。これは、システムが最初に開始されるときの状況であり、タイムアウトを介して、あるいは、ユーザがスマートカード１０を挿入せずにリモート操作装置１に接触したためにサービスが終了する場合にも発生する。
（ｉ）ユーザがスマートカード１０をリーダ１へと挿入し、タッチパネル８を押下する。
（ｉｉ）押下イベントがイベントマネージャ４９０４に送信され、イベントマネージャ４９０４はパケットの形式を変更し、ランチャ４９１０へと転送する。
（ｉｉｉ）ランチャ４９１０はパケットを受信し、いずれのサービスもアクティブでないことを認識し、スマートカード１０のサービス識別子（ベンダ識別子及びアプリケーション識別子）及びサービス固有識別子（カード識別子）を伴ってディレクトリサービス４９１２に問合せを行なう。
（ｉｖ）問合せは適切なアプリケーション４９２０の位置を戻し、この位置をランチャ４９１０が取り込む。アプリケーション４９２０は、一般的に、ネットワーク２２０のいずれかの場所のサーバコンピュータ上の記憶装置からリモートで供給されるが、ランチャ４９１０に対してローカルに実行されなければならない場合もある。高度なシステムでは、アプリケーションはランチャからリモートで実行されても良い。
（ｖ）ランチャ４９１０は、イベントマネージャ４９０４に新規のアプリケーション４９２０が開始されることを通知する。
（ｖｉ）アプリケーション４９２０は、実行されるランチャへのダウンロードを終了すると、ランチャ４９１０により開始される。
（ｖｉｉ）アプリケーション４９２０はイベントマネージャ４９０４との接続を開始し、イベントマネージャ４９０４がその接続を受け入れると、ランチャ４９１０に登録する。これは、アプリケーション４９２０がどのサービスグループの一部であるか及びアプリケーションがどの機能を実行可能であるのかを含む。
（ｖｉｉｉ）ランチャ４９１０は、新規のアプリケーション４９２０にフォーカスが得られることを通知する。

【０３７８】

この段階でアプリケーション４９２０は開始され、イベントを受信できる状態になる。リーダ１により生成されるＰＲＥＳＳメッセージ、ＲＥＬＥＡＳＥメッセージ、及びＭＯＶＥメッセージは、そのアプリケーションにより意図される限り、イベントマネージャ４９０４によりアプリケーション４９２０へと転送される。アプリケーション４９２０は、登録が完了するまでいかなる形であろうとイベントマネージャ４９０４と対話することがで

きない。更に、イベントマネージャ４９０４はイベントをアプリケーションへと転送することはない。イベントマネージャ４９０４が登録されていないアプリケーション４９２０から受信するアプリケーション登録イベント以外のいかなるイベントも破棄されることになる。

【０３７９】

１３．６　アプリケーションの開始、制御、及び停止

図５６（ａ）及び図５６（ｂ）は、ソフトウェアアーキテクチャ４９００を組み込むシステム６００上でユーザにサービスを提供するために、複数のアプリケーション４９２０のうちの１つのアプリケーション（アプリケーション＃１～＃ｎ）を開始、制御、及び停止する方法５６００を示す。方法５６００のプロセスは、システム６００ＡのＣＰＵ２０５又はシステム６００ＢのＣＰＵ４３０５などのＣＰＵにより実行される。方法５６００を示すソフトウェアプログラムは、システム６００ＡのＣＤ－ＲＯＭ２１２又はシステム６００Ｂのメモリ４３０６などのメモリ媒体に格納される。ユーザがスマートカード１０をリーダ１に挿入し、タッチパネル８を押下して所望の印を選択すると、リーダ１のＣＰＵ４５はスマートカード１０からカードヘッダ１１００及び選択された印と関連付けられるデータを読み、選択された印と関連付けられた押下イベント（例えば、押下メッセージ）をイベントマネージャ４９０４に送信する。イベントマネージャ４９０４はパケットの形式を変更する。イベントマネージャ４９０４は、押下パケット（例えば、ＥＭ－ＲＥＡＤＥＲ　ＰＲＥＳＳ）をランチャ４９１０に送信する。ソフトウェアプログラムはＣＰＵにより実行されるが、このＣＰＵは、カードインタフェース（デーモン）４９０２がリーダ１から押下イベントを受信し、それをイベントマネージャ４９０４に送信すると、少なくとも同じ計算装置のカードインタフェース（デーモン）４９０２、イベントマネージャ４９０４、ランチャ４９１０、及びアプリケーション４９２０を実行する。これに対し、ソフトウェアプログラムが別々の計算装置の各ＣＰＵにより実行される場合、イベントマネージャ４９０４を実行する第１の計算装置の第１のＣＰＵがステップ５６０３から５６０８を実行し、少なくともランチャ４９１０及びアプリケーション４９２０を実行する第２の計算装置の第２のＣＰＵがステップ５６０９から５６３６を実行する。

【０３８０】

ステップ５６０３において、イベントマネージャ４９０４を実行することによって、ＣＰＵはカードインタフェース４９０２を介してリーダ１から押下イベントを受信し、次のステップ５６０５において、押下イベント中のサービス識別子（ベンダ識別子及びアプリケーション識別子）が既に実行中のアプリケーション４９２０のうちのフロントアプリケーション（例えば、アプリケーション＃１）のサービス識別子と一致するか否かを判定する。次のステップ５６０５においてマッチングテーブルを使用してサービス識別子がフロントアプリケーション（例えば、アプリケーション＃１）のサービス識別子と一致すると判定される場合、次のステップ５６０８においてイベントマネージャ４９０４を実行することによって、ＣＰＵは押下パケットをフロントアプリケーションへと転送し、方法５６００は終了する。アプリケーション４９２０の各アプリケーションと対応するサービス識別子との間の関係を有するテーブルが、メモリ２０６又はメモリ４３０６のＲＡＭに格納される。ステップ５６０５において、サービス識別子がフロントアプリケーションのサービス識別子と一致しないと判定される場合、次のステップ５６０７において、ＣＰＵはイベントマネージャ４９０４からランチャ４９１０へと押下パケットを転送する。次のステップ５６０９において、ランチャを実行することによって、ＣＰＵはサービス識別子を伴ってディレクトリサービス４９１２に問合せを行ない、サービス識別子に対応する新規のアプリケーション（例えば、アプリケーション＃２）の位置を受信する。次のステップ５６１１において、ランチャ４９１０を実行することによって、ＣＰＵはその位置から新規のアプリケーションを取り込む。次のステップ５６１３において、ランチャ４９１０を実行することによって、ＣＰＵは新規のアプリケーション（例えば、アプリケーション＃２）を実行する。次のステップ５６１５において、ＣＰＵは新規のアプリケーションとイベントマネージャ４３０６との間の接続を開始する。イベントマネージャ４３０６が接続を受

け入れると、ＣＰＵは新規のアプリケーションをランチャ４９１０に登録し、アプリケーションはランチャ４９１０にどのサービスグループの一部であるかを通知する。次のステップ５６１６において、ＣＰＵはメモリ２０６又はメモリ４３０６のＲＡＭに格納されたサービスグループテーブルを使用して、新規のアプリケーションが現在実行中のアプリケーションとサービスグループを共有するか否かを判定する。各サービス識別子と対応するサービスグループとの間の関係を有するテーブルは、メモリ２０６又はメモリ４３０６中のメモリのＲＡＭに格納される。例えば、テーブルにおいて、サービス識別子１（アプリケーション＃１）及びサービス識別子３（アプリケーション＃３）がサービスグループＡに対応し、サービス識別子２（アプリケーション＃２）及びサービス識別子４（アプリケーション＃４）がサービスグループＢに対応する。次のステップ５６１６において、新規のアプリケーションが現在実行中のアプリケーションとサービスグループを共有すると判定される場合、次のステップ５６３５においてランチャ４９１０を実行することによって、ＣＰＵは現在のアプリケーション（フロントアプリケーション）にフォーカスが失われることを通知する。次のステップ５６３６においてランチャ４９１０を実行することによって、ＣＰＵは新規のアプリケーションにフォーカスが得られることを通知し、方法５６００は終了する。この場合、ＣＰＵは現在のアプリケーション（フロントアプリケーション）の実行をバックグラウンドで継続するが、リーダ１からはイベントを受信しない。現在のアプリケーションを実行することによって、ＣＰＵは同報メッセージ及びメッセージを特定のアプリケーションに送信することができるが、サービスグループから外れることはできない。

ステップ５６１６において、新規のアプリケーションが現在実行中のアプリケーションとサービスグループを共有しないと判定される場合、ステップ５６１７においてランチャ４９１０を実行することによって、ＣＰＵは現在実行中のアプリケーションに終了するように通知し、タイムアウトを設定する。次のステップ５６２１においてランチャ４９１０を実行することによって、ＣＰＵはタイムアウトを待ち、新規のアプリケーションを除く残りのアプリケーションを終了させる。次のステップ５６２３においてランチャ４９１０を実行することによって、ＣＰＵはイベントマネージャ４９０４に終了した又は終了させられたアプリケーションを通知する。次のステップ５６３６においてランチャ４１０を実行することによって、ＣＰＵは新規のアプリケーションにフォーカスが得られることを通知し、方法５６００は終了する。この場合、ＣＰＵは新規のアプリケーションを実行中であり、ＥＭ－ＲＥＡＤＥＲ ＰＲＥＳＳ、ＥＭ－ＲＥＡＤＥＲ－ＲＥＬＥＡＳＥ、及びＥＭ－ＲＥＡＤＥＦ ＭＯＶＥなどのそのアプリケーション用の押下パケットを受信する。システム６００Ａ又は６００Ｂは１つだけアプリケーションを有する新規のサービスを実行中である。

【０３８１】

１３．７　２つのアプリケーション間でのデータの受け渡し

本節では、図４８のソフトウェアアーキテクチャを組み込むシステム６００上でデータグラムプロトコルを使用して２つのアプリケーション４９２０（アプリケーション＃１）及び４９２０（アプリケーション＃２）間でデータの受け渡しを行なうプロセスを説明する。この方法では、送信側アプリケーション＃１が受信側アプリケーション＃２のアプリケーション識別子（Ｘｉｄ）を知っている必要がある。

（ｉ）送信側アプリケーション＃１は送信を希望するデータを収集する。

（ｉｉ）送信側アプリケーション＃１はランチャ４９１０に現在のサービスグループで実行中のアプリケーションのリストを要求する。

（ｉｉｉ）ランチャ４９１０はアプリケーション＃１に現在のサービスグループ中の全てのアプリケーションのリストを送信する。このリストは、アプリケーションが提供した記述文字列のみならず、各アプリケーションがランチャ４９１０に実行可能であると通知した各機能を含む。このリストの順序では、直前のアプリケーションが先頭に列挙される。

（ｉｖ）送信側アプリケーション＃１は、データの適切な受信者の有無を調べる。受信者が存在しない場合、処理の進め方を決定するのはアプリケーション＃１の役割である。ア

プリケーション＃１は、例えば、データを送信しなくても良く、場合によっては、ユーザに別のスマートカードを挿入するように要求しても良い。カードの挿入により所要のアプリケーションが開始されるであろう。
（ｖ）適切な受信者が存在する場合、送信側アプリケーション＃１はイベントマネージャ４９０４を介して受信側アプリケーション＃２にデータを送信する。
（ｖｉ）イベントマネージャ４９０４はメッセージヘッダをチェックして送信側アプリケーション＃１がデータ長及び送信者フィールドを正しく記入していることを確認し、メッセージを受信側アプリケーション＃２に渡す。このようなアプリケーション＃２が実行されていない場合、イベントマネージャ４９０４はメッセージを破棄し、送信側アプリケーション＃１にエラーメッセージを戻す。

【０３８２】
１３．８　掲示板へのデータの掲示
本節では、ソフトウェアアーキテクチャ４９００を組み込むシステム６００上で共通の掲示板にデータを掲示するプロセスを説明する。
（ｉ）ポスティングアプリケーション４９２０は掲示板に掲示することを希望するデータを収集する。
（ｉｉ）ポスティングアプリケーション４９２０はデータをその簡潔な記述と共にイベントマネージャ４９０４に送信する。

【０３８３】
１３．９　掲示板からのデータの取得
本節では、ソフトウェアアーキテクチャ４９００を組み込むシステム６００上で別のアプリケーションにより予め掲示板に掲示されているデータを取得するプロセスを説明する。
（ｉ）要求元アプリケーション＃２は、イベントマネージャ４９０４に掲示板上のメッセージのリストを要求する。
（ｉｉ）イベントマネージャ４９０４は、アプリケーション＃２に掲示板上のメッセージのリストを送信する。このリストは、データの簡潔な記述、メッセージを掲示板に掲示したアプリケーション４９２０に対するアプリケーション識別子（Ｘｉｄ）、及び掲示板上の全てのメッセージに対するメッセージ識別子を含むであろう。
（ｉｉｉ）アプリケーション＃２は、イベントマネージャ４９０４にメッセージ識別子を用いて特定のメッセージを要求することも、ランチャ４９１０に現在実行中の全てのアプリケーションのリストを要求することもできる。
（ｉｖ）アプリケーション＃２が実行中のアプリケーションのリストを要求した場合、ランチャ４９１０はそれをアプリケーション＃２に送信する。このリストは、アプリケーション識別子（Ｘｉｄ）と対応するアプリケーションがランチャ４９１０に実行可能であると報告した機能のリストとを含むであろう。
（ｖ）要求元アプリケーション＃２は、求める機能を実行するアプリケーションからの全て又は一部のメッセージを見つけることができる。

【０３８４】
１３．１０　掲示板からのデータの削除
本節では、ソフトウェアアーキテクチャ４９００を組み込むシステム６００上で同じアプリケーション又は別のアプリケーションにより予め掲示板に掲示されているデータを削除するプロセスを説明する。
（ｉ）要求元アプリケーション＃２は、イベントマネージャ４９０４に掲示板上のメッセージのリストを要求する。
（ｉｉ）イベントマネージャ４９０４は、アプリケーション＃２に掲示板上のメッセージのリストを送信する。このリストは、データの簡潔な記述、ポスティングアプリケーションのアプリケーション識別子（Ｘｉｄ）、及び掲示板上の全てのメッセージに対するメッセージ識別子を含むであろう。
（ｉｉｉ）アプリケーション＃２は、メッセージ識別子を指定することによってイベントマネージャ４９０４に特定のメッセージを削除するように要求する。

【０３８５】
１３．１１　適用例
例Ａ：　カード配列
実施される可能性のあるアプリケーションカード配列は幾つも存在する。アーキテクチャ４９００では、アプリケーション４９２０に対してどのカード配列又はカード配列の組み合わせが採用されるかに関して制約がない。
【０３８６】
図５１Ａに示すサービスグループ内の順次カード配列では、アプリケーションの特定の集合に対するスマートカード１０は指定の順序で挿入される必要がある。例えば、カードＡの後には、カードＢ、カードＣが続き、抜き取り及び／又は再挿入の場合でも同じ配列に従う。
【０３８７】
サービスグループ内の階層カード配列では、アプリケーションの特定の集合に対するカードは図５１Ｂに示すようにツリー状に挿入される必要がある。カードＡが挿入される場合、カードＢ又はカードＣのみを挿入しても良い。カードＢが抜き取られる場合、カードＡを再挿入しなければならない。カードＣが挿入される場合、カードＤのみが挿入されても良く、カードＤが抜き取られる場合、カードＣのみが挿入されても良い。
【０３８８】
サービスグループにおける完全網目状カード配列により、アプリケーションの集合に対するカードを任意の順序で挿入・使用できる。
【０３８９】
例Ｂ：　ピザ注文サービス
従来のピザ注文アプリケーションでは、ピザの種類に関して複数の選択肢（ｖｅｇｅｔａｒｉａｎ、ｓｕｐｒｅｍｅ、及びｍｅａｔ　ｌｏｖｅｒｓなど）が提示されているが、トッピングのカスタマイズ又は特別割引の利用に対する機能は提供されていない。
【０３９０】
アーキテクチャ４９００の下でＪｏｅ’ｓ　Ｐｉｚｚｅｒｉａサービスグループを構成するアプリケーションの集合の一例は以下のようになるであろう：
（ｉ）Ｊｏｅ’ｓ　Ｐｉｚｚａメニュー、
（ｉｉ）トッピングスペシャリスト、
（ｉｉｉ）現在の特別割引、及び
（ｉｖ）個人識別子
これらのアプリケーションの各々は、他のアプリケーションと協働し、フル機能のピザ注文サービスを創設するように仕向けることができる。Ｊｏｅ’ｓ　Ｐｉｚｚａメニューアプリケーションは、ピザの種類（ｖｅｇｅｔａｒｉａｎ、ｓｕｐｒｅｍｅなど）、ドリンク（コーラ、ライムなど）、及びサイドメニュー（ガーリックブレッド、パスタなど）を顧客が選択することができるユーザインタフェースを提供する。また、このアプリケーションは、現在の注文の買い物かご形式のリストを保持し、注文をリセットするためのボタン及び完了するためのボタンをスマートカードに設ける。
【０３９１】
トッピングスペシャリストアプリケーションは、顧客が現在注文予定のピザのリスト内を移動し、選択したピザにカードの表面に印刷された一連のトッピングの中からトッピングを追加／削除できるユーザインタフェースを提供する。注文可能なピザのリストは実行中のＪｏｅ’ｓ　Ｐｉｚｚａメニューアプリケーションから得られる。ピザのトッピングの変更は、ピザの注文の修正のためにＪｏｅ’ｓ　Ｐｉｚｚａメニューアプリケーションへと伝えられるであろう。
【０３９２】
現在の特別割引アプリケーションは、Ｊｏｅ’ｓ　Ｐｉｚｚｅｒｉａで利用可能な現在の特別割引のリスト内を移動するためのコントロールを提供する。選択される特別割引は、現在の注文に適用されるようにＪｏｅ’ｓ　Ｐｉｚｚａメニューアプリケーションへと伝

えられる。
【０３９３】
個人識別子アプリケーションは、ユーザが提供を希望する詳細によって、ユーザの自宅の住所及び自宅の電話番号をＪｏｅ’ｓ Ｐｉｚｚａメニューアプリケーションへと選択的に伝える方法を提供する。
【０３９４】
例Ｃ： 写真現像サービス
従来の写真アルバムアプリケーション及びＴシャツアプリケーションでは、現在選択されている写真の送受信のためにクリップボードが共有される（ファイルとして）。しかし、写真の修整（例えば、トリミング又は輝度の向上）のための機能、あるいは、複数の連結されたカードを有する機能はない。これらの連結されたカードは１本のフィルム全体を表し、各カードは現在最大２０枚の写真のみを含み、各写真はボタンとして機能するのに十分な大きさのアイコンにより表される。
【０３９５】
アーキテクチャ４９００に関して、写真現像サービスは以下の１組のカードを有するように設計されても良い：
（ｉ）Ｆｉｌｍ＿１ａ、
（ｉｉ）Ｆｉｌｍ＿１ｂ、
（ｉｉｉ）Ｔシャツプリンタ、及び
（ｉｖ）フォトエンハンサ
Ｆｉｌｍ＿１ａカード及びＦｉｌｍ＿１ｂカードは、それぞれ４０枚の写真を含む１本のＡｄｖａｎｔｉｘ（Ｋｏｄａｋ Ｃｏｒｐ，ＵＳＡの登録商標）全体を表す。いずれのカードを最初に挿入しても良い。いずれかのカードが一度挿入されると、挿入されたカードの表面に印刷された写真への直接アクセスを用いることで双方のカードにわたる写真一式へのアクセス権が提供される。これは、スライドショー機能が双方のカードに対応する写真を循環させるであろうことを意味する。各カードは、写真現像サービスグループ中の別のアプリケーションを用いてユーザ用のサービスグループクリップボードに特定の写真参照を追加するためのボタンを有し、アプリケーションは現在閲覧中の写真への参照を戻す機能を提供するであろう。
【０３９６】
Ｔシャツプリンタアプリケーションは、直前に見た写真（フィルムアプリケーションから得られるものへの参照）を使用してＴシャツ転写物をインスタントで印刷するか、あるいは、クリップボードにある１組の写真からＴシャツ転写物を作成する機能を提供する。
【０３９７】
簡易な写真編集サービスの一部として、フォトエンハンサアプリケーションは、直前に見た写真（Ｔシャツアプリケーション及びフィルムアプリケーションのうちの直前にフォアグラウンドにあった方から得られる）に対して動作する。フォトエンハンサは、自動トリミング、鮮鋭化、ぼかし、淡色化、暗色化などの操作を提供しても良い。このとき、変更をフォトサーバに戻して永続化することができる。
【０３９８】
例Ｄ： ビデオ電子メールサービス
従来のビデオ電子メールアプリケーションは、カードの表面に記載されるビデオ電子メールユーザにビデオ電子メールメッセージを送信する手段を提供する。設計を幾らか変更すれば、アーキテクチャ４９００によるビデオ電子メールサービスを創設することが可能である。このサービスでは、スマートカード名刺をアドレス帳の所有者に供給する複数ユーザからアドレス帳を編纂することができる。ビデオ電子メールサービスを形成するアプリケーションは以下の通りである：
（ｉ）ビデオ電子メール送信、
（ｉｉ）ビデオ電子メールメールボックス、
（ｉｉｉ）ビデオ電子メールアドレス帳、及び

（ｉｖ）名刺
ビデオ電子メール送信アプリケーションは、挿入される個人識別カード又は挿入される名刺からアドレスが得られる点を除き、従来のアプリケーションとほぼ同様に動作する。
【０３９９】
ビデオ電子メールメールボックスアプリケーションは、リモートサーバからビデオ電子メールメッセージを取得する機能を提供すると共に、ビデオ電子メール送信アプリケーションにより返信アドレスとして使用される送信者のアドレスを提供する。
【０４００】
アドレス帳機能は、ビデオ電子メールアドレス帳アプリケーションにより提供される。このアプリケーションにより、ユーザは挿入される様々な名刺、個人識別子カード、又はビデオ電子メールメールボックスカードからアドレスのリストを増強することができる。ビデオ電子メール送信アプリケーションにより使用するため、アドレスのリストから１つ以上のエントリが選択されても良い。
【０４０１】
例Ｅ：　買い物かごサービス
従来のソフトウェアアーキテクチャでは、オンラインショッピングを提供するアプリケーションは、買い物かご手段、注文手段、料金請求手段、及び発送手段を含む独自の購入システムを維持する必要があった。アーキテクチャ４９００の一部として利用可能な特徴を利用する買い物かごサービスは、これらの機能が各オンラインショッピングアプリケーションから分離できるようにし、他の機能のためのユーザインタフェース領域を広げるであろう。このような買い物かごサービスの一部を形成するであろうアプリケーションは以下の通りである：
（ｉ）電子配送買い物かご、
（ｉｉ）Ｄａｖｙ　Ｊｏｎｅｓオンライン、及び
（ｉｉｉ）Ｐａｃｅ　Ｂｒｏｓ．オンライン
電子配送買い物かごアプリケーションは、総合買い物かご管理機能、支払機能、及び注文機能を提供する。
【０４０２】
Ｄａｖｙ　Ｊｏｎｅｓオンラインアプリケーション及びＰａｃｅ　Ｂｒｏｓ．オンラインアプリケーションは、対応するデパートから仕入れ可能な品目のリストを関連する品目記述と共に閲覧する機能を提供する。購入を希望する品目が見つかったときには、電子配送買い物かごアプリケーションを経由してその品目を買い物かごに追加し、注文・配送に備えることができる。
【０４０３】
先の説明から明らかなように、管理プロセスの部門分け及びアプリケーションの適切な起動を介して柔軟性の拡張を許容するカードインタフェースシステムを実現するのにアーキテクチャ４９００を使用しても良い。これにより、アプリケーションは機能的結果を達成するために協力的に動作することができる。更に、複雑度に見合ったプラットフォームで手順を動作させる能力を介して、種々の複雑度のハードウェアプラットフォームからアーキテクチャ４９００の種々のコンポーネントを動作させることができる。このようなプラットフォームは、処理能力の限られたローエンドのセットトップボックスから、家庭用ＰＣ及びリモートサーバコンピュータにまで及ぶ。特に、「ダム」セットトップボックスの場合、カードデーモン４９０２はセットトップボックスの内部から１つ以上のリモートサーバコンピュータの全てのプロセスのバランスにより実行されるであろう。これに対し、スマートセットトップボックス又は家庭用パーソナルコンピュータの場合、全てのプロセスは、ネットワーク２２０を介する外部通信が必要不可欠な場合を除き、１つのハードウェアの内部から動かされるであろう。
【０４０４】
また、アーキテクチャ４９００は、不正なデータ吸上げ及び詐欺行為からユーザ及びベンダの双方を保護するために、特定のアプリケーションに適した機密保護モデルをサポート

するように拡張することができる。
【０４０５】
イベントコマンドの導管として作用するイベントマネージャ４９０４により、アーキテクチャは通信プロトコルのある範囲のバージョンにわたって開発されたアプリケーションと共に動作することができる。このようなアーキテクチャは、通常、時の経過と共に開発される可能性があるからである。
【０４０６】
アーキテクチャ４９００により、カードインタフェースシステム６００は、カードアプリケーションが予期される動作モードに従っていないときでも機能を継続することができる。これは、アプリケーションの突然の終了、コマンドによる終了の拒否、及びシステム６００への不正な又は余分なデータの送信を含む。アーキテクチャ４９００は、同じサービスグループに所属するアプリケーションの各カードによってマルチカードアプリケーションをサポートし、それにより、カードが抜き取られて新規のカードが挿入されるときにアプリケーション実行の維持が保証される。
【０４０７】
１３．１２　アプリケーション管理システム
コンピューティングリソースを過負荷状態にしたり、適切な応答を保証したりすることなく複数のアプリケーションが同時に動作可能であるようにするために、サービスグループの概念、その確立、及びその消滅を利用してアーキテクチャ４９００の説明を行なってきた。
【０４０８】
複数のアプリケーションを考慮する際の代替のアプローチは、アプリケーション間のデータフローがデータの作成者からデータの消費者への方向であると解釈することから生じる。図５５は有向グラフを示し、グラフ方向は、群機能を実行するために消費者から作成者へと向かう。この場合、Ｔシャツは名前と表面に転写される写真とを有し、そのデータは複数の他のアプリケーションから得られる。このようなグラフ構造内のアプリケーションの管理は、グラフの各ノードのアクセス可能性によって決まる。具体的には、グラフ中のノードが到達不可能になると、そのノードにおけるアプリケーションは認識している機能を実行することができないので、そのアプリケーションは終了させられるべきである。更に、そのアプリケーションの生成物の消費者がそのサービスに対する登録を解除するときには、ノードへのリンクは削除されるべきである。アプリケーションは開始されるとツリーに配置される。アプリケーションが消費者が望む種類の作成者である場合、アプリケーションはツリー中のそのノードの下に配置される。
【０４０９】
上述のように、アプリケーション４９２０は、アーキテクチャ２００に関して先に説明したサービス識別子と同等の対応するベンダ識別子及びアプリケーション識別子により参照される。アプリケーション識別子（すなわち、Ｘｉｄ）は、既に実行中のアプリケーションを起動する際の迅速なマッチング用のユニークキーとして使用される。アプリケーション識別子は、ベンダ識別子及びカード識別子（カード識別子はアーキテクチャ２００を参照して先に説明したサービス固有識別子と同等である）と共にカード上に格納されるものと、開始されるときにシステムによりアプリケーションに割り当てられるものとの２つが存在する（後者のアプリケーション識別子は、ここではコンポーネント識別子又はＸｉｄとも呼ばれ、前者のアプリケーション識別子は上述のようにサービス識別子に関連する）。
【０４１０】
各アプリケーションは、識別のためにＸｉｄを使用することで必要に応じて機能の作成者又は消費者として登録しても良い。アプリケーションは、ユーザインタラクションを介して特定の時点で何が必要かを認識している。例えば、ユーザはアプリケーション内を「写真追加」画面まで移動する。ここで、アプリケーションはフォト消費者として登録しても良い。この点に関して、サービスグループアプローチは実際には大まか過ぎるので、登録

はサービスグループではなく機能に基づいて行なわれるのが好ましい。更に、このアプローチでは、サービスグループ中の全てのアプリケーションがそのサービスグループにより提供される全ての機能をサポートしない限り、作成者が必要とする特定のサービスを提供することができない可能性があるときに、アプリケーションは消費者／作成者関係において別のアプリケーションにリンクすることができない。

【０４１１】

アプリケーションは他のアプリケーションから２つ以上の機能を要求する可能性があるので、このようなモデルは実現用の２つのオプションを提示する。

【０４１２】

１．グラフ中の各ノードは、他のどのノードに対しても接続を１つだけ有する。これは、接続が消費者／作成者関係に含まれるサービスのリストも含まなければならないことを意味する。消費者がサービスに対する登録を解除するたびにリストエントリが削除される。接続用のサービスのリストが空になると、接続は削除される。接続が削除されると、その接続によりリンクされる作成者もチェックを受ける。作成者ノードが既に他のノードに接続していない場合、そのノードは削除されるであろう。

【０４１３】

２．このオプションは、サービスのリストを保持する代わりに、各サービスが消費者／作成者ノード間における別々の接続である点を除いては（１）と同様である。このため、２つのアプリケーション間には複数の接続が存在するであろう。消費者がサービスに対する登録を解除するとき、その接続は削除される。作成者が既に接続されていない場合、消費者は終了する。

【０４１４】

この提案は、それぞれが現在リーダ１に挿入されているスマートカードと関連付けられるアプリケーションを特定のユーザ動作以外のイベントにより終了させることができる点において問題である。これは、ユーザの観点から混乱を招く恐れがある。従って、アプリケーションの終了に対する代替のアプローチが望まれる。

【０４１５】

このような代替のアプローチでは、アーキテクチャ４９００は、上述のように特定のサービスグループのメンバである任意のアプリケーション４９２０に特に依存することなく動作しても良いが、「主」サービスグループと呼ばれるグループの一時情報を介して動作する。主サービスグループは、任意のアプリケーション４９２０が、作成者である場合、消費者である場合、作成者兼消費者である場合、及び作成者でも消費者でもない場合のいずれに分類されるかによって判定される２つ以上の現在のアプリケーション間の一時機能的関係から生じる。

【０４１６】

アプリケーション４９２０に対するこのような管理システムは、同じサービスグループ中のアプリケーションの作成者／消費者対又はアプリケーションが双方の基準を満たす場合には単一のアプリケーションが登録されるときに形成される「主」サービスグループの概念を中心とする。例えば、アプリケーションＡｃ及びＡｐが同時に動作するときにはサービスグループＡが主となり、作成者－消費者対に適合する。一方、作成者－消費者対に適合するＡｃＢｐ又はＡｐＢｃは主サービスグループを作成しない。この管理システムによると、主サービスグループが形成されるとき、そのグループを共有しない全てのアプリケーションが終了させられる。双方が同時に登録される場合、主サービスグループは、第２の主サービスグループと共に存在しても良い。例えば、アプリケーション＃１が開始されてＡｐＢｐを登録し、アプリケーション＃２が開始されてＡｃＢｃを登録する場合、Ａ及びＢは主になる。２つ以上の主サービスグループが存在するためには、開始される新規のアプリケーションが作成者－消費者対を確立する各グループに登録するときにそれらのサービスグループが形成されなければならない。サービスグループを形成するアプリケーションの作成者／消費者対は、主サービスグループが主グループの「従属者」になった後に登録される。従属グループの従属グループが形成されても良い。従属者の従属者は、従属

者の作成者が既に第２の従属者に対する消費者として登録されているときに形成される。

【０４１７】

このような管理構造の最終結果は、ユーザが望む最終結果を達成するためにデータの受け渡しを行なう相互作用アプリケーションのツリー又はグラフの作成及び解体である。具体的には、このような結果は、ピザ注文サービスに対する上述の例Ｂとは対照的に、利用されるアプリケーションのフェースから容易に明らかにならない可能性がある。このアプリケーション管理構造は、以下の例を参照することで最も良く説明される。

【０４１８】

以下の例は、複数のアプリケーションを参照する。その詳細が以下の表４に記載される。

【０４１９】

【表４】

表4

カードアプリケーション名	記述	サービスグループメンバ（ｐ＝作成者、ｃ＝消費者、ｎ＝どちらでもない）
ＩＤ１	識別詳細カード	Ｚｐ　Ｃｐ
ＩＤ２	識別詳細カード	Ｚｐ　Ｑｐ
ＰｈｏｔｏＩＤ	写真識別カード	Ｚｐ　Ｑｐ　Ａｐ
Ｐｈｏｔｏ１	写真カード	Ａｐ　Ｆｐ
Ｐｈｏｔｏ２	写真カード	Ｍｐ　Ａｐ
ＰＩＮ	個人識別番号カード	Ｐｐ
Ｂａｎｋ	エレクトロニックバンキングカード	Ｂｎ
Ｐｉｚｚａ	ピザ注文カード	Ｒｎ
Ｔ－ｓｈｉｒｔ	Ｔシャツ製造	Ｔｐ
ＣａｒｄＭａｋｅｒ	他のカード作成に使用されるカード	Ｓｐ

【０４２０】

例Ｆ：

この例では、ユーザはカード上に受信者の名前、定番のメッセージ、及び写真を有するグリーティングカードを作成することを望む。表４のカードを使用する第１のステップは、ユーザがＣａｒｄＭａｋｅｒアプリケーションカードをリーダ１へと挿入することであろう。このような動作によりアプリケーションが開始され、サービスグループＡ及びＺの消費者として登録される。アプリケーションは、そのサービスグループへの所属を動的に変更しても良い。例えば、ＣａｒｄＭａｋｅｒが開始され、ユーザに以前に作成したカードと同一のカードの作成を希望するか否かを尋ねる画面表示を提示しても良い。「ＮＯ」と回答すると、新規のカードが作成されることになるので、ＣａｒｄＭａｋｅｒはＩＤ１及びＰｈｏｔｏ１に消費者として登録する。この段階におけるプロセスツリーは図５３Ａに示される。次に、ユーザは写真が要求されていることを認識し、ＣａｒｄＭａｋｅｒアプリケーションを抜き取り、Ｐｈｏｔｏ１アプリケーションを挿入することによってその写真を提供する。ＣａｒｄＭａｋｅｒアプリケーションはまだ機能を実行しなければならないので、リーダ１から抜き取っても動作を継続する。Ｐｈｏｔｏ１アプリケーションの挿入により、図示するようにＡｃ及びＡｐに主サービスグループをクレートに詰めるが、これは、ＣａｒｄＭａｋｅｒアプリケーションが写真を要求し、Ｐｈｏｔｏ１アプリケーションがその写真を供給することができることを意味する。Ｐｈｏｔｏ１アプリケーションは、写真にアクセスするためにＰＩＮを必要とするので、構成は図５３Ｂに表すようになる。Ｐｈｏｔｏ１上の全ての写真が使用のためのロック解除にＰＩＮを必要とする訳ではないので、考慮中の場合のように、処理を進めるのにＰＩＮを必要とするときにはＰｈｏｔｏ１はＰｃとしてのみ登録する。図５３Ｃによると、続いてＰＩＮカードが提供される。図５３Ｃから明らかなように、第２の作成者－消費者対が形成され、この場合、ＰＩＮの提供によりＰｈｏｔｏ１カードはユーザにより選択された写真をＣａｒｄＭａｋｅｒア

プリケーションに供給することができる。これらのタスクが完了すると、プロセスツリーの左側の枝が消去され、図５３Ｄに示すように、対応する「実行済」アプリケーションがイベントマネージャ４９０４から登録を解除する。プロセスを完了する次のステップは、所望の名前を有するカードを挿入することである。ここでは、この名前は、図５３Ｅに示すアプリケーションＩＤ１から得られる。このアプリケーションが所望の名前を供給するので、ＣａｒｄＭａｋｅｒアプリケーションは満足し、それにより、他の全てのアプリケーションは登録を解除し、終了できるようになる。ＣａｒｄＭａｋｅｒアプリケーションは他のアプリケーションとのインタラクションなしに所要のカードを出力することができる。

【０４２１】

代替のアプローチでは、写真及び名前の双方にアクセスするのにＰＩＮアプリケーションが必要とされても良い。双方の写真カードに対するＰＩＮが同じである場合に限り、ＰＩＮアプリケーションカードは１度だけの挿入で良い。図５４に示すようなプロセスツリーが形成されるであろう。この例では、ＰｈｏｔｏＩＤは作成されるカードの受信者の写真を有しても良く、Ｐｈｏｔｏ１は写真に重ねる魅力的な背景写真を有しても良いので、ＰｈｏｔｏＩＤ及びＰｈｏｔｏ１が使用される。

【０４２２】

図５４は、プロセスツリー中のノードに対する複数のリンクが許容され、到達不可能なノード（リンクなしのノード）上のアプリケーションが終了させられることを例証する。

【０４２３】

実行中のアプリケーションに対する上限は７に設定されるのが好ましい。この数を超える場合、アプリケーションの終了によりプロセスツリー中の最古の葉アプリケーションが開始される。

【０４２４】

以上の記述では、本発明の幾つかの構成及びこれらの構成に対する変形を説明しているに過ぎず、本発明の趣旨から逸脱することなく、変形及び／又は変更を行なうことができる。各実施形態は例示のためのものであり、限定することを意図しない。

【図面の簡単な説明】

【図１】

図１は、読取り装置及び関連するカードの斜視図である。

【図２】

図２は、図１に示すカードの反対側の斜視図である。

【図３】

図３は、図１に示すカードを線ＩＩＩ－ＩＩＩに沿って切断したときの長手方向の横断面図である。

【図４】、

【図５】

図４及び５は、図１に示すカードの代替のカード構成の裏面の斜視図である。

【図６（ａ）】

図６（ａ）は、カードインタフェースシステムのハードウェアアーキテクチャを示す図である。

【図６（ｂ）】

図６（ｂ）は、カードインタフェースシステムの別のハードウェアアーキテクチャを示す図である。

【図７】

図７は、図６（ａ）及び図６（ｂ）の汎用コンピュータの概略ブロック図である。

【図８】

図８は、カードインタフェースシステムアーキテクチャの概略ブロック図である。

【図９】

図９は、カードインタフェースシステムの概略ブロック図である。

【図１０】
図１０は、図１のリーダの内部構成を示す概略ブロック図である。
【図１１】
図１１は、図１のカードに格納されたカードヘッダのデータ構造を示す図である。
【図１２】
図１２は、図１１のヘッダの各フィールドの記述を示す図である。
【図１３】
図１３は、図１１のヘッダに含まれる各フラグの記述を示す図である。
【図１４】
図１４は、図１のカードに対するオブジェクト構造の各フィールドの記述を示す図である。
【図１５】
図１５は、図１４のオブジェクト構造に対するフラグの記述を示す図である。
【図１６】
図１６は、図１４のオブジェクト構造に対する各オブジェクト種別の記述を示す図である。
【図１７】
図１７は、図１４のオブジェクト構造によるユーザインタフェースオブジェクト構造に対する各フィールドの記述を示す図である。
【図１８】
図１８は、図１４のオブジェクト構造による各ユーザインタフェースオブジェクトフラグに対する記述を示す図である。
【図１９】
図１９は、図１のリーダから送信されるメッセージヘッダの形式を示す図である。
【図２０】
図２０は、図１９のヘッダに対するメッセージイベント種別を列挙するテーブルを示す図である。
【図２１】
図２１は、単純なメッセージの形式を示す図である。
【図２１（ａ）】
図２１（ａ）は、ＩＮＳＥＲＴメッセージの形式を示す図である。
【図２２】
図２２は、ＭＯＶＥメッセージの形式を示す図である。
【図２３】
図２３は、ＰＲＥＳＳメッセージ及びＲＥＬＥＡＳＥメッセージの形式を示す図である。
【図２４】
図２４は、図６のシステム内のメッセージの流れを示すデータフロー図である。
【図２５】
図２５は、図１のリーダにより実行される読取り方法を示すフローチャートである。
【図２６】
図２６は、図２５の方法の間に実行される図６のシステムを初期化する方法を示すフローチャートである。
【図２７】
図２７は、図２５の方法の間に実行される図１のカードをチェックする方法を示すフローチャートである。
【図２８】
図２８は、図２５の方法の間に実行される図１のリーダのタッチパネルを走査する方法を示すフローチャートである。
【図２９】
図２９は、図２５の方法の間に実行される１０ミリ秒待機方法を示すフローチャートであ

る。
【図３０】
図３０は、図６のシステムのイベントの概要を示すフローチャートである。
【図３１】
図３１は、図３０のプロセスの間にイベントマネージャにより実行されるプロセスを示すフローチャートである。
【図３２】
図３２は、図３０のプロセスの間に実行される新規のアプリケーションを開始するための方法を示すフローチャートである。
【図３３】
図３３は、図３０のプロセスの間に実行されるアプリケーションを終了する方法を示すフローチャートである。
【図３４】
図３４は、永続的アプリケーションに対する現在のセッションを終了する方法を示すフローチャートである。
【図３５】
図３５は、フォーカス変更を実行する方法を示すフローチャートである。
【図３６】
図３６は、ランチャにより実行される方法の概要を示すフローチャートである。
【図３７】
図３７は、図３６の方法の間に実行されるアプリケーションを変更する方法を示すフローチャートである。
【図３８】
図３８は、図３６の方法の間に実行される新規のアプリケーションを登録する方法を示すフローチャートである。
【図３９】
図３９は、ランチャからイベントを受信するときにアプリケーションにより実行される方法を示すフローチャートである。
【図４０】
図４０は、ランチャからイベントを受信するときにブラウザコントローラアプリケーションにより実行される方法を示すフローチャートである。
【図４１】
図４１は、ブラウザアプリケーション方法を示すフローチャートである。
【図４２】
図４２は、システム６００のセットトップボックスをより詳細に示す概略ブロック図である。
【図４３】
図４３は、「ボトムエントリ」型のリーダの斜視図である。
【図４４】
図４４は、図４３のリーダの平面図である。
【図４５】
図４５は、図４３のリーダにカードを挿入するユーザを示す図である。
【図４６】
図４６は、カードが完全に挿入された後に図４３のリーダを操作するユーザを示す図である。
【図４７（ａ）】
図４７（ａ）は、図４４の線Ｖ－Ｖに沿って切断したときの長手方向の横断面図である。
【図４７（ｂ）】
図４７（ｂ）は、カードがリーダのレセプタクルに部分的に挿入された場合の図４７（ａ）と同様の図である。

【図４７（ｃ）】
図４７（ｃ）は、カードがリーダのテンプレートレセプタクルに完全に挿入された場合の図４７（ａ）と同様の図である。
【図４８】
図４８は、更なるカードインタフェースシステムアーキテクチャの概略ブロック図である。
【図４９】
図４９は、カードとアプリケーションとの関係を示す概略ブロック図である。
【図５０】
図５０は、アプリケーションとサービスグループとの関係を示す図である。
【図５１Ａ】、
【図５１Ｂ】、
【図５１Ｃ】
図５１Ａから図５１Ｃは、図４８のアーキテクチャ内の多様なカード配列を示す図である。
【図５２】
図５２は、図４８のアーキテクチャに対するスマートカードに格納される制御テンプレートデータを示す図である。
【図５３Ａ】、
【図５３Ｂ】、
【図５３Ｃ】、
【図５３Ｄ】、
【図５３Ｅ】
図５３Ａから図５３Ｅは、マルチカードアプリケーション構造の一例を示す図である。
【図５４】
図５４は、図５３Ａから図５３Ｅの目的を達成するための代替のアプローチを示す図である。
【図５５】
図５５は、マルチアプリケーション方法の有向グラフ表現を示す図である。
【図５６（ａ）】、
【図５６（ｂ）】
図５６は、アプリケーションを開始する方法を示す図である。
【図５７】
図５７は、図１１のカードヘッダに続く１つ以上のオブジェクト構造を示す図である。
【図５８】
図５８は、図８のディレクトリサービスにより実行されるプロセスの概要を示すフローチャートである。

【図１】

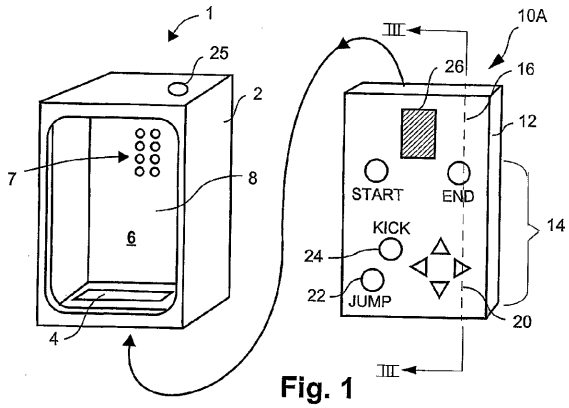


Fig. 1

【図２】

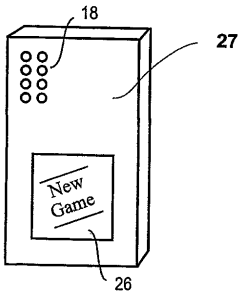


Fig. 2

【図３】

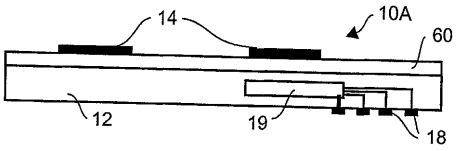


Fig. 3

【図４】

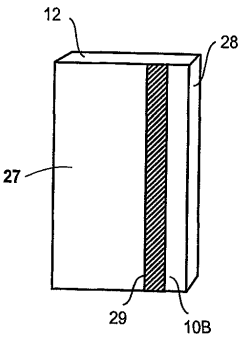


Fig. 4

【図５】

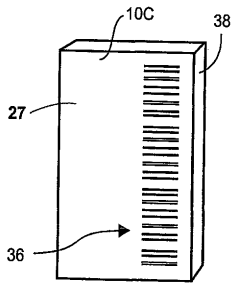


Fig. 5

【図６（ａ）】

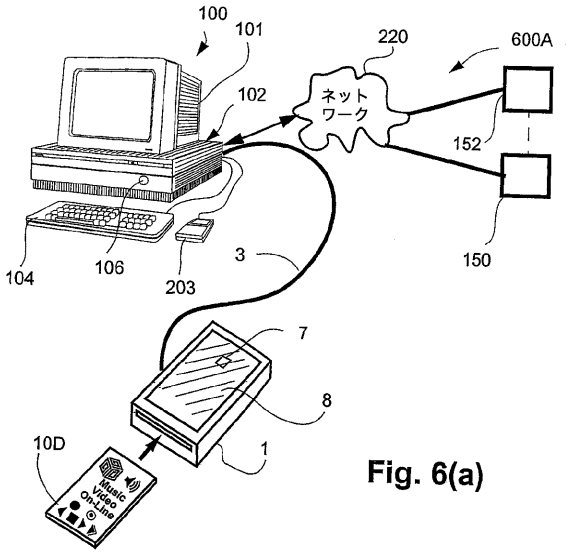


Fig. 6(a)

【図６（ｂ）】

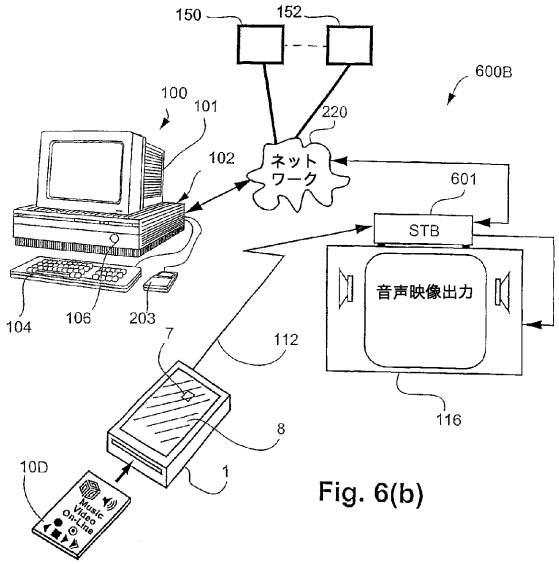


Fig. 6(b)

【図７】

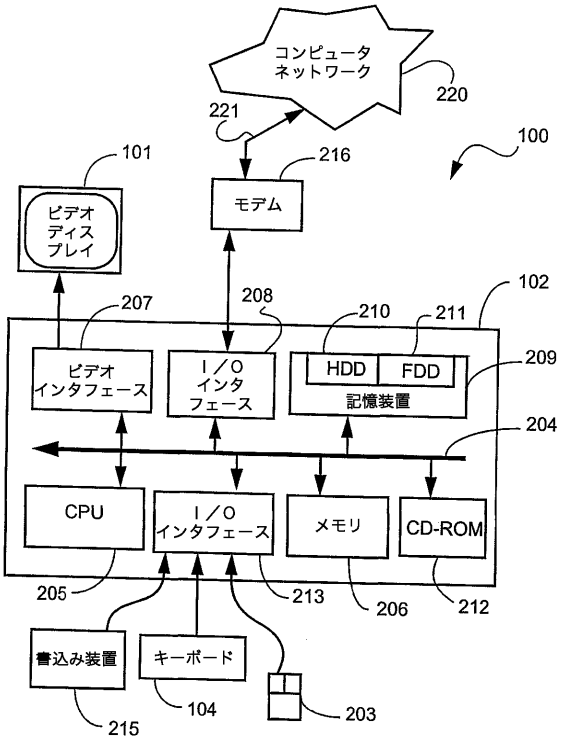


Fig. 7

【図８】

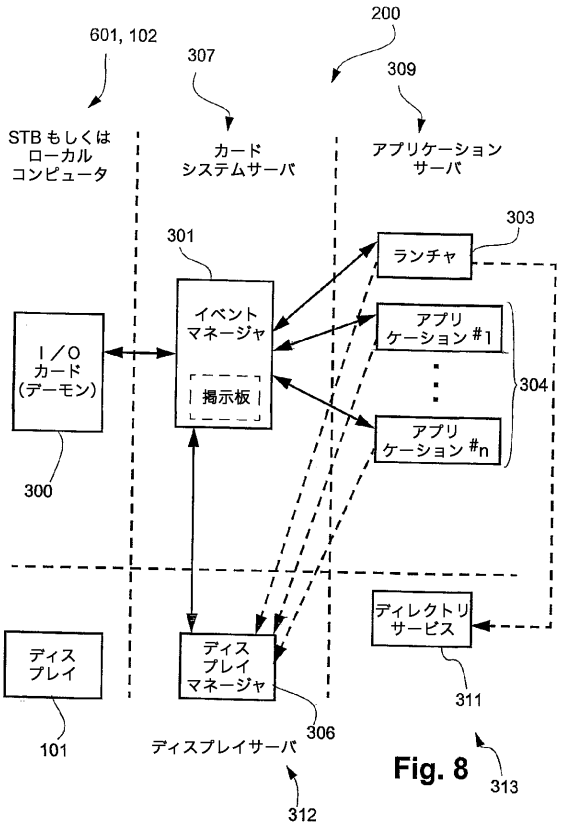


Fig. 8

【図９】

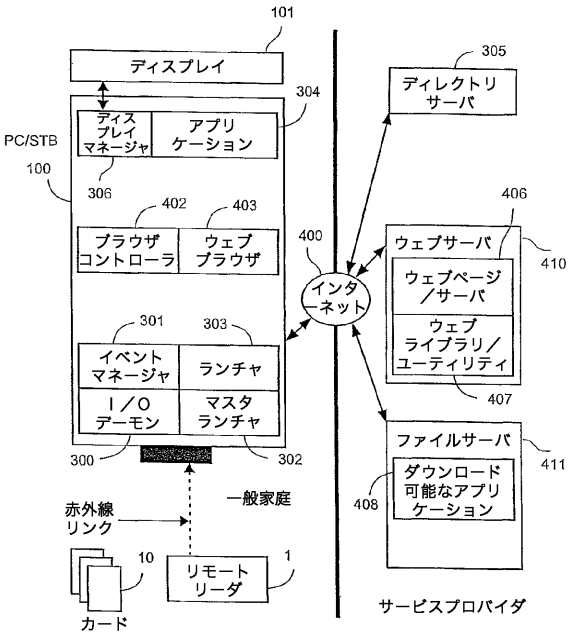


Fig. 9

【図１０】

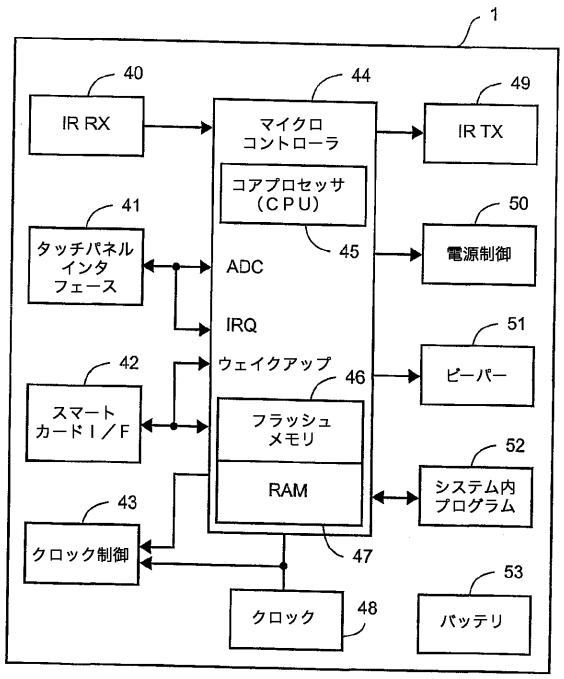


Fig. 10

【図１１】

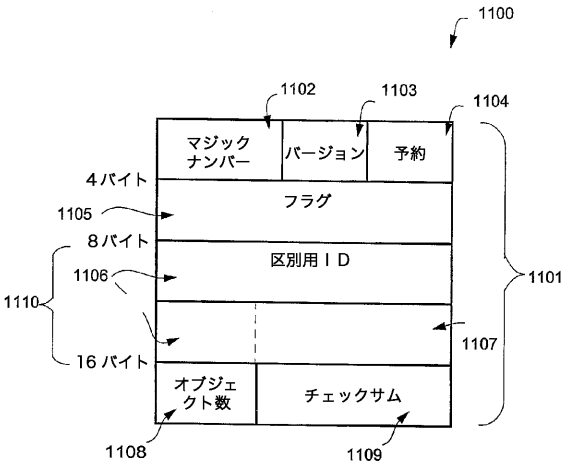


Fig. 11

【図１２】

フィールド番号	記述（カードヘッダ）
マジックナンバー	２バイトのマジックナンバー。 これを有効なカードとして指定する定数。 現在、「ｉ」のＡＳＣＩＩ値＋ 「C」のＡＳＣＩＩ値として定義される
バージョン	１バイトのバージョン番号。 各バージョン増加分はカードレイアウトの 下位のバージョンと互換性のあるリーダにより 読むことができないレイアウトの変更を指定する。 この文書ではカード形式のバージョン１ （０ｘ０１）を記述する
予約	このデータは今後の使用のために予約される。 その値は０に設定されなければならない
フラグ	このカードに対する４バイトのフラグ。（図１３参照。） 全ての未使用ビットは０でなければならない
区別用ＩＤ	８バイトの区別用識別子。 区別用識別子はサービス識別子及び サービス固有識別子の２つのフィールドを含む。 サービス識別子は５バイトであり、 カードと関連付けられたサービスを識別する。 サービス固有識別子は３バイトのサービス 固有値である
オブジェクト数	１バイト。このヘッダに続くオブジェクトの数。 ０の可能性もある
チェックサム	カードチェックサムであり、２バイト。 カードチェックサムは１６ビットであり、 チェックサムを除くカード上の全てのデータ バイトの符号無し整数の合計である

Fig. 12

【図１３】

名前	記述（プリカードフラグ値）	値（１６進）
ビープ停止	リーダユニットがデフォルトで音声 フィードバックを供給するのを停止 する。このビットが設定されている 場合、ＵＩ要素オブジェクトにおいて ＵＩ要素の「ビープ反転」フラグが 設定されていない限り、ＵＩ要素が 押下されたときでもリーダは音声 フィードバックを発行しない	0x0000 0001
ＭＯＶＥ イベントなし	ユーザが指をリーダの表面で 動かしたときにリーダユニットが マウスとして機能するのを停止する	0x0000 0002
イベント 座標なし	リーダがＰＲＥＳＳイベント、 ＲＥＬＥＡＳＥイベント、 及びＭＯＶＥイベントに対する座標を 送信するのを停止する。Ｘ値及び Ｙ値は値０を伴って送信される	0x0000 0004

Fig. 13

【図１４】

名前	記述（オブジェクト構造）	長さ
種別	オブジェクトの種別（図１６参照）	１バイト
オブジェクトフラグ	このオブジェクトと関連付けられる一般的なオブジェクトフラグ（図１５参照）。注：オブジェクト種別に固有の追加フラグはオブジェクトのデータフィールド内で指定される	１バイト
長さ	このオブジェクトに続くデータの長さ。この値は０の可能性がある	２バイト
データ	このオブジェクトと関連付けられるデータ。このデータの構造はオブジェクトの種別によって決まる	可変長

Fig. 14

【図１５】

名前	記述（プリオブジェクトフラグ値）	値（１６進）
非アクティブ	リーダに対してオブジェクトが有効であるが、その種別に関わらず無視されることを示す	0x01

Fig. 15

【図１６】

名前	記述（オブジェクト種別）	値（１６進）
ＵＩオブジェクト	ＵＩカードボタン	0x10
カードデータ	このカードのみに関連するデータを含む	0x20
固定長データ	カード上に固定長のデータブロックを格納するのに使用することができるオブジェクト	0x30
リーダ挿入	カードが挿入されたときにリーダに指示を与えるのに使用することができるオブジェクト	0x40
操作なし	カード上の空きスペースのブロックを埋めるのに使用されるオブジェクト	0x01
操作なし（１バイト）	標準オブジェクトヘッダをもたない１バイトのオブジェクト。通常のオブジェクトヘッダには小さすぎるカード上のスペースを埋めるのに使用される	0x00

Fig. 16

【図１７】

フィールド	記述（ユーザインタフェースオブジェクト構造）	サイズ
フラグ	カード上のこのＵＩ要素に固有のフラグ	１バイト
X1	このオブジェクトの矩形の左下隅の座標のＸ値	１バイト
Y1	このオブジェクトの矩形の左下隅の座標のＹ値	１バイト
X2	このオブジェクトの矩形の右上隅の座標のＸ値	１バイト
Y2	このオブジェクトの矩形の右上隅の座標のＹ値	１バイト
データ	このオブジェクトと関連付けられる０以上のバイトのデータ。このフィールドのサイズはオブジェクトデータサイズ－上述のフィールドの合計サイズにより判定される	可変長

Fig. 17

【図１８】

名前	記述（ＵＩオブジェクト用のフラグ）	値
ビープイネーブル反転	このフラグによりボタンはカードヘッダ中のビープ停止フラグの逆の値を有する。ヘッダにおいてビープ停止フラグが設定されていない場合、このフラグによりボタンはビープ音を発生しない。逆の場合、ボタンはビープ音を発生する	0x01
オートリピート	ボタンの上で押下状態が保たれるとき、ボタンと関連付けられたメッセージが自動的に繰り返される	0x02
押下時データ送信禁止	このフラグによりボタンはＰＲＥＳＳイベントの際にボタンと関連付けられたデータを送信しない。デフォルトはＰＲＥＳＳイベントの際にボタンと関連付けられたデータを送信することである	0x04
リリース時データ送信禁止	このフラグによりボタンはＲＥＬＥＡＳＥイベントの際にボタンと関連付けられたデータを送信しない。デフォルトはＲＥＬＥＡＳＥイベントの際にボタンと関連付けられたデータを送信することである	0x0a

Fig. 18

【図１９】

フィールド	記述（メッセージヘッダ形式）	バイト
プリアンブル	メッセージのプリアンブル。値は常に０ｘＡＡ ０ｘ５５（ビットシーケンス１０１０１０１０００１０１０１）。これは、イベントマネージャがメッセージの始めを容易に探し出せるようにするためである	2
バージョン	このメッセージが使用するＵＩカードＩＲメッセージプロトコルのバージョン。今回のプロトコルのバージョンはバージョン１である（バージョンフィールドは０ｘ０１）	1
種別	メッセージ種別。これは図２０において与えられる値のうちの１つである	1
リーダＩＤ	メッセージを送信したリーダの１６ビットＩＤ。この番号はリーダのバッテリが交換されるときに変更される疑似ランダムに生成される数である。これはアプリケーションと共に複数のリーダが使用されているときにリーダを区別するのに必要とされる	2
サービス	カード上に格納されたサービス識別子	5
サービス固有	カード上に格納されたサービス固有識別子	3

Fig. 19

【図２０】

名前	記述（メッセージ種別コード）	コード
INSERT	カードがリーダに挿入された	'I'
REMOVE	カードがリーダから抜き取られた	'E'
PRESS	タッチパネルが押下された	'P'
RELEASE	タッチパネルの押下が解除された	'R'
MOVE	押下位置が移動したが押下は解除されていない	'M'
BADCARD	カードが挿入されたが検証に合格しなかった	'B'
LOW_BATT	リーダのバッテリの電力が低下している	'L'

Fig. 20

【図２１】

フィールド	記述（単純なメッセージ形式）	バイト
ヘッダ	図１９により定義されるようなメッセージヘッダ	14
チェックサム	メッセージチェックサム。これはメッセージ中の全バイトの合計である	1
チェックサム'	チェックサムの１の補数	1

Fig. 21

【図２１（ａ）】

フィールド	記述（ＩＮＳＥＲＴメッセージ形式）	バイト
ヘッダ	図１９により定義されるようなメッセージヘッダ	14
長さ	データのバイト数。０の可能性がある	2
データ	カード上のカードデータオブジェクトからのデータ	長さ
チェックサム	メッセージチェックサム。これはメッセージ中の全バイトの合計である	1
チェックサム'	チェックサムの１の補数	1

Fig. 21(a)

【図２２】

フィールド	記述（ＭＯＶＥメッセージ形式）	バイト
ヘッダ	図１９により定義されるようなメッセージヘッダ	14
X	接触位置のＸ座標	1
Y	接触位置のＹ座標	1
チェックサム	メッセージチェックサム。これはメッセージ中の全バイトの合計である	1
チェックサム'	チェックサムの１の補数	1

Fig. 22

【図２３】

フィールド	記述（ＰＲＥＳＳメッセージ形式／ＲＥＬＥＡＳＥメッセージ形式）	バイト
ヘッダ	図１９により定義されるようなメッセージヘッダ	14
X	接触位置のＸ座標	1
Y	接触位置のＹ座標	1
長さ	データのバイト数。０の可能性がある	2
データ	ユーザインタフェース要素と関連付けられるデータ	長さ
チェックサム	メッセージチェックサム。これはメッセージ中の全バイトの合計である	1
	チェックサムの１の補数	1

Fig. 23

【図２４】

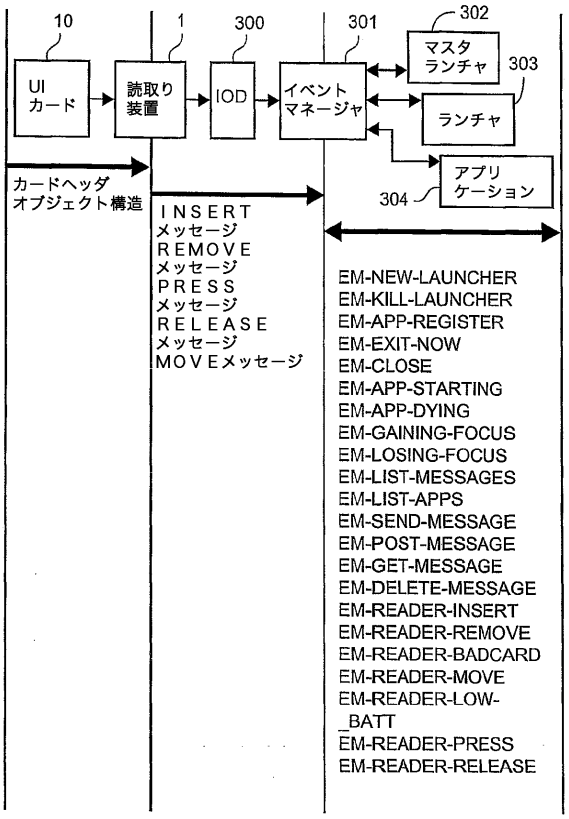


Fig. 24

【図２５】

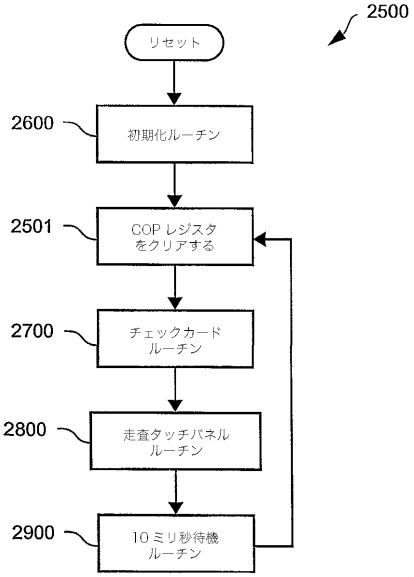


Fig. 25

【図２６】

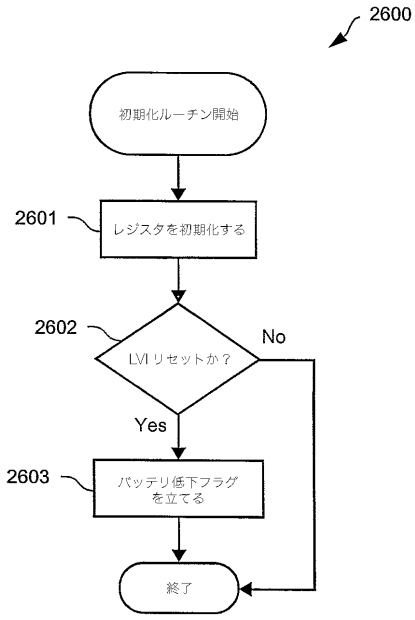


Fig. 26

【図２７】

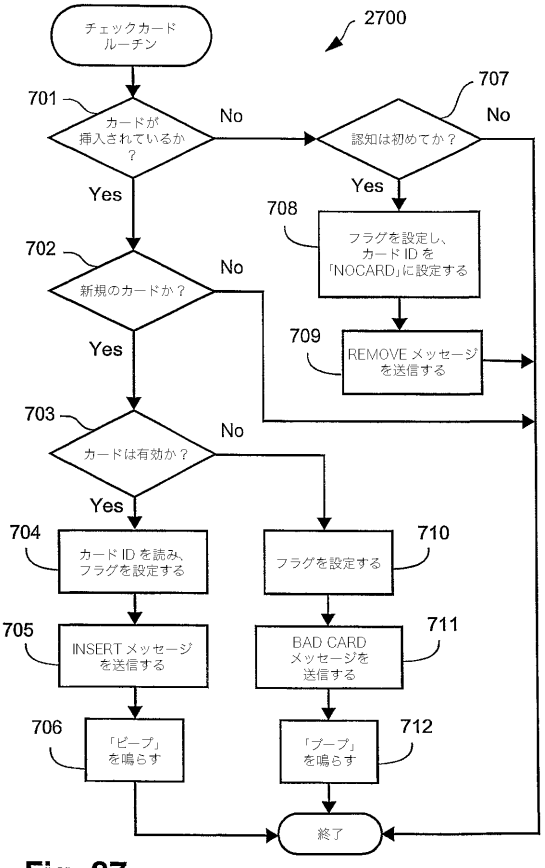


Fig. 27

【図２８】

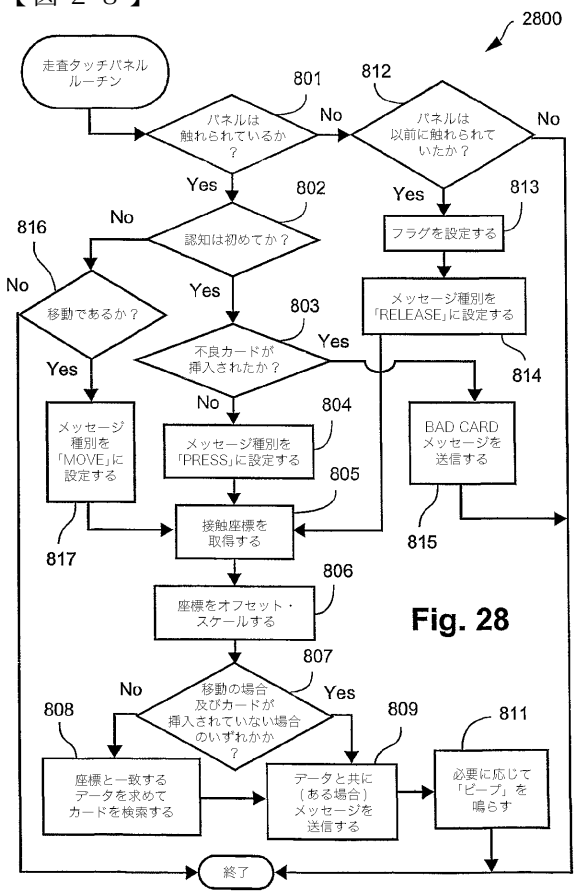


Fig. 28

【図２９】

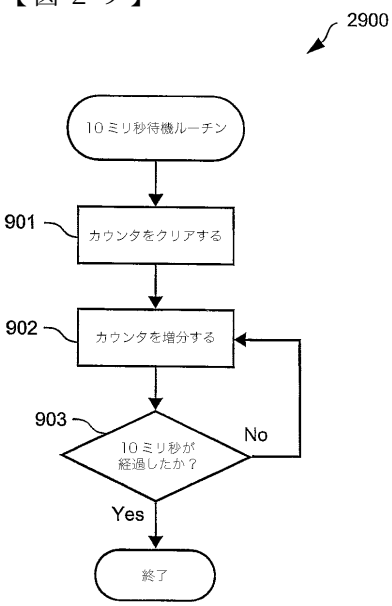


Fig. 29

【図３０】

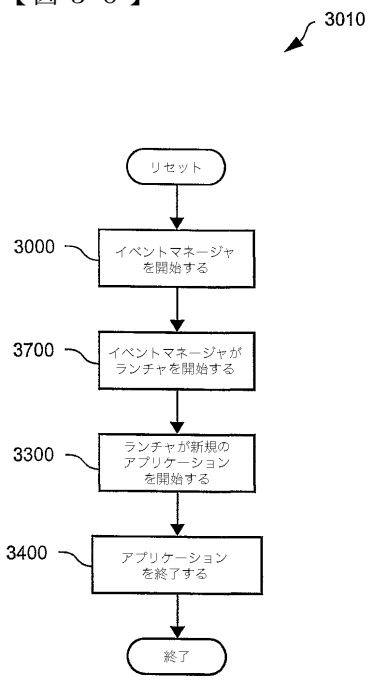


Fig. 30

【図３１】

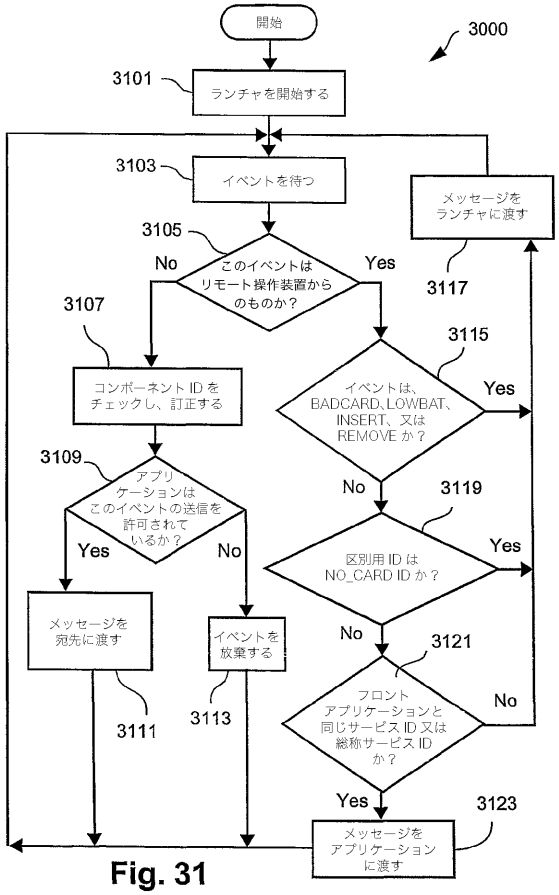


Fig. 31

【図３２】

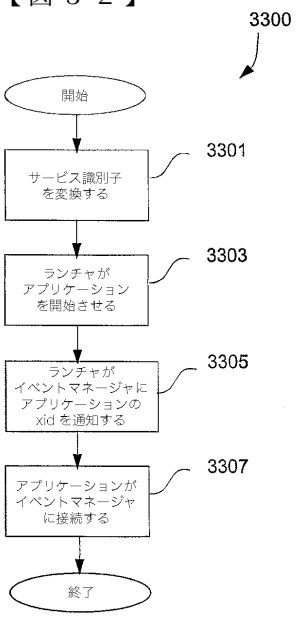


Fig. 32

【図３３】

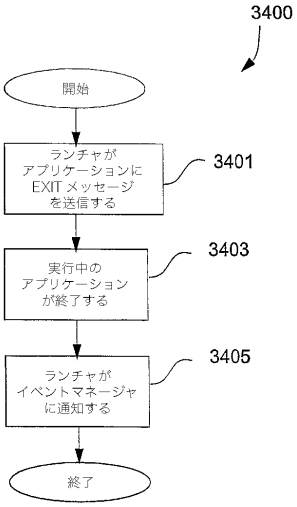


Fig. 33

【図３４】

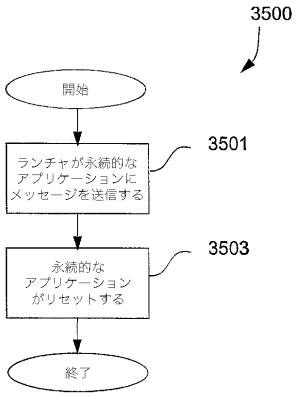


Fig. 34

【図３５】

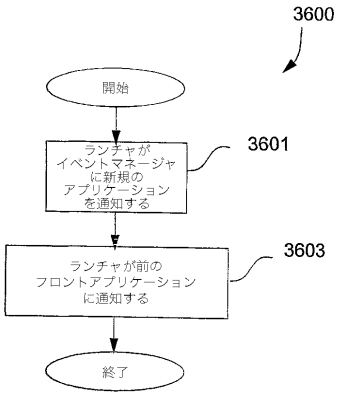
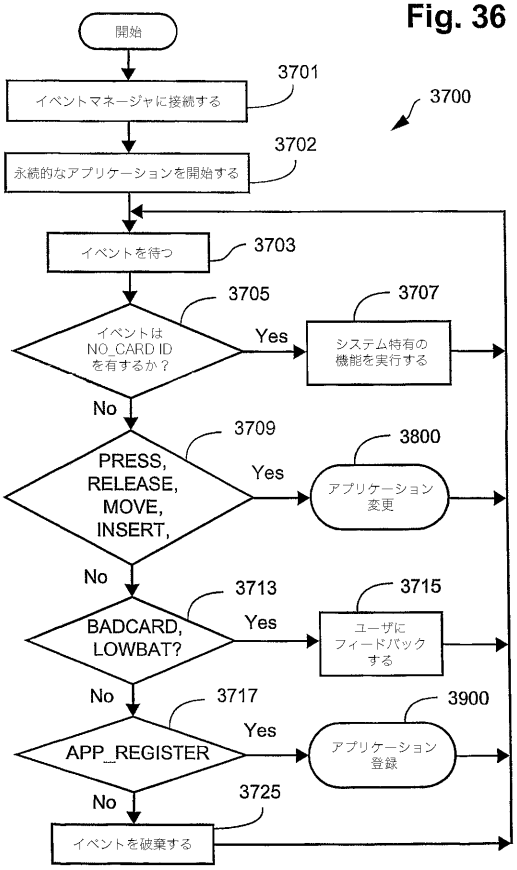


Fig. 35

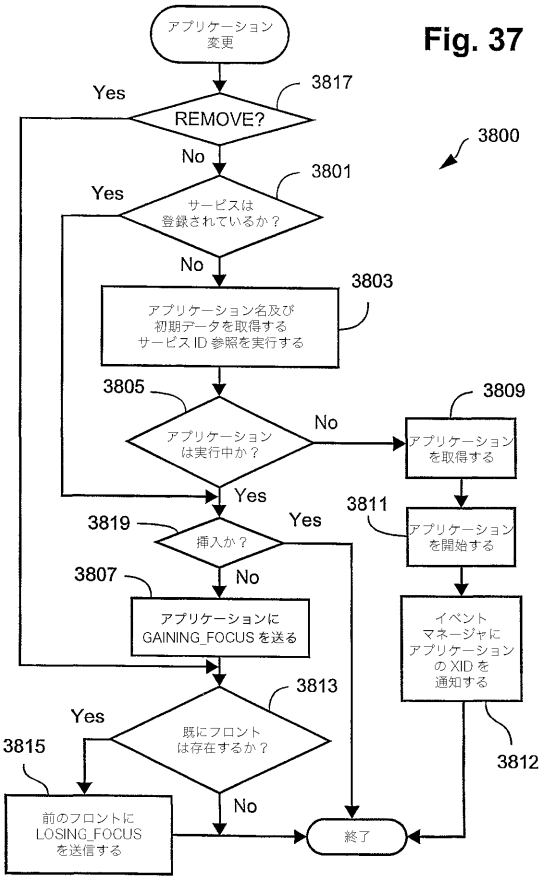
【図３６】

Fig. 36



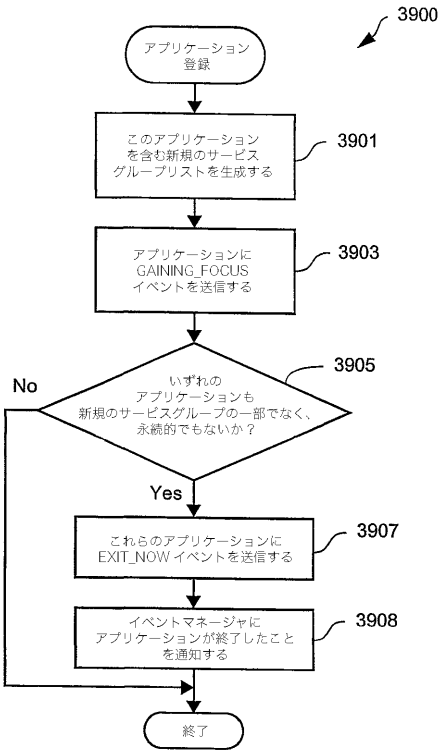
【図３７】

Fig. 37



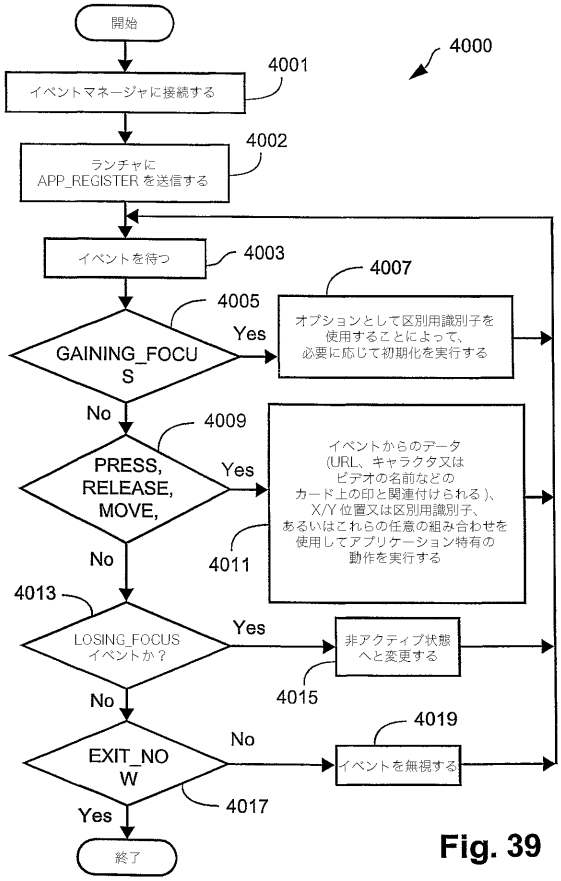
【図３８】

Fig. 38



【図３９】

Fig. 39



【図４０】

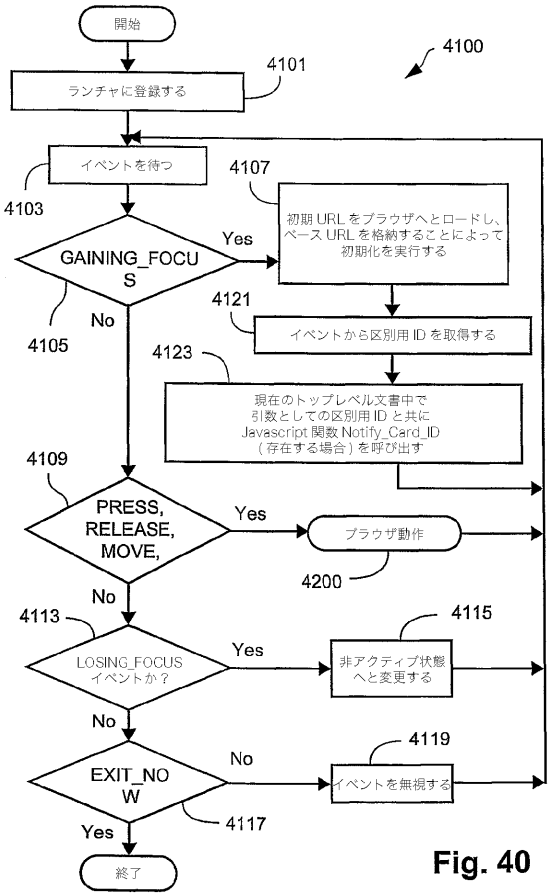
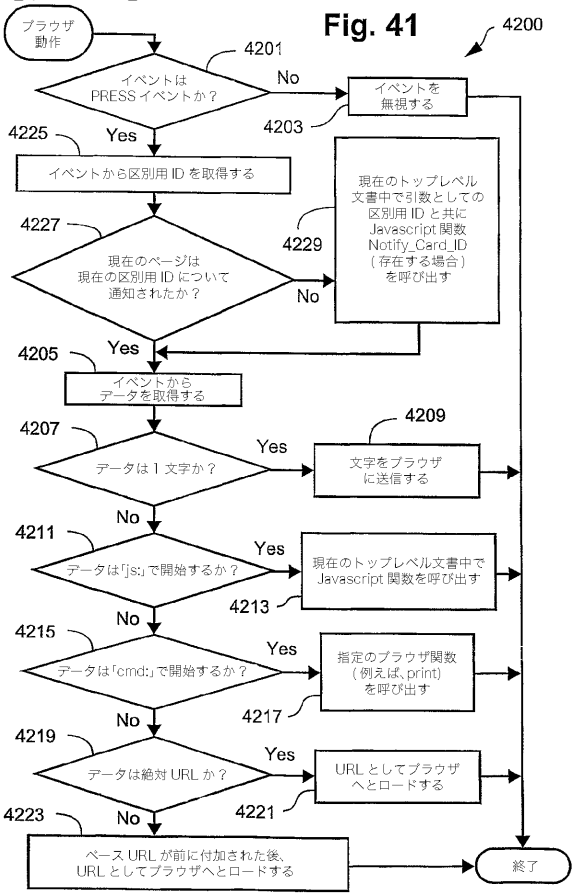


Fig. 40

【図４１】

Fig. 41



【図４２】

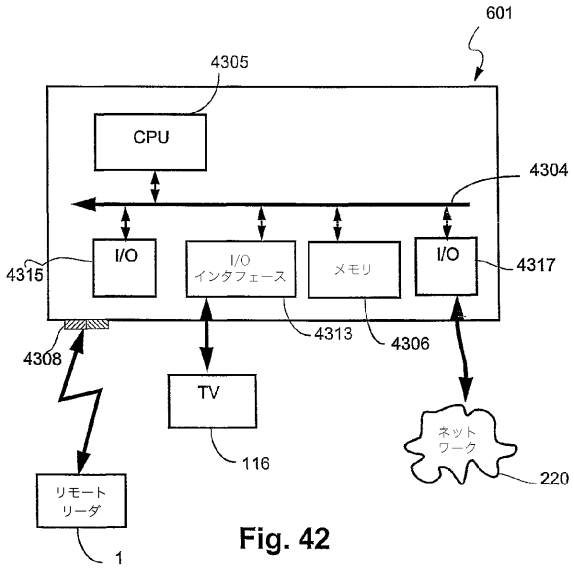


Fig. 42

【図４３】

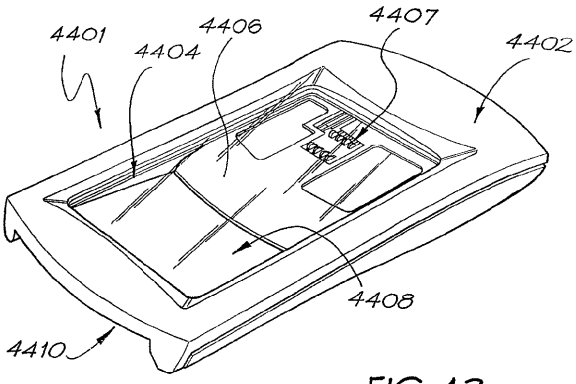


FIG. 43

【図４４】

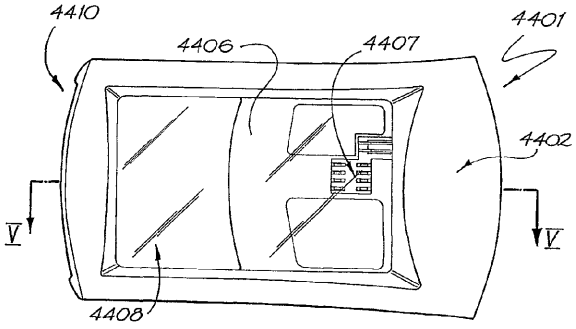


FIG. 44

【図４５】

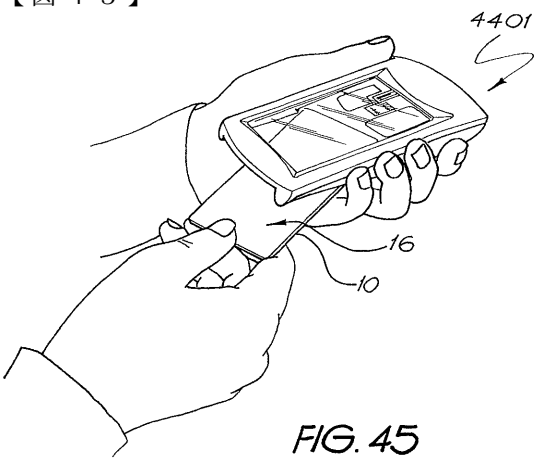


FIG. 45

【図４６】

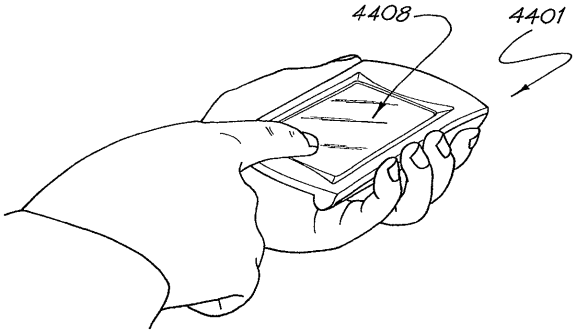


FIG. 46

【図４７（ａ）】

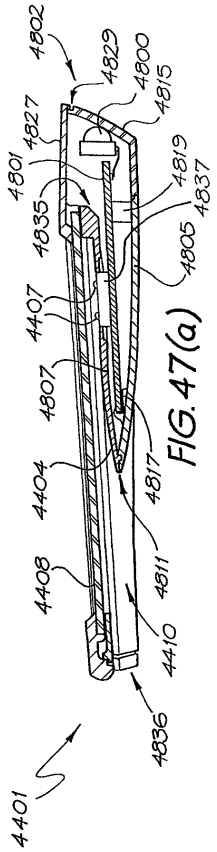


FIG. 47(a)

【図４７（ｂ）】

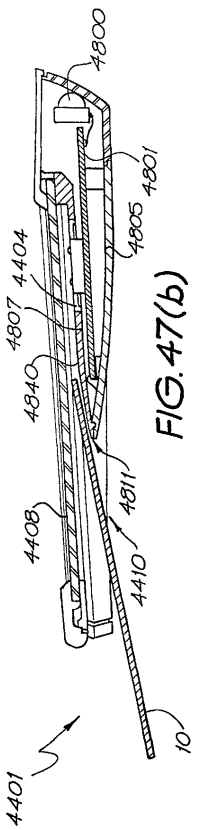


FIG. 47(b)

【図４７（ｃ）】

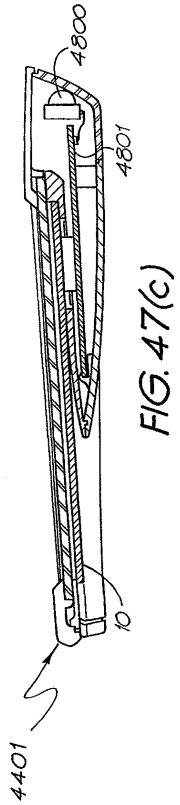


FIG. 47(c)

【図４８】

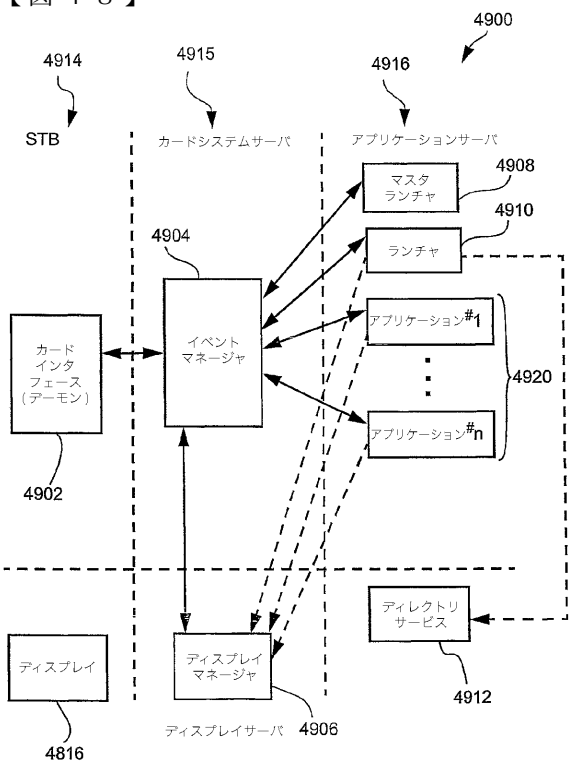


Fig. 48

【図４９】

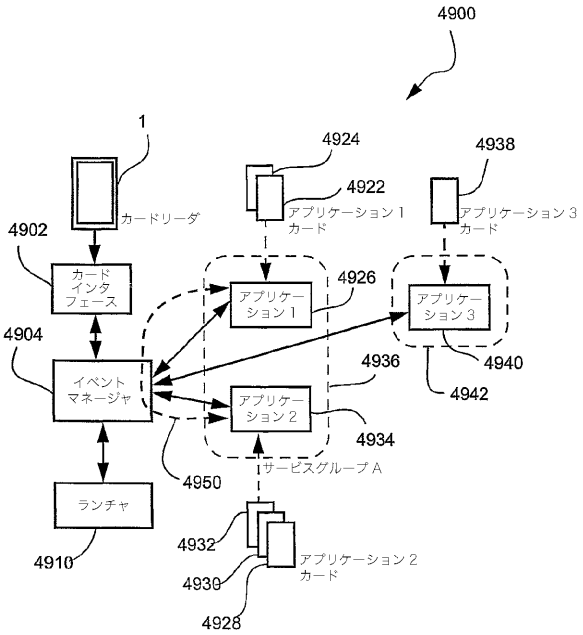


Fig. 49

【図５０】

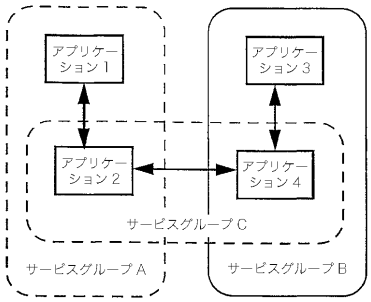


Fig. 50

【図５１Ｂ】

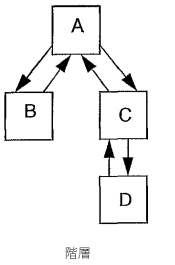


Fig. 51B

【図５１Ａ】

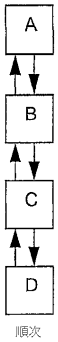


Fig. 51A

【図５１Ｃ】

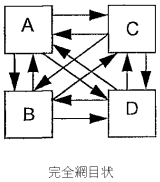


Fig. 51C

【図５２】

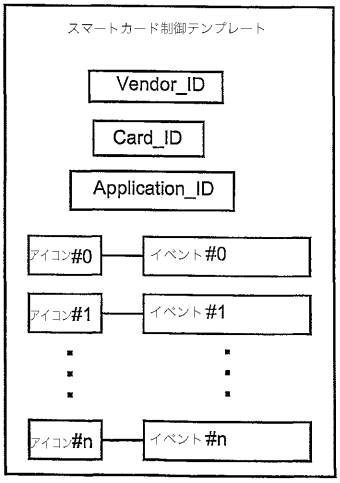


Fig. 52

【図５３Ａ】

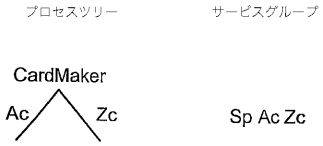


Fig. 53A

【図５３Ｂ】

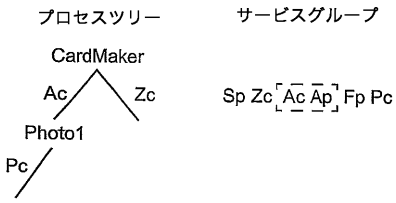


Fig. 53B

【図５３Ｃ】

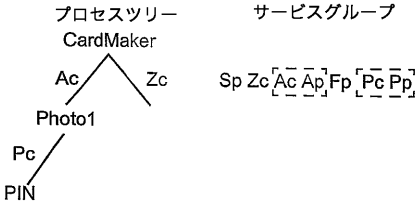


Fig. 53C

【図５３Ｄ】

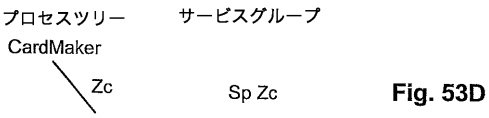


Fig. 53D

【図５３Ｅ】

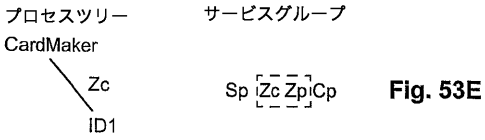


Fig. 53E

【図５４】

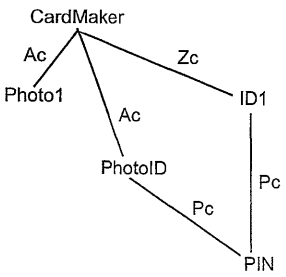


Fig. 54

【図５５】

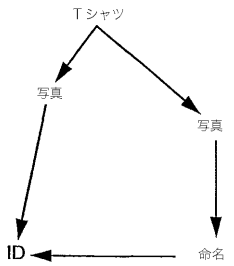


Fig. 55

【図５６（ａ）】

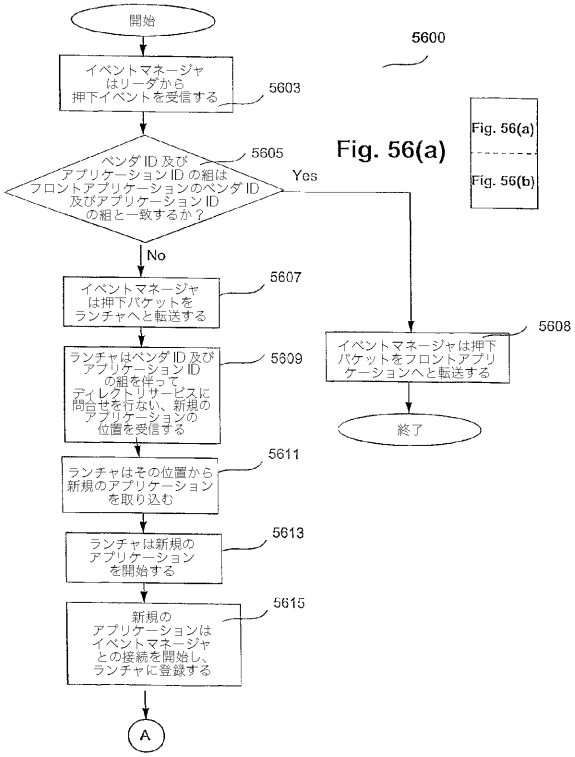


Fig. 56(a)

【図５６（ｂ）】

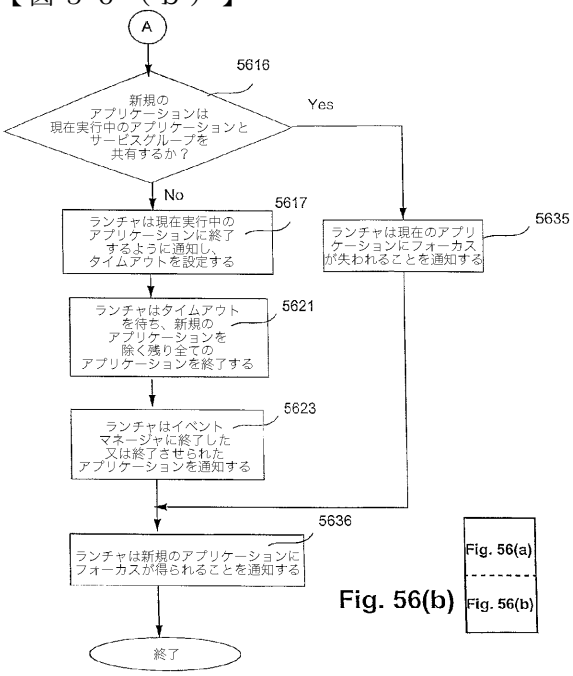


Fig. 56(b)

【図５７】

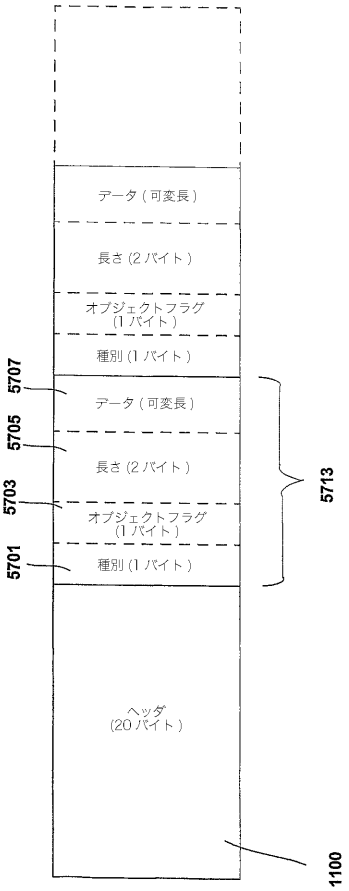


Fig. 57

【図５８】

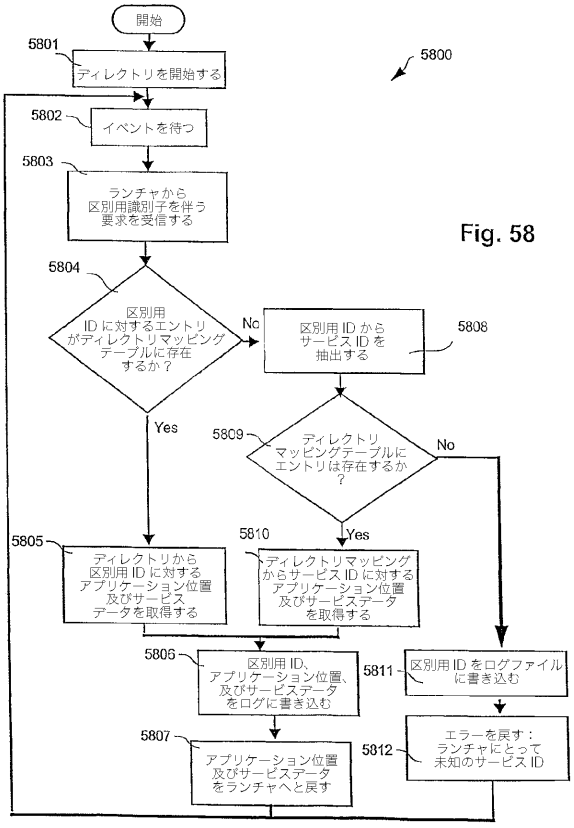


Fig. 58

【国際公開パンフレット】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
21 March 2002 (21.03.2002)

PCT

(10) International Publication Number
WO 02/23320 A1

(51) International Patent Classification[7]: G06F 3/023, G06K 19/07

(21) International Application Number: PCT/AU01/01142

(22) International Filing Date:
12 September 2001 (12.09.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
PR 0073 12 September 2000 (12.09.2000) AU
PR 5593 8 June 2001 (08.06.2001) AU

(71) Applicant *(for all designated States except US)*: **CANON KABUSHIKI KAISHA** [JP/JP]; 30-2, Shimomaruko 3-chome, Ohta-ku, Tokyo 146 (JP).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **YAP, Sue-Ken** [AU/AU]; 19/9 Burley Street, Lane Cove, NSW 2066 (AU). **SMEALLIE, Robert** [AU/AU]; 16 Cypress Street, Normanhurst, NSW 2076 (AU). **SIMPSON-YOUNG, William** [NZ/AU]; 118 Balaclava Road, Eastwood, NSW 2122 (AU). **NEWMAN, Andrew, Timothy, Robert** [AU/AU]; 16 Burton Street, Glebe, NSW 2037 (AU).

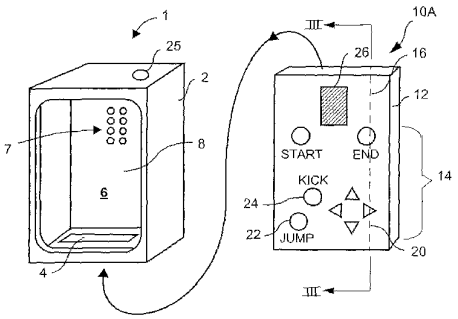
(74) Agent: **SPRUSON & FERGUSON**; GPO BOX 3898, Sydney, NSW 2001 (AU).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF,

[Continued on next page]

(54) Title: CARD READING DEVICE FOR SERVICE ACCESS



(57) Abstract: A read device (1) for reading an interface card (10) is disclosed. The card (10) is configured for insertion into the read device (1). The card (10) comprises indicia (14) formed thereon and a memory (19) having data stored therein for communicating with an external device. The read device (1) comprises a substantially transparent touch sensitive membrane (8) arranged to overlay the interface card (10) upon receipt of the card (10) in the read device (1). The read device (1) also comprises a central processing unit for sending a service identifier, and a specific portion of the data to the external device. The specific data is related to a user selected indicia (14), wherein the external device provides a service identified by the service identifier upon receipt of the data.

WO 02/23320 A1

CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Published:
— *with international search report*

CARD READING DEVICE FOR SERVICE ACCESS

Technical Field of the Invention

The present invention relates to a control template or smart card for use with a remote reader device and, in particular, to a card interface system for providing a service. The invention also relates to a computer program product including a computer readable medium having recorded thereon a computer program for a card interface system.

Background Art

Control pads of various types are known and used across a relatively wide variety of fields. Typically, such pads include one or more keys, buttons or pressure responsive areas which, upon application of suitable pressure by a user, generate a signal which is supplied to associated control circuitry.

Unfortunately, prior art control pads are somewhat limited, in that they only allow for a single arrangement of keys, buttons or pressure sensitive areas. Standard layouts rarely exist in a given field, and so a user is frequently compelled to learn a new layout with each control pad they use. For example many automatic teller machines ("ATMs") and electronic funds transfer at point of sale ("EFTPOS") devices use different layouts, notwithstanding their relatively similar data entry requirements. This can be potentially confusing for a user who must determine, for each control pad, the location of buttons required to be depressed. The problem is exacerbated by the fact that such control pads frequently offer more options than the user is interested in, or even able to use.

Overlay templates for computer keyboards and the like are known. However these are relatively inflexible in terms of design and require a user to correctly configure a system with which the keyboard is associated, each time the overlay is to be used.

One known system involves a smart card reading device intended for the remote control of equipment. Such, for example, allows a television manufacturer, to manufacture a card and supply same together with a remote control housing and a television receiver. A customer is then able to utilise the housing, in conjunction with the card, as a remote control device for the television receiver. In this way the television manufacturer or the radio manufacturer need not manufacture a specific remote control device for their product, but can utilise the remote control housing in conjunction with their specific card. However, the above described concept suffers from the disadvantage in that control data (e.g. PLAY, RECORD, REWIND commands etc.,) stored upon the card, and to be used for controlling an associated apparatus, comes from the manufacturer of the apparatus and is thus limited in its application.

Another known system involves an operating card reading device known as a 'remote commander' used for remote-controlling a video device, audio device etc. The operating card of this known system includes a card identification mechanism for identifying which mode the remote commander is operating in and as such what control data is to be transmitted from the remote commander. The operating card identification mechanism can be in the form of either electrodes/notches formed on side surfaces of the cards or identification information stored within the operating cards. The operating card identification mechanism can be configured in order to enable the remote commander to send commands for either a video tape recorder or for a television receiver, depending on the configuration of the identification mechanism. Again, this known system suffers from the disadvantage in that control data (e.g. PLAY, RECORD, REWIND commands etc.,) to be used for controlling the video tape recorder or television, comes from the manufacturer of the apparatus and is thus limited in its application. Further, the operating card identification mechanism must be configured each time the user wishes to change the

apparatus to be controlled and is restricted to the operating card such that the identification mechanism can not be used to interact with the video device, audio device etc., to be controlled.

Still another known smart card system includes optics for receiving information from a television channel and a modem for providing real-time two way communication with an application running on a remote service provider. This known smart card system is used for remote service transactions such as an existing home shopping application. In accordance with this known system, information including home shopping program information, an item name, an item description, an item price and item commercial and programming re-run times, can be down-loaded to a smart card. The smart card can then use the access information along with the modem of the smart card to automatically dial a home shopping program automated service computer to place an order. However, again this system is limited in its application since the access information must be down-loaded to the smart card each time the smart card is to be used to purchase an item and can only be used to purchase the item specified by the item name and description.

The above-described systems all lack flexibility and are all limited in their respective applications. These systems are all used with pre-running applications and there is no interaction with the application other than that specified by the manufacturer.

Summary of the Invention

It is an object of the present invention to substantially overcome, or at least ameliorate, one or more disadvantages of existing arrangements.

According to one aspect of the present invention there is provided a read device for reading an interface card, said card being configured for insertion into said read device, and wherein said card comprises indicia formed thereon and a memory having

data stored therein for communicating with an external device, said read device comprising:

a substantially transparent touch sensitive membrane arranged to overlay said interface card upon receipt of said card in said read device,

a central processing unit for sending a service identifier, and a specific portion of said data to said external device, said specific data being related to a user selected indicia, wherein said external device provides a service identified by the service identifier upon receipt of said data.

According to another aspect of the present invention there is provided a program to be executed in a read device having a receptacle to receive an interface card, said card comprising a substrate and indicia formed on said substrate, said program comprising:

code for reading a service identifier and a specific portion of data stored within a memory of said card, said specific data being related to a user selected indicia; and

code for a sending said service identifier and said specific data to an external device, whereby a service is provided by said external device, said service being identified by the service identifier.

According to still another aspect of the present invention there is provided a data processing method for a read device that has a receptacle to receive an interface card, said card comprising a substrate with indicia formed thereon, said method comprising the steps of:

reading a service identifier and a specific portion of data stored within a memory of said card, said specific data being related to a user selected indicia; and

sending said service identifier and said specific data to an external device, whereby a service is provided by said external device, said service being identified by the service identifier.

According to still another aspect of the present invention there is provided a read device for reading an interface card, said card comprising indicia formed thereon and a memory having data stored therein, and wherein said card is configured for insertion into said read device, said read device comprising:

read means for reading a service identifier and a specific portion of data stored in the card, said specific data being related to user selected indicia; and

send means for sending said service identifier and said data to an external device, wherein said external device provides a service, said service being identified by the service identifier.

According to still another aspect of the present invention there is provided a card interface system comprising an interface card and a read device, said card being configured for insertion into said read device, and said card comprising at least a substrate with indicia formed thereon, said read device comprising a receptacle to receive said interface card said system comprising:

a memory on said card for storing a service identifier and data related to a user selected indicia; and

a central processing unit integrally formed within said read device for sending said service identifier and said data to an external device in order that a user of said card receives a service from said external device, said service being identified by the service identifier.

According to still another aspect of the present invention there is provided an electronic card reader for reading an electronic card, said electronic card having at least one indicia formed thereon and an electronic memory having data stored therein for controlling data controlled equipment, said electronic reader comprising:

a touch sensitive substantially transparent membrane having an upper surface configured to be depressible by a user of said reader;

a receptacle shaped to receive said electronic card, wherein said electronic card received therein and said indicia can be viewed through said touch sensitive membrane; and

an electronic circuit coupled to said membrane to read a specific portion of said data from said memory according to depression of said membrane associated with a specific one of said indicia, and for sending said specific data together with a service identifier to said data controlled equipment, said data controlled equipment providing a function controlled by receipt of said specific data, wherein said function is associated with said service identifier.

According to still another aspect of the present invention there is provided a read device for a interface card, said data comprising a substrate having at least one indicia formed thereon, said read device comprising:

a receptacle shaped to receive said interface card;

a substantially transparent touch sensitive membrane arranged to overlay said interface card upon receipt of said interface card in said receptacle such that said indicia can be viewed through said touch sensitive membrane; and

an electronic circuit for coupling to a memory component of said interface card and for reading data related to one of said indicia selected by a user of said read device via said membrane, wherein said electronic circuit is configured to send said data together with a service identifier to an external device, whereby said external device provides a service associated with said service identifier upon receipt of said read data.

According to still another aspect of the present invention there is provided a read device having a receptacle to receive an interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for determining a validity of said card that was inserted into said read device and sending to an external device a service identifier and data stored in the card, said data being related to a user pressed indicia, in order that a card user receives a service identified by the service identifier according to said pressed indicia.

According to still another aspect of the present invention there is provided a read device having a substantially transparent touch sensitive membrane arranged to overlay a detachable interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for determining a validity of said card that was inserted into said read device and sending to an external device a service identifier and user press coordinates pressed on said touch sensitive membrane to select said indicia, in order that a card user receives a service identified by the service identifier according to said pressed indicia.

According to still another aspect of the present invention there is provided a read device having a substantially transparent touch sensitive membrane arranged to overlay a detachable interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for distinguishing a identifier stored in said card and sending to an external device a service identifier and user press coordinates pressed on said touch sensitive membrane to select said indicia or a service identifier and user press coordinates pressed on said touch sensitive membrane to select said indicia, based on said

result of said distinction, in order that a card user receives a service identified by the service identifier according to said pressed indicia.

According to still another aspect of the present invention there is provided a read device having a substantially transparent touch sensitive membrane arranged to overlay a detachable interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for detecting a user press touch on said on said touch sensitive membrane and sending a touch coordinates of said user press touch to an external device, wherein said touch coordinates is used as a cursor information.

According to still another aspect of the present invention there is provided a read device having a receptacle to receive an interface card , said device comprising:

a central processing unit for reading an information that affects functions that said card perform in said read device, performing the functions based on said information.

According to still another aspect of the present invention there is provided a read device having a receptacle to receive an interface card, said device comprising:

a central processing unit for generating a session identifier identifying a current session of a card insertion, which is a number that is incremented each time a card is inserted into said read device and sending said session identifier to a external device, in order to determining a validity of said inserted card in said external device.

Other aspects of the invention are also disclosed.

Brief Description of the Drawings

One or more embodiments of the present invention will now be described with reference to the drawings, in which:

Fig. 1 is a perspective view of a read device and an associated card;

Fig. 2 is a perspective view of an opposite side of the card shown in Fig. 1;

Fig. 3 is a longitudinal cross-sectional view of the card shown in Fig. 1 taken along the line III - III;

Figs. 4 and 5 are perspective views of the rear face of alternative arrangements of cards to the card shown in Fig. 1;

Fig. 6(a) shows a hardware architecture of a card interface system;

Fig. 6(b) shows another hardware architecture of a card interface system ;

Fig. 7 is a schematic block diagram of the general-purpose computer of Figs. 6(a) and 6(b);

Fig. 8 is a schematic block diagram representation of a card interface system architecture;

Fig. 9 is a schematic block diagram representation of a card interface system;

Fig. 10 is a schematic block diagram showing the internal configuration of the reader of Fig. 1;

Fig. 11 shows the data structure of a card header as stored in the card of Fig. 1;

Fig.12 shows a description of each of the fields of the header of Fig. 11;

Fig. 13 shows a description of each of the flags contained in the header of Fig. 11;

Fig. 14 shows a description of each of the fields of the object structure for the card of Fig. 1;

Fig. 15 shows a description of the flag for the object structure of Fig. 14;

Fig. 16 shows a description of each of the object types for the object structure of Fig. 14;

Fig. 17 shows a description of each of the fields for a user Interface Object Structure according to the object structure of Fig. 14;

Fig. 18 shows a description for each of the user Interface object flags according to the object structure of Fig. 14;

Fig. 19 shows the format of a message header that is sent from the reader of Fig. 1;

Fig. 20 shows a table listing message event types for the header of Fig. 19;

Fig. 21 shows the format of a simple message;

Fig. 21(a) shows the format of an INSERT message;

Fig. 22 shows the format of a MOVE message;

Fig. 23 shows the format of PRESS and RELEASE messages;

Fig. 24 is a data flow diagram showing the flow of messages within the system of Fig. 6;

Fig. 25 is a flow diagram showing a read method performed by the reader of Fig. 1;

Fig. 26 is a flow diagram showing a method of initialising the system of Fig. 6, performed during the method of Fig. 25;

Fig. 27 is a flow diagram showing a method of checking the card of Fig. 1, performed during the method of Fig. 25;

Fig. 28 is a flow diagram showing a method of scanning the touch panel of the reader of Fig.1, performed during the method of Fig. 25;

Fig. 29 is a flow diagram showing a wait 10ms method, performed during the method of Fig. 25;

Fig. 30 is a flow diagram showing an overview of events of the system of Fig. 6;

Fig. 31 is a flow diagram showing processes performed by the event manager during the process of Fig. 30;

Fig. 32 is a flow diagram showing a method for starting a new application, performed during the process of Fig. 30;

Fig. 33 is a flow diagram showing a method of ending an application performed during the process of Fig. 30;

Fig. 34 is a flow diagram showing a method of closing a current session for a persistent application;

Fig. 35 is a flow diagram showing a method for performing a focus change;

Fig. 36 is a flow diagram showing an overview of a method performed by the launcher;

Fig. 37 is a flow diagram showing a method of changing an application, performed during the method of Fig. 36;

Fig. 38 is a flow diagram showing a method of registering a new application, performed during the method of Fig. 36;

Fig. 39 is a flow diagram showing a method performed by an application when receiving events from the launcher;

Fig. 40 is a flow diagram showing a method performed by the browser controller application when receiving events from the launcher;

Fig. 41 is a flow diagram showing a browser application method;

Fig. 42 is schematic block diagram showing the set top box of the system 600 in more detail;

Fig. 43 is a perspective view of a "bottom-entry" reader;

Fig. 44 is a plan view of the reader of Fig. 43;

Fig. 45 shows a user inserting a card into the reader of Fig. 43;

Fig. 46 shows a user operating the reader of Fig. 43 after a card has been fully inserted;

Fig. 47(a) is a longitudinal cross-sectional view along the line V-V of Fig. 44;

Fig. 47(b) is a view similar to Fig. 47(a), with a card partially inserted into the receptacle of the reader;

Fig. 47(c) is a view similar to Fig. 47(a), with a card fully inserted into the template receptacle of the reader.

Fig. 48 is a schematic block diagram representation of a further card interface system architecture;

Fig. 49 is a schematic block diagram representation showing the relationships between cards and applications;

Fig. 50 illustrates the relationships between applications and service groups;

Figs. 51A to 51C illustrates different types of card orderings within the architecture of Fig. 48;

Fig. 52 illustrates the control template data stored in the smart card for the architecture of Fig. 48;

Figs. 53A to 53E illustrate an example of a multi-card application structure;

Fig. 54 shows an alternative approach to achieve the end of Figs. 53A to 53E;

Fig. 55 shows a directed graph representation of a multi-application method;

Fig. 56 shows a method of starting an application;

Fig. 57 shows one or more object structures following the card header of Fig. 11; and

Fig. 58 is a flow diagram, showing an overview of the process performed by the directory service of Fig. 8.

Detailed Description including Best Mode

Where reference is made in any one or more of the accompanying drawings to steps and/or features, which have the same reference numerals, those steps and/or features

have for the purposes of this description the same function(s) or operation(s), unless the contrary intention appears.

The embodiments disclosed herein have been developed primarily for use with remote control systems, automatic tellers, video game controllers and network access, and will be described hereinafter with reference to these and other applications. However, it will be appreciated that the invention is not limited to these fields of use.

For ease of explanation the following description has been divided into Sections 1.0 to 13.0, each section having associated subsections.

1.0 CARD INTERFACE SYSTEM OVERVIEW

Fig. 1 shows a remote reader 1, having a housing 2, which defines a card receptacle 4 and a viewing area 6. Data reading means are provided in the form of exposed electrical contacts 7 and associated control circuitry (not shown). The remote reader 1 also includes sensor means in the form of a substantially transparent pressure sensitive membrane forming a touch panel 8 covering the viewing area 6. The remote reader 1 disclosed herein has been described with a substantially transparent pressure sensitive membrane forming the touch panel 8, however it will be appreciated by one skilled in the art that alternative technology can be used as a substantially transparent touch panel. For example, the touch panel can be resistive or temperature sensitive. The remote reader 1 is configured for use with a user interface card, which, in the cards shown in Figs. 1 to 3, takes the form of an electronic smart card 10A. The smart card 10A includes a laminar substrate 12 with various control indicia 14 in the form of a four way directional controller 20, a "jump button" 22, a "kick button" 24, a "start button" and an "end button" printed on an upper face 16 thereof. Other non-control indicia, such as promotional or instructional material, can be printed alongside the control indicia. For

example, advertising material 26 can be printed on the front face of the smart card 10A or on a reverse face 27 of the card 10A, as seen in Fig. 2.

As seen in Fig. 3, the smart card 10A includes storage means in the form of an on-board memory chip 19 for data associated with the control indicia. The smart card 10A also includes electrical data contacts 18 connected to the on-board memory chip 19 corresponding with the exposed contacts 7 on the remote reader 1.

As seen in Fig. 3, the upper face 16 may be formed by an adhesive label 60 upon which are printed control indicia 14, in this case corresponding to the "End Button" and the Right arrow "button" of the directional controller 20. The label 60 is affixed to the laminar substrate 12. A home user can print a suitable label for use with a particular smart card 10A by using a printer, such as a colour BUBBLEJET ™ printer manufactured by Canon, Inc. Alternatively, the control indicia 14 can be printed directly onto the laminar substrate or separate adhesive labels can be used for each of the control indicia.

In use, the smart card 10A is inserted into the card receptacle 4, such that the pressure sensitive touch panel 8 covers the upper face 16 of the smart card 10A. In this position, the control indicia are visible within the viewing area 6 through the transparent pressure sensitive touch panel 8.

The exposed contacts 7 and associated circuitry of the reader 1 are configured to read the stored data associated with the control indicia 14 from the memory chip 19, either automatically upon insertion of the smart card 10A into the control template receptacle 4, or selectively in response to a signal from the remote reader 1. The signal can, for example, be transmitted to the smart card 10A via the exposed contacts 7 and data contacts 18.

Once the data associated with the control indicia 14 has been read, a user can press areas of the pressure sensitive touch panel 8 on or over the underlying control

indicia 14. By sensing the pressure on the pressure sensitive touch panel 8 and referring to the stored data, the remote reader 1 can deduce which of the control indicia 14 the user has selected. For example, if the user places pressure on the pressure sensitive touch panel 8 adjacent the "kick button" 24, the remote reader 1 is configured to assess the position at which the pressure was applied, refer to the stored data, and determine that the "kick" button 24 was selected. This information can then be used to control an external device, for example, an associated video game console (of conventional construction and not shown). It will be appreciated from above that the control indicia 14 are not, in fact buttons. Rather, the control indicia 14 are user selectable features which, by virtue of their corresponding association with the mapping data and the function of the touch panel 8, operate to emulate buttons traditionally associated with remote control devices.

In one advantageous implementation, the remote reader 1 includes a transmitter (of conventional type and not shown), such as an infra-red (IR) transmitter or radio frequency (RF) transmitter, for transmitting information in relation to indicia selected by the user. As seen in Fig. 1, the remote reader 1 incorporates an IR transmitter having an IR light emitting diode (LED) 25. Upon selection of one of the control indicia 14, the remote reader 1 causes information related to the selection to be transmitted to a remote console (not shown in Fig.1) where a corresponding IR or RF receiver can detect and decode the information for use in controlling some function, such as a game being played by a user of the reader 1.

Any suitable transmission method can be used to communicate information from the remote reader 1 to the remote console, including direct hard wiring. Moreover, the remote console itself can incorporate a transmitter, and the remote reader 1 a receiver, for communication in an opposite direction to that already described. The communication from the remote console to the remote reader 1 can include, for example, handshaking

data, setup information, or any other form of information desired to be transferred from the remote console to the remote reader 1.

Turning to Fig. 4, there is shown a control card 10B. The control card 10B includes a laminar substrate 12, which bears control indicia (not illustrated). In the control card 10B the storage means takes the form of a magnetic strip 29 formed along an edge 28 of the reverse face 27 of the control card 10B. The stored data associated with the control indicia may be stored on the magnetic strip 29 in a conventional manner. A corresponding reader (not shown) for this arrangement includes a magnetic read head positioned at or adjacent an entrance to the corresponding control template receptacle. As the control card 10B is slid into the card receptacle, the stored data is automatically read from the magnetic strip 29 by the magnetic read head. The reader 1 may then be operated in a manner corresponding to the card 10A of Fig. 1.

Fig. 5 shows another card in the form of a control card 10C, in which the storage means takes the form of machine-readable indicia. In the card 10C of Fig. 5, the machine readable indicia takes the form of a barcode 36 formed along an edge 38 of the reverse face 27 of the card 10C. The stored data is suitably encoded, and then printed in the position shown. A corresponding controller (not shown) for the card 10C of Fig. 5 includes an optical read head positioned at or adjacent an entrance to the associated control template receptacle. As the card 10C is slid into the control receptacle, the stored data is automatically read from the barcode 36 by the optical read head. Alternatively, the barcode can be scanned using a barcode reader associated with the reader immediately prior to inserting the card 10C, or scanned by an internal barcode reader scanner once the card 10C has completely been inserted. The card 10C may then be operated in a manner again corresponding to the card 10A of Fig. 1. It will be appreciated that the position, orientation and encoding of the barcode can be altered to suit a particular application.

Moreover, any other form of machine readable indicia can be used, including embossed machine-readable figures, printed alpha-numeric characters, punched or otherwise formed cut outs, optical or magneto optical indicia, two dimensional bar codes. Further, the storage means can be situated on the same side of the card 10A or 10B or 10C as the control indicia.

Fig. 6(a) shows a hardware architecture of a card interface system 600A. In the system 600A, the remote reader 1 is hard wired to a personal computer system 100 via a communications cable 3. Alternatively, instead of being hardwired, a radio frequency or IR transceiver 106 can be used to communicate with the remote reader 1. The personal computer system 100 includes a screen 101 and a computer module 102. The computer system 100 will be explained in more detail below with reference to Fig. 7. A keyboard 104 and mouse 203 are also provided.

The system 600A includes a smart card 10D which is of similar configuration to the smart card 10A described above. The smart card 10D is programmable and can be created or customised by a third party, which in this case can be a party other than the manufacturer of the card 10D and/or card reader 1. The third party can be the ultimate user of the smart card 10D itself, or may be an intermediary between the manufacturer and user. In accordance with the system 600A of Fig. 6(a), the smart card 10D can be programmed and customised for one touch operation to communicate with the computer 100 and obtain a service over a network 220, such as the Internet. The computer 100 operates to interpret signals sent via the communications cable 3 from the remote reader 1, according to a specific protocol, which will be described in detail below. The computer 100 performs the selected function according to touched control indicia, and can be configured to communicate data over the network 220. In this manner the

computer 100 can permit access to applications and/or data stored on remote server computers 150, 152 and appropriate reproduction on the display device 101.

Fig. 6(b) shows a hardware architecture of a card interface system 600B. In the system 600B, the remote reader 1 can be programmed for obtaining a service locally at a set top box 601, that couples to an output interface, which in this example takes the form of an audio-visual output device 116, such as a digital television set. The set-top box 601 operates to interpret signals 112 received from the remote reader 1, which may be electrical, radio frequency, or infra-red (IR), and according to a specific protocol which will be described in detail below. The set top box 601 can be configured to perform the selected function according to touched control indicia and permit appropriate reproduction on the output device 116. Alternatively, the set top box 601 can be configured to convert the signals 112 to a form suitable for communication and cause appropriate transmission to the computer 100. The computer 100 can then perform the selected function according to the control indicia, and provide data to the set-top box 601 to permit appropriate reproduction on the output device 116. The set top box 601 will be explained in more detail below with reference to Fig. 42.

In one application of the system 600B, the smart card 10D can be programmed for obtaining a service both remotely and locally. For instance, the smart card 10D can be programmed to retrieve an application and/or data stored on remote server computers 150, 152, via the network 220, and to load the application or data on to the set top box 601. The latter card can be alternatively programmed to obtain a service from the loaded application on the set top box 601.

Unless referred to specifically, the systems 600A and 600B will be hereinafter generically referred to as the system 600. Further, unless referred to specifically, the

smart cards 10A, 10B, 10C and 10D will be hereinafter generically referred to as the smart card 10.

Fig. 7 shows the general-purpose computer system 100 of the system 600, which can be used to run the card interface system and to run software applications for programming the smart card 10. The computer system 100 includes a computer module 102, input devices such as a keyboard 104 and mouse 203, output devices including the printer (not shown) and the display device 101. A Modulator-Demodulator (Modem) transceiver device 216 is used by the computer module 102 for communicating to and from the communications network 220, for example connectable via a telephone line 221 or other functional medium. The modem 216 can be used to obtain access to the Internet, and other network systems, such as a Local Area Network (LAN) or a Wide Area Network (WAN).

The computer module 102 typically includes at least one central processing unit (CPU) 205, a memory unit 206, for example formed from semiconductor random access memory (RAM) and read only memory (ROM), input/output (I/O) interfaces including a video interface 207, and an I/O interface 213 for the keyboard 104 and mouse 203, a write device 215, and an interface 208 for the modem 216. A storage device 209 is provided and typically includes a hard disk drive 210 and a floppy disk drive 211. A magnetic tape drive (not illustrated) is also able to be used. A CD-ROM drive 212 is typically provided as a non-volatile source of data. The components 205 to 213 of the computer module 201, typically communicate via an interconnected bus 204 and in a manner, which results in a conventional mode of operation of the computer system 102 known to those in the relevant art. Examples of computers on which the arrangement described herein can be practised include IBM-computers and compatibles, Sun Sparcstations or alike computer system evolved therefrom.

Typically, the software programs of the system 600 are resident on the hard disk drive 210 and read and controlled in their execution by the CPU 205. Intermediate storage of the software application programs and any data fetched from the network 220 may be accomplished using the semiconductor memory 206, possibly in concert with the hard disk drive 210. In some instances, the application programs can be supplied to the user encoded on a CD-ROM or floppy disk and read via the corresponding drive 212 or 211, or alternatively may be read by the user from the network 220 via the modem device 216. Still further, the software can also be loaded into the computer system 102 from other computer readable medium including magnetic tape, ROM or integrated circuits, a magneto-optical disk, a radio or infra-red transmission channel between the computer module 102 and another device, a computer readable card such as a smart card, a computer PCMCIA card, and the Internet and Intranets including email transmissions and information recorded on Websites and the like. The foregoing is merely exemplary of relevant computer readable media. Other computer readable media are able to be practised without departing from the scope of the invention defined by the appended claims.

The smart card 10 can be programmed by means of a write device 215 coupled to the I/O interface 213 of the computer module 102. The write device 215 can have the capability of writing data to the memory on the smart card 10. Preferably, the write device 215 also has the capability of printing graphics on the top surface of the smart card 10. The write device 215 can also have a function reading data from the memory on the smart card 10. Initially, the user inserts the smart card 10 into the write device 215. The user then enters the required data via the keyboard 104 of the general purpose computer 102 and a software application writes this data to the smart card memory via the write

device 215. If the stored data is encoded for optical decoding such as using a barcode, the write device can print the encoded data onto the smart card 10.

Fig. 42 shows the set top box 601 of the system 600, which can be used to interpret signals 112 received from the remote reader 1. The set top box 601 in some implementations essentially is a scaled version of the computer module 102. The set top box 601 typically includes at least one CPU unit 4305, a memory unit 4306, for example formed from semiconductor random access memory (RAM) and read only memory (ROM), and input/output (I/O) interfaces including at least an I/O interface 4313 for the digital television 116, an I/O interface 4315 having an IR transceiver 4308 for receiving and transmitting the signals 112, and an interface 4317 for coupling to the network 220. The components 4305, 4306, 4313, 4315 and 4317 of the set top box 601, typically communicate via an interconnected bus 4304 and in a manner which results in a conventional mode of operation. Intermediate storage of any data received from the remote reader 1 or network 220 may be accomplished using the semiconductor memory 4306. Alternatively, the set top box can include a storage device (not shown) similar to the storage device 209.

The card interface system 600 will now be explained in more detail in the following paragraphs.

2.0 CARD INTERFACE SYSTEM SOFTWARE ARCHITECTURE

2.1 Software Architecture Layout

A software architecture 200 for the hardware architectures depicted by the system 600, is generally illustrated in Fig. 8. The architecture 200 can be divided into several distinct process components and one class of process. The distinct processes include an I/O interface 300, which may be colloquially called an "I/O daemon" 300, an event manager 301, a display manager 306, an (application) launcher 303 and a directory

service 311. The class of process is formed by one or more applications 304. In the architecture 200 described herein , there exists one I/O daemon 300, one event manager 301, one display manager 306 and one launcher 303 for every smart card remote connection, usually formed by the set-top box 601, and one master launcher (not shown) for each computer 100 (e.g. the server computers 150, 152) that is running the launchers 303, and at least one directory service 311 for all systems. The Directory service 311, is queried by the launcher 303 to translate service data into a Resource Locater (eg. URL) that indicates a name or location of a service or the location or name of an application 304 to be used for the service.

In this form, the architecture 200 can be physically separated into six distinct parts 101, 307, 309, 312, 313 and 601 as shown by the dashed lines in Fig. 8, each of which can be run on physically separate computing devices. Communication between each of the parts of the system 600 is performed using Transport Control Protocol/ Internet Protocol (TCP/IP) streams. Alternatively, each of the parts 101, 307, 309, 312, 313 and 601 can be run on the same machine.

In the system 600A of Fig. 6(a), all of the process components 300, 301, 303, 304 and 306 can be run on the computer 100. The event manager 301, the launcher 303 and display manager 306 are preferably all integrated into one executable program which is stored in the hard disk 209 of the computer 100 and can be read and controlled in its execution by the CPU 205. The directory service 311 runs on the same computer 100 or on a different computer (e.g. server 150) connected to the computer 100 via the network 220.

In the system 600B of Fig. 6(b), all of components 300 to 304 and 306 can run from the set-top-box 601. In this instance, the components 300 to 304 and 306 can be stored in the memory 4306 of the set top box 601 and can be read and controlled in their execution

by the CPU 4305. The directory service 311 can run on the computer 100 and can be stored in the memory 206 of the computer 100 and be read and controlled in its execution by the CPU 205. Alternatively, the directory service 311 can be run on the set top box 601 or its function performed by the launcher 303.

Alternatively, if the set-top-box 601 is not powerful enough to run the system 600 locally, only the I/O daemon 300 need run on the set-top-box 601 and the remainder of the architecture 200 (i.e. process components 301, 303, 304, 306 and 311) can run remotely on the other servers (150, 152) which can be accessed via the network 220. In this instance, the I/O daemon 300 can be stored in the memory 4306 of the set top box 601 and can be read and controlled in its execution by the CPU 4305. Again, the functional parts of such a system can be divided as shown in Fig. 8.

2.1.1 I/O Daemon

The I/O daemon 300 is a process component that converts datagrams received from the remote reader 1 into a TCP/IP stream that can be sent to the event manager 301 and vice versa (e.g. when using a two-way protocol). Any suitable data format can used by the remote reader 1. The I/O daemon 300 is preferably independent of any changes to the remote reader 1 data format, and can work with multiple arrangements of the remote reader 1. In one advantageous implementation of the system 600, the I/O daemon 300 is integrated into the event manager 301.

In the system 600A, the I/O daemon 300 is started when a user starts the smart card system 600 by powering up the computer 100 and the event manager 301 has been started. Alternatively, the I/O daemon 300 is started when a user starts the system 600 by turning on the set-top box 601.

The I/O daemon 300 will be explained in more detail below with reference to section 9.0.

2.1.2 Event Manager

The event manager 301 forms a central part of the architecture 200 in that all communications are routed through the event manager 301. The event manager 301 is configured to gather all events that are generated by the remote reader 1 and relayed by the I/O daemon 300. These events are then redistributed to the various process components 300 to 304, and 306 and running applications. The event manager 301 is also configured to check that an event has a valid header, correct data length, but is typically not configured to check that an event is in the correct format. An "event" in this regard represents a single data transaction from the I/O daemon 300 or the launcher 303 or applications 304.

Any changes in protocol between different systems can be dealt with by the event manager 301. Where possible, events can be rewritten to conform to the data format understood by any presently running application 304. If such is not possible, then the event manager 301 reports an error to the originating application 304. When different data formats are being used, for example with a system running multiple smart cards, the event manager 301 preferably ensures that the smallest disruption possible occurs.

The event manager 301 does not have any presence on the display screen or other output device 116. However, the event manager 301 can be configured to instruct the display manager 306 as to which application is presently required (i.e. the "front" application) and should currently be displayed on the display 101. The event manager 301 infers this information from messages passed to the applications 304 from the launcher 303 as will be explained in more detail below with reference to section 10.0.

The event manager 301 can be configured to always listen for incoming I/O daemon connections or alternatively, can start the system 600. The method used is dependent on the overall configuration of the system 600. In this connection, the event

manager 301 can start the system 600 or the set top box 601 can use the incoming connection of the I/O daemon 300 to start the system 600. The event manager 301 will be described in more detail below with reference to section 7.0.

2.1.3 Master Launcher

Where a thin client computer is being utilised and multiple launchers 303 are running with each launcher 303 being responsible for one set top box, a master launcher (not shown) which communicates directly with the event manager 301 can be used. The master launcher is used to start the launcher 303 corresponding to each of the event managers 301 if more than one event manager is running on the system 600. Initially, when the I/O daemon 300 connects to the event manager 301, the event manager 301 requests that the master launcher start a first process for the event manager 301. This first process is generally the launcher 303 for any smart card application 304. The master launcher can also be configured to shut down the launcher 303 of an application 304 when the event manager 301 so requests, and for informing the event manager 301 that the launcher 303 has exited.

There is preferably one master launcher running for each physically separate server (e.g. 150, 152) that is running an associated smart card application 304. This one master launcher handles the requests for all event managers that request launchers on a particular server. When run on a computer 100, as seen in Fig. 7, the master launcher commences operation either before or no later than at the same time as the rest of the system 600. In this instance, the master launcher is started first.

The master launcher can be integrated into the event manager 301, for example, when an associated launcher is running on the same computer as the event manager 301.

2.1.4 Launcher/First Application

In one advantageous implementation of the system 600, the first process started by the insertion of a smart card 10 into the remote reader 1 is the launcher 303. In specific systems, specific applications may be commenced, for example an automatic teller machine can start a banking application. Another example includes the use of restricted launchers that only start a specified sub-set of applications. The launcher 303 is an application that starts other applications for a specific event manager 301. The launcher 303 starts and ends applications and can also start and end sessions. The launcher 303 also informs the event manager 301 when applications are starting and ending, and tells the applications 304 when they are receiving or losing focus, or when they need to exit. In this regard, where a number of applications 304 are operating simultaneously, the application 304 that is currently on-screen is the application having focus, also known as the "front application". When another application is about to take precedence, the launcher 303 tells the front application that it is losing focus, thereby enabling the current application to complete its immediate tasks. The launcher 303 also tells the new application 304 that it is gaining focus, and that the new application 304 shall soon be changing state. The launcher 303 is also configured to force an application to exit.

The launcher 303 may receive certain events such as "no-card", "low battery" and "bad card" events generated by the remote reader 1. The launcher 303 also receives events that are intended for applications that are not currently the front application, and the launcher 303 operates to correctly interpret these events.

The launcher 303 is preferably only started when a request is generated by the event manager 301 to request the launcher 303 to be started. The launcher 303 can also be told to exit and forced to exit by the event manager 301.

The launcher 303 is preferably the only process component that needs to communicate with the directory service 311. When the launcher 303 is required to start a

new application 304, the launcher 303 queries the directory service 311 with service data, and the directory service 311 returns a location of the application 304 and service data associated with the new application 304. The service data is sent to the new application 304 as initialisation data in an event, referred to herein as the EM_GAINING_FOCUS event. The application location specifies the location of the application 304 to be run. This may be local, for implementations with a local computer, or networked. If the application location is empty, then the launcher 303 has to decide which application to start based on the service data.

The launcher 303 can also be configured to start any applications, for example browser controllers that will generally always be running while the system 600 is operating. Such applications are referred to as persistent applications. Applications can also be started by the launcher 303 either as a response to the first user selection on a corresponding smart card 10, or at the request of another one of the applications 304.

The launcher 303 can be integrated into the event manager 301 in some implementations of the system 600.

The launcher 303 will be explained in more detail below with reference to section 10.0.

2.1.5 Display Manager

The display manager 306 selects which smart card application 304 is currently able to display output on the display screen 101. The display manager 306 is told which application 304 can be displayed by an EM_GAINING_FOCUS event originating from the launcher 303. This event can be sent to the display manager 306 directly, or the event manager 301 can send copies of the event to the display manager 306 and the intended recipient.

Generally, the only application 304 that should be attempting to display output should be the front application. The display manager 306 can provide consistent output during the transfer between applications having control of the display. The display manager 306 may need to use extrapolated data during changeovers of applications as the front application.

The architecture 200 can be configured such that the display manager 306 is not needed or the role of the display manager 306 may be assumed by the other parts 301or 303, of the architecture 200.

2.1.6 Directory Service

The directory service 311 is configured to translate service identifiers that are stored on smart cards 10, into resource locators (e.g. a URL) that indicate the location of the services or the location of an application associated with a service. The directory service 311 is also configured to translate optional service data. The directory service 311 allows the launcher 303 associated with a particular card 10 to decide what to do with a resource locator, for example, download and run the associated application 304 or load the resource locator into a browser application. The translation by the directory service can be performed using a distributed lookup system.

2.1.7 Applications

The applications 304 associated with a particular smart card 10 can be started by the launcher 303 associated with that smart card 10 in a response to a first button press on a corresponding card. Each application 304 can be a member of one or more service groups, described in detail later in this specification. An application 304 can be specified to not be part of any service group in which case the application will never be run with other applications. An application can become part of a service group once the

application is running and can remove itself from a service group when the application is the currently front application.

Some applications can be started when the system 600 is started and these applications, for example a browser control application or a media playing application can be always running. These persistent applications can be system specific or more generally applicable.

Fig. 9 is a schematic block diagram representation of a card interface system, including the process components 301 to 306 described above. In the system of Fig. 9, the remote reader 1 communicates with a computer 100 via an IR link in conjunction with an I/O daemon 300 for controlling the IR link. Further, the computer 100 is configured for communicating to and from a communications network in this case represented by the Internet 400 to a Web server 410. In this instance, some of the applications 304 accessible utilising the smart cards 10 and remote reader 1 can be Web pages 406 associated with different smart cards 10. The Web libraries 407 contain functions (e.g. JavaScript functions) and classes (e.g. Java classes) that can be included with web pages for use with the smart card 10. The Web pages 406 can be accessed with a running application called the Web browser 403. In the system of Fig. 9, the event manager 301 is configured to receive an event from the remote reader 1. The event is then sent to the launcher 303, which can be configured to send a message to the browser controller 402, which controls the Web browser 403. The process for starting an application or browser session will be explained in more detail below. The launcher 303 can also be configured to download applications 408 as well as running applications from a file server 411 which is also connected to the computer 100 via the Internet 400.

3.0 READER

The remote reader 1 is preferably a hand-held, battery-powered unit that interfaces with a smart card 10 to provide a customisable user interface. As described above, the remote reader 1 is intended for use with a digital television, a set top box, computer, or cable television equipment to provide a simple, intuitive interface to on-line consumer services in the home environment.

Figs. 43 and 44 show a reader 4401 similar to the reader 1 described above. The reader 4401 is configured for the reading of the card 10. The reader 4401 is formed of a housing 4402 incorporating a card receptacle 4404 and a viewing area 4406. The receptacle 4404 includes an access opening 4410 through which a smart card 10, seen in Fig. 1, is insertable.

An upper boundary of the viewing area 4406 is defined by sensor means in the form of a substantially transparent pressure sensitive membrane 4408 similar to the membrane 8 described above. Arranged beneath the membrane 4408 is data reading means provided in the form of an arrangement of exposed electrical contacts 4407 configured to contact complementary contacts of the smart card 10.

The card 10 is inserted into the reader 4401 via the access opening 4410 as shown in Fig. 45. The configuration of the reader 4401 allows a user to hold the reader 4401 in one hand and easily insert the smart card 10 into the reader 4401 with the user's other hand. When the smart card 10 is fully inserted into the reader 4401, the pressure sensitive membrane 4408 fully covers the upper face 16 of the smart card 10. The viewing area 4406 preferably has substantially the same dimensions as the upper face 16 of the card 10 such that the upper face 16 is, for all intents and purposes, fully visible within the viewing area 4406 through the transparent pressure sensitive membrane 4408.

Fig. 46 shows a user operating the reader 4401 after a card has been fully inserted.

Referring to Figs. 47(a) to 47(c), the housing 4402 is formed of a substantially two part outer shell defined by a top section 4827 that surrounds the membrane 4408, and a base section 4805 which extends from a connection 4829 with the top section 4827 to a location 4811 below and proximate the transverse centre of the membrane 4408. The base section 4805 incorporates a facing end 4815 formed from infrared (IR) transparent material thereby permitting IR communications being emitted by the reader 4401.

The location 4811 defines a point of connection between the base section 4805 a card support surface 4807 which extends through a plane in which the contacts 4407 lie to an interior join 4835 that sandwiches the membrane 4408 between the surface 4807 and the top section 4827. The access opening 4410 is substantially defined by the space between the location 4811 and a periphery 4836 of the housing 4402, seen in Fig. 47(a).

The contacts 4407 extend from a connector block 4837 mounted upon a printed circuit board (PCB) 4801, the PCB 4801 being positioned between the base section 4805 and the support surface 4807 by way of the two mountings 4817 and 4819. Arranged on an opposite side of the PCB 4801 to the connector block 4837 is electronic circuitry (not shown), electrically connected to the connectors 4407 and the touch sensitive membrane 4408 and configured for reading data from the card 10 according to depression of the membrane 4408. Also mounted from the PCB 4801 is an infrared light emitting diode (LED) 4800 positioned adjacent the end 4815 which acts as an IR window for communications with a device (e.g. the set top box 601) to be controlled.

Fig. 47(b) shows a similar view to Fig. 47(a), with the smart card 10 partially inserted through the access opening 4410 into the receptacle 4404. As can be seen in Fig. 47(b), the support surface 4807 has an integrally formed curve contour 4840 that leads downward from the plane of the contacts 4407 towards the join 4811. This configuration allows the reader 4401 to receive the smart card 10 such that the smart card 10 may be

initially angled to the plane of the receptacle 4404, as seen in Fig. 47(b). The configuration of the curve contour portion 4840 of the support surface 4807 guides the smart card 10 into a fully inserted position under the force of the user's hand. Specifically, as the card 10 is further inserted, the curvature of the support surface 4807 guides the card 10 into the plane of the contacts 4407 and receptacle 4404.

Fig. 47(c) shows a similar view to Fig. 47(a), with the smart card 10 fully inserted into the receptacle 4404. In this position, the card 10 lies in the plane of the receptacle 4404 and the contacts 4407 which touch an associated one of the data contacts (not seen) of the smart card 10, and the smart card 10 is covered by the pressure sensitive membrane 4408. Further, the contacts 4407 are preferably spring contacts that act to provide a force against the card 10 and associated with the membrane 4408, sufficient for the card 10 to be held within the receptacle by a neat interference fit.

In the following description references to the reader 1 can be construed as references to a reader implemented as the reader 1 of Fig. 1 or the reader 4401 of Figs. 43 to 47(c).

Fig. 10 is a schematic block diagram showing the internal configuration of the remote reader 1 in more detail. The remote reader 1 includes a microcontroller 44 for controlling the remote reader 1, coordinating communications between the remote reader 1 and a set top box 601, for example, and for storing mapping information. The microcontroller 44 includes random access memory (RAM) 47 and flash (ROM) memory 46. The microcontroller 44 also includes a central processing unit (CPU) 45. The microcontroller 44 is connected to a clock source 48 and a clock controller 43 for coordinating the timing of events within the microcontroller 44. The CPU 45 is supplied with electrical power from a 5 volt battery 53, the operation of the former being

controlled by a power controller 50. The microcontroller 44 is also connected to a beeper 51 for giving audible feedback about card entry status and for "button" presses.

Infra-red (IR) communications are preferably implemented using two circuits connected to the microcontroller 44, an IR transmitter (transmitter) 49 for IR transmission and an IR receiver (receiver) 40 for IR reception.

The pressure sensitive touch panel 8 of the remote reader 1 communicates with the microcontroller 44 via a touch panel interface 41. A smart card interface 42 connects to the electrical contacts 7.

An in-system programming interface 52 is also connected to the microcontroller 44, to enable programming of the microcontroller 44 by way of the microcontroller FLASH memory 46 with firmware. The firmware will be explained in further detail later in this document with reference to section 6.0.

The internal configuration of the remote reader 1 will now be described in further detail.

3.1 Low Power Mode Lifetime

The power controller 50 is operable to provide two power modes, one being a low-power "sleep" mode, and another being an active mode. The low power mode lifetime is the lifetime of the battery 53 expressed in years. When the remote reader 1 is not functioning and is in the low power mode, the lifetime can be between greater than 2 years.

If the reader 1 is in sleep mode and a user presses the touch panel 8, the remote reader 1 then comes out of sleep mode, and the CPU 45 calculates the touch co-ordinates and sends a serial message by infra-red transmission. The battery 53 should preferably remain serviceable for the current supply requirements of more than 100,000 button presses.

3.2 Service Life

The service life is defined as the period of time that the remote reader 1 can be expected to remain serviceable, not including battery replacement. The service life is related to the Mean Time Between Failures (MTBF) figure and is usually derived statistically using accelerated life testing. The service life of the remote reader 1 can thus be greater than 5 years.

3.3 Microcontroller

The microcontroller 44 of the remote reader 1 has an 8 bit central CPU with 4096 bytes of FLASH memory 46 and 128 bytes of random access memory 47. The microcontroller 44 preferably operates on a supply voltage from 3 to 5 Volts and has flexible on-board timers, interrupt sources, 8 bit analog to digital converters (ADC), clock watchdog and low voltage reset circuits. Preferably, the microcontroller 44 also has high current output pins and can be programmed in circuit with only a few external connections.

3.4 Clock Source

The main clock source 48 for the remote reader 1 is preferably a 3 pin 4.91MHz ceramic resonator with integral balance capacitors. The frequency tolerance is 0.3%. While such tolerance is not as good as a crystal, such is however adequate for serial communications and is much smaller and cheaper than a crystal.

3.5 Beeper

The beeper 51 is included with the remote reader 1 to give audible feedback about card entry status and for button presses. The beeper 51 is preferably a piezo-ceramic disk type.

3.6 Infra-red Communications

As described above, infra-red (IR) communications are preferably implemented using two circuits, an IR transmitter 49 for IR transmission and an IR receiver 40 for IR reception. The two circuits 40 and 49 are preferably combined on a printed circuit board (e.g the PCB 4801 of Fig. 47) within the remote reader 1. The printed circuit board 4801 can be connected to the microcontroller 44 by a 4 way flat printed cable. Large bulk decoupling capacitors (not shown) are required on the PCB 4801 to provide surge currents, which are required when transmitting.

3.7.1 Infra-red Transmission

IR transmission is preferably by means of an infra-red Light Emitting Diode (LED) (e.g. the LED 4800 of Fig 47(a)) forming part of the IR transmitter 49.

3.7.2 Infra-red Reception

The IR receiver 40 is preferably integrated with an infra-red filter, a PIN diode, an amplifier and discriminator circuitry into a single device. Received serial information passes directly from this device to an input port of the microcontroller 44. This port can be programmed to generate an interrupt on receiving data allowing speedy storage and processing of incoming signals.

3.8 CPU / Memory Card Interface

The remote reader 1 can preferably support smart cards 10 as defined by the International Standards Organisation (ISO) standards 7816-3 and ISO 7810. Three and five volt CPU cards (i.e. cards with an embedded microprocessor) with T=0 and T=1 protocols can also be supported as are 3 and 5V memory cards.

The electrical contacts 7 used to make contact between the card 10 and the microcontroller 44 are preferably a surface mount connector with 8 sliding contacts and a "card in" switch. In accordance with the ISO requirements the following signals must be provided:

- Pin 1 - VCC - Supply voltage;
- Pin 2 - RST - Reset signal. Binary output to card;
- Pin 3 - CLK - Clock signal, Binary output to card;
- Pin 4 - RFU - Reserved, leave unconnected;
- Pin 5 - GND – Ground;
- Pin 6 - VPP - Programming voltage, not required, link to GND, VCC or open;
- Pin 7 - I/O - Data I/O, bi-directional signal; and
- Pin 8 - RFU - Reserved, leave unconnected.

The RST and I/O pins are preferably connected directly to the microcontroller 44. All pins except the power supplies are equipped with series termination and transient voltage suppressor diodes to prevent electrostatic discharge problems.

3.9 CPU Card Power Supply

As described above, the microcontroller 44 requires a 3 - 5 Volt power supply for operation. The 5 Volt supply can be generated from a 3V Lithium coin cell operating as the battery 53 by means of the power controller 50 in the form of a regulated 5V charge-pump DC-DC converter chip.

3.10 Touch Sensitive Interface

As described above, the pressure sensitive touch panel 8 of the remote reader 1 communicates with the microcontroller 44 via a touch panel interface 41. The touch panel interface 41 provides an analog signal according to the position of the touch on the touch panel 8. This analog signal is then communicated to the microcontroller 44.

The calculation of touch co-ordinates requires bottom and left touch panel 8 contacts (not shown) to be connected to the inputs of an analog to digital converter on the microcontroller 44.

A touch on the touch panel 8 can preferably be used to wake up the remote reader 1 from sleep mode. A resistive connection from the left screen contact to a sleep WAKE UP port as illustrated provides this feature. Note that during in-system programming, up to 8 volts may be applied to a pin on the microcontroller 44 referred to as the Interrupt Request Pin (IRQ) so a clamping diode needs to be fitted to this pin to prevent device damage. In this instance, it is the internal pull up on the IRQ pin that actually provides the bias required to detect touch panel 8 presses.

3.11 Battery

As described above, the remote reader 1 uses a battery 53. A 5 Volt lithium coin cell can be used as the battery 53 to power all the circuitry of the remote reader 1.

3.12 In System Programming

The microcontroller supports in-system programming (ISP) options. The in-system programming interface 52 is used in the remote reader 1 to perform programming of the microcontroller 44 such as programming of the microcontroller FLASH ROM memory 46 with firmware.

3.13 Printed Circuit Boards and Interconnection

The remote reader 1 can include two printed circuit boards (PCB), instead of the one PCB 4801 of the reader 4401, as follows:

(i) an infra-red (IR) PCB which holds the infra-red diode, drive FET and receiver; and

(ii) a main PCB (e.g. the PCB 4801 of Fig. 47(a)) which holds all the other components 40 to 53 mentioned above.

Both of the PCB boards described above are preferably double sided designs using standard grade FR4, 1.6mm PCB material. The main PCB preferably utilises surface

mount components since the thickness of the finished PCB is critical and preferably components are restricted to a height of approximately 3mm max.

The IR PCB can use through hole parts but again there are preferably stringent component height restrictions imposed. The interconnection of the two PCBs is via a custom designed 4 way flat printed cable (FCA). This interfaces to the two PCBs via a surface mount FCA connector that is the same part used to interface to the touch panel 8.

3.14 Low Power Mode

When the remote reader 1 has not been used for a short period of time, pre-programmed firmware preferably puts the unit into the low-power mode to conserve battery life. In low-power mode, the supply voltage is switched off to all current consuming components, the ports of the microcontroller 44 are set into a safe sleep state and the clock 48 is stopped. In this state the current consumption of the remote reader 1 is less than 5 μA. A P-channel FET can be used to control the supply of power to the current consuming components.

There are three alternative preferred methods to wake the remote reader 1 up from low power mode as follows:

- touch the touch panel 8;
- insert a card into the card receptacle 4; and
- remove and re-insert the battery 53.

The card insert wake up enables the remote reader 1 to always beep when a card is inserted, regardless of whether the unit is in low power mode or not. The touch and card insert wake ups are handled by the IRQ pin as illustrated on the microcontroller 44. It is important that this pin is set to "edge trigger" only so that only a new touch or card insert wakes the microcontroller up. If IRQ sensitivity is set to "level" trigger then

inadvertently leaving the touch panel 8 pressed, for example when the remote reader 1 is packed in luggage, would prevent the remote reader 1 from entering low power mode.

3.15 Interrupts and Resets

The microcontroller 44 firmware for the remote reader 1 uses two external and one internal interrupt sources. External interrupts come from the IRQ pin for low power mode wake up. The internal interrupt is triggered by a timer overflow and is used to time various external interfaces. These interrupts are serviced by pre-programmed firmware procedures.

There are four possible reset sources for the microcontroller as follows:

- low supply voltage reset at 2.4 Volts;
- illegal firmware op-code reset;
- Computer Operating Properly (COP) reset if firmware gets stuck in a loop; and
- ISP reset forced onto a RESET pin when in-system programming (ISP) starts.

4.0 CARD DATA FORMAT

The format of data for the card 10 described above will be described in the following paragraphs. For memory cards such as the control card 10B as described in relation to Fig. 4, data conforming to the format to be described will be copied directly onto the card. For the CPU cards described above, data conforming to the format to be described can be loaded as a file into the file system of the CPU of the card.

The card 10 described above preferably stores a data structure that describes various card properties and any user- interface indicia printed on the card. The cards 10 can also include global properties that specify attributes such as information about the card, vendor and a service. User-interface objects, if present, specify data to associate with areas of the surface of the card 10.

The user-interface objects as described herein, represent mapping data, which relate predetermined areas, or iconic representations directly imprinted on a surface of the card 10, to commands or addresses (eg: Uniform Resource Locators (URLs)). The mapping data includes coordinates which typically define the size and location of user Interface Elements (eg: predetermined areas) on the card 10. In this connection, the term user interface element typically refers to indicia on the card 10, whilst the term user interface object typically refers to the data related to a particular indicia. However, these terms are used interchangeably throughout the following description.

The user-interface objects are preferably stored directly on the card 10. Alternatively, the user-interface objects can be stored not on the card 10 itself, but in the system 600. For example, the card 10 can store, via an on-card memory, a barcode or a magnetic strip, a unique identifier, which is unique to cards 10 having substantially similar user interface elements and layout. The unique identifier together with the coordinates determined from the touch panel 8, as a result of a user press, can be transmitted by the reader 1 to the computer 100 or to the set top box 601, of the system 600. The system 600 having the user-interface objects stored on the computer 100, set top box 601 or a server 150, can perform the mapping from the determined coordinates to a corresponding command, address or data relevant to a service associated with the card 10 and the user press, in order to provide a desired function represented by the user interface element on the card 10. In this instance, the data related to the user selected indicia as described above takes the form of coordinates determined by the reader 1 as a result of a user press on a portion of the touch panel 8 which overlays the desired indicia.

In the cards (e.g. 10) described above, data stored by the card 10 includes a card header followed by zero or more objects as described in the following sections.

4.1 Card Header

Fig. 11 shows the data structure of a card header 1100 as stored in the smart card 10. The header 1100 includes a number of rows 1101, each of which represent four bytes of data. The data is preferably in 'big endian' format. The complete header is 20 bytes long and includes the following fields (described in more detail in Fig. 12):

(i) magic number field 1102, which includes a constant specifying a card as being a valid memory card. For example, the magic number field 1102 can be used to check or verify that a propriety card belonging to a particular manufacture is being used.

(ii) versions field 1103, which includes each version increment that specifies a change in the card layout that can not be read by a reader which is compatible with lower versions of the layout;

(iii) reserved field 1104, this field is reserved for future use;

(iv) flags field 1105, which includes flags for a card (see Fig. 13);

(v) distinguishing identifier field 1110, which includes two fields – a service 1106 and a service specific field 1107. The service field 1106 identifies the service of a corresponding card and the service specific field 1107 optionally contains a service-specific value;

(vi) a number of objects field 1108, which includes a number value representing how many objects follow the header. This field can be set to zero; and

(vii) a checksum field 1109, which includes a card checksum of all data on the card excluding the checksum itself.

Fig. 12 provides a description of the content of the various (number) fields described with reference to Fig. 11. In particular, the distinguishing ID number field 1110 comprises an eight byte distinguishing identifier. The distinguishing identifier includes two portions, unit pieces of data, namely, a service identifier and a service-specific identifier.

Preferably, the distinguishing identifier is arranged so that the service identifier occupies five bytes and the service-specific identifier occupies three bytes of the total distinguishing identifier value.

The service identifier contained in the field 1106 distinguishes one service from another or distinguishes one vendor from another. That is, for example, a service can be associated with an application that provides the service to a card user as distinct from a vendor who can provide multiple services to the card user by providing multiple applications.

The service identifier can be an identifier to identify the application to be used or application location (e.g. URL). Also, generic cards may be added to the System 600A or 600B and they are a special use of the Service identifier. The Generic cards are cards with a special Service identifier that can be used to provide input to a current application already running. The special value for the service 0x0000000001 is known as "the generic service identifier " and is used on "generic cards". A generic card can be used to send data to the front application already running. These are used, for example, for keypads that can be used to send text input to any application or a card with personal details that also may be used to submit this information to any application.

The service – specific identifier contained in the field 1107 can be optionally used by the vendor of a particular service to provide predetermined functions associated with that particular service. The use of the service-specific identifier is substantially dependent upon the application 304 run on the system 600. For example, the service identifier together with the service-specific identifier can be used as a unique identifier for a card 10. This unique identifier can be used to gain or deny access to a specific feature associated with a particular service, to reproduce a specific-service identifier in a log file in order to confirm or verify that a particular card 10 having that value was used to access

a service, and to provide a unique identifier that can be matched up with a corresponding value in a database in order to retrieve information about the user of the service (eg: name, address, credit card number etc).

Another example of a use for the service-specific identifier can include providing information about a mechanism or mode of distribution of the cards 10 (e.g. by mail, bus terminal kiosks, handed out on a train etc). Further, the service-specific identifier, can identify what data should be loaded into the system 600 when a service is accessed.

The foregoing is not intended to be an exhaustive list of possible uses or applications of the service-specific identifier but a small sample of possible applications and there are many other applications of the service-specific identifier of field 1107.

4.1.1 Card Flags

The flags field 1105 of the header 1100 of Fig. 11 may include three flags as follows:

(i) Don't beep;

(ii) No move events; and

(iii) No event co-ordinates.

Fig. 13 shows a description of each of the above flags. The above flags affect the functions that a smart card 10 can perform in a remote reader 1, as is defined by the description of each flag. An example, of a user interface element as referred to in Fig. 13 is a "button" on the card 10. user interface elements will be explained in further detail later in this document.

4.2 Objects

As shown in Fig. 57, immediately following the card header 1100 of Fig. 11 can be zero or more object structures 5713 defining the objects of a particular card 10 and

forming part of the data stored on the card 10. Each object structure 5713 comprises four fields as follows:

(i) a type field 5701;

(ii) an object flags field 5703;

(iii) a length field 5705; and

(iv) a data field 5707.

The structure of the data field 5707 depends on the object type as will be described below.

Fig. 14 shows a description of each of the fields 5701, 5703, 5705 and 5707 of the object structure 5713. The flags field 5703 of the object structure 5713, preferably includes an inactive flag. Fig. 15 shows a description of the inactive flag.

There are preferably five object types provided for the cards 10A, 10B, 10C and 10D described above, as follows:

(i) user Interface objects (i.e. data defining a button on the card 10);

(ii) Card Data;

(iii) Fixed Length Data;

(iv) Reader Insert;

(v) No operation; and

(vi) No operation (single byte).

Fig. 16 shows a description of each of the above object types (i) to (vi).

4.2.1 user Interface Object

Each user interface object defines a rectangular area on the card 10 and some quantity of associated data that is transmitted when the user touches an area of the panel 8 over the corresponding rectangular area of the card 10. The origin for the co-ordinate mapping system is the top left of the smart card 10 as if it was an ISO standard memory

smart card held in a portrait view with the chip contacts 18 facing away from the viewer and towards the bottom of the card 10. For any reader 1 that does not use this card orientation, the values of the corner points must be adjusted by the reader 1 so as to report a correct "button" press.

The user interface (element) object structure preferably has six fields as follows:

(i) a flags field;

(ii) an X1 field;

(iii) an Y1 field;

(iv) an X2 field;

(v) a Y2 field; and

(vi) a data field which typically includes data associated with the user interface element for example, a URL, a command, a character or name.

Fig. 17 shows a description of each of the above fields for the described user interface object structure. A press on the pressure sensitive touch panel 8 is defined to be inside a particular user interface object if:

(i) the X value of the press location is greater than or equal to the X1 value of the associated user interface object and is strictly less than the X2 value for that particular user interface object; and

(ii) the press Y value for the press location is greater than or equal to the Y1 value of the particular user interface element and strictly less than the Y2 value.

Overlapping user interface elements is allowed. If a press is within the bounds of more than one user interface element then the object sent is determined by a Z order. The order of the user interface elements on the card defines the Z ordering for all of the user interface elements on that particular card. The top user interface element is the first user

interface element for a particular card 10. The bottom user interface element is the last user interface element for that particular card 10. This allows for non-rectangular areas to be defined. For example, to define an "L" shaped user interface element, a first user interface object would be defined with zero bytes in the data field, and a second user interface object would be defined to the left and below the first user interface object but overlapping the user interface object.

The location of a press is to be reported in "fingels", which represent finger elements (analogous to "pixels" which represent picture elements). The height of a fingel is defined to be 1/256th of the length of an ISO memory smart card and the width is defined to be 1/128th of the width of an ISO memory smart card. The behaviour associated with each element may be modified with one or more flags. Each user interface element preferably has four associated flags as follows:

(i) Invert Beep Enable;

(ii) Auto repeats;

(iii) Do Not Send Data on Press; and

(iv) Do Not Send Data on Release.

Fig. 18 shows a description for each of the user interface element flags.

4.2.2 Card Data

The Card Data object is used to store data which is specific to a particular card. The data layout for this object has no fixed form. The contents of the Card Data object are sent from the reader 1 as part of the INSERT message when the card 10 is inserted into the reader 1.

4.2.3 Fixed Length Data

The fixed length data object is used to define a fixed length block on the card that can be written to by the computer 100, for example.

4.2.4 Reader Insert

The reader insert object is used to store instructions for the remote reader 1 when a particular card is inserted. This can be used, for example, to instruct the reader 1 to use a specific configuration of IR commands to allow communication with a specific set top box or TV.

4.2.5 No Operation

The No Operation object is used to fill in unused sections between other objects on a particular card. Any data stored in the no operation object is ignored by the remote reader 1. Any unused space at the end of the card 10 does not need to be filled in with a no operation object.

4.2.6 No Operation (One Byte)

The No Operation (One Byte) object is used to fill gaps between objects that are too small for a full object structure. These objects are only one byte long in total.

5.0 READER PROTOCOL

The remote reader 1 uses a datagram protocol that supports both uni-directional and bi-directional communication between the remote reader 1 and the set top box 601 or computer 100, for example. The format used for messages from the remote reader 1 as a result of user interactions with the remote reader 1 are of a different format than those that are sent to the remote reader 1.

5.1 Message Types

There are at least seven message event types that can be sent by the remote reader 1. These events are as follows:

- INSERT: When a card 10 is inserted into the remote reader 1, and the card 10 is validated, an INSERT event is generated by the remote reader 1 and an associated message is transmitted. This message announces the card 10 to a receiver (e.g. the set

top box 601). The INSERT message preferably includes the particular distinguishing identifier and allows applications to be started or fetched immediately upon card 10 insertion rather than waiting until the first interaction takes place. The INSERT message preferably includes the contents of the card data object from the card 10 inserted into the reader 1 if an object of this type is present on the card 10.

- REMOVE: When a card 10 is removed from the remote reader 1, a corresponding REMOVE event is generated and a REMOVE message is sent to the particular receiver associated with the remote reader 1. Like the INSERT message, the associated distinguishing identifier is transmitted along with the message. As the distinguishing identifier cannot be read from the now removed card 10, the distinguishing identifier is stored in the memory 47 of the remote reader 1. This is a useful optimisation as the distinguishing identifier is required for all other messages and reading the distinguishing identifier from the card 10 each time the distinguishing identifier is required can be too slow. INSERT and REMOVE messages are not relied upon by the system 600 to control processing. The system 600 is configured to infer missing messages if a message is received and is not immediately expected. For example, if an application detects two INSERT messages in a row, then an application can assume that it has missed the REMOVE message associated with the card of the first INSERT message as it is not possible to have two cards inserted at one time in present arrangement. The application can then take whatever action is required prior to processing the second INSERT message.

- Another example of where a missing message can occur is where a hand-held, infrared connected reader 1, as compared with a wired reader, is being used. Often a user does not point the reader 1 directly at a receiver when inserting or removing cards. This problem can be corrected by the system 600 inferring the INSERT or

REMOVE operations based on differing distinguishing identifiers in consecutive PRESS and RELEASE pairs.

- BAD CARD: If an invalid card is inserted, then the remote reader 1 is preferably configured to generate a BAD CARD event and to send a BAD CARD message. This message allows an associated receiver to take some action to alert the user to the invalid card.
- PRESS: When a touch is detected by the remote reader 1, a PRESS event is generated and a PRESS message is sent to an associated receiver. The PRESS message contains details of the associated card, the position of the press and the data associated with the user-interface element at that particular position. If there is no user interface element defined for that position (including if there is no user interface elements defined on the card 10 at all) a PRESS message is sent containing details of the associated card and the position of the press. If there is no card present in the remote reader 1 when a PRESS event is generated then a PRESS message is sent containing the special "NO_CARD" identifier (i.e. eight bytes of zero - 0x00) and the position of the press.
- RELEASE: A RELEASE event complements the PRESS event and a RELEASE message can be sent in order to inform the application program of the system 600 that a PRESS has been lifted. Every PRESS event preferably has a corresponding RELEASE event. Readers can allow multiple presses to be registered or provide other events that may occur between PRESS and RELEASE messages.
- MOVE: If, after processing a PRESS event, the touch position changes by a certain amount then the finger (or whatever is being used to touch the card) is assumed to be moving. MOVE EVENTS are generated and MOVE messages are sent until the touch is lifted. MOVE events auto-repeat by re-sending the last MOVE messages when the touch position remains stationary. The repeated sending finishes when the

touch is lifted and a corresponding RELEASE message is sent. Unlike PRESS and RELEASE events there is no user-interface object involved with MOVE events.

- LOW BATT: A LOW BATT event is generated and a LOW BATT message is sent when the battery 53 in the remote reader 1 is getting low. This message is sent after user interactions to increase the chance that the message will be received by the rest of the system 600. The sending of the LOW BATT message does not prevent the remote reader 1 from entering a low power state.

5.2 Data Formats

The preferred data format of the reader protocol used in the system 600 is a fixed size header followed by a variable length data field which can be zero bytes or more in length, followed by an eight bit check-sum and complement.

5.2.1 Message Header

The message header is preferably of a fixed length and is prepended (i.e. appended to, but in front of) to all messages sent from the remote reader 1. It is necessary to keep the message header as small as possible due to any bandwidth restrictions that may be imposed. Fig. 19 shows the format of the message header that is sent from a remote reader 1.

Service and service-specific identifiers can be assigned, by a smart card identification authority, to a vendor when the vendor registers a particular service. The service and service-specific identifier are the same for every message from a given card. A service specific identifier is preferably set by a vendor for use with their application. The Reader identifier is also in the header of each message. This identifier can be used by an application 304 to distinguish different users, for example, in a multi-player game.

Fig. 20 shows a table listing the message event types that have been described above.

5.2.2 Simple Messages

A number of message types are considered simple in that they consist solely of the message header described above followed by the message checksum byte and its complement. For example, a BADCARD message, a LOW_BATT message and a REMOVE message are simple messages.

Fig. 21 shows the format of a simple message.

5.2.3 MOVE Messages

MOVE messages are formed of the message header described above followed by two fields defining the co-ordinates of the touch position on the touch panel 8 of the remote reader 1. Fig. 22 shows the format of a MOVE message.

5.2.4 PRESS and RELEASE Messages

Fig. 23 shows the format of PRESS and RELEASE messages. PRESS and RELEASE messages, like MOVE messages contain the message header and touch co-ordinates. In addition, PRESS and RELEASE messages send data associated with the user-interface element if the touch position matches a user-interface element defined on the card. This data is of variable length, the actual size being defined by a corresponding card 10. If the touched position does not match a user-interface element defined on the card (including if no user-interface elements are defined on the card), zero bytes of data associated with user interface elements are sent. If there is no card 10 in the reader 1 then the service identifiers are all set to zero (ie 0x00) and zero bytes of data associated with the user-interface elements are sent. The data associated with the user interface element normally corresponds to the data associated with the user interface element defined on the card but may be modified or generated by processing on the card 10 or reader 1.

Fig. 24 is a data flow diagram showing the flow of the above-described messages within the system 600. As seen in Fig. 24, the card header 1100 and object structure 5713

are read by the CPU 45 of the remote reader 1 which sends a corresponding INSERT, REMOVE, PRESS, RELEASE, MOVE, BADCARD or LOW BAT message to the event manager 301 via the I/O daemon 300. As will be described in more detail below, the event manager 301 has twenty-one core messages, which are sent to and received from the ML 302, launcher 303 and applications 304.

5.2.5 INSERT Messages

INSERT messages are formed of the message header described above and the contents of the card data object from the inserted card 10. Fig. 21A shows the format of an INSERT message.

6.0 READER FIRMWARE

6.1 Overview

The microcontroller 44 has non-volatile memory 46 embedded within which can be programmed with the firmware to be described in detail below. The firmware working in concert with the microcontroller 44 and peripheral hardware (e.g. the computer 100) can thus dictate the functional requirements of the remote reader 1.

6.2 Code Type

In an attempt to minimise the cost of the remote reader 1 to a user, memory on the remote reader 1 is preferably minimised. As a result the application program written for the remote reader 1(i.e. the firmware) must be as compact and fast as is possible.

6.3 Resource Constraints

The microcontroller 44 has the following characteristics:

6.3.1 Non-volatile memory

The flash memory 46 is configured with 4096 bytes of FLASH ROM and can be utilised for firmware storage. The FLASH ROM is re-programmable but in the case of mass production a MASK ROM part can be utilised.

6.3.2 Random Access Memory (RAM)

The RAM 47 is configured as 128 bytes of RAM for use by the firmware.

6.4 Interrupts

The remote reader 1 uses two of the numerous interrupt sources supported by the microcontroller 44. These interrupts can be described as follows:

6.4.1 Received Data Interrupt

An infrared (IR) serial data receiver generally generates a falling edge when incoming data is received. This data has to be sampled and buffered as quickly as possible. One port of the microcontroller 44 doubles as an input timing capture pin which can initiate an interrupt on the falling edge.

6.4.2 Timer Overflow Interrupt

The microcontroller 44 has a free-running 16-bit timer which can be programmed to generate an interrupt when it overflows. In conjunction with the 4.91MHz clock source and pre-scale factor of 64, this equates to an interrupt every 3.41 seconds. An interrupt service routine increments a counter which triggers the suspension to low power mode preferably after about one minute of inactivity.

6.5 Resets

The microcontroller 44 supports five reset sources and the remote reader 1 is preferably configured to use all of reset sources. These reset sources can be described as follows:

6.5.1 Power On Reset (POR)

The POR reset is initiated when a new battery is fitted to the remote reader 1. The microcontroller 44 includes a circuit that detects the power on condition and generates a reset.

6.5.2 Low Voltage Inhibit (LVI) Reset

The LVI reset is initiated when a circuit (not shown) within the microcontroller 44 detects that the supply voltage has fallen below 2.4 Volts. When this kind of reset occurs a flag is set in a Reset Status Register (RSR) and an initialisation routine can deduce that the battery 53 is becoming depleted. For example, when infrared data is being transmitted, the infrared LED consumes high current as it is being pulsed. If the battery 53 is depleted, the supply voltage can dip under the 2.4 Volt threshold during transmission causing an LVI reset. After reset the battery 53 voltage recovers and the LVI reset does not occur until the next high current drain. This gives the remote reader 1 a chance to flag the failing of the battery 53 to an associated set-top box or remote equipment so that the user can be prompted to replace the battery 53.

6.5.3 Computer Operating Properly (COP) Reset

The COP reset is configured to reset the microcontroller 44 if the microcontroller 44 gets stuck doing a particular operation for an inordinate amount of time. The COP circuit takes the form of a counter that generates a reset if the counter is allowed to over-flow. The COP register must be written at predetermined time intervals to avoid a COP reset.

6.5.4 Illegal Address / Opcode Reset

An Illegal Address/Opcode Reset is generated by the microcontroller 44 if it encounters either an address out of a predetermined range or an opcode that does not conform to predefined conditions. This reset cannot be turned off but should only be in evidence during code debugging.

6.5.5 Hardware Reset

A hardware reset is generated by driving a 'Reset' pin on the microcontroller 44 low during normal operation. Additionally, if the microcontroller 44 is in low power mode, a falling edge on the Interrupt Request (IRQ) pin also generates a hardware reset. This reset is the mechanism used to wake the microcontroller 44 out of low power mode in the firmware. The IRQ pin is preferable for this function since it can be configured to be edge sensitive only, not level sensitive as the reset pin is.

6.6 Memory Card / CPU Card Interface

The firmware preferably supports only memory card peripherals using an Integrated Circuit Protocol (e.g. the I^2C protocol) . Alternatively, the firmware can support CPU card formats.

6.7 Power Consumption

The firmware plays a critical role in conserving the life of the battery 53. All operations performed by the microcontroller 44 are optimised so as to be performed as quickly as possible while wasting as little power as possible. As soon as the remote reader 1 has been inactive for a time (e.g. 1 minute) the microcontroller 44 suspends to low power mode to conserve battery life still further. Low power mode consumes about 1000 times less current than normal operating mode so efficient suspension to this mode is very desirable. The firmware controls the state of the microcontroller 44 ports during low power mode.

6.8 Device Programming

The microcontroller 44 is able to be programmed using an In-System program (ISP) function supported by an embedded monitor within the microcontroller 44. Monitor code is typically factory set by a manufacturer and can not be altered.

Programming of the microcontroller 44 for specific hardware can be performed using an In-Circuit Simulator (ICS) kit and a monitor-mode download cable. This cable uses the VCC, GND, RST, IRQ and PTB0 pins on the microcontroller 44. Source code to be programmed can be delivered, for example, from a Windows™ 95 development environment via a computer serial port to the ICS hardware and from there via the download cable to the microcontroller 44 pins. This programming method is ideal for firmware development and testing, but may be altered for mass production. A monitor-mode programming model is preferred in the microcontroller and an embedded programming jig for production can be used. Test points for programming signals can be provided to allow for production ISP. If the firmware is mask programmed into the microcontroller 44 then device programming will not be required.

6.9 Firmware Programming Sequence

The programming of the firmware will be described with reference to the reader 1 being operative coupled to a local computer 100.

6.9.1 The Main Loop

Fig. 25 is a flow diagram showing the read method 2500 performed by the remote reader 1 of the system 600 incorporating the software architecture 200. The method 2500 begins after a reset event, as described above, has been generated and the method 2500 is executed by the CPU 45. The method of Fig. 25 is configured in a "paced loop" manner. That is, the method 2500 is paced by a routine, which generates a 10ms delay. This delay gives adequate service to the necessary routines while providing good latency for the handling of interrupts.

At the first step 2600, an initialisation routine is performed by the CPU 45. The initialisation routine is performed in order to initialise configuration registers and will be explained below with reference to the flow diagram of Fig. 26. The method 2500

continues at the next step 2501, where the computer operating properly (COP) register is cleared indicating that the firmware is not stuck in any recurring loops. At the next step 2700 a check card process is performed by the CPU 45, in order to check for any changes in the presence and validity of a particular smart card 10. The check card process will be explained in more detail below with reference to the flow diagram of Fig. 27. The method 2500 continues at the next step 2800, where a scan touch panel process is performed by the CPU 45 to check for any touches on the touch panel 8 by the user. At the next step 2900, a wait 10 ms routine is performed by the CPU 45, and the method 2500 then returns to step 2501.

6.9.1 The Initialisation Process

After a reset from any one of the five sources described above all configuration registers require correct initialisation. If an LVI reset was received then a "possibly depleted battery" flag is set. Fig. 26 is a flow diagram showing a method 2600 of initialising the system 600 incorporating the software architecture 200. The method 2600 is executed by the CPU 45 and begins at step 2601 where all registers are initialised to a predetermined default state. At the next step 2602, a check is performed by the CPU 45 to determine if the reset was an LVI reset. If the reset was not an LVI reset at step 2602, then the method 2600 concludes. Otherwise the method 2600 proceeds to step 2603 where the possibly depleted battery flag is set and then the method 2600 concludes.

6.9.2 The Check Card Process

Fig. 27 is a flow diagram showing a method 2700 of checking the card 10 of the system 600 incorporating the software architecture 200. As described above, the method 2700 checks for changes in the presence and validity of a smart card 10 in the remote reader 1 and responds accordingly. The method 2700 is performed by the CPU 45 and begins at step 701 where if a smart card 10 is inserted in the remote reader 1, then the

method 2700 proceeds to step 702. At step 702, if the card 10 is a new card (i.e. in the previous state there was no card in the reader 1), then the method 2700 proceeds to step 703. Otherwise, the method 2700 concludes. At the next step 703, the "magic number" and "checksum" fields are read from the card header stored in the memory 19 of the card 10, and are checked for correctness. If the "magic number" and "checksum" are correct, then the method 2700 proceeds to step 704. The method 2700 continues at step 704, where the distinguishing identifier is read from the card header and the "No MOVE events" and "No Event Co-ordinates" flags are set. The Card Data, if present, is also read from the card at this step 704. At the next step 705, an INSERT message, including the Card Data if present, is sent to computer 100, and the INSERT message is processed by the CPU 205. Then at step 706, a "BEEP" is sounded and the method 2700 concludes.

If the "magic number" and "checksum" fields are not correct (ie: the card 10 is not valid) at step 703, then the method 2700 proceeds to step 710 where the don't beep, no move events and event co-ordinate flags are set. At the next step 711, a BAD CARD message is sent to the computer 100, and the BAD CARD message is processed by the CPU 205. Then at step 712, a "BOOP" is sounded and the method 2700 concludes.

If a smart card 10 is not inserted in the remote reader 1 at step 701, then the method 2700 proceeds to step 707. At step 707, if this is the first operation of the reader 1 after the reset then the method 2700 concludes. Otherwise, the method 2700 proceeds to step 708 where the "Don't beep", "No MOVE Events" and "No Event Co-ordinates" flags are set and the distinguishing identifier stored in memory 47 is set to "NO_CARD". At the next step 709, a REMOVE message is sent to the computer 100, and the REMOVE message is processed by the CPU 205. The method 2700 concludes after step 709.

6.9.3 The Scan Touch Panel Routine

Fig. 28 is a flow diagram showing a method 2800 of scanning the touch panel 8 of the reader 1 of the system 600 incorporating the software architecture 200. As described above, the scan touch panel routine checks for touch panel touches that equate with card button presses and responds accordingly. The method 2800 is executed by the CPU 45 and begins at step 801 where if the panel 8 is being touched, then the method 2800 proceeds to step 802. Otherwise, the method 2800 proceeds to step 812, where if the panel 8 has been touched previously then the method 2800 proceeds to step 813. Otherwise, the method 2800 concludes.

At step 813, the "don't beep", "no move events" and "event co-ordinate" flags are set. Then at step 814, the message type is set to RELEASE and the method 2800 proceeds to step 805.

The method 2800 continues at step 802, where if this is the first time that the touch has been noticed since there was no touch, then the method 2800 proceeds to step 803. At the next step 803, the CPU45 determines if a bad card has been inserted into the reader 1 by checking the result of step 703, then in the case that a bad card has been inserted into the reader 1, the method 2800 proceeds to step 815. Then at step 815, a BAD Card message is sent to the computer 100, the BAD CARD message is stored in memory 206, and the method 2800 concludes. If it was determined at step 803 that the card 10 was valid, by checking the result of step 703, or that no card was inserted into the reader 1, by the checking of step 701, then the method 2800 proceeds to step 804, where the type of message is set to PRESS in the message header of Fig. 19. At the next step 805, the CPU45 determines the touch coordinates (i.e. X, Y coordinates of user press location) via the touch panel interface 41. Then at the next step 807, the offset and scale functions are applied to the coordinates. The offset and scale functions map the coordinate space of the touch panel 8 to the coordinate space of the card 10. The method

2800 continues at the next step 807, where if the CPU45 determines that the sent message was a MOVE and/or no card was inserted into the reader 1, by checking step 701, then the method 2800 proceeds directly to step 809. Otherwise, the method 2800 proceeds to step 808 and the memory 19 of the card 10 is searched in order to find the first user interface element whose X1, Y1, X2, Y2 values form a range within which the touch coordinates fall and data associated with matched user interface element is read from the card 10. At the step 809, the message is sent along with any data to the associated computer 100, and the CPU 205 in the computer 100 processes the message. The method 2800 continues at the next step 811, where a BEEP sound is sounded and the method 2800 concludes.

If this is not the first time that a touch has been noticed since there was no touch, at step 802, then the method 2800 proceeds to step 816. At step 816, if the touch detected at step 801 was a move, then the method 2800 proceeds to step 817. Otherwise the method 2800 concludes. At step 817, the message type is set to MOVE and the method 2800 proceeds to step 805. For example, a MOVE message can be sent along with the X, Y coordinates of a touch position as defined by Figs. 19 and 22, a PRESS and RELEASE message can be sent along with X, Y coordinates of a touch position and data associated with a user interface object (i.e. one of Indicia 14) as defined by Figs. 19 and 23. If it was determined at step 807 that the message was a MOVE, at step 809, then the CPU 45 sends a MOVE message to the computer 100. The CPU205 processes X, Y coordinates as cursor information and moves a cursor that is displayed on the Video Display 101. In this case, the next RELEASE message can be interpreted as a command to select the displayed object at the cursor position (eg to execute a program, select an item or load a URL). Further, if NO Event Coordinates (see Fig. 13) have been set in the card 10, then the reader 1 may send the data associated with a user interface object to the event

manager 301in the computer 100 or STB 601 without sending the X, Y coordinates of the touch position.

In addition, if the application 304 has a user interface Object structure such as that shown in Fig. 17, and a matching function such as at step 808, then the reader 1 may send X, Y coordinates of a touch position to the application 304. As a result, the CPU 205 executes the same matching function to read data associated with the user interface object from the event manager 301 and provides the card user, a service (e.g. game) identified by a service identifier 1106 associated with the read data. For example, at step 4205 of Fig. 41, the CPU205 determines if data is in the data field of a message. If data is in the data field, then CPU205 reads the data and processes the data at the next steps in Fig. 41. If data is not in the data field, then the CPU205 reads the X, Y coordinates from the message and executes the matching function for the coordinates to get data associated with user pressed indicia. Alternatively, the event manager 301, using the user interface object structure available to the event manager 301, can perform this function.

Therefore, if a card user uses the reader 1 (without inserting a card 10) as a mouse by moving his or her finger on the touch pane l 8, the user can select one of the STB services on a STB menu displayed on the TV display. Also, if the card user uses the reader 1 with an inserted card 10 and selects some indicia 14, the user receives a service (e.g. game) from the computer 100 or STB 601. In particular, if the user selects a START indicia, a desired game can be executed in the computer 100 or STB 601 and an object in the game kicks a ball according to the selection of a KICK indicia 14.

By defining per-card flag values in advance for the card 10, various types of cards 10 can be provided to a user. For example, if a flag (i.e. information) of "NO Move Events" has been set in a card 10 in advance, the reader 1 can be configured to not perform as a mouse based on the flag. On the other hand, if a flag of "NO Move Events"

has not been set in the card 10 in advance, then the reader 1 can be configured to perform as a mouse based on the flag.

As shown in Fig.13, the reader 1 has a default condition in which the reader 1 provides audio feedback, acts as a mouse and sends coordinates for press, release and more events. Alternatively, the reader 1 can provide a default condition in which the reader 1 does not provide audio feedback, act as a mouse and send coordinates.

If the reader 1 is configured to perform the 'beep function' using the per-card flag values, the reader 1 sounds a "beep" and executes a method in accordance with the flow diagrams shown in Figs. 27 and 28. Further, if the reader 1 is configured to perform the 'mouse function' using the per-card flag values, then the reader 1 acts as a mouse and executes a method in accordance with the flow diagrams of Figs. 27 and 28. Still further, if the reader 1 is configured to perform the 'matching function' using the per-card flag values, then the reader 1 sends coordinates for press, release and move events and executes a method in accordance with the flow diagrams of Figs. 27 and 28.

The matching function is also executed in the EM301 as at step 808 of Fig. 28. The card 10 can also be configured as a card having only the mouse function and/or a basic function (e.g. sending the EM301 data associated with indicia selected by a user). Therefore, by combining each per-card flag value randomly, various types of cards 10 can be provided to a user.

As described herein , the service identifier 1106 is an indispensable identifier for the system 600. By sending at least a service identifier 1106 in the distinguishing identifier 1110, to the EM 301, a service can be provided to a user.

The service specific identifier 1107 described above is preferably set by a vendor for use with a particular application. Therefore, if the vendor defines a unique service specific identifier 1107 for each card 10, then the card 10 would be unique. If the service

specific identifier 1107 is being used to provide information about a means by which particular cards have been distributed (e.g. by mail, handed out on a train), then the service specific identifier 1107 can be added to a file which gives a record of which cards have been used to access the service for later use in determining how effective different distribution means have been used.

6.9.4 The Wait 10ms Process

Fig. 29 is a flow diagram showing a wait 10ms routine 2900. The wait 10 ms routine 2900 loops so as to consume CPU cycles until 10ms has elapsed. The delay process 2900 is executed by the CPU 45 and begins at step 901 where a predefined process counter is cleared. At the next step 902, the counter is incremented. Then at the step 903, if 10ms has not elapsed, then the method 2900 returns to step 902. Otherwise the delay process 2900 concludes.

7.0 EVENT MANAGER

The event manager 301 is one of the process components of the software architecture 200. The event manager 301 enforces the rules of the architecture 200 and ensures consistent behaviour between the other process components.

7.1 Role in the System

Most communications pass through the event manager 301 and the event manager 301 is the only component of the architecture 200 that all process components except the directory service 311 components need to be able to directly communicate with. The event manager 301 acts as the enforcer of the rules of the architecture 200, and the event manager 301 does not necessarily have to be configured as one distinct program. The event manager 301 can also be formed of trusted relays or other separate process components that perform part of the event manager role. This can be done for efficiency or security reasons for example.

The event manager 301 may incorporate various other parts of the software architecture 200 such as the I/O daemon 300 and the launcher 303. The event manager 310 may even incorporate an application such as a browser controller.

The event manager 301 can communicate with every process component of the system 600 except the directory service 311 either directly or through a trusted relay. These components include the I/O daemon 300, launcher 303 and any of the applications 304. The event manager 301 can use any suitable communications method to communicate with the other process components. The preferred communication method is Transmission Control Protocol / Internet Protocol (TCP/IP) due to it's nearly universal implementation but other OS specific methods, such as Unix™ sockets, etc can also be used. When the process components are integrated together the method used to communicate can be internal data passing between separate threads.

The event manager 301 is preferably configured to be immune to interference from other process components which includes other processes being able to kill the event manager 301 or being able to starve the event manager 301 of CPU time or network bandwidth. This ensures that the event manager 301 can remain in ultimate control of the system 600.

7.2 Internal Requirements

The event manager 301 performs non-blocking I/O to all the other process components 300, 303, 304 and 306 of the architecture 200 by methods such as polling (NB: polling is not recommended due to the CPU load), interrupt driven I/O, having a separate thread reading and writing from each component or any other suitable method that achieves the same goal. This ensures that one component is not starved out by another component and also reduces user wait time.

The event manager 301 is also configured to check all incoming data for validity and to repair the data if possible before output. This includes data from trusted components. The event manager 301 is preferably also fail safe. If the event manager 301 receives unexpected data from one of the components 300, 303, 304, or 306, then the event manager 301 is configured to deal with the data and not exit unless it is absolutely unavoidable.

The event manager 301 can be required to be running for a considerable length of time and it is configured so as to ensure that performance does not degrade over time. The event manager 301 is preferably configured to assume that the transmission mechanism is reliable for communication with any component that is using a predetermined event manager protocol (i.e. EM-protocol) but assumes that the transmission mechanism used to communicate with the remote reader 1, via the I/O daemon 300, is unreliable and parts of the incoming data may be incorrect or missing.

7.3 Procedures

The event manager 301 is a direct participant in some of the operations of the system 600 but also transparently takes part in many of the other operations of the architecture 200. The event manager 301 is transparent in that it uses data packets as they pass through it without modifying them. The procedures will be explained in more detail below particularly with reference to section 8.0.

Fig. 30 is a flow diagram showing an overview process 3010 of events performed by the system 600 incorporating the software architecture 200. The process 3010, is executed by the CPU 205 depending on the configuration of the system 600. The process 3010 begins at step 3000 where a system initialisation routine is performed, with the initialisation routine including starting the event manager 301. At step 3000 the I/O daemon is typically also started with the event manager 301.

At the next step 3700 the event manager 301 starts the launcher 303. Then at the step 3300, the event manager 301 passes a message to the launcher 303, enabling the launcher 303 to determine which application 304 to execute, and the launcher 303 then starts the corresponding application 304. The process 3010 continues at the next step 3400, where once the currently running application 304 is no longer needed, for instance, when a new card 10 is inserted into the reader 1, the launcher 303 provides an exit message to the running application in order to end the execution of the running application. All applications are terminated when the system 600 is powered down (or switched off).

Fig. 31 is a flow diagram showing a method 3000 of receiving an event performed by the event manager 301. The method 3000 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3000 can be executed by the CPU 4305 in set top box implementations. The method 3000 begins at step 3101, where the launcher 303 is started. At the next step 3103, the event manager 301 receives an event. If the event received at step 3103 is not from the remote reader 1 at the next step 3105, then the method 3000 proceeds to step 3107 where the component identifier (XID) is checked and corrected if necessary. The method 3000 continues at the next step 3109, where if the new application sending an event is allowed to send the event, then the method 3000 proceeds to step 3111. At step 3111, the event is sent to a destination process component and the method 3000 returns to step 3103. If the sending application is not allowed to send the event at step 3109, then the method 3000 proceeds to step 3113, where the event is dropped and the method 3000 returns to step 3103.

If the event is from the remote reader 1 at step 3105, then the method 3000 proceeds to step 3115. If the event is a BADCARD, LOWBAT, INSERT or REMOVE event at step 3115 then the method 3000 proceeds to step 3117. Otherwise the method 3000 proceeds to step 3119. At step 3117, the event is passed to the launcher 303 and the

method 3000 returns to step 3103. If the distinguishing identifier is the NO_CARD identifier at step 3119, then the corresponding message is passed to the launcher 303 at step 3117. Otherwise the method 3000 proceeds to step 3121, where the service identifier portion of the distinguishing identifier is compared with the service identifier used in determining the current front application. If the service identifier is not the same as that which has been used to determine the front application and the service identifier portion of the distinguishing identifier is not the special generic service identifier, then the method 3000 proceeds to step 3117 where this message is passed to launcher 303 . Otherwise, the method 3000 proceeds to step 3123, where the event is sent to the front application and the method 3000 returns to step 3103.

7.4 Focus Change

The event manager 301 can safely ignore any EM_LOSING_FOCUS events that are not for the current front application. The event manager 301 needs to watch for EM_GAINING_FOCUS messages for which applications becoming the front application as well as the service identifiers that are associated with that application. The event manager 301 can safely ignore multiple EM_GAINING_FOCUS events that are to the same application with the same service identifier as well as any EM_LOSING_FOCUS events to applications that are not the currently front application. Messages that are ignored are passed on as normal.

7.5 Reader Messages

The event manager 301 is also responsible for distributing the messages to the correct component. The event manager 301 is configured to follow certain predetermined protocol rules, which will be described in detail below.

7.6 Restrictions on Sending Messages

A further role of the event manager 301 is to enforce predetermined restrictions on the transmitting of messages.

8.0 EVENT MANAGER PROTOCOL

The event manager protocol (EM-protocol) is the protocol used to communicate between all components of the architecture 200 except for the directory service 311. Generally all messages are configured to go through the event manager 301 before being passed onto an intended recipient. The EM-protocol is a datagram based protocol that is implemented on top of a reliable communications protocol, for example, Transport Control Protocol/Internet Protocol (TCP/IP). The event manager 301 is configured to assume that all data being sent will arrive unchanged and in the correct order. The event manager 301 does not assume that there is a reliable method of synchronisation between the process components of the architecture 200.

All multi-byte values are sent in Internet byte order (i.e. big-endian). The exception to this is the 'distinguishing identifier' values representing services, which are sent as blocks of several single bytes and are always treated as such (i.e. the distinguishing identifier values are never stored as a number typically because of the byte ordering issues).

8.1 Communication Methods

The event manager protocol is preferably configured to assume a TCP/IP like method of communication between the components of the architecture 200 and the system 600 hardware components. Alternatively, any known method of communication that ensures reliable transport can be used. For example, an operating system specific method such as 'Unix sockets' can be used. The data can be passed between the process components 301, 303, 304 and 306 directly via internal data structures in a multi-threaded application, for example.

In the case of architectures where an alternative method of communication between the components is being used, the problem of byte-ordering must be taken into account. If it is possible that applications can run on a machine that has different byte orderings or is required to communicate with components that expect the data in network byte order, which all components assume by default, then all affected communications can be done in network byte order.

8.2 Data Format

8.2.1 Basic data types

Some abbreviations that are used in the following paragraphs to refer to data types are as follows:

- int8: An eight bit signed value;
- uint8: An eight bit unsigned value;
- int16: A 16 bit signed value;
- uint16: A 16 bit unsigned value;
- int32: A 32 bit signed value;
- uint32: A 32 bit unsigned value; and
- xid_t: A 32 bit unsigned value.

8.2.2 Component Addressing

Every addressable process component in the architecture 200 is assigned a 32 bit unsigned value referred to as an 'xid' (or component identifier). This number is unique within the boundaries of each individual system 600 instance. Some xids of the process components are always the same. These are:

- Event Manager 301: EM_EVENT_MANGER_XID
- Master Launcher : EM_MASTER_LAUNCHER_XID
- Launcher 303: EM_FIRST_APP_XID

- Display Manager 306: EM_DISPLAY_MANAGER_XID

The xid value is divided up into a one byte type field and a three byte identifier. The different types are shown in Table 1 below.

Table 1

Value	Type
Internal xid's	These xid values are not routable and can be used internally by all components. They are dropped if seen by the EM
Core System xid's	These identify the core system components of a user interface Card system. These components include the EM, Launcher and Master Launcher.
Standard Application	These identify standard applications that are started and ended by the Launcher as needed.
Special application	These identify special applications that aren't controlled by the standard rules for starting and ending applications. They are applications that are written to provide the user interface card system with functionality that can be controlled by other applications such as a video on demand player or a browser controller.
Readers	Readers are assigned xids by the EM. These xids are unique to each reader that is used to access the system for the duration of the EM. If the event manager and therefore the system is restarted then the reader xids will change.

8.3 Message types

There are twenty-two core messages in the EM-protocol, which preferably have the following labels:

- EM_NEW_LAUNCHER
- EM_KILL_LAUNCHER
- EM_APP_REGISTER
- EM_EXIT_NOW
- EM_CLOSE
- EM_APP_STARTING
- EM_APP_DYING
- EM_GAINING_FOCUS
- EM_LOSING_FOCUS
- EM_LIST_MESSAGES
- EM_LIST_APPS
- EM_SEND_MESSAGE
- EM_POST_MESSAGE
- EM_GET_MESSAGE
- EM_DELETE_MESSAGE
- EM_READER_INSERT
- EM_READER_REMOVE
- EM_READER_BADCARD
- EM_READER_MOVE
- EM_READER_PRESS
- EM_READER_RELEASE
- EM_READER_LOW_BATT

These messages will be explained in more detail in the following paragraphs.

8.3.1 Message Header

The messages sent within the system 600 have a header portion preferably including the following information:

version: This represents the version number of the protocol being used by the component. This should always be set to EM_PROTOCOL_VERSION, which is defined in library headers to be the version used by the library.

type: This represents the type of message that a header proceeds and is set to one of the message types listed above and described below. The length of the messages is assigned the label dataLength .

reserved: This represents that the value in these two bytes is reserved and should be set to zero.

timestamp: This represents the timestamp of a data packet.

to_xid: This represents the destination xid of a particular packet. This is the final destination of the packet and should only be set to the event manager if that is the intended final recipient.

from_xid: This represents the source xid of the packet.

dataLength: This represents the length of the data that follows a header. This value can be zero. Different types of messages impose different requirements on the data following the message header. Components should not assume the length of a message from the type. The number of bytes in the dataLength field is always read even if this is different to the correct size of the message to insure that the stream can only be corrupted by an incorrect dataLength.

8.3.2 EM_NEW_LAUNCHER

The EM_NEW_LAUNCHER message is sent when the event manager 301 requires a new launcher 303. This message is only sent between the event manager 301 and the

Master Launcher if the software architecture 200 includes such a Master Launcher. The packet containing this message also contains information that a new launcher needs to connect to the event manager 301. The EM_NEW_LAUNCHER message preferably includes the following information:

port: This represents the port number that the event manager 301 is listening for new connection on.

host: This represents the host name of the machine running the event manager 301.

8.3.3 EM_KILL_LAUNCHER

The EM_KILL_LAUNCHER message is sent when the event manager 301 wants the Master Launcher to kill the current launcher 303. The EM_KILL_LAUNCHER message has no data associated with it.

8.3.4 EM_APP_REGISTER

The EM_APP_REGISTER message is sent when an application is starting up to the launcher 303 and informs the rest of the components of the architecture 200 that it is now ready to receive messages. Any messages that an application 304 sends before it has registered will be discarded by the event manager 301.

The EM_APP_REGISTER message preferably includes the following information:

xid: This represents the component identifier that was assigned to the application by the associated launcher 303. The remainder of the information sent cannot be represented by the structure as the remaining fields are of variable length. The data following the xid is a series of null terminated strings with a maximum length of 256 characters not including the terminating null, consisting of the lower and upper case characters a-z, the numbers 0-9 and the characters (.,-_). If the strings are longer than 256 characters they will be truncated at 256 characters.

Application Name: this represents a name that is used to identify the present application to other applications.

Service Group: this represents one or more names of service groups that the application wishes to be a part of.

An application that is persistent, such as a browser controller, only needs to register once. Such a persistent application does not need to register every time it gets an EM_GAINING_FOCUS event.

8.3.5 EM_EXIT_NOW

The EM_EXIT_NOW message is sent by the launcher 303 to an application when the application is about to be forced to exit. The EM_EXIT_NOW message has no data associated with it.

8.3.6 EM_CLOSE

The EM_CLOSE message is sent to persistent applications to indicate that the current session is closed and to return the application to its startup state. Once this message is received by an application, the application is required to treat the next EM_GAINING_FOCUS event as the start of a new session rather than as a change in input/output focus. The EM_CLOSE message has no associated data.

8.3.7 EM_APP_STARTING

The EM_APP_STARTING message is sent by the launcher 303 to the event manager 301 when an application is about to start. The EM_APP_STARTING message preferably includes the following information:

xid: This represents the component identifier of the application that is about to start.

8.3.8 EM_APP_DYING

The EM_APP_DYING message is sent by the launcher 303 to the event manager 301 when an application has exited. The EM_APP_DYING message is sent only after the launcher 303 is certain that the application has finished. The EM_APP_DYING message preferably includes the following information:

xid: This represents the component identifier of the application that has exited.

8.3.9 EM_GAINING_FOCUS

The EM_GAINING_FOCUS message is sent to an application by the launcher 303 when the application 304 is about to start receiving input from the remote reader 1. The EM_GAINING_FOCUS message preferably includes the following information:

id: This represents the distinguishing identifier of the remote reader 1 messages that will be sent to an application.

Data: This represents extra data that is to be sent to the application when it is about to receive focus. This is specific to each service and it is up to the application to interpret the data. The extra data is not checked for byte ordering issues and this should be dealt with by the application. Any multi-byte data is sent by applications in network byte order and assumed to be in this order by the receiving application. An example of this data, when the receiving application is a browser controller, is a URL which the browser controller is being instructed to load.

8.3.10 EM_LOSING_FOCUS

The EM_LOSING_FOCUS message is sent when an application 304 is about to lose input/output focus from the remote reader 1 and the display 101. The EM_LOSING_FOCUS message has no extra data.

8.3.11 EM_LIST_APPS

The EM_LIST_APPS message is sent when an application wishes to know what other applications are also running at a point in time. The EM_LIST_APPS message is returned to the application with the data field containing the application list. This message does not need to be addressed to any of the process components 301 to 306. The event manager 301 ensures that the EM_LIST_APPS message is sent to the correct component, which is usually the launcher 303, regardless of the to_xid field of the header. It is the role of the receiving component to decide which applications to list.

When used as a reply, the EM_LIST_APPS message has two formats. The first is the format used when the EM_LIST_APPS is sent as a request and the second is the format when it is sent as a reply. The request has no extra data associated with it.

The EM_LIST_APPS message preferably includes the following information:

app_xid: This represents the xid of the application being described.

app_desc: This represents the name string given to the launcher 303 when the application first registers.

8.3.12 EM_SEND_MESSAGE

The EM_SEND_MESSAGE message can be sent between any two concurrently running applications in the system 600. There is no structure imposed on this message by the architecture 200 but communicating applications need to agree on a common data structure.

8.3.13 EM_LIST_MESSAGES

The EM_LIST_MESSAGES message is used to get a list of all messages currently on a message board, which is used in the architecture 200. The message board will be described in more detail below with reference to section 8.4.7.1. The EM_LIST_MESSAGES message should be sent to the launcher 303. The

EM_LIST_MESSAGES message has a request and reply format. The request format has no data associated with it. The reply preferably includes the following information:

message_count: This represents the number of messages currently on the message board and can be equal to zero.

Messages: This represents a variable number (i.e. equal to message_count) of variable sized structures that have the following structure:

Each message preferably includes the following information:

message_id: This represents the message identifier of this message.

poster_id: This represent the xid (component identifier) of the component that posted this message.

mime_type: This represents the Multipurpose Internet Mail Extention-type (MIME-type) of the data associated with this message and is a null terminated string which can be of zero length in which case the terminating zero is still present.

message_desc: This represents the description of this message that was assigned when the message was posted by the posting application. This is a null terminated string that is at most 255 characters long not including the terminating zero. The length of this string can be zero in which case the terminating zero is still present.

8.3.14 EM_POST_MESSAGE

The EM_POST_MESSAGE message is used to post some data to the message board used in the architecture 200. These messages last until there is a service group change and can be accessed by any application that is running. The EM_POST_MESSAGE messages can also be deleted by any currently running

application and are not assumed to be totally reliable. Once the message has been posted it is returned to the application that posted it to inform the application of the message identifier of the message. These messages are sent to the launcher 303 by the application. The message from the application (i.e. the application that posted the message) includes the following information:

message_desc: This represents a description of the message and is a null terminated string that can be at most 255 characters long not including the terminating zero. The description can be zero bytes in length but must still have a terminating zero.

mime_type: This represents the MIME type of the message data that is being posted. The MIME type is not required but there must still be a terminating zero.

message_data: This represents the data to be posted to the message board.

The message returned to the application preferably includes the following information:

message_id: This represents the message identifier by which this message can be retrieved or deleted .

8.3.15 EM_GET_MESSAGE

The EM_GET_MESSAGE message is used to retrieve a message from the message board. It is sent containing the message identifier of the message that the component wishes to retrieve and it is returned to the component either containing the message or an error that there is no message with that identifier. These messages are sent to the launcher 303 by an application 304.

The information included when requesting the message is as follows:

message_id: This represents the message identifier of the message the application wishes to retrieve.

flags: This is a flags word. All unused bits should be set to zero. The flag description is shown in Table 2 below:

Table 2

Flag	Description	Value
EM_GM_DELETE	Delete the message from the message board after it has been sent	0x01

The reply has the following information:

error: If an error occurred then this will be set to one of the values in Table 3 below.

Table 3

Value	Description
EM_GM_NO_ERROR	No error occurred. The message is in the message field.
EM_GM_NO_SUCH_MESSAGE	No message exists with that message identifier on the message board.

message_id: This represents the message identifier of the message that was retrieved.

mime_type: This represents the MIME type of the message that was retrieved. This is a null terminated string. If this message has no MIME type associated with it then the string is zero length but the terminating zero is still present.

message: If no error occurred then this field will contain the data posted on the message board. The length is determined by the dataLength value in the header minus the size of the error field

8.3.16 EM_DELETE_MESSAGE

The EM_DELETE_MESSAGE message is used to delete messages from the message board. It is not an error to delete a message that does not exist. These messages are sent to the launcher 303 by the front application. The EM_DELETE_MESSAGE preferably includes the following information:

message_id: This represents the message identifier of the message that is to be deleted.

8.3.17 user Interface Card Reader Messages

The user interface card reader messages are generated by the remote reader 1 and are encapsulated by the event manager 301 so that they conform with the event manager protocol. There are three types of messages that are generated by the remote reader 1. These messages are "simple" messages, "move" messages and "press/release" messages. Move messages are simple messages with co-ordinates added, and press/release messages are simple messages with data and coordinates added.

8.3.17.1 Simple Messages

The following messages are simple messages:

- EM_READER_INSERT
- EM_READER_REMOVE
- EM_READER_BADCARD
- EM_READER_LOW_BATT

These simple messages preferably include the following information:

id: This represents the distinguishing identifier that was sent by the remote reader 1 and has no meaning for BADCARD messages.

8.3.17.2 Move Messages

The EM_READER_MOVE messages preferably include the following information:

id: This represents the distinguishing identifier that was sent by the remote reader 1, and is set to all zeros for no card messages.

X: This represents the x value.

Y: This represents the y value.

8.3.17.3 Press/Release Messages

EM_READER_PRESS and EM_READER_RELEASE messages preferably includes the following information:

id: This represents the distinguishing identifier that was sent by the remote reader 1.

x: This represents the x value.

y: This represents the y value.

data: This represents any data that was associated with the press or release (associated with the user interface-element data).

8.4 Procedures

The following paragraphs describe the main procedures that each process component of the architecture 200 follow.

8.4.1 Starting a new Application

Fig. 32 is a flow diagram showing detail of the method 3300 of starting a new application and performed whenever the launcher 303 starts a new application. The method 3300 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3300 can be executed by the CPU 4305 in set top box implementations. The method 3300 begins at the first step 3301 where the launcher 303 performs a mapping to translate the service identifier into a URL. At the next step 3303, the launcher 303 fetches and starts the application informing it of an event manager host-name and port number. The method 3300 continues at the next step 3305, where the launcher 303 sends the event manager 301 an EM_APP_STARTING message informing

the event manager 301 of the xid of the starting application. At the next step 3307, the new application connects to the event manager 301 and sends the launcher 303 an EM_APP_REGISTER message. Further, there is normally a focus change to the new application.

8.4.2 Ending an Application

Fig. 33 is a flow diagram showing a method 3400 of ending an application in the system 600 incorporating the software architecture 200. The method 3400 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3400 can be executed by the CPU 4305 in set top box implementations. This method is used whenever the launcher 303 terminates a running application. The method 3400 begins at step 3401, where the launcher 303 sends the running application an EM_EXIT_NOW message. The launcher 303 sets a time out at this point to give the application a chance to exit cleanly. At the next step 3403, the running application cleans up and exits. Alternatively, the application ignores the EM_EXIT_NOW message and the launcher 303 times out and forces the application to quit. Then at step 3405, the launcher 303 sends the event manager 301 an EM_APP_DYING to tell it that the application has exited and that the launcher 303 should discard any waiting data and close the connection to the application if the connection is still open, and the method 3400 concludes.

8.4.3 Closing a persistent application's session

Fig. 34 is a flow diagram showing a method 3500 of closing the current session of a persistent application on the system 600 incorporating the software architecture 200. The method 3500 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3500 can be executed by the CPU 4305 in set top box implementations. The method 3500 is analogous to the application ending but the application does not actually close. The method 3500 begins at step 3501, where the

launcher 303 sends the persistent application an EM_CLOSE message. At the next step 3503, the persistent application resets to its initial state, and the method 3500 concludes. This may involve closing connections to outside servers, loading a default web page etc. The next EM_GAINING_FOCUS event that the persistent application receives is assumed to be the start of a new session.

8.4.4 Focus Change

Fig. 35 is a flow diagram showing a method 3600 of performing a focus change on the system 600 incorporating the software architecture 200. The method 3600 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3600 can be executed by the CPU 4305 in set top box implementations. The method 3600 is used to tell an application that it is about to gain or lose input/output focus, which is not a signal for the application to exit. At the first step 3601, the launcher 303 makes the decision to change the application that currently has input/output focus and sends the application that is to receive input focus an EM_GAINING_FOCUS event typically based on a card change. The sending of this event is used by the event manager 301 to decide which application should receive input/output focus based on predetermined conditions. Then at the step 3603, the launcher 303 sends the previous front application an EM_LOSING_FOCUS event, and the method 3600 concludes. This message is less critical and is not sent when the current front application remains the same, but still needs the EM_GAINING_FOCUS (i.e. in the case of a browser controller where the EM_GAINING_FOCUS events are used to tell the browser controller 402 the base URL).

8.4.5 Message Passing

There are two distinct types of message passing between applications supported by the architecture 200. Through the message board that is as persistent as the current

service group, and a direct message method where two components communicate with each other directly as described below.

8.4.5.1 Message Board

One component of the architecture 200, typically the launcher 303, maintains a message board and the event manager 301 knows which component does this. The message board is formed of a list of messages that are assigned a 32 bit unsigned number as an identifier by the process component managing the message board. The messages are formed of a text description, an optional MIME type for the message data and the message itself. An application can request a list of all messages currently on the message board by sending an EM_LIST_MESSAGES message. This will return with the text descriptions of all messages currently on the message board with their associated message identifiers. The application can then request a specific message by sending a EM_GET_MESSAGE with the message identifier of the message that it requires. It is possible that a message could be deleted between getting a listing of the message board and actually requesting a message. The error field of the EM_GET_MESSAGE message reply is configured to indicate this.

8.4.5.2 Direct Communication

Two applications can send each other arbitrary data directly, by using direct communication. This is performed by one application sending the other application the data by using an EM_SEND_MESSAGE message. The two applications need to agree on a data format for these messages and byte ordering issues need to be taken into account. To get the component identifier of the other application, an application can request to be sent a list of all running applications by sending a EM_LIST_APPS message. This message returns a list of all publicly visible applications that are currently running.

8.5 Reader Messages

This section outlines the rules used by the event manager 301 to route the EM_READER_* messages. The following messages are always sent to the launcher 303 regardless of which application currently has focus.

- EM_READER_INSERT
- EM_READER_REMOVE
- EM_READER_BADCARD
- EM_READER_LOW-BATT

The following messages are sent to the currently front application if the messages are from cards 10 that have the same service identifier in their corresponding fields1106 as the currently front application. A service-specific identifier is not taken into account in this comparison. If the service identifier is different to the currently front application or the distinguishing identifier is the NO_CARD present value (i.e. all zeroes) then the message is sent to the launcher 303 as previously described.

- EM_READER_PRESS
- EM_READER_RELEASE
- EM_READER_MOVE

8.6 Restrictions on Sending Messages

To improve the security and stability of the system 600, there are preferably restrictions placed on the sending of messages. Any messages that breach these rules will be discarded by the event manager 301.

8.6.1 Restrictions for all components

No component except the remote reader 1 will be allowed to send EM_READER_* messages.

8.6.2 Restrictions on the event manager

The event manager 301 is the enforcer of the rules and as such can send any messages necessary. The event manager 301 is configured to only need to generate EM_KILL_LAUNCHER and EM_NEW_LAUNCHER messages but it can copy messages and send the copies to process components that are not the target component. The event manager 301 also handles all transmissions between components.

8.6.3 Restrictions on the Launcher

The launcher 303 sends messages to all components 301 to 306 of the architecture 200. The messages that the launcher 303 can not send are as follows:

- EM_KILL_LAUNCHER
- EM_NEW_LAUNCHER

8.6.4 Restrictions on Applications

Applications only send the following messages to other applications (which includes the launcher 303):

- EM_APP_REGISTER
- EM_SEND_MESSAGE
- EM_LIST_APPS
- EM_POST_MESSAGE
- EM_GET_MESSAGE
- EM_DELETE_MESSAGE
- EM_LIST_MESSAGES

8.7 Component Procedure Lists

This section lists the functions, which each component of architecture 200 is involved in.

8.7.1 Event Manager

The event manager 301 is a direct participant in the following procedures:

- System Initialisation

- System Startup
- Starting a new Application
- Ending an Application
- Focus Change
- Message Passing
- Reader Messages

8.7.2 Launcher

The Launcher 303 is a participant in the following procedures:

- System Initialisation
- System Startup
- Starting a new Application
- Ending an Application
- Focus Change
- Message Passing (in some instances)
- Reader Messages (in some instances)

8.7.3 Applications

The Applications 304 are participants in the following procedures:

- Starting a new Application
- Ending an Application
- Closing a session if the application is persistent.
- Focus Change
- Message Passing
- Reader Messages (in some instances)

9.0 I/O DAEMON

The I/O daemon 300 is responsible for transporting the data being sent from the remote reader 1 to the event manager 301, and vice versa for a two-way protocol. The I/O daemon 300 is configured to be able to read from the hardware of the system 600 either directly or through operating system drivers that are interface with the remote reader 1, for example, an IR link or standard serial hardware connection. The I/O daemon 300 is also required to listen on a TCP/IP port to wait for the event manager 301 to connect, at which point the I/O daemon 300 sends data from the remote reader 1 to the event manager 301 encapsulated in a TCP/IP stream.

The I/O daemon 300 does not communicate with the rest of the system 600 except to send the remote reader 1 data to the event manager 301, and vice versa in optional two way protocol arrangements between the I/O daemon 300 and the remote reader 1.

While the functionality of the I/O daemon 300 must be present in the system 600, the I/O daemon 300 does not have to be a separate component. For example, the I/O daemon 300 can be integrated into the event manager 301 if the event manager 301 is running on the same machine as the hardware used to interface with the remote reader 1.

The I/O daemon 300 is configured to run on minimum hardware for the instance where the rest of the system 600 is running remotely.

9.1 Requirements

9.1.1 General Requirements

The platform upon which the I/O daemon 300 is implemented must be configured be able to receive signals from (and optionally transmit signals to) a remote reader 1. The platform also preferably has a TCP/IP stack or other reliable communications method implemented on it to communicate with the other parts of the system (i.e. the event manager (EM) 301). The I/O daemon 300 can be required to do multiplexed I/O, and the I/O system of the architecture 200 is preferably configured to support multiplexed I/O.

The architecture 200 is preferably configured to assign a port that the I/O daemon 300 will be listening on, for example, as a command line argument.

9.1.2 Internal Requirements

The I/O daemon 300 is not required to understand the protocol used by the remote reader 1. The I/O daemon 300 is only required to forward all data that it receives to any listening EM (event manager). The I/O daemon 300 is not required to correct any errors of transmission from the remote reader 1 unless it is supported by the transport protocol of the communications link (i.e. through error correcting codes or similar). If the transport protocol being used supports error detection but not correction then any data that does not pass the error check can be passed onto the event manager 301.

9.1.3 External Interface Requirements

The I/O daemon 300 is preferably able to accept one or more TCP/IP connections. The data stream that is sent to the event manager 301 is the content of the data sent by the remote reader 1. All header and footer information that is transmitted as part of the communications protocol used is preferably stripped off and the byte ordering is big endian. If the communication method of the architecture 200 ever becomes unusable (e.g. due to an error arising) then the I/O daemon 300 closes all connections as soon as the error condition arises.

9.2 External Interface

The external interface (not shown) of the I/O daemon 300 is intentionally simplistic to allow it to be run on minimum hardware. The I/O daemon 300 is preferably configured in the following manner.

9.2.1 Start-up Procedure

The I/O daemon 300 listens on a TCP/IP port that is specified to it in some manner, for example, by command line arguments. The exact method of informing the I/O

daemon 300 of the TCP/IP port is implementation specific. The communications hardware used to communicate with the remote reader 1 is initialised if required and the method to read data that is sent from the remote reader 1 is configured to be ready to receive data. While the I/O daemon 300 is waiting for a connection, the I/O daemon 300 consumes the data that is being sent by the remote reader 1 so that when a connection is made, only new data is being sent. This new data is not required to start on a message boundary.

9.2.2 Connection from an event manager

If a connection arrives on the TCP/IP port then the I/O daemon 300 is configured to accept the connection and begin transmitting any data received from the remote reader 1 down the connection. If the I/O daemon 300 is already connected to an event manager (EM) 301 then the I/O daemon 300 has two options. Firstly, the I/O daemon can accept the connection and send all data down all currently connected event managers. This option is provided for system debugging purposes. The second method is to reject the second connection and continue to send the data to the already connected EM. Any encryption of the stream can be handled externally by some other method, such as port tunnelling.

9.2.3 Connection from an event manager closing

If at any time the connection to the event manager 301 is closed, then the I/O daemon 300 is configured to discard any data from the remote reader 1 that is waiting to be sent to that event manager 301. If this is the only event manager connected then the I/O daemon 300 is configured to return to an initial startup state whereby the I/O daemon 300 consumes data being sent by the remote reader 1 and waits for a connection.

9.2.4 Unrecoverable error is encountered

If the I/O daemon 300 detects an error that cannot be dealt with and will cause the I/O daemon 300 to exit, then the I/O daemon 300 is configured to close all connections to any EMs to inform the EMs that the I/O daemon 300 has detected an error. Examples of these errors include if the hardware that is being used to communicate with the remote reader 1 becomes unavailable or if the I/O daemon 300 receives a signal that would cause it to exit. The I/O daemon 300 is configured to close all connections as soon as an error is experienced.

10.0 LAUNCHER

The launcher 303 is the process component that enforces site specific rules such as allowed applications and basic application configuration rules. The launcher 303 allows the other component processes 300, 301, 304, 305 and 306 of the system architecture 200 to be used in a wide range of applications from a general home set top box 601 to a very specific application (e.g. an automatic teller machine (ATM)). A launcher 303 can be specifically written for each network or installation.

The launcher 303 is configured with special privileges. For example, the launcher 303 can be configured to be the first component to connect to the event manager 301 as the system 600 starts up. Further, the launcher 303 receives all "LOW_BATT", "BADCARD", "INSERT", and "REMOVE" messages sent by the remote reader 1 and also receives all "PRESS", "RELEASE" and "MOVE" messages that originate from a card other than the smart card 10 that the front application is associated with at any one point in time. The launcher 303 also receives PRESS, RELEASE and MOVE messages with a special "NO_CARD" distinguishing identifier. The launcher 303 also has control over which application is the front application via the EM_GAINING_FOCUS and EM_LOSING_FOCUS events.

The launcher 303 is configured to decide when applications need to be started and made to exit. The launcher 303 is also used to start and stop applications although this is not always the case. This role can be undertaken by another application at the instruction of the launcher 303, for instance, in the case where the applications 304 are run on separate machines to the rest of the components of the architecture 200.

The events that are sent to the launcher 303 instead of being sent to the current front application allow the launcher 303 to make decisions on which application(s) are to be running at the any moment in time and being configured to force applications to exit means that the launcher 303 can enforce which applications are to be currently running. The launcher 303 is also required to inform the event manager 301 when it is starting and stopping applications.

Fig. 36 is a flow diagram, showing an overview of the method 3700 performed by the launcher 303. The method 3700 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3700 can be executed by the CPU 4305 in set top box implementations or by the CPU of a remote server. The method 3700 begins at the first step 3701, where the launcher 303 connects to the event manager 301, and then continues to a next step 3702 where persistent applications are started. At the next step 3703, the launcher 303 waits for an event and when an event is received the launcher 303 proceeds to step 3705. If the event is the NO_CARD identifier at step 3705, then the process proceeds to step 3707. Otherwise the method 3700 proceeds to step 3709. At step 3707, the launcher 303 performs a predetermined system specific function (e.g. displays a message on the display 101) in response to the NO_ CARD identifier and the method 3700 returns to step 3703.

If the event at decision step 3705 is determined not to be a NO_CARD identifier, another decision step 3709 is entered to determine whether or not the event is a PRESS,

RELEASE, REMOVE or MOVE. If this decision step 3709 returns a "yes", that is, the event is one of the aforementioned events, then the method 3700 proceeds to step 3800. Otherwise the method 3700 proceeds to a further decision step 3713. At step 3800, the launcher 303 changes the application in accordance with the process steps described with reference to the flow diagram Fig. 37. The method 3700 returns to step 3703.

If the event at step 3709 is not one of the PRESS, RELEASE, REMOVE or MOVE events, then a decision step 3713 is entered. This decision step 3713 makes a determination on a BADCARD or LOW_BATT event. If the event is a BADCARD or LOW_BATT event at step 3713, then the method 3700 proceeds to step 3715, otherwise the method 3700 proceeds to step 3717. At step 3715, the launcher 303 gives the user feedback on the event that has occurred (e.g. displaying a "Low Battery" message on the display 101 if the LOW_BATT event is determined or a "Incorrect Card" upon determination of a BADCARD event) and the method 3700 returns to step 3703. If the event at decision step 3713 is neither a BADCARD or LOW_BATT event, then step 3717 is entered.

If the event is an APP_REGISTER event at step 3717, then the method 3700 proceeds to step 3900, "Application Registering". Otherwise the method 3700 proceeds to step 3725. At step 3900, the application is registered as described herein with reference to Fig. 38 (i.e. the application informs the other components 301, 302 and 306 that it is now ready to receive messages, as described above with reference to section 8.3.4) and the method 3700 returns to step 3703. A method of registering an application in accordance with step 3900, will be described in more detail below with reference to the flow diagram of Fig. 38. At step 3725, the event is discarded and the method 3700 returns to step 3703.

Fig. 37 is a flow diagram showing the method 3800 of changing an application, which is performed by the launcher 303. The method 3800 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3800 can be executed by the CPU 4305 in set top box implementations or by the CPU of a remote server. The method 3800 begins at step 3817, where if a REMOVE message has been received by the launcher 303 then the process proceeds directly to step 3813. Otherwise, the method 3800 continues to decision step 3801. At decision step 3801, if the service represented by the event is associated with an application that is registered, then the method 3800 proceeds directly to step 3819. Otherwise, the method 3800 continues to step 3803, where a service identifier lookup is performed to determine the location and/or name of a new application and any initial data associated with the new application. For example, the initial data may be a URL to load into a browser 403 or a media file to be loaded into a media player application. At the next step 3805, if the application is already running the method 3800 proceeds to step 3819. Otherwise, the method 3800 proceeds to step 3809, where the new application is retrieved from applications 304. At the next step 3811, the new application is started as the front application, and at step 3812 the event manager 301 is notified of the component identifier (Xid) of this new front application.

Decision step 3819 is entered either from step 3801 if the service represented by the event is associated with an application that is registered or if the application is already running. At step 3819, if it is determined that an INSERT message is received by the launcher 303, then the method 3800 concludes. Otherwise, the method 3800 proceeds to step 3807, where the new application is sent a GAINING_FOCUS event indicating that the new application will soon be changing state. After the new application is sent a GAINING_FOCUS event, or as a result of a REMOVE event detected at decision step 3817, control is passed to decision step 3813. At step 3813 it is determined if there is an

existing front application, if there is no previously front application, then method 3800 concludes. Otherwise, a LOSING_FOCUS event is sent to the previous front application enabling the previous front application to complete immediate tasks, before the method 3800 concludes.

Fig. 38 is a flow diagram showing the method or process 3900 of registering a new application, which is performed by the launcher 303. The method 3900 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3900 can be executed by the CPU 4305 in set top box implementations, or by the CPU of a remote server. The process 3900 begins at step 3901, where a new service group list, including the application, referred to with reference to step 3900 of Fig. 36, is generated. At the next step 3903, a GAINING_FOCUS event is sent to this application. Then at the step 3905, if any applications are not part of the new service group and are not persistent, the method 3900 proceeds to step 3907. Otherwise the method 3900 concludes. At step 3907, any applications which are not part of the service group are sent an EXIT_NOW event, and the method 3900 proceeds to a next step 3908 where the event manager 301 is notified that the applications, which were not part of the new service group, have been terminated. The method 3900 then concludes.

Fig. 39 is a flow diagram showing the process steps 4000 performed by an application when receiving events from the launcher 303. The method 4000 can be executed by the CPU 205 for computer implementations. Alternatively, the method 4000 can be executed by the CPU 4305 in set top box implementations or by the CPU of a remote server. The method steps 4000 begins at step 4001, where the launcher 303 connects to the event manager 301 and then the method 4000 proceeds to step 4002. At step 4002, the application is registered by sending an APP_REGISTER message to the launcher 303. Following the flowchart shown in Fig 39, to the next step 4003, the

application waits for events and when an event is received the process proceeds to step 4005. If the event is a GAINING_FOCUS event at step 4005, then the method 4000 proceeds to step 4007. Otherwise the method 4000 proceeds to step 4009. At step 4007, the application is initialised if necessary, optionally using the distinguishing identifier and optionally using the data field of the GAINING_FOCUS event. This data field used for initialisation may include a URL to load, a filename to load, etc. Control returns to waiting for events at step 4003.

If the event is a PRESS, RELEASE or MOVE event at step 4009, then the method 4000 proceeds to step 4011. Otherwise the method 4000 proceeds to step 4013. At step 4011, an application specific action is performed in response to the event. The application specific action is performed using data from the event (i.e. data associated with an indicium on the card 10, (eg URL, character or video name)), the X/Y position or distinguishing identifier or any combination of these.

The application specific action is typically associated with an indicium on the card 10. For example, an indicium can be associated with a particular URL and when the indicium is pressed the URL may be accessed. Therefore, for example, the computer 100 or STB 601 can download desired programs from a Web Page that was designated by the URL, and a card user can receive the service (i.e program download) from the system 600. Further, an indicium can be associated with a particular memory address and when the indicium is pressed the memory address can be used to data store at the memory address. Therefore, for example, the computer100 or STB 601 can download desired image data from memory or from a file server on a network, which was designated by the memory address, and a card 10 user can receive the service (e.g. image data download) from the system 600. After step 4011, the method 4000 returns to step 4003 as shown in Fig. 39.

The process steps 4000, according to the flowchart of Fig. 39 as described above, filters through to step 4013 if an event is not determined to be any one of a GAINING_FOCUS, PRESS, RELEASE or MOVE event at the corresponding decision steps 4005 or 4009. If the event is a LOSING_FOCUS event then at step 4013 the method 4000 proceeds to step 4015. Otherwise the method 4000 proceeds to decision step 4017. At step 4015, the application reverts to an inactive state and the method 4000 returns to step 4003. If the event is an EXIT_NOW event at step 4017, then the method 4000 concludes. Otherwise the method 4000 proceeds to step 4019, where the event is ignored and the method 4000 returns to step 4003.

Fig. 40 is a flow diagram showing the method 4100 performed by the browser controller 402 application when receiving events from the launcher 303. The method 4100 can be executed by the CPU 205 for computer implementations. Alternatively, the method 4100 can be executed by the CPU 4305 in set top box implementations, or by the CPU of a remote server. The method 4100 begins at step 4101, where the browser application sends an APP_REGISTER message to the launcher 303. At the next step 4103, the browser application waits for events and when an event is received the method 4100 proceeds to step 4105. If the event is a GAINING_FOCUS event at step 4105, then the method 4100 proceeds to step 4107. Otherwise the method 4100 proceeds to step 4109. At step 4107, the application is initialised if necessary. For example, the application reads the data field of the GAINING_FOCUS message and, if the data field represents a URL, the application loads that URL. Initialisation is performed on the browser controller 402, by loading an initial URL into the browser application 403 and storing the base of the URL. The method 4100 continues at the next step 4121, where the distinguishing identifier is determined from the event. At the next step 4123, a JavaScript call back function (preferably known as the Notify_Card_ID) is called in the

current top-level document with the distinguishing identifier 1110 as the argument, and then the method 4100 returns to step 4103.

If the event is a PRESS, RELEASE or MOVE event at step 4109, then the method 4100 proceeds to step 4200. Otherwise the method 4100 proceeds to step 4113. At step 4200, a browser application specific action is performed in response to the event. The browser application specific action will be described in more detail below with reference to the flow diagram of Fig. 41. After step 4200, the method 4100 returns to step 4103.

If the event is a LOSING_FOCUS event at step 4113, then the method 4100 proceeds to step 4115. Otherwise the method 4100 proceeds to step 4117. At step 4115, the browser application reverts to an inactive state and the method 4100 returns to step 4103.

If the event is an EXIT_NOW event at step 4117, then the method 4100 concludes. Otherwise the method 4100 proceeds to step 4119. At step 4119, the event is ignored and the method 4100 returns to step 4103.

Fig. 41 is a flow diagram showing a browser application method 4200 executing on the system 600 incorporating the software architecture 200. The method 4200 can be executed by the CPU 205 for computer implementations. Alternatively, the method 4200 can be executed by the CPU 4305 in set top box implementations or by the CPU of a remote server. The method 4200 begins at step 4201, where if the event is a PRESS event then the method 4200 proceeds to step 4225. Otherwise the method 4200 proceeds to step 4203, where the event is ignored and the method 4200 concludes. At step 4225, the distinguishing identifier is determined from the event. At the next step 4227, if the current page has been notified about the current distinguishing identifier then the method 4200 proceeds to step 4205. Otherwise, the method 4200 proceeds to step 4229, where the JavaScript call back function known as the Notify_Card_ID is called in the current

top-level document with the distinguishing identifier as the argument, and then the method 4200 proceeds to step 4205.

At step 4205, data is retrieved from the event. At the next step 4207, if the data is a single character then the method 4200 proceeds to step 4209. Otherwise the method 4200 proceeds to step 4211. At step 4209, the character is sent to the browser application 403, and the method 4200 concludes. This may be used to provide the same effect as a user pressing a key on a keyboard or a button on a conventional remote control. The current page may provide an action which is performed on receipt of a given keypress using existing methods such as those provided by Hyper Text Mark-up Language (HTML).

If the data starts with "js:" at step 4211, then the method 4200 proceeds to step 4213. Otherwise the method 4200 proceeds to step 4215. At step 4213, a JavaScript function in the current top-level document is called and the method 4200 concludes. The specified data may optionally include an argument for the JavaScript function. For example, the data "js:hello" would indicate that the browser controller is to call the JavaScript function "hello", and the data "js:hello(world)" would indicate that the browser controller is to call the JavaScript function "hello" with the argument "world".

If the data starts with "cmd:" at step 4215, then the method 4200 proceeds to step 4217. Otherwise the method 4200 proceeds to step 4219. At step 4217, a specified browser function is called and the method 4200 concludes. For example, the data "print" would result in the browser controller instructing the data "back" would result in the browser controller instructing the browser to return to the previously displayed page.

If the data is an absolute URL at step 4219, then the method 4200 proceeds to step 4221. Otherwise the method 4200 proceeds to step 4223. At step 4221, the data is loaded into the browser application 403 as a URL and the method 4200 concludes.

At step 4223, the data is loaded into the browser application 403 as a URL after the base URL has been appended, and the method 4200 concludes.

A variation on the browser controller application described above with reference to Fig. 40, is a program controller, which provides control of a software program. The software program can include any program, which is normally controlled with one or more keypress events (e.g. like a keyboard keypress event or the equivalent on a game controller). The program controller can be used to provide card-based control of an existing software program such as an interactive game. The program controller process behaves substantially as described with reference to Fig. 40 with the following exceptions. If the event at step 4105 is a GAINING_FOCUS event, then the program controller process proceeds to a step of getting a Resource Locator, for the software program to be controlled, from the GAINING_FOCUS message. The process then proceeds to a step of getting and starting the software program specified by the resource locator. The program controller process then proceeds to step 4103. Further, at step 4109, instead of testing for a PRESS, RELEASE or MOVE event, this particular variation in the method 4100 would substantially check for a PRESS event. If the event is a PRESS event, the process proceeds to the steps of getting the data from the event, taking the first character from that data, and effecting a keypress of that character resulting in the same effect as if a user had typed that character on a keyboard.

10.1 Special Routing Rules for the Launcher

The launcher 303 has a special set of routing rules and the launcher 303 always receives the following events:

- EM_REMOTE_INSERT
- EM_REMOTE_REMOVE
- EM_REMOTE_BADCARD

The launcher also receives EM_REMOTE_PRESS, EM_REMOTE_RELEASE and EM_REMOTE_MOVE messages if a service identifier does not match a currently front application or if the distinguishing identifier represents the NO_CARD present identifier (i.e. all zeroes). For the purposes of determining whether or not messages match, the service-specific identifier is ignored.

The launcher 303 can be configured to explicitly make itself the front application by sending itself a EM_GAINING_FOCUS event. In this instance, all messages will be sent to the launcher 303 regardless of the service identifier of the message. The launcher 303 is not required by the protocol to respond to any of these messages.

10.2 Sample Implementations

This section outlines several examples of launcher configuration.

10.2.1 Generic Launcher

A generic launcher can be used in an open set-top-box or computer environment with broad-band Internet connectivity. In accordance with this configuration, the launcher 303 assumes that there are applications that can be downloaded to a local machine or designated remote machine and run. A generic launcher can also be configured to accommodate the use of applications that use the browser 403 via the browser controller 402.

The generic launcher can be configured to download applications as well as support persistent applications. The computer 100 running the system 600 preferably has a reasonably fast Internet connection available. In this instance, some of the applications 304 can be web pages with JavaScript that is handled by a persistent application called the browser controller 402, as described above. Further some of the applications 304 can be designed to work together. The generic launcher preferably also assumes that the

communications link used by the remote reader 1 is unreliable (i.e. an IR link) so messages can be lost.

10.2.2 Rules for the generic launcher

The following rules are the rules that are preferably used by the launcher 303 to define the system 600.

- EM_REMOTE_PRESS and EM_REMOTE_RELEASE events that have the NO_CARD present identifier (i.e. all zeroes) are used as a cue that the user wishes to exit from the front application. This could result in the system 600 either generating a "Please insert a card" message on the display 101 or returning to an earlier application, depending on the configuration of the system 600.
- EM_REMOTE_BADCARD events cause the launcher 303 to provide the users with feedback indicating that the card is faulty.
- EM_REMOTE_INSERT, EM_REMOTE_REMOVE are not relied upon to provide the bounds of the session because of the assumed unreliable communications method from the remote reader 1 to the event manager 301.
- If the launcher 303 receives an EM_REMOTE_PRESS, EM_REMOTE_RELEASE or an EM_REMOTE_MOVE message then the launcher 303 does a service mapping, and if the service identifier resolves to a downloadable application then the corresponding application is downloaded and run. The mapping is done by querying the Directory Server 305 with the service information from cards. The values returned from the Directory Server 305 are an application location and associated service data. The application location specifies the location of the application or a value the launcher recognises as a local application. The service data is the initialisation data that is sent to the application in the EM_GAINING_FOCUS

message. If the application location is empty the launcher 303 is configured to decide which application to use based upon the service data which will be a URL.

- When a new application registers with an EM_APP_REGISTER message the specified service groups are compared with a currently running set of applications and if there is no overlap then all other currently running applications are told to exit. The new application is made the current front application (using an EM_GAINING_FOCUS event) and the previously front application is sent an EM_LOSING_FOCUS event. If this occurs and the service identifier resolves to a web page then the focus is changed, using an EM_GAINING_FOCUS message, to the browser controller 402 with the address (location) of the web page in the data field. The data field is returned in the query that told the launcher 303 that the service identifier resolved to that web page. In this situation, an EM_LOSING_FOCUS event is also sent to the current front application. All other applications are told to exit.

10.3 An Example Single Use System

The architecture 200 can be configured for use with a single specialised application. In this instance, the launcher 303 can be used where it is advantageous to have a physical token (e.g. a bank card) where part or all of the user interface can be printed onto the token. The example described below is in the form of an automatic teller machine, and whilst this example is described in terms of a specific specialised application it should not be read as being limited to automatic teller machines. Such a system can be configured to be able to use a single or at least very limited number of cards. In this system no other applications 304 are started regardless of the card that is entered. The launcher 303 takes the role of a single application 304 as well as that of a system controller. No modifications are made to the event manager 301.

A single use system can be used in an automatic teller machine for example. A bank can produce personalised bank cards with commonly used options on the cards that are used as the sole or supplementary interface for an automatic teller machine. In this instance, the automatic teller machine preferably contains an event manager 301 and other core process components of the architecture 200. In this specific example the communications link between the remote reader 1 and the event manager 301 must also be reliable.

10.3.1 Rules

The following rules can be used by a launcher 303 to define a single use system bank teller machine example :

- Any events that do not come from cards associated with a participating bank could cause the launcher to display an incompatible card screen on the terminal.
- EM_REMOTE_BADCARD events are ignored.
- EM_REMOTE_INSERT events are used to start the transaction.
- EM_REMOTE_REMOVE events are used to end the transaction.
- EM_REMOTE_PRESS, EM_REMOTE_RELEASE and EM_REMOTE_MOVE events are treated as a user interaction. These are preferably handled directly by a launcher as that is the one application that is running.
- Service mappings to an external Directory Server are never done. If the card is not one that a particular automatic teller machine (ATM) knows about then the card should be rejected.

These rules are examples of how a single use system can be configured to provide a specific application in the form of an ATM.

10.4 Directory service Operation

Fig. 58 is a flow diagram, showing an overview of the process 5800 performed by the Directory Service 311. The process 5800 is executed by the CPU 205 of a computer 100, which performs the role of a Directory Service 311. The software program as shown in Fig.58 is stored in a memory medium such as Memory206 or CD-ROM212 in the system 600A or Memory 4306 in the system 600B. The process 5800 begins at the first step 5801, where the Directory Service 311 is started. At the next step 5802, the CPU waits for incoming events from a Launcher 303. The events are sent from Read Device 1 to Launcher 303 via Event Manager 301. At the next step 5803, the CPU receives a request from a Launcher 303, which contains a Distinguishing identifier, which is to be mapped by the Directory Service 311. The connection between the Launcher 303 and the Directory Service 311 is shown in Fig. 8.

At the next step 5804, the CPU searches a directory-mapping table to check if the table has an entry corresponding to the Distinguishing identifier. The directory-mapping table typically contains relations between Service identifiers and corresponding application location (e.g. URL) and service data and additionally contains relations between Distinguishing identifiers and the corresponding application location and service data. Typically, the relation involving the Service identifier is used with respect to cards 10 for which the Directory Service 311 is intended to maintain service-level information for all cards 10 which can be used for that service (for example, the location of the application 304 which is to be executed to provide the service for the card 10). Typically, the relation involving the Distinguishing identifier is used with respect to cards 10 for which the Directory Service 311 is intended to maintain information specific to the actual cards 10 or groups of cards 10 which have identical service-specific identifiers (for example, the location of a media file which is to be played to provide the service for the card 10). The directory-mapping table is typically stored in hard disk 210 or in memory 206. At step

5804, if there is an entry for the Distinguishing identifier in the directory mapping table, at the next step 5805, the CPU retrieves the application location and service data from this entry and moves to step 5806. At step 5804, if there is not an entry for the Distinguishing identifier in the table, the CPU at step 5808 extracts the Service identifier from the Distinguishing identifier by taking the relevant portion of this value (typically the first 5 bytes as is indicated in Fig. 11). At the next step 5809, the CPU searches the directory-mapping table for an entry corresponding to the Service identifier. If one is found, the CPU retrieves the application location and service data from this entry at the next step 5810 and moves to step 5806. If one is not found, at step 5811, an entry is placed in a log file indicating that a request had been made for the specific Distinguishing identifier and, at step 5812, an error is returned to the Launcher 303 indicating that the Service identifier part of the Distinguishing identifier supplied is not known by this Directory Service 311. The flow then continues to step 5802.

At step 5806, where a Distinguishing identifier or a Service identifier has been successfully found, the Distinguishing identifier and corresponding application location and service data is written to a log file and the CPU returns the application location and service data to the Launcher 303 which made the request. Flow then continues to step 5802 to wait for another event.

11. GENERAL

Typically, applications 304 are resident on the hard disk drive 210 and read and controlled in their execution by the CPU 205. Intermediate storage of programs and any data fetched from the network 220 can be accomplished using the semiconductor memory 206, possibly in concert with the hard disk drive 210. In some instances, the applications

304 will be supplied to the user encoded on a CD-ROM or floppy disk and read via the corresponding drive 212 or 211, or alternatively may be read from the network 220 via the modem device 216. Other mechanisms for loading software application into a computer system 100 from other computer readable medium include magnetic tape, a ROM or integrated circuit, a magneto-optical disk, a radio or infra-red transmission channel between the computer module 102 and another device, a computer readable card such as a smart card, a computer PCMCIA card, and the Internet and/or Intranets including email transmissions and information recorded on Websites and the like. The foregoing is merely exemplary of relevant computer readable media. Other computer readable media are also possible including combinations of those described above .

Alternatively, the process components 301 to 306 described above can be implemented in dedicated hardware as one or more integrated circuits performing the described functions or sub-functions. Such dedicated hardware may include graphic CPUs, digital signal CPUs, or one or more microCPUs and associated memories. An examples of dedicated hardware is the set top box 601 for a television described with reference to Fig. 6(b) above.

12. OTHER VARIATIONS

12.1 A Session Identifier

As described above, the distinguishing identifier is included in every INSERT, REMOVE, PRESS, RELEASE and MOVE message sent from the reader 1 to the computer 100 or set-top box 601. As an alternatively, the distinguishing identifier can be sent in connection with an INSERT message only. In this instance, upon insertion of a new card 10, the reader 1 generates a session identifier (not illustrated). The session identifier identifies a current session of a card insertion. The session identifier, for example, can be a pseudo-random number (which can be represented with 2 bytes of data)

or the session identifier can be a number that is incremented each time a card is inserted (and reset to zero when a predetermined value is reached). The reader 1 sends an INSERT message to the computer 100 or the set-top box 601, which includes a distinguishing identifier as previously described above and a session identifier which is generated for each new insertion. All subsequent PRESS, RELEASE and MOVE messages need not include the distinguishing identifier but will include the session identifier and user interface object data or press coordinates previously described.

When using a session identifier, the system 600 performs as described above with reference to Figs. 6(a) and 6(b), except that the event manager 301, upon receiving an INSERT message from a reader 1, stores the session identifier as the current session identifier and a distinguishing identifier as the current distinguishing identifier. When the event manager 301 receives a PRESS, RELEASE or MOVE message, the event manager 301 checks that the session identifier is equal to the current session identifier. If so, the event manager 301 sets a distinguishing identifier used in all messages to the current distinguishing identifier. Otherwise, if the session identifier is not equal to the current session identifier, the event manager 301 informs the user, via the display manager 306, and the display device 101, that a message has been received without a corresponding INSERT message. The user, for example, is then requested to remove and reinsert the card 10.

12.2 Other Characteristics of a user Press

As described above, the sending of information relates to the pressing, moving and releasing of an object (typically with a finger or stylus) on the touch panel 8 of the reader 1. However, the reader 1 can send additional information pertaining to an interaction from the touch panel 8 to the computer 100 or set-top box 601 for use by the system 600. For example, the additional information can represent a length of time or an

amount of pressure exerted upon the touch panel 8 as a result of a press. This additional information can be incorporated in the PRESS messages sent from the reader 1 to the system 600 and with the EM_READER_PRESS messages sent within the system 600. In this instance, the information is passed to an application 304 corresponding to the card inserted in the reader 1. An application can make use of the additional information to provide, for example, an added effect on a particular action. For example, the application can use pressure information, when associated with a press on an indicium indicating an increase in (audio) volume, to determine an amount of increase in volume. That is, the harder the press on the selected indicium, the higher the rate of increase in the volume and conversely, the softer the press on the selected indicia the lower the rate of increase.

Another example of the use of additional information in relation to a length of time (or duration) of an interaction with a touch panel 8 is described below. If a press is of very short duration, the press can to be considered to be a "tap". On the other hand, a press of very long duration can be considered as a persistent "holding down" of a keypress. In this instance, additional information can add an extra dimension to a mode of interacting with an instant software application. For instance, a "tap" on the touch panel 8 can be an instruction to the software application to select an item displayed at a current (on-screen) cursor position.

12.3 No Coordinates

A PRESS and RELEASE message can be configured not to include coordinate data of a user's interaction with the touch panel 8. In this instance, coordinate data is only sent from the reader 1 to the system 600 in conjunction with a MOVE message. The advantage of not including coordinate data in a PRESS and RELEASE message is a size reduction of messages sent by a reader 1 to the system 600, where an applications 304

does not require coordinate information for mapping from coordinates to user interface element data.

12.4 Two-way protocol

A one-way or a two-way protocol can be used for communication between a reader 1 and a computer 100 or set-top box 601. The description of the reader 1 hardware with reference to Fig. 10, and the I/O Daemon described with reference to Figs. 8 and 9 included a sending of information from a reader 1 to computer 100 or set-top box 601 and vice versa. The sending of information back to a reader 1 from a computer 100 or set top box 601 can be used to change the data stored on a card 10. For example, changing user interface object data stored on the memory chip of a smart card 10.

A two-way protocol can also be used to enable hand-shaking in the protocol. For example, a two-way protocol between a reader 1 and a set-top box 601 or computer 100 can be used so that the system 600 can acknowledge the receipt of an INSERT message sent when a card is inserted in the reader 1. A system 600 which supports a two-way protocol should also provide an additional message in the event manager protocol, in order to allow an application to send a request in order to modify a portion of the stored data on a card 10, sent to the I/O daemon 300 via the event manager 301. The I/O daemon 300 can then send a message to the reader 1 to bring about a requested action. For example, if the system 600 uses a two-way protocol then the system 600 can provide a security mechanism to ensure that applications can not modify cards without the permission of a user or without a system-defined privilege. In one example of such a system, the event manager 301 can present a displayed message to a user asking if it is OK for the application to modify a currently inserted card. The user can assent to the proposal by pressing a first region of the touch panel 8 and dissent to the proposal by pressing a second region of the touch panel 8. If the user assents to the modification of

the card 10 then the event manager 301 can allow the request from the application 304 to be passed onto the I/O daemon 300 and then on to the reader 1. On the other hand, if the user dissents from the modification, the event manager 301 drops the message and the information is not sent to the reader 1.

12.5 Alternative Read Device

In the above system600A and 600B, the Read device 1 has a substantially transparent touch sensitive membrane arranged to overlay the card 10. To reduce a cost of the Read Device 1, instead of the touch sensitive membrane, the Read Device 1 may has a plurality of user operable switches positioned around the receptacle into which the smart card 10 is insertable for reading the data, the distinguishing identifier and relation information to associate the data with each switch. Therefore the user can select at least one of the switches that correspond to at least one indicia on the card, since the operable ones of the switches are associated with indicia on the smart card visually. In this case CPU45 reads the data corresponding to a switch pressed by the user based on the relation information and the distinguishing identifier from the card 10 and sends them to Event Manager 301.

13.0 ALTERNATIVE SOFTWARE ARCHITECTURE

A further software architecture 4900 for the hardware architecture depicted by the system 600, is generally illustrated in Fig. 48 and represents an alternate software architecture to that described in previous sections. The alternative architecture 4900 is configured to be scaled from very low hardware requirements at the users home (ie. a simple set-top box), up to a powerful home system, where for example the set-top box 601 functionality is implemented on personal computing system. Further, the alternative architecture 4900 is preferably implemented within the hardware system 600.

13.1 Structure

The architecture 4900 is divided into six distinct processes and one class of process. The distinct processes include a smart card interface 4902, referred to as an I/O daemon as in the architecture 200, an event manager 4904, a display manager 4906, a master launcher 4908, an (application) launcher 4910 and a directory service 4912. The class of process is formed by one or more smart card applications 4920. In the architecture 4900 there exists one card daemon 4902, one event manager 4904, one display manager 4906 and one launcher 4910 for every smart card remote connection, usually formed by the set-top box 601, but only one master launcher 4908 for each computer that is running the launchers 4910, and at least one directory service 4912 for all systems.

In this form, the architecture 4900 can be physically separated into three distinct parts 4914, 4915 and 4916, as shown by the dashed lines in Fig. 48, each of which can be run on physically separate computing devices. Communication between each of the parts of the system is performed using TCP/IP streams as with the architecture 200.

The I/O daemon 4902 is a process that converts datagrams received from the smart card remote reader 1 into a TCP/IP stream. The I/O daemon 4902 is not intended to understand the data format used by the reader 1, but to operate independent of any changes in the smart card remote data format, and thus provides the capability to work with multiple versions of the reader 1.

The I/O daemon 4902 is started when the user starts the system 600 which, in the case of the set-top box system 600B, is when the set-top box 601 is turned on. For the computer system 600A, the I/O daemon 4902 may be started when the user starts the smart card system after the event manager 4904 and master launcher 4908 have been started.

The event manager 4904 forms a central part of the architecture 4900 in that all communications are routed through the event manager 4904. The event manager 4904 is responsible for gathering all events that are generated by the smart card remote reader 1 and relayed by the I/O daemon 4902. These events are then redistributed to the various processes and running applications.

A further role of the event manager 4904 is to isolate misbehaving applications from other well-behaved applications. In this regard, any events passed through the event manager 4904 are guaranteed to be correct to the extent that the event manager 4904 can check the event. The event manager 4904 is required to check that an event has a valid header and the correct data length, but is typically not configured to check if the data is in the correct format.

Any changes to the protocol between different versions are also to be dealt with by the event manager 4904. If possible, the events are to be rewritten to conform with the version of the data format that the operating application 4920 understands. If such is not possible, then the event manager 4904 reports an error to the originating application 4920. When different data format versions are being used, the event manager 4904 ensures that the smallest disruption possible occurs.

The display manager 4906 operates in concert with those applications 4920 operating to control which operating application 4920 has priority with respect to the particular output device 116, typically a display (e.g. 116). It is the role of the display manager 4906 to select which video stream is sent to the display 116, this information being obtained from the respective launcher 4910 of the application 4920, via the event manager 4904. Generally only the front (ie. foreground) application will produce a video display stream. Further, the display manager 4906 may operate to maintain a constant

output stream from the inconsistent input streams and may fill-in some parts of the output stream with extrapolated data.

The event manager 4904 is not responsible for deciding when an application 4920 needs to be started/ended or for actually starting or terminating an application 4920. These operations are both the responsibility of the launchers 4908 and 4910, to be discussed below. Moreover, the event manager 4904 does not have any presence on the users screen or other output device 116. Any system related feedback, such as the display of the initial insert of a smart card, is performed by the launcher 4910.

For the system 600B of Fig. 6(b) incorporating the alternative architecture 4900, there will typically be an event manager 4904 running for every set-top box 601 that is allowed to connect to the system 600B. For the system 600A incorporating the architecture 4900, the event manager 4904 will be started when the smart card system 600A, is started after the master launcher 4908 has been started.

The role of the master launcher 4908 is to start the launchers 4910 at the request of any of the event managers 4904. When the I/O daemon 4902 connects to the event manager 4904, the event manager 4904 requests the master launcher 4908 to start a first process for the event manager 4904. This first process will generally be a launcher 4910 for any smart card application 4920. The master launcher 4908 is also responsible for shutting down the launcher 4910 of an application 4920 when the event manager 4904 so requests, and for informing the event manager 4904 that the correct launcher 4910 has exited.

For the system 600B of Fig 6(b) incorporating the alternative architecture 4900, there will always be one master launcher 4908 running for each physically separate server 150, 152 running smart card applications 4920. This one master launcher 4908 handles

the requests for all event managers 4904 that request launchers 4910 on that server. For the system 600A, the master launcher 4908 commences operation either before or no later than at the same time as the rest of the smart card system.

The card directory service 4912 is provided to translate vendor-application value (Service identifier value) stored within smart cards 10 into an application location such as Uniform Resource Locators (URLs) that each point to the application 4920 associated with a vendor-application pair (Service identifier) which will be described. The directory service 4912 can be split into a number of parts by changing the launcher 4910 so that applications 4920 can run on separate systems to the launcher 4910. The directory service 4912 performs this function using a distributed look-up system where the query is passed on to another directory server if the directory service currently in possession of the query does not know the answer. Such a distributed system allows each directory server to have a limited knowledge of the transition from vendor-application ID pairs to URL's, but to still be able to translate all ID's to URL's. This provides a number of advantages including a simpler database at each directory, is more robust and permits servers to become inoperable (i.e. crash or be removed from service) whilst still permitting queries.

Referring to Fig. 52, the control template customisation information that distinguishes the smart card 10 from traditional smart cards includes a tuple of data from by a vendor identifier, a card identifier and an application identifier. The vendor identifier and the application identifier pair are equivalent to the service identifier described above for the architecture 200. Also, the card identifier is equivalent to the service-specific identifier described above for the architecture 200. Further, associated with each of the icons 4804 is corresponding data that, when a user presses on the touch panel over the icon 4804, is sent as event data that, when passed to the particular application 4920, implements a particular operation within that application. Further

detection of user actions may be incorporated, for example to detect the release of an icon, as distinct from a depression of that icon, and also to detect moving depression, where the user may scribe a finger across the touch panel 8 to perform a particular function. On each such action, event data stored on the card can be sent, which may be read from a different location of memory on the card in each case. The service identifier implemented in this alternative architecture 4900 as a vendor-application identifier pair allows the vendor, of an application associated with a smart card, to be distinguished from other. For deployments of the architecture 4900 where there is no need to distinguish a vendor of the application associated with the smart card, the vendor identifier and the application identifier can be treated as a single value: a service identifier.

The first process started by the insertion of a smart card 10 into a reader 1 will be, in a generalised system (e.g. home), a launcher 4910. In specific systems, specific applications may be commenced. For example a banking teller would start a banking application. Another example includes the use of restricted launchers that only start a specified sub-set of applications. The launcher 4910 is a smart card application that starts other applications 4920 for a specific one event manager 4904. It is the decision of the launcher 4910 to start and end applications 4920 and to actually start and terminate applications 4920. The launcher 4910 informs the event manager 4904 when applications 4920 are starting and ending, and tells applications 4920 that they are receiving or losing focus, or when they need to exit. In this regard, where a number of applications 4920 are operating simultaneously, the application that is currently on-screen is the application having focus. When another application is about to take precedence, the launcher 4910 tells the current application that it is losing focus, thereby enabling the current application to complete its immediate tasks, and tells the new application it is

gaining focus, and that the new application shall soon be changing state. The launcher 4910 must be able to force a program to exit.

The first application 4920 started (ie. usually the launcher 4910) is given special privileges, and receives "NO_CARD", "Bad_CARD" and "POWER_OFF" events generated from the remote reader 1. The first application 4920 also receives events that are intended for applications 4920 that are not the current front application, and the launcher 4910 operates to correctly interpret these events. Such is related to the specific applications mentioned above, so that the launcher correctly interprets any changes. The launcher 4910 is an application 4920 but having special rights, including the right to start and shut down other applications.

The launcher 4910 is preferably only started when the event manager 4904 requests the launcher 4910 to be started. The launcher 4910 can also be told to exit by the event manager 4904.

Applications are started by the launcher 4910 either as a response to the first user selection on a corresponding smart card 10, or at the request of another one of the application 4920. In this regard, the architecture 4900 provides a substantial enhancement over conventional arrangements through each application 4920 being organised during its programming, as a member of one or more application service groups.

13.2 Application Service Groups

An application service group is comprised of a number of smart card applications 4920 that act co-operatively, as opposed to merely simultaneously, to provide a particular set of functions. Applications 4920 that form part of a service group are permitted to run simultaneously, and also share a communication means (ie. the event

manager 4904) by which data may be exchanged. Each such application 4920 is a process or sub-process that provides a set of functions corresponding to a particular user interface or set of user interfaces. Such an application 4920 may or may not have a visible display.

With reference to the example represented in Fig. 49, a service group is initiated once an application 4920 that forms part of that service group is started and registers the particular service group with the event manager 4904. As seen in Fig. 49, a first application 4926 has associated therewith two smart cards 4924 and 4926, and a second application 4934 is operable with smart cards 4928, 4930 and 4932. Accordingly, upon insertion of the card 4922 into the reader 1, the card daemon 4902 communicates that occurrence with the event manager 4904 which, via the launcher 4910 commences application 4926. The commencement of the application 4926 enables a service group 4936, this group also including application 4934. Applications that correspond to the currently established service group may be started by inserting the relevant smart cards. For example, removal of the card 4922 and insertion of the card 4932 operates to launch application 4934, maintaining the service group 4936 as being active. Further, the starting of applications that form part of the same service group does not cause other applications from the same service group to terminate. Rather, the other applications are kept running in the background.

Termination of a service group is initiated either by touching on an empty remote reader 1, or by inserting a smart card corresponding to a different service group, such as the card 4938, corresponding to application 4940 in service group 4942. Termination of a service group causes all the applications that are currently running as part of that service group to be similarly terminated.

Applications running under the same service group may communicate with each other via the event manager 4904 by way of a service-group defined protocol 4950 as seen in Fig.49. In the protocol 4950, the format and contents of data packets sent between applications (e.g. 4926 and 4934) should be defined by the authors of those applications that coexist within the same service group).

Seen in Fig. 50 is another feature of service groups within the architecture 4900, where a service group may contain one or more applications that may run as part of any other service group. These applications provide services that may be required across service groups. An example of such an application is a personal identification service that can provide the postal address and credit card details of a user (once the user has agreed to provide those details). In this respect, such a service may form a component of numerous other services or transactions that require a financial transaction, these including on-line shopping and banking.

The design of applications to support the architecture 4900 may or may not be the same as existing approaches to application design depending on whether applications developed require the new features provided by the architecture 4900. Existing applications will still function with some modification under the architecture 4900. An example of such a modification is where each application that runs under an existing architecture can be assumed to have a service group that has the same name as the running application (ie. each application forms its own service group having only one member), or some other method of choosing a group name that is unique, including not having a group name for existing applications or applications that do not work with other applications.

Applications within the same service group need not operate on the same physical hardware, and may not be able to communicate directly with each other by using operating system defined methods. Two methods of communication are preferably implemented in the event manager 4904 to provide a standard method of inter-application communication. These methods are:

(i) a datagram based protocol where a message is sent by one application to another; and

(ii) a protocol based on a message board, where messages are posted by applications 4920 to a common area from which any application 4920 in the same service group are able to read the messages.

The event manager 4904 imposes no structure on the data that is passed between applications 4920. All the messages are just blocks of data of known length. Any other structure that is imposed on the data only needs to be understood by the applications of the particular service group. The blocks of data may be given types (e.g. raw data, .wav, .doc, etc.) which are stored by the event manager 4904 by the posting application.

A datagram method is used to allow the sending of arbitrary length data from one application in a service group to another application in the same service group, and require that the sending application knows the identification (ID) number (also referred to above as the Xid) of the receiving application. The ID number is generated by the corresponding launcher 4910 when the application 4920 is started to uniquely identify that application 4920. The ID number is unique only in the context of the event manager 4904. In this fashion, many running applications can have the same ID number but every ID number will be unique amongst all the applications 4920 that are connected to the same event manager 4904 to which the particular application is connected. It is the responsibility of the corresponding launcher 4910 to ensure that this occurs, although the

event manager 4904 can detect when duplicate ID numbers are about to be used and prevent the new application from starting.

To send a message using the datagram method, the sending application retrieves the Xid of the destination application from the event manager 4904 and then sends the message via the event manager 4904 to the destination application using this Xid to address the message. The event manager 4904 does nothing to the packet that contains the message except to ensure that the data length and sender fields of the header are correct.

For the datagram method to be available, the event manager 4904 must provide the applications with some method of determining what other applications are running in their service group. This information must also include some method for applications to identify what other applications are capable of. Such is performed in the architecture 4900 using a list of function strings that the application lists when the application registers with the event manager 4904. This list of functions is service specific as the event manager 4904 does not need to understand them in any way. Only other applications in the service need to understand what each function string means.

The event manager 4904 may impose some upper limit on the size of messages that can be passed using this method.

In the architecture 4900, the message board mentioned above allows data to be broadcast to all applications in the service group at once and also allows the applications in the service group to store data in a central repository. This removes the need for any one application to be always present in a service group. The message board also allows smart cards, and therefore applications in a service group, to be inserted/run in an arbitrary order. Applications post the data they contribute to the service group onto the

message board and when an action needs to be taken by an application, the application can examine the message board for the data that is required.

To post data to the message board, the posting application sends to the event manager 4904 the data desired to be posted, a description string, and in some instances some form of typing information (e.g. a MIME type). If the application does not supply the type information, the event manager 4904 will assign the data a default type (e.g. default binary data, the MIME type application/octet-stream). The event manager 4904 then assigns this message a message identifier, which is used to identify the message in the message board. This message identifier is used to retrieve the message from the message board by other applications. The message identifier is also used by the posting application to remove the message from the message board. The message board, and any messages remaining on the message board corresponding to a service group are destroyed when a service group is terminated.

To retrieve a message from the message board, an application must find the message identifier of the message that is required. The application can obtain a listing of the messages on the message board, which will contain the message identifier, poster identifier and the message description of each of the messages on the board. The second method involves also obtaining a listing of running applications from the event manager 4904. This provides the application with the functions that each application provides for the service. The application requesting the message from the message board can then cross-reference the application identifier (Xid) of the application from which it needs the information, against the poster identifier on the message board, and then retrieve all messages posted by that application.

The format of both the messages and the message descriptions on the message board is decided by the service group and may be totally arbitrary. The event manager 4904 does not force any structure upon the data.

To support such a method of communication, the event manager 4904 is required to maintain the message board. To the event manager 4904, the message board appears simply as a list of known length data blocks. When an application posts a message to the message board, the event manager 4904 stores the data and its length. When an application reads a message from the message board, the event manager 4904 sends the data to the application. The event manager 4904 also creates a listing the contents of the message board for applications that request such a listing.

The event manager 4904 may limit the total size of messages that each application can post as well as the total size of all messages that can be posted by all applications in a service group, so that each application has a message size limit and each service group has a message size limit. The number of messages an application and service group may post may also be limited. The size of the descriptions of the messages may also be limited to a maximum length.

13.3 System Initialisation

This section describes the process of initially starting the system 600 incorporating the software architecture 4900 of Fig. 48. It is relevant to the computer system 600A as well as a distributed set-top box system 600B.

Firstly, the master launcher 4908 is started and listens over the network 220 for a reply over a communication port. The event manager 4904 is then started and makes a connection to the master launcher 4908.

This order of starting these two core parts of the architecture 4900 is arbitrary in the case of the system 600A, but has distinct advantages when used in a set-top box system 600B. In the system 600B the master launcher 4908 is already running when the event manager 4904 is started, it is possible to start more event managers when more users subscribe to the service, and to reduce the number of running event managers when users leave the service.

13.4 System Start-up

This section describes the process of starting a smart card system incorporating the hardware architecture of Fig. 6A or 6B and the alternative software architecture of Fig. 48. This description assumes that there is already an event manager 4904 and a master launcher running and they have an open connection.

(i) The I/O daemon 4902 is started and initiates a connection to the event manager 4904.

(ii) The event manager 4904 accepts the connection from the I/O daemon 4902. It is at this stage that any service accounting can be performed. For instance if the user hasn't paid the bill then the connection can be refused.

(iii) The event manager 4904 requests a new launcher 4910 from the master launcher 4908 informing the master launcher 4908 what port the event manager 4904 is listening on, and then waits for an incoming connection.

(iv) The master launcher 4908 starts a new launcher 4910 and gives the new launcher 4910 the address and port number of the event manager 4904.

(v) The new launcher 4910 initiates a connection with the event manager 4904.

(vi) The event manager 4904 accepts the connection.

The system 600 is now ready to start applications 4920 as the user inserts smart cards into the reader 1 and initiates a first button press.

13.5 Starting the First Smart Card Service

This section describes the process of starting a smart card service if no other service is running on the system 600 incorporating the software architecture of Fig. 48. This is the situation when the system is first initiated and can also occur if a service terminates, either though a time-out or because the user touched the remote 1 with no smart card 10 inserted.

(i) The user inserts the smart card 10 into the reader 1 and presses the touch panel 8.

(ii) The pressed event is sent to the event manager 4904 which reformats the packet and forwards it onto the launcher 4910.

(iii) The launcher 4910 receives the packet and recognises that no service is active and queries the directory service 4912 with the service identifier (the vendor identifier and the application identifier) and the service-specific identifier (the card identifier) of the smart card 10.

(iv) The query returns the location of the appropriate application 4920, which the launcher 4910 then fetches. The application 4920 will generally be sourced remotely from storage on a server computer somewhere in the network 220, but may need to be run locally to the

launcher 4910. In advanced systems, the application may be run remotely from the launcher.

(v) The launcher 4910 informs the event manager 4904 that a new application 4920 is starting.

(vi) When the application 4920 has finished downloading to the launcher where it is to be run, it is started by the launcher 4910.

(vii) The application 4920 initiates a connection with the event manager 4904 and when the event manager 4904 has accepted the connection, the application 4920 registers with the launcher 4910. This includes what service groups that application 4920 is part of and what functions the application is capable of performing.

(viii) The launcher 4910 tells the new application 4920 that it is gaining focus.

The application 4920 at this stage has started and capable of receiving events. PRESS, RELEASE and MOVE messages generated from the reader 1 are forwarded to the applications 4920 by the event manager 4904 so long as they are intended from that application. The application 4920 cannot interact with the event manager 4904 in any way until registered has been completed. Further, the event manager 4904 will not forward events to the application and any events that are not application registration events that the event manager 4904 receives from an application 4920 that has not registered, will be discarded.

13.6 Starting, controlling and stopping an Application

Fig. 56 (a) and (b) show a method 5600 of starting, controlling and stopping an application (a application #1- #n) of applications 4920 to provide a service to a user on

the system 600 incorporating the software architecture 4900. The process of method 5600 is executed by CPU such as CPU 205 in system 600A or CPU 4305 in system 600B. A software program indicating the method 5600 is stored in a memory medium such as CD-ROM212 in system 600A or Memory 4306 in system 600B. When a user inserts the smart card 10 into the reader 1 and presses the touch panel 8 to select desired indicia, CPU45 in the reader 1 reads Card Header 1100 and data associated with the selected indicia from the smart card 10 and sends the pressed event (e.g. Press Message) associated with the selected indicia to the event manager 4904 that reformats the packet. The event manager 4904 sends the pressed packet (e.g. EM-READER PRESS) to Launcher 4910. The software program is executed by the CPU that executes at least Card Interface (Demon) 4902, Event Manager 4904, Launcher 4910 and Applications 4920 in same computing device, when Card Interface (Demon) 4902 receives the pressed event from the reader 1 and sends it to Event Manager 4904. On the other hand, if the software program is executed by each CPU in a separate computing device, a first CPU in a first computing device executing Event Manager 4904 executes steps from 5603 to 5608 and second CPU in a second computing device executing at least Launcher 4910 and applications 4920 executes steps from 5609 to 5636.

At step 5603, by executing Event Manager 4904, the CPU receives the pressed event from the reader 1 via Card Interface 4902 and at the next step 5605 the CPU determines if the Service Identifier (the vendor identifier and application identifier) in the pressed event matches that of a front application (e.g. application #1) of applications 4920 already running. If it is determined that the Service identifier matches that of the front application (e.g. application #1) using a matching table at the next step 5605, by executing Event Manager 4904 at the next step 5608 the CPU forwards the pressed packet to the front application and the method 5600 concludes. The table having a relation between each

application of applications 4920 and corresponding service identifier is stored in a RAM in Memory 206 or Memory 4306. If it is determined that the service identifier does not match that of the front application at the step 5605, at the next step 5607 the CPU forwards the pressed packet from Event Manager 4904 to Launcher 4910. At the next step 5609, by executing Launcher the CPU queries the directory server 4912 with the service identifier and receives location of the new application (e.g. application #2) corresponding to the service identifier. At the next step 5611, by executing Launcher 4910, the CPU fetches the new application from the location. At the next step 5613, by executing Launcher 4910, the CPU executes the new application (e.g. application #2). At the next step 5615, the CPU initiates a connection between the new application and Event Manager 4306 and when Event Manager 4306 has accepted the connection, the CPU registers the new application with Launcher 4910 and also the application tells the Launcher 4910 which service groups it is part of. At the next step 5616, the CPU determines if the new application shares a service group with a currently running application using a service group table stored in a RAM in Memory 206 or Memory 4306. The table having a relation each service identifier and corresponding service group is stored in the RAM in Memory in Memory 206 or Memory 4306. For example, in the table, service identifier 1 (application #1) and service identifier 3 (application #3) correspond to a service group A and service identifier 2 (application #2) and service identifier 4 (application #4) correspond to a service group B. At the next step 5616 if it is determined that the new application shares the service group with the currently running application, at the next step 5635 by executing Launcher 4910 the CPU tells the current application (the front application) that it is losing focus. At the next step 5636 by executing Launcher 4910 the CPU tells the new application that it is gaining focus and the method 5600 concludes. In this case, the CPU is still executing the current application

(the front application) in the background but no longer receives any events from the reader 1. By executing the current application the CPU can still send broadcast messages and messages to specific applications but cannot remove itself from service groups.

At the step 5616 if it is determined that the new application does not share the service group with the currently running application, at step 5617 by executing Launcher 4910 the CPU tells the applications that are currently running to exit and sets time-out. At the next step 5621 by executing Launcher 4910 the CPU waits for time-out then terminates any remaining applications except the new application. At the next step 5623 by executing Launcher 4910 the CPU informs the Event Manager 4904 of the applications which have exited or been terminated. At the next step 5636 by executing Launcher 410 the CPU tells the new application that it is gaining focus and the method 5600 concludes. In this case the CPU is now executing the new application and receives pressed packet such as EM-READER PRESS, EM-READER-RELEASE and EM-READEF MOVE that are intended for it. The system 600A or 600B is now running a new service with only one application within the service.]

13.7 Passing Data Between Two Applications

This section describes the process of passing data between two applications 4920 (application #1) and 4920 (application #2) using the datagram protocol on the system 600 incorporating the software architecture of Fig. 48. This method requires that the sending application #1 know the application identifier (Xid) of the receiving application #2.

(i) The sending application #1 gathers the data that it wishes to send.

(ii) The sending application #1 asks the launcher 4910 for the list of applications that are running in the current service group.

(iii) The launcher 4910 sends the application #1 the list of all applications in the current service group. This list includes the functions that each

application has told the launcher 4910 that it can perform as well as the descriptive string the application provided. This list is order with the most recent application listed first.

(iv) The sending application #1 looks to see if there is a suitable recipient for the data. If there is not, then it is up to the application #1 to decide how to proceed. The application #1 could, for example, not bother sending the data, or possibly ask the user to insert another smart card 10, which will start the required application.

(v) If there is a suitable recipient then the sending application #1 sends the data to the receiving application #2 via the event manager 4904.

(vi) The event manager 4904 checks the message header to ensure that the sending application #1 has correctly filled out the data length and sender fields and then passes the message to the receiving application #2. If there is no such application #2 running, then the event manager 4904 discards the message and sends an error message back to the sending application #1.

13.8 Posting Data to a Message Board

This section describes the process of posting data to a common message board on the system 600 incorporating the software architecture 4900.

(i) The posting application 4920 gathers the data that it wishes to post on the message board.

(ii) The posting application 4920 sends the data to the event manager 4904 along with a short description of the data.

13.9 Retrieving Data From a Message Board

This section describes the process of retrieving data that has been previously been posted to the message board by another application on the system 600 incorporating the software architecture 4900.

(i) The requesting application #2 asks the event manager 4904 for a list of messages on the message board.

(ii) The event manager 4904 sends the application #2 the list of messages on the message board. This list will contain the short description of the data, the application identifier (Xid) for the application 4920 that posted the message to the message board and the message identifier for all messages on the message board.

(iii) The application #2 can then ask the event manager 4904 for a particular message by its message identifier, or the application #2 can request the list of all applications currently running from the launcher 4910.

(iv) If the application #2 has asked for the list of running applications the launcher 4910 will then send it to the application #2. This list will contain the application identifier (Xid) and the list of functions the corresponding application reported to the launcher 4910 that the corresponding application can perform.

(v) The requesting application #2 can then find all or some messages from the applications that perform the functions that it is looking for.

13.10 Removing Data From a Message Board

This section describes the process of removing data that has been previously posted to the message board by the same application, or another application on the system 600 incorporating the software architecture 4900.

(i) The requesting application #2 asks the event manager 4904 for a list of messages on the message board.

(ii) The event manager 4904 sends the application #2 the list of messages on the message board. This list will contain the short description of the data, the application identifier (Xid) of the posting application and the message identifier for all messages on the message board.

(iii) The application #2 can then ask the event manager 4904 to remove a particular message by specifying the specific message identifier.

13.11 Application Examples

Example A: Card Orderings

A number of potential application card orderings exist that may be implemented. The architecture 4900 places no restriction on which card ordering, or combination of card orderings is adopted for an application 4920.

Sequential card ordering in an service group, illustrated in Fig. 51A, requires that smart cards 10 for a particular set of applications to be inserted in a specified order. For example, card A followed by card B followed by card C, with removal and/or reinsertion following the same ordering.

Hierarchical card ordering in a service group and requires the cards for a particular set of applications to be inserted in a tree-like fashion as illustrated in Fig. 51B where if card A is inserted, only cards B or C may be then inserted. If card B is removed, card A must be reinserted. If card C is inserted, only card D may be inserted, and if card D is removed, only card C may be inserted.

A fully-meshed card ordering in a service group permits cards for a set of applications to be inserted and used in any order.

Example B: Pizza Ordering Service

With a prior art pizza ordering application, a number of choices for pizza type are presented (such as vegetarian, supreme and meat lovers), but no functionality is provided for customisation of the toppings or to make use of special offers.

An example set of applications that would make up a Joe's Pizzeria service group under the architecture 4900 could be as follows:

(i) Joe's Pizza Menu;

(ii) Topping Specialist;

(iii) Current Specials; and

(iv) Personal identifier.

Each of these applications can be made to work with the other applications to create a fully featured pizza ordering service. The Joe's Pizza Menu application provides a user interface that allows a customer to select a pizza type (vegetarian, supreme etc.), drinks (cola, lime etc.) and side orders (garlic bread, pasta, etc.). This application also keeps a shopping-basket style list of the current order, and provides buttons on the smart card for resetting the order, and completing the order.

The Topping Specialist application provides a user interface that allows a customer to move through a list of currently ordered pizzas, and to add/remove toppings to a selected pizza from a set of toppings printed on the surface of the card. The list of pizzas available is obtained from a running Joe's Pizza Menu application. Changes made to the toppings of a pizza will propagate back to the Joe's Pizza Menu application for modification of the pizza order.

The Current Specials application provides controls to navigate through a list of current special offers available from Joe's Pizzeria. Any specials selected are communicated to a Joe's Pizza Menu application for addition to an existing order.

The Personal identifier application provides a method of selectively communicating the home address and home phone number of the user to the Joe's Pizza Menu application depending on the details that a user wishes to supply.

Example C: Photo Lab Service

In prior art Photo Album and T-Shirt applications, a clipboard is shared (as a file) for communication of currently selected photographs. There is no facility however, for modification of a photograph (for example cropping, or increasing the brightness), or to have a number of linked cards that represent a full roll of film, with each card currently only containing a maximum of 20 photographs, each photograph being represented by an icon large enough to act as a button.

With the architecture 4900, a Photo Lab service may be designed that would have the following set of cards:

(i) Film_1a,

(ii) Film_1b;

(iii) T-Shirt printer; and

(iv) Photo Enhancer.

The Film_1a and Film_1b cards represent a complete roll of Advantix (trade mark of Kodak Corp. of USA) film containing 40 photographs each, and may be inserted with either card first. Once either card is inserted, access is provided to the complete set of photographs spanning both cards with direct access to photos that are printed on the surface of the inserted card. This means that a slideshow function would cycle through the photographs corresponding to both cards. Each card would also have buttons for adding a particular photograph reference to the service group clipboard for user with another application in the Photo Lab service group, and the application would also provide a function returning a reference to the photograph currently being viewed.

The T-Shirt printer application provides the ability to either instantly print a T-Shirt transfer using the most recently viewed photograph (a reference to which is obtained from the Film application), or to compose a T-Shirt transfer from the set of photos residing on the clipboard.

As part of a simple photo editing service, the Photo Enhancer application operates on the most recently viewed photograph (obtained either from the T-Shirt application, or the Film application - whichever was most recently in the foreground). The Photo Enhancer may provide such operations as automatic crop, sharpen, blur, lighten darken etc., with the changes able to be pushed back to the photo server and made permanent.

Example D: Video Email Service

Prior art video email applications provide a means to send video email messages to video email users appearing on the surface of the card. With some re-design it is possible to create a Video Email service according to the architecture 4900 in which an address book can be compiled of users that supply their smart card business cards to the owner of the address book. Applications forming the Video Email service are:

(i) Video Email Send;

(ii) Video Email Mailbox;

(iii) Video Email Address Book; and

(iv) Business Card.

The Video Email Send application operates in much the same way as the prior art application, with the exception that an address may be obtained from an inserted personal identification card, or an inserted Business Card.

The Video Email Mailbox application provides functions for retrieving video email messages from a remote server, and can also provide the address of senders for use as a reply address with the Video Email Send application.

Address book functionality is provided by the Video Email Address Book application. This application allows a user to build up a list of addresses from different Business Cards, personal identifier cards, or Video Email Mailbox cards that have been inserted. One or more entries from the list of addresses may be selected for use with a Video Email Send application.

Example E: Shopping Basket Service

With conventional software architectures, applications that provided online shopping needed to each maintain their own purchasing system, including a shopping basket, ordering, billing, and shipping means. A shopping basket service designed to make use of the features available as part of the architecture 4900 would allow these functions to be split out of each online shopping application, leaving more user interface area for other functions. Applications that would form part of such a Shopping Basket Service are:

(i) E-Deliver Shopping Basket;

(ii) Davy Jones Online; and

(iii) Pace Bros. Online.

The E-Deliver Shopping Basket application provides an overall shopping basket management facility, payment, and ordering facilities.

Davy Jones Online, and Pace Bros. Online applications provide facilities for browsing through a list of available items for purchase, with associated item descriptions, from corresponding department stores. When an item is found that a user wants to purchase, the item can be added to the shopping basket for future ordering and delivery by way of the E-Deliver Shopping Basket application.

It will be appreciated from the forgoing, that the architecture 4900 may be used to implement a card interface system that affords expanded flexibility through sectionalising management processes and through the judicious launching of applications.

This has permitted applications to be operated co-operatively to achieve a functional result. Further, such enables the various components of the architecture 4900 to be operated from hardware platforms of varying complexity through the capacity to operate procedures on platforms commensurate with their complexity. Such platforms range from low end set-top boxes with limited processing power, to home PC's, and remote server computers. Specifically, with a "dumb" set-top box, the card daemon 4902 would be run from within the set-top box and the balance of all processes from one or more remote server computers. Conversely, with a smart set-top box or home-style personal computer, all processes may be operated from within the one piece of hardware, excepting for where external communications via the network 220 is essential.

The architecture 4900 is also extensible to support security models appropriate to a particular application in order to protect both users and vendors from unauthorised data siphoning and fraud.

By virtue of the event manager 4904 acting as a conduit of event commands, the architecture is able to operate with applications developed over a range of versions of the communication protocol, as such would typically be developed over the course of time.

The architecture 4900 allows the card interface system 600 to continue to function even when card applications are not complying with expected modes of operation. This includes applications unexpectedly exiting, refusing to exit on command, and sending incorrect or excessive data to the system 600. The architecture 4900 supports multi-card applications by virtue of each card in the application belonging to the same service group, thereby ensuring that the application is maintained running when a card is removed and a new card inserted.

13.12 Application Management System

The architecture 4900 has been described above utilising the concept of service groups, their establishment, and their extinction, in order to permit multiple applications to operate simultaneously without overloading computing resources and ensuring adequate response.

An alternative approach in considering multiple applications arises from interpreting data flow between applications as being from producers of data to consumers of data. Fig. 55 shows a directed graph, with the graph direction flowing from consumers to producers for performing a collective function, in this case a T-shirt having a name and a photograph transferred to its surface, that data being derived from a number of other applications. The management of applications within such a graph structure depends upon the accessibility of nodes of the graph. Specifically, when a node becomes unreachable in the graph, the application at that node should be terminated, since, at that stage, that application is unable to perform a cognisant function. Further, links to a node should be removed when a consumer of that application's product de-registers for that service. When an application starts, the application is placed in the tree. If the application is a producer of a type that a consumer wants, the application is placed under that node in the tree.

As described above, the applications 4920 are referenced by their corresponding vendor identifier and application identifier which together are equivalent to the service identifier described above for the architecture 200. . The application identifier (or Xid) is used as a unique key for quick matching when starting-up an already running application. There are two application identifiers, the one stored on the card with the vendor identifier and the card identifier (A card identifier is equivalent to the service-specific identifier described above with reference to the architecture 200) , and one assigned by the system to applications when they start (the latter application identifier being referred to herein

also as the component identifier or Xid, the former application identifier being related to the service identifier as described above).

Each application may register, using its Xid for identification, as a producer or consumer of a functionality on a needs basis. The application knows what it needs at a certain point in time by way of user interaction. For example, the user may navigate through the application to an "add photo" screen, at which point the application may register as a photo consumer. Registration in this regard is preferably be on the basis of a functionality, rather than a service group, as a service group approach would be too general for practical purposes. Further, such wouldn't allow an application to be linked to another in a consumer/producer relationship when the producer may not be able to provide the specific service that the producer requires unless all the applications in a service group support all functionality's offered by that service group.

Such a model presents two options for implementation, since an application may require two or more functions from any other applications:

1. Each node in the graph has only one connection to any other node. This means that the connection must also contain a list of the service included in the consumer/producer relationship. Each time a consumer de-registers for a service the list entry is removed. When the list of services for a connection becomes empty, the connection is removed. When a connection is removed, any producer that is linked by that connection is also checked. If the producer node is no longer connection to any other, that node may be removed.
2. This option is similar to (1) above except that instead of keeping a list of services, each specific service is a separate connection between the consumer/producer node. Thus, there may be multiple connections between two

applications. When a consumer de-registers for a service, that connection is removed. If the producer is no longer connected, the consumer is terminated.

Such proposals are problematic in that each allows the application associated with the smart card presently in the reader 1 to be terminated by an event other than a specific user action. This may be confusing from the user point of view. An alternative approach to termination of an application is therefore desired.

In such an alternative approach, the architecture 4900 may be operated without specific dependence upon any application 4920 being a member of a specific service group as described above, but through the transient formation of what is referred to herein as a "dominant" service group. A dominant service group arises from any transient functional relationship between two or more current applications being determined from whether any application 4920 is classed as either a producer, a consumer, both a producer and consumer, or neither a producer nor a consumer.

Such a management system for the applications 4920 revolves around the concept of the "dominant" service group being formed when a producer/consumer pair of applications, or a single application where that application meet both criteria, in the same service group are registered. For example simultaneous operation of applications Ac and Ap will cause service group A to be dominant and satisfies a producer-consumer pair, whereas AcBp or ApBc whilst satisfying a producer-consumer pair, will not create a dominant service group. According to the management system, when a dominant service group is formed, all applications not sharing that group are terminated. The dominant service group may exist in conjunction with a second dominant service group, provide both are registered simultaneously. For example, if Application#1 starts and registers ApBp and Application#2 starts and registers AcBc, A and B are then dominant. For two or more dominant service groups to exist, they must be formed when a new application

starting registers for each group establishing a producer-consumer pair. A producer/consumer pair of applications forming a service group registered after a dominant service group becomes a "subsidiary" of the dominant group. A subsidiary group of a subsidiary group may also be formed. A subsidiary of a subsidiary is formed when a producer of the subsidiary that was already registered as a consumer for the second subsidiary.

The net effect of such a management structure is the creation, and subsequent dismantling, of a tree or graph of interacting applications that pass data there between to achieve a final result desired by the user. Specifically, such a result may not be readily apparent from on the face of the applications being utilised, in contrast to Example B above for the pizza ordering service. This application management structure is best described with reference to the examples below.

The examples below make reference to a number of applications, details of which are described in Table 4 below.

Table 4

Card Application Name	**Description**	**Service Group Member** (p=producer, c=consumer, n=neither)
ID1	Identification detail card	Zp Cp
ID2	Identification detail card	Zp Qp
PhotoID	photograph identification card	Zp Qp Ap
Photo1	photograph card	Ap Fp
Photo2	photograph card	Mp Ap

PIN	personal identification number card	Pp
Bank	electronic banking card	Bn
Pizza	pizza ordering card	Rn
T-shirt	T-shirt manufacture	Tp
CardMaker	card used for making other cards	Sp

Example F:

In this example, it is desired by the user to create a greeting card having the recipient's name, a standard message, and a photograph on the card. A first step using the cards of Table 4 would be for the user to insert the CardMaker application card into the reader 1. Such an action commences that application and registers that application as a consumer of service groups A and Z. Applications may dynamically change their service group membership. For example, CardMaker may start and present the user with a screen display asking if the user wants to make a card identical to the card created on a previous occasion. Upon answering "NO", CardMaker registers as a consumer for ID1 and Photo1 since a new card will be made. A process tree for this stage appears as shown in Fig. 53A. Next, the user knows that a photograph is required, and provides that photograph by removing the CardMaker application and by inserting the Photo1 application. The CardMaker application remains in operation upon removal from the reader 1 since, its processes have yet to perform a function. The insertion of Photo1 application crates a dominant service group in Ac and Ap as illustrated, meaning that the CardMaker application requires a photograph and the Photo1 application can supply that photograph. The Photo1 application, requires a PIN to access the photograph and the arrangement is thus as represented in Fig. 53B. Not all photographs on the Photo1 card may require a

PIN to unlock them for use, so Photo1 only registers as Pc when it requires a PIN to proceed, such as in the present case. The PIN card is then provided according to Fig. 53C. As seen from Fig. 53C, a second producer-consumer pair is formed, and in this case the provision of the PIN, allows the Photo1 card to supply the photograph selected by the user to the CardMaker application. Those tasks having been completed, the left branch of the process tree is extinguished and those corresponding "performed" applications de-register from the event manger 4904, as shown in Fig. 53D. The next step to complete the process is to insert a card having the desired name, which in this case comes from the application ID1 as shown in Fig. 53E. This application supplies the required name and the CardMaker application is thus satisfied, thereby permitting all other applications to de-register and terminate. The CardMaker application can then output the required card without interaction with any other application.

In an alternative approach, the PIN application may be required to access both the photograph and the name. As such, the PIN application card need only be inserted the once only if the PIN for both photo cards is the same, and a process tree such as that shown in Fig. 54 may be formed. In this example PhotoID and Photo1 are used since PhotoID may have a picture of the recipient of the card being made, and Photo1 may have an attractive background picture to place over the photo.

Fig. 54 demonstrates that multiple links to nodes in the process tree are permitted, and that applications on unreachable nodes (being those with no links) are terminated.

Preferably, an upper limit on running applications is set to be seven (7). If this number is exceeded, termination of applications commences with the oldest leaf application in the process tree.

The foregoing describes only some arrangements and variations on those arrangements of the present invention, and modifications and/or changes can be made thereto without departing from the scope and spirit of the invention, the embodiments being illustrative and not restrictive.

The claims defining the invention are as follows:

1. A read device for reading an interface card, said card being configured for insertion into said read device, and wherein said card comprises indicia formed thereon and a memory having data stored therein for communicating with an external device, said read device comprising:

a substantially transparent touch sensitive membrane arranged to overlay said interface card upon receipt of said card in said read device,

a central processing unit for sending a service identifier, and a specific portion of said data to said external device, said specific data being related to a user selected indicia, wherein said external device provides a service identified by the service identifier upon receipt of said data.

2. A read device according to claim 1, wherein a service identifier is set by a vendor for use by an application.

3. A read device according to claim 2, wherein said service identifier is assigned to said vendor by a central authority.

4. A read device according to claim 1, wherein the central processing unit sends a read device identifier, read from the read device, to said external device.

5. A read device according to claim 1, wherein the central processing unit sends an insert message to the external device when said card is inserted into said read device.

6. A read device according to claim 1, wherein the central processing unit sends a remove message to said external device when said card is removed from said read device.

7. A read device according to claim 1, wherein the central processing unit sends a press message to the external device when at least one of said indicia is pressed via said touch sensitive membrane.

8. A read device according to claim 1, wherein the central processing unit sends a release message to the external device upon release of said at least one of said pressed indicia.

9. A read device according to claim 1, wherein the central processing unit sends a move message upon a press position being moved without being released.

10. A read device according to claim 1, wherein the central processing unit sends a bad card message when said an invalid card is inserted into said read device.

11. A read device according to claim 1, wherein the central processing unit sends a low battery message when a charge level of a battery powering said read device reaches a predetermined threshold.

12. A read device according to claim 1, wherein coordinates of a users press position on said touch sensitive membrane are sent to said read device.

13. A read device according to claim 12, wherein coordinates are calculated by an application which was provided by a vendor.

14. A read device according to claim 1, wherein the card stores said service identifier and said data in a memory chip.

15. A read device according to claim 1, wherein the external device is a set-top-box.

16. A read device according to claim 1, wherein the external device is a personal computer.

17. A program to be executed in a read device having a receptacle to receive an interface card , said card comprising a substrate and indicia formed on said substrate, said program comprising:

code for reading a service identifier and a specific portion of data stored within a memory of said card, said specific data being related to a user selected indicia; and

code for a sending said service identifier and said specific data to an external device, whereby a service is provided by said external device, said service being identified by the service identifier.

18. A program according to claim 17, wherein the program is stored on a computer-readable medium in said read device.

19. A data processing method for a read device that has a receptacle to receive an interface card, said card comprising a substrate with indicia formed thereon, said method comprising the steps of:

reading a service identifier and a specific portion of data stored within a memory of said card, said specific data being related to a user selected indicia; and

sending said service identifier and said specific data to an external device, whereby a service is provided by said external device, said service being identified by the service identifier.

20. A read device for reading an interface card, said card comprising indicia formed thereon and a memory having data stored therein, and wherein said card is configured for insertion into said read device, said read device comprising:

read means for reading a service identifier and a specific portion of data stored in the card, said specific data being related to user selected indicia; and

send means for sending said service identifier and said data to an external device, wherein said external device provides a service, said service being identified by the service identifier.

21. A card interface system comprising an interface card and a read device, said card being configured for insertion into said read device, and said card comprising at least a substrate with indicia formed thereon, said read device comprising a receptacle to receive said interface card said system comprising:

a memory on said card for storing a service identifier and data related to a user selected indicia; and

a central processing unit integrally formed within said read device for sending said service identifier and said data to an external device in order that a user of said card receives a service from said external device, said service being identified by the service identifier.

22. An electronic card reader for reading an electronic card, said electronic card having at least one indicia formed thereon and an electronic memory having data stored therein for controlling data controlled equipment, said electronic reader comprising:

a touch sensitive substantially transparent membrane having an upper surface configured to be depressible by a user of said reader;

a receptacle shaped to receive said electronic card, wherein said electronic card received therein and said indicia can be viewed through said touch sensitive membrane; and

an electronic circuit coupled to said membrane to read a specific portion of said data from said memory according to depression of said membrane associated with a specific one of said indicia, and for sending said specific data together with a service identifier to said data controlled equipment, said data controlled equipment providing a function controlled by receipt of said specific data, wherein said function is associated with said service identifier.

23. A read device for a interface card, said data comprising a substrate having at least one indicia formed thereon, said read device comprising:

a receptacle shaped to receive said interface card;

a substantially transparent touch sensitive membrane arranged to overlay said interface card upon receipt of said interface card in said receptacle such that said indicia can be viewed through said touch sensitive membrane; and

an electronic circuit for coupling to a memory component of said interface card and for reading data related to one of said indicia selected by a user of said read device via said membrane, wherein said electronic circuit is configured to send said data together with a service identifier to an external device, whereby said external device provides a service associated with said service identifier upon receipt of said read data.

24. A read device having a receptacle to receive an interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for determining a validity of said card that was inserted into said read device and sending to an external device a service identifier and data stored in the card, said data being related to a user pressed indicia, in order that a card user receives a service identified by the service identifier according to said pressed indicia.

25. A read device according to claim 24, wherein the service is identified by an application that is executed in the external device.

26. A read device having a substantially transparent touch sensitive membrane arranged to overlay a detachable interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for determining a validity of said card that was inserted into said read device and sending to an external device a service identifier and user press coordinates pressed on said touch sensitive membrane to select said indicia, in order that a card user receives a service identified by the service identifier according to said pressed indicia.

27. A read device according to claim 26, wherein the service is identified by an application that is executed in the external device.

28. A read device having a substantially transparent touch sensitive membrane arranged to overlay a detachable interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for distinguishing a identifier stored in said card and sending to an external device a service identifier and user press coordinates pressed on said touch sensitive membrane to select said indicia or a service identifier and user press coordinates pressed on said touch sensitive membrane to select said indicia, based on said result of said distinction, in order that a card user receives a service identified by the service identifier according to said pressed indicia.

29. A read device according to claim 28, wherein the service is identified by an application that is executed in the external device.

30. A read device having a substantially transparent touch sensitive membrane arranged to overlay a detachable interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for detecting a user press touch on said on said touch sensitive membrane and sending a touch coordinates of said user press touch to an external device, wherein said touch coordinates is used as a cursor information.

31. A read device having a receptacle to receive an interface card , said device comprising:

a central processing unit for reading an information that affects functions that said card perform in said read device, performing the functions based on said information.

32. A read device having a receptacle to receive an interface card , said device comprising:

a central processing unit for generating a session identifier identifying a current session of a card insertion, which is a number that is incremented each time a card is inserted into said read device and sending said session identifier to a external device, in order to determining a validity of said inserted card in said external device.

33. A read device for reading a user interface card, said card being configured for insertion into said read device, and wherein said card comprises a plurality of indicia formed thereon and a memory having stored therein a service identifier for communicating to an external device to provide an identified service, said read device comprising:

a receptacle for receiving said card and providing visibility of said plurality of indicia upon receipt of said card in said read device;

a selection mechanism for allowing a user to select at least one of said plurality of visible indicia;

a central processing unit coupled to said selection mechanism for generating data related to said user selected indicia and for reading said service identifier from said memory; and

a transmission unit for sending said indicia related data and said service identifier to said external device, wherein upon receipt of said data and service identifier said external device provides a service identified by the service identifier and said data.

34. A read device as claimed in claim 33, wherein the generating of data related to said user selected indicia includes reading data associated with said selected indicia previously stored in said memory.

35. A read device for reading a user interface card, said card being configured for insertion into said read device, and wherein said card comprises a plurality of indicia formed thereon , said read device comprising:

a receptacle for receiving said card and providing visibility of said plurality of indicia upon receipt of said card in said read device;

a selection mechanism for allowing a user to select at least one of said plurality of visible indicia;

a central processing unit coupled to said selection mechanism for generating data related to said user selected indicia and for reading from a memory of said card a service identifier for communicating to an external device to provide a service ; and

a transmission unit for sending said indicia related data and said service identifier to said external device, wherein upon receipt of said data and service identifier said external device provides a service identified by the service identifier and said data.

36. A read device as claimed in claim 35, wherein the generating of data related to said user selected indicia includes reading data associated with said selected indicia previously stored in said memory.

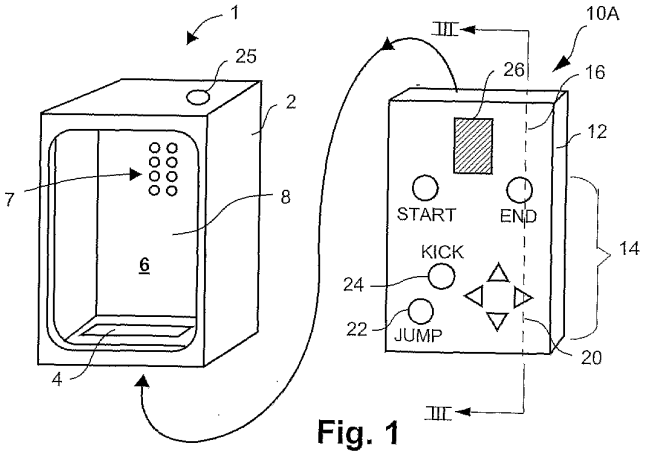

1
25
2
7
8
6
4
III
10A
26
16
12
START
END
KICK
14
24
22
JUMP
20
III

Fig. 1

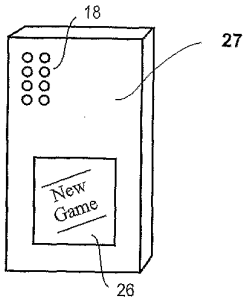

18
27
New Game
26

Fig. 2

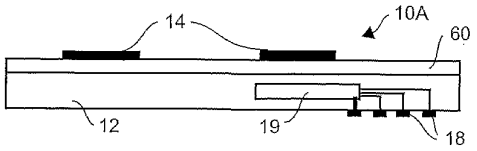


Fig. 3

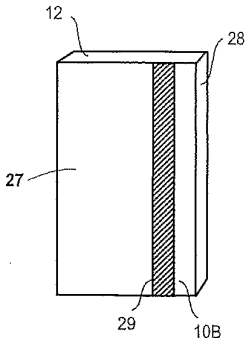


Fig. 4

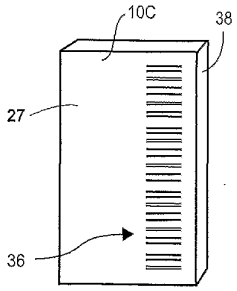


Fig. 5

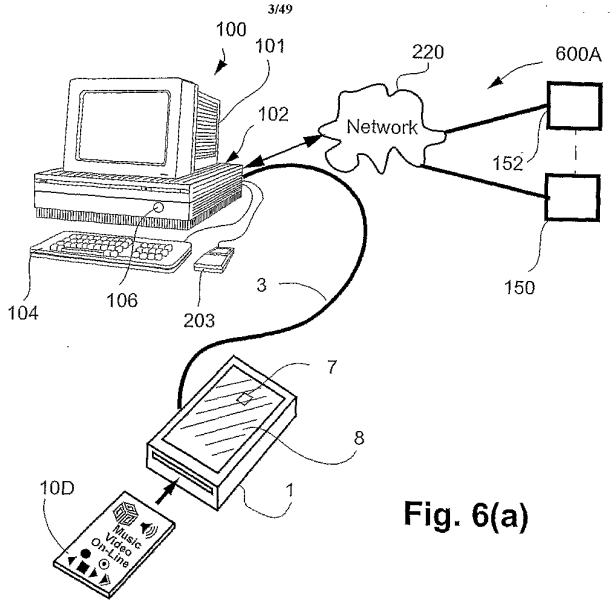

100
101
102
220
600A
Network
152
150
104
106
203
3
7
8
1
10D
Music
Video
On-line

Fig. 6(a)

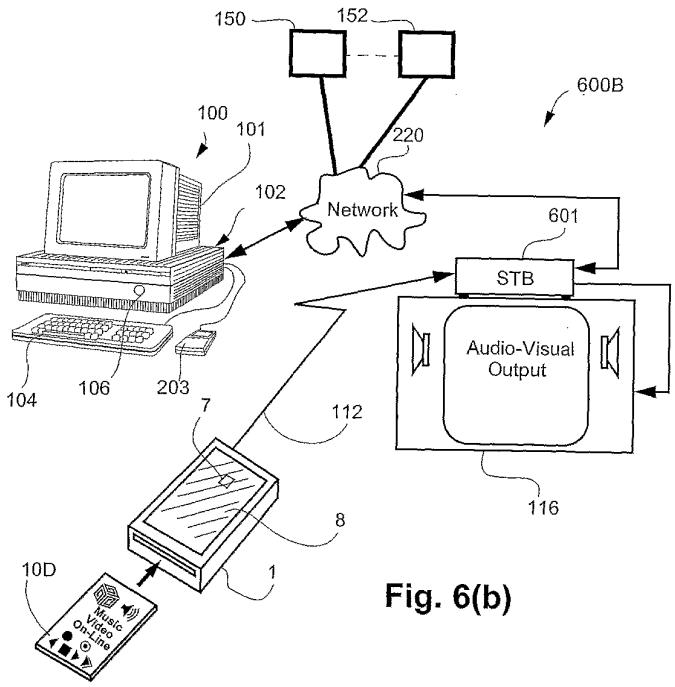

150
152
600B
100
101
220
102
Network
601
STB
Audio-Visual
Output
104
106
203
7
112
116
8
10D
1
Music
Video
On-Line

Fig. 6(b)

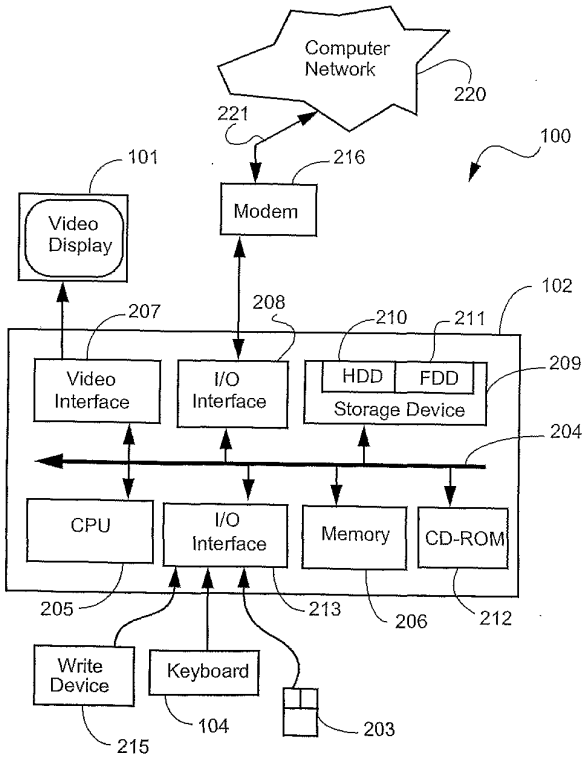

Computer Network
220
221
100
101
216
Modem
Video Display
102
207
208
210
211
209
Video Interface
I/O Interface
HDD
FDD
Storage Device
204
CPU
I/O Interface
Memory
CD-ROM
205
213
206
212
Write Device
Keyboard
104
203
215

Fig. 7

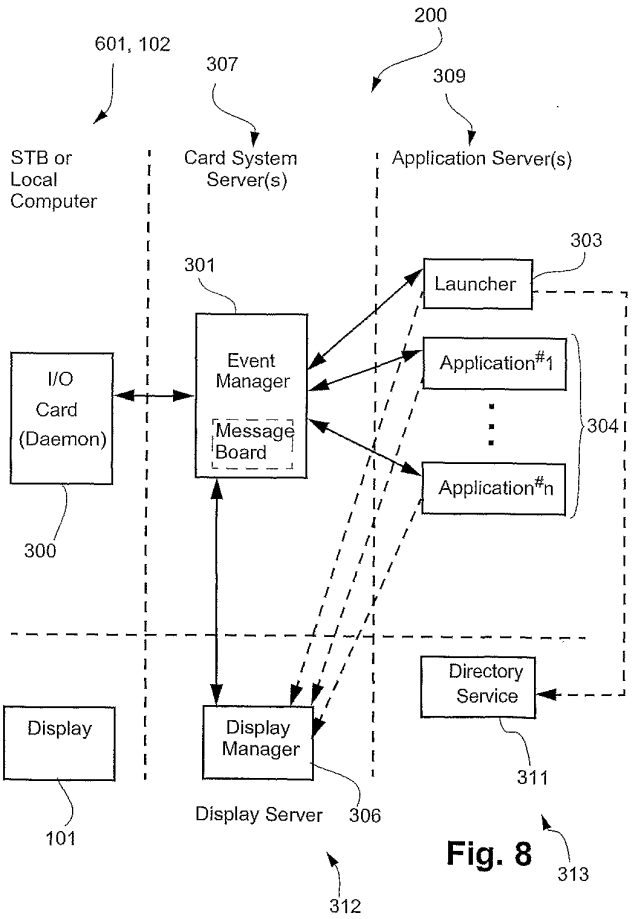

200
601, 102
307
309
STB or Local Computer
Card System Server(s)
Application Server(s)
303
301
Launcher
Event Manager
Application#1
I/O Card (Daemon)
Message Board
304
Application#n
300
Directory Service
Display
Display Manager
311
Display Server
306
101
312
313

Fig. 8

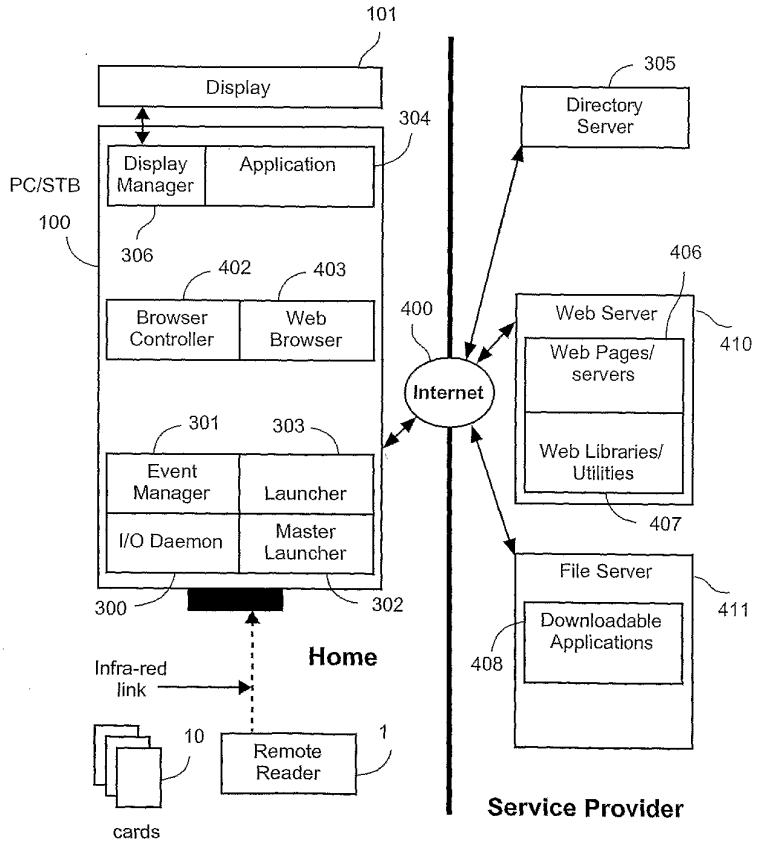
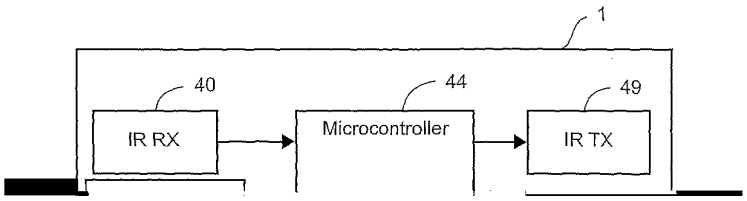

101
Display
304
Display Manager
Application
PC/STB
100
306
402
403
Browser Controller
Web Browser
400
301
303
Event Manager
Launcher
I/O Daemon
Master Launcher
300
302
Home
Infra-red link
10
Remote Reader
1
cards
Internet
305
Directory Server
406
Web Server
Web Pages/ servers
Web Libraries/ Utilities
410
407
File Server
411
Downloadable Applications
408
Service Provider

Fig. 9


1
40
IR RX
44
Microcontroller
49
IR TX

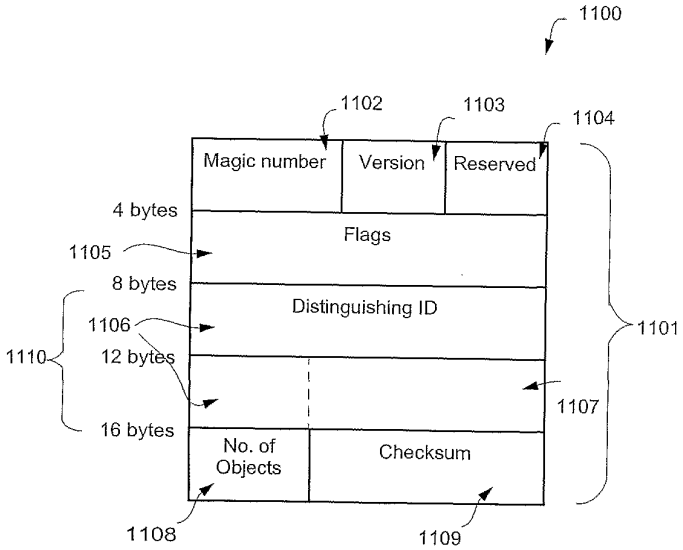

1100
1102
1103
1104
Magic number
Version
Reserved
4 bytes
Flags
1105
8 bytes
Distinguishing ID
1106
1101
1110
12 bytes
1107
16 bytes
No. of Objects
Checksum
1108
1109

Fig. 11

Field Number	Description (Card Header)
Magic Number	Two byte magic number. A constant that specifies this as being a valid card. Currently defined as the ASCII value for 'i' followed by the ASCII value for 'C'.
Version	One byte version number. Each version increment specifies a change in the card layout that can not be read by a reader that is compatible with lower versions of the layout. This document describes version 1(0x01) of the card format.
Reserved	This data is reserved for future use. Its value must be set to zero.
Flags	Four bytes of flags for this card. (See Fig. 13.) All non-assigned bits must be zero.
Distinguishing ID	Eight byte distinguishing identifier. Distinguishing identifiers include two fields - service identifier and service-specific identifier. The service identifier is five bytes and identifies the service associated with the card. The service-specific identifier is three bytes of service-specific value.
Number of Objects	One byte. The number of objects following this header. Can be zero.
Checksum	Card checksum, 2 bytes. The card checksum is sixteen bit, unsigned integer sum of all data bytes on the card excluding the checksum.

Fig. 12

Name	Description (Pre-Card Flag Values)	Value (hex)
Don't Beep	Stops the reader unit providing audio feedback by default. If this bit is set the reader will not issue any audio feedback when a UI element is pressed unless that element has the "INVERT BEEP" flag set in the UI Element object	0x0000 0001
No MOVE Events	Stops the reader unit from acting as a mouse when the user moves their finger around on the reader surface	0x0000 0002
No Event Co-ordinates	Stops the reader unit from sending co-ordinates for PRESS, RELEASE and MOVE events. X and Y values are sent with value zero.	0x0000 0004

Fig. 13

Name	Description (Object Structure)	Length
Type	The type of object (see Fig. 16).	1 byte
Object Flags	The general object flags that are associated with this object (see Fig. 15). Note: Additional flags specific to an object type are specified within the data field of the object.	1 byte
Length	The length of the data following this object. This value can be zero.	2 bytes
Data	The data associated with this object. The structure of this data is dependent on the type of object.	Variable

Fig. 14

Name	Description (Pre-Object Flag Values)	Value (hex)
Inactive	Indicates to the reader that the object is valid but is to be ignored regardless of it's type.	0x01

Fig. 15

Name	Description (Object Types)	Value (hex)
UI Object	A UICard button.	0x10
Card Data	Contains data that relates specifically to this card.	0x20
Fixed Length Data	An object that can be used to store fixed length blocks of data on the card.	0x30
Reader Insert	An object that can be used to give instructions to the reader when the card is inserted.	0x40
No Operation	An object that is used to fill blocks of empty space on the card.	0x01
No Operation (Single byte)	A single byte object that doesn't have a standard object header. Used to fill spaces on the card that are too small for a normal object header.	0x00

Fig. 16

Field	Description (User Interface Object Structure)	Size
Flags	Flags specific to this UI element on the card.	1 byte
X1	X value of the bottom-left hand corner co-ordinate of this object's rectangle.	1 byte
Y1	Y value of the bottom-left hand corner co-ordinate of this object's rectangle.	1 byte
X2	X value of the top-right hand corner co-ordinate of this object's rectangle.	1 byte
Y2	Y value of the top-right hand corner co-ordinate of this object's rectangle.	1 byte
Data	Zero or more bytes of data associated with this object. The size of this field is determined by the object data size minus the combined size of the above fields.	Variable

Fig. 17

Name	Description (Flags for UI Object)	Value
Invert Beep Enable	This flag causes this button to have the inverse of the don't beep flag in the card header. If the Don't Beep flag isn't set in the header, this flag causes this button not to beep and vice versa.	0x01
Auto-repeats	Messages associated with this button automatically repeat when the press is held on the button.	0x02
Don't Send Data on Press	This causes this button not to send the data associated with this button in the press event. The default is to send the data associated with the button in the press event.	0x04
Don't Send Data on Release	This causes this button not to send the data associated with this button in the release event. The default is to send the data associated with the button in the release event.	0x0a

Fig. 18

Field	Description (Message Header Format)	Bytes
Preamble	Preamble to the message. Value is always 0xAA 0x55 (bit sequence 10101010 01010101). This is to make it easier for the EM to find the beginning of a message.	2
Version	The version of the UICard IR message protocol this messages uses. This version of the protocol is version 1(0x01 in the version field.)	1
Type	Type of message. This is one of the values given in Fig. 20	1
Reader ID	The 16 bit id of the reader that sent the message. This number is a pseudorandom generated number that is changed when the battery is replaced in the reader. This is needed to distinguish readers when multiple readers are being used with applications.	2
Service	Service identifier as stored on the card.	5
Service-specific	Service-specific identifier as stored on the card.	3

Fig. 19

Name	Description (Message Type Codes)	Code
INSERT	A card has been inserted into the reader.	'I'
REMOVE	The card has been removed from the reader.	'E'
PRESS	The touch panel has been pressed.	'P'
RELEASE	The press on the touch panel has been released.	'R'
MOVE	The press position has moved but the press has not been released.	'M'
BADCARD	A card had been inserted however it has not passed validation	'B'
LOW_BATT	The battery in the reader is getting flat.	'L'

Fig. 20

Field	Description (Simple Message Format)	Bytes
Header	Message header as defined by Fig. 19.	14
Checksum	Message checksum. This is the sum of all the bytes in the message.	1
Checksum'	The 1's complement of the checksum.	1

Fig. 21

Field	Description (INSERT Message Format)	Bytes
Header	Message header as defined by Fig. 19.	14
Length	The number of bytes of data. Can be zero.	2
Data	The data from a Card Data object on the card.	Length
Checksum	Message checksum. This is the sum of all the bytes in the message.	1
Checksum'	The 1's complement of the checksum.	1

Fig. 21(a)

Field	Description (Move Message Format)	Bytes
Header	Message header as defined by Fig. 19.	14
X	The X co-ordinate of the touch position.	1
Y	The Y co-ordinate of the touch position.	1
Checksum	Message checksum. This is the sum of all the bytes in the message.	1
Checksum'	The 1's complement of the checksum.	1

Fig. 22

Field	Description (Press and Release Message Format)	Bytes
Header	Message header as defined by Fig. 19.	14
X	The X co-ordinate of the touch position.	1
Y	The Y co-ordinate of the touch position.	1
Length	The number of bytes of data. Can be zero.	2
Data	The data associated with the user interface element.	Length
Checksum	Message checksum. This is the sum of all the bytes in the message.	1
Checksum'	The 1's complement of the checksum.	1

Fig. 23

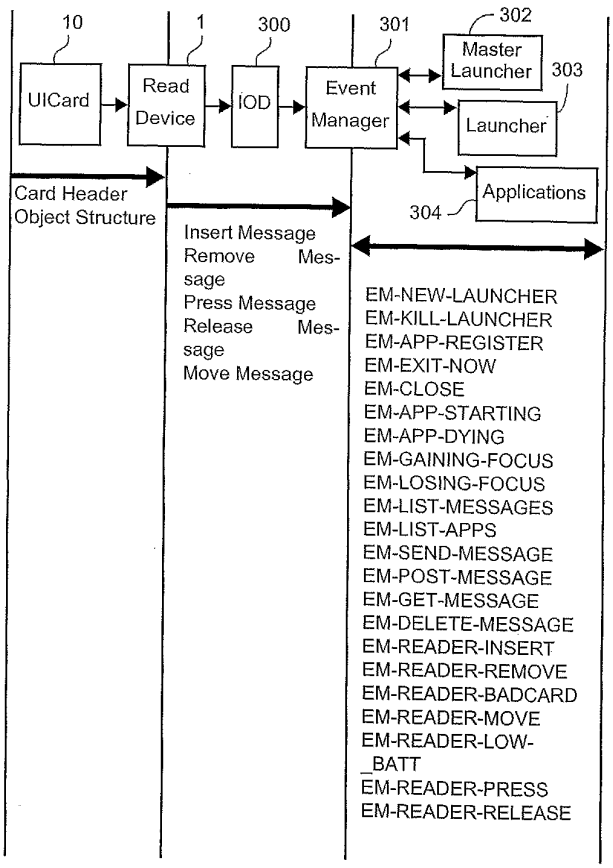

10
1
300
301
302
UICard
Read Device
IOD
Event Manager
Master Launcher
303
Launcher
Applications
304
Card Header Object Structure
Insert Message
Remove Message
Press Message
Release Message
Move Message
EM-NEW-LAUNCHER
EM-KILL-LAUNCHER
EM-APP-REGISTER
EM-EXIT-NOW
EM-CLOSE
EM-APP-STARTING
EM-APP-DYING
EM-GAINING-FOCUS
EM-LOSING-FOCUS
EM-LIST-MESSAGES
EM-LIST-APPS
EM-SEND-MESSAGE
EM-POST-MESSAGE
EM-GET-MESSAGE
EM-DELETE-MESSAGE
EM-READER-INSERT
EM-READER-REMOVE
EM-READER-BADCARD
EM-READER-MOVE
EM-READER-LOW-_BATT
EM-READER-PRESS
EM-READER-RELEASE

Fig. 24

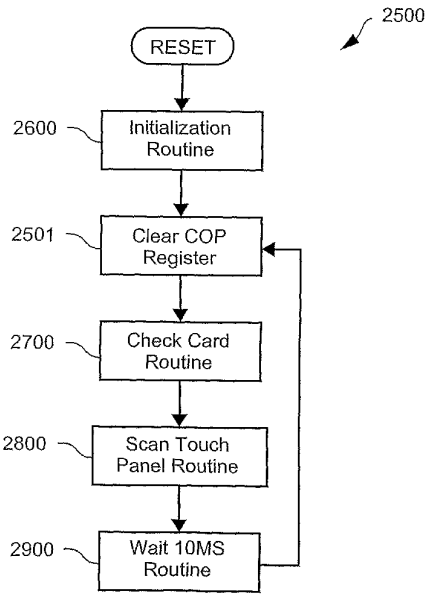

2500
RESET
2600
Initialization Routine
2501
Clear COP Register
2700
Check Card Routine
2800
Scan Touch Panel Routine
2900
Wait 10MS Routine

Fig. 25

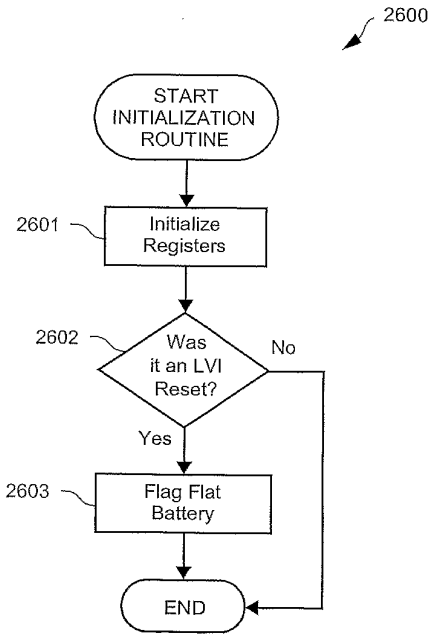

2600
START INITIALIZATION ROUTINE
2601
Initialize Registers
2602
Was it an LVI Reset?
No
Yes
2603
Flag Flat Battery
END

Fig. 26

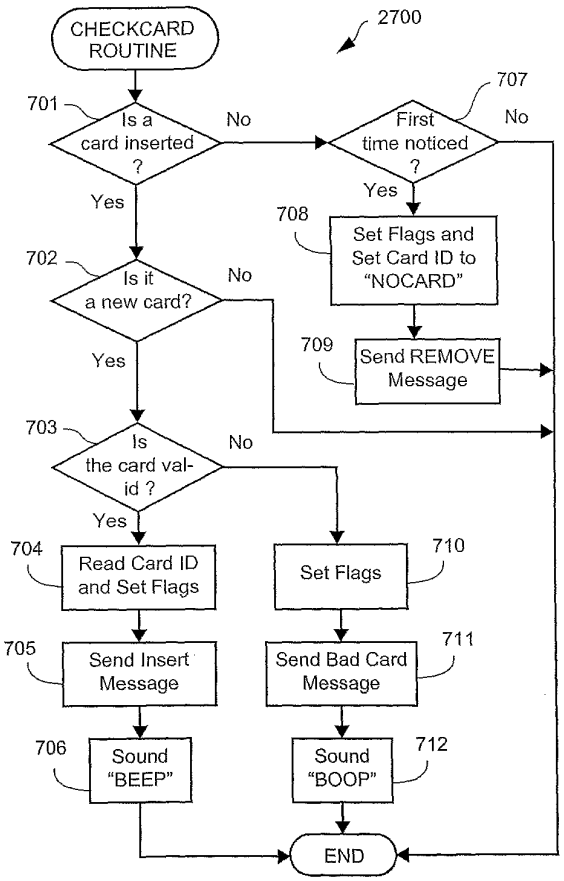

CHECKCARD ROUTINE
2700
701
Is a card inserted ?
No
Yes
707
First time noticed ?
No
Yes
708
Set Flags and Set Card ID to "NOCARD"
709
Send REMOVE Message
702
Is it a new card?
No
Yes
703
Is the card valid ?
No
Yes
704
Read Card ID and Set Flags
705
Send Insert Message
706
Sound "BEEP"
710
Set Flags
711
Send Bad Card Message
712
Sound "BOOP"
END

Fig. 27

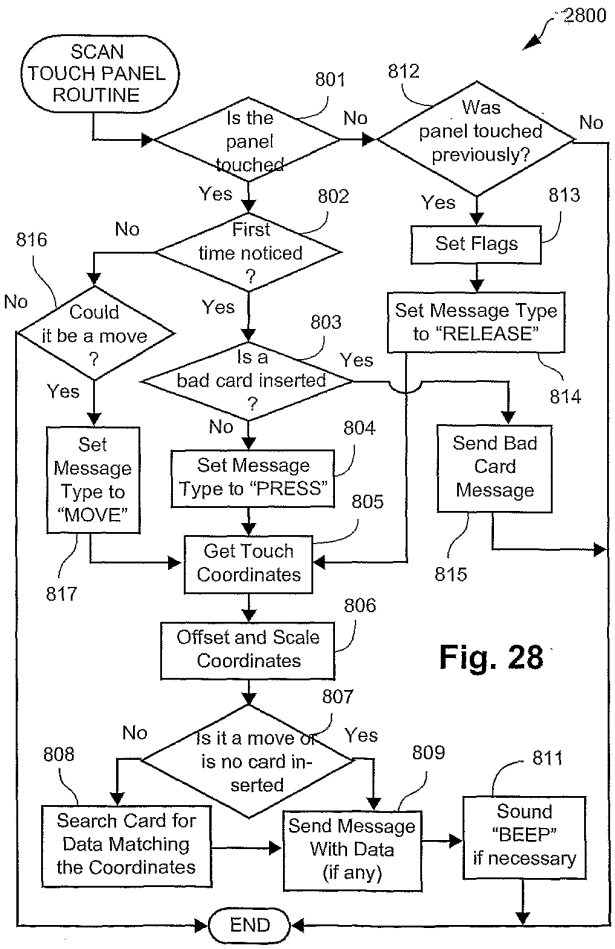

2800
SCAN TOUCH PANEL ROUTINE
801
Is the panel touched
No
812
Was panel touched previously?
No
Yes
802
First time noticed ?
Yes
813
Set Flags
816
No
Could it be a move ?
No
Yes
803
Is a bad card inserted ?
Yes
Set Message Type to "RELEASE"
814
Yes
No
804
Set Message Type to "PRESS"
Send Bad Card Message
Set Message Type to "MOVE"
805
Get Touch Coordinates
815
817
806
Offset and Scale Coordinates
807
No
Is it a move or is no card in-serted
Yes
808
809
811
Search Card for Data Matching the Coordinates
Send Message With Data (if any)
Sound "BEEP" if necessary
END

Fig. 28

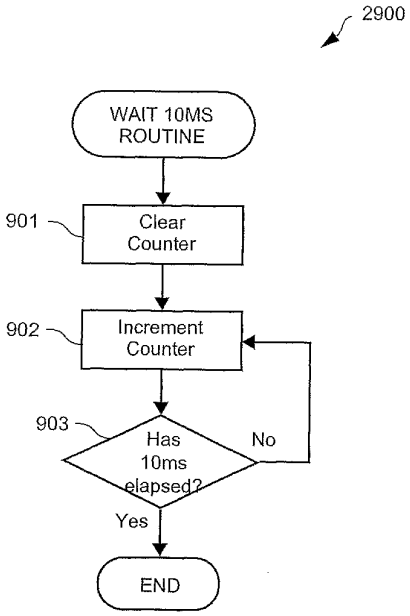

2900
WAIT 10MS ROUTINE
901
Clear Counter
902
Increment Counter
903
Has 10ms elapsed?
No
Yes
END

Fig. 29

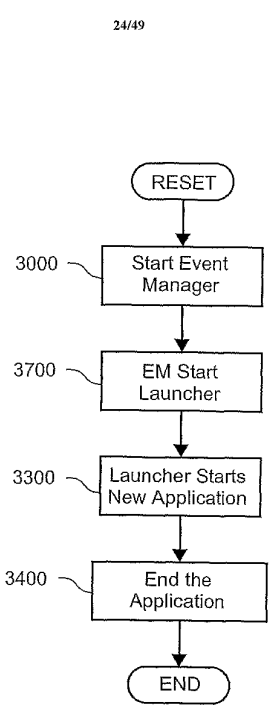

3010
RESET
3000
Start Event Manager
3700
EM Start Launcher
3300
Launcher Starts New Application
3400
End the Application
END

Fig. 30

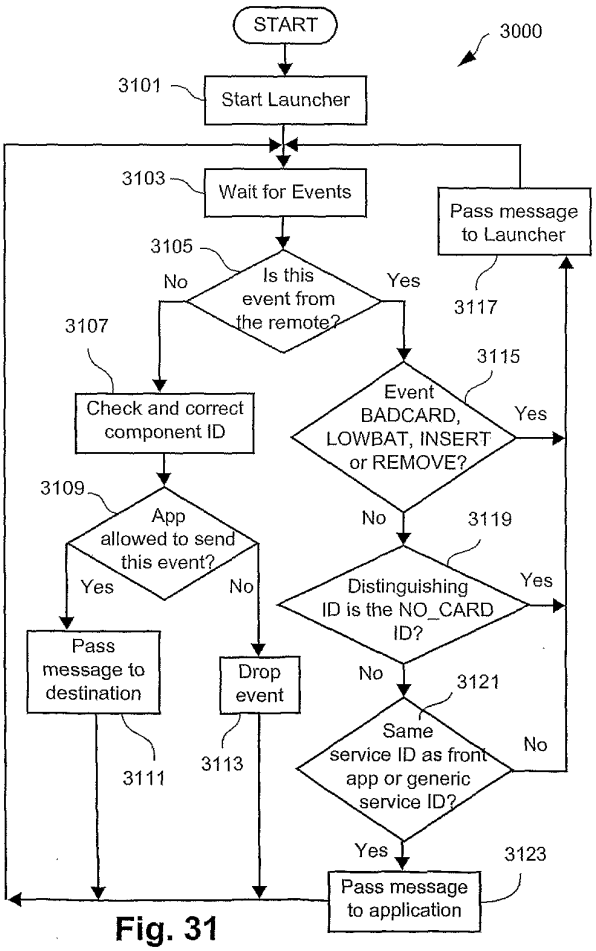

START
3000
3101
Start Launcher
3103
Wait for Events
Pass message to Launcher
3105
Is this event from the remote?
No
Yes
3117
3107
3115
Check and correct component ID
Event BADCARD, LOWBAT, INSERT or REMOVE?
Yes
3109
App allowed to send this event?
No
3119
Yes
No
Distinguishing ID is the NO_CARD ID?
Yes
Pass message to destination
Drop event
No
3121
Same service ID as front app or generic service ID?
No
3111
3113
Yes
3123
Pass message to application

Fig. 31

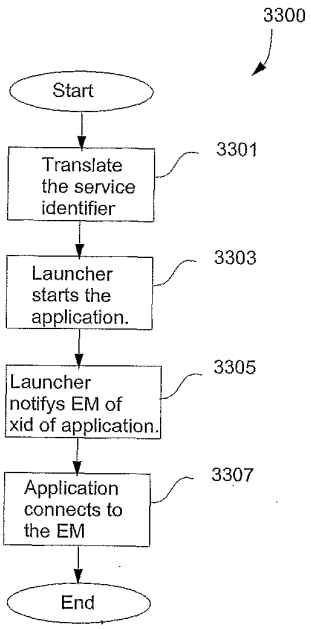

3300
Start
Translate the service identifier
3301
Launcher starts the application.
3303
Launcher notifys EM of xid of application.
3305
Application connects to the EM
3307
End

Fig. 32

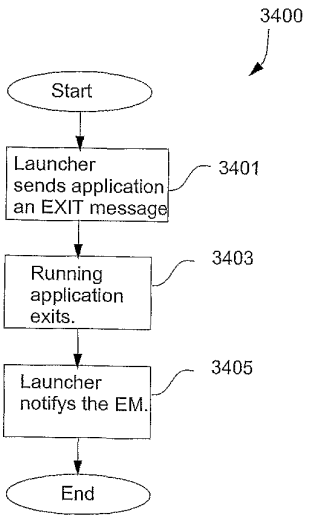

3400
Start
Launcher sends application an EXIT message
3401
Running application exits.
3403
Launcher notifys the EM.
3405
End

Fig. 33

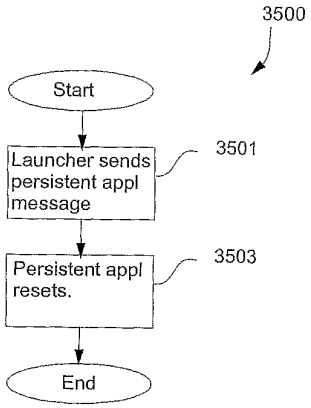

3500
Start
Launcher sends persistent appl message
3501
Persistent appl resets.
3503
End

Fig. 34

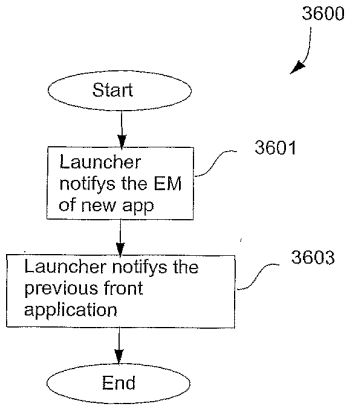

3600
Start
Launcher notifys the EM of new app
3601
Launcher notifys the previous front application
3603
End

Fig. 35

Fig. 36

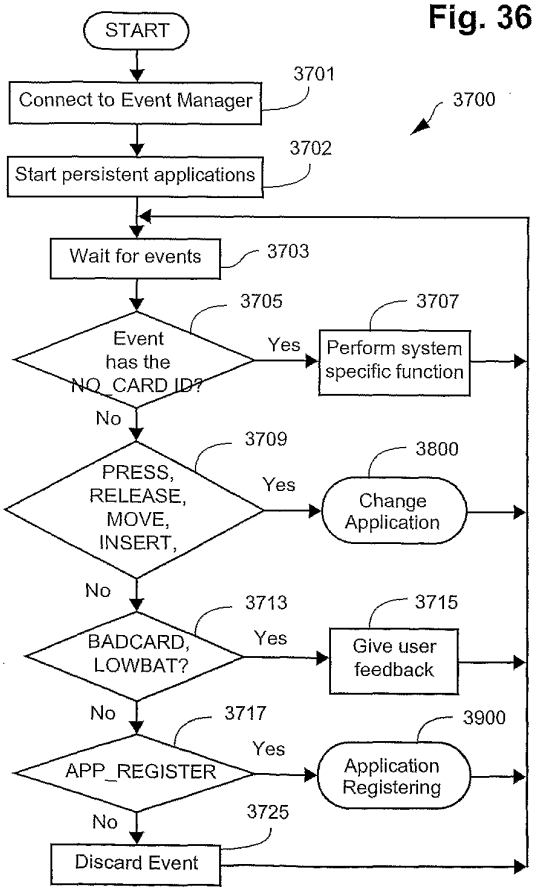
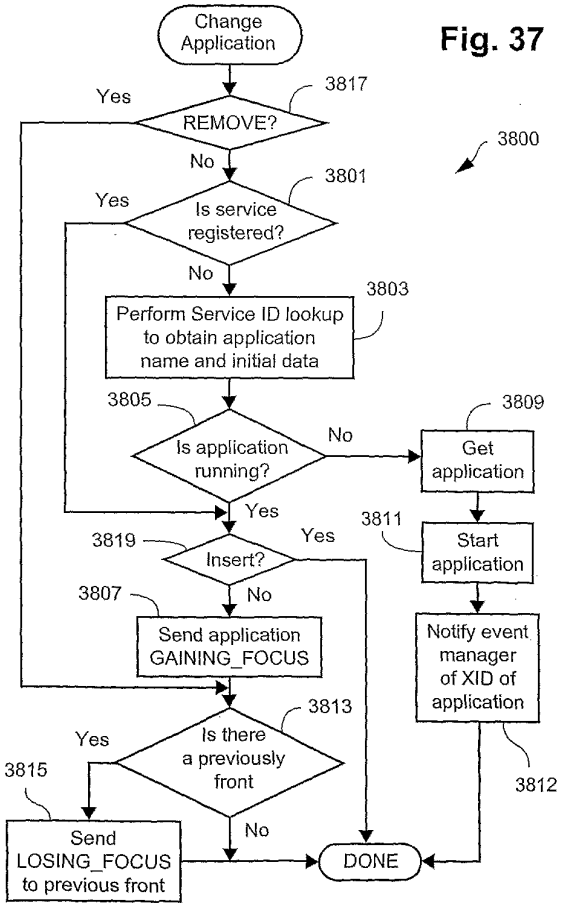

START
Connect to Event Manager
3701
3700
Start persistent applications
3702
Wait for events
3703
Event has the NO_CARD ID?
3705
Yes
Perform system specific function
3707
No
PRESS, RELEASE, MOVE, INSERT,
3709
Yes
Change Application
3800
No
BADCARD, LOWBAT?
3713
Yes
Give user feedback
3715
No
APP_REGISTER
3717
Yes
Application Registering
3900
No
Discard Event
3725

Fig. 37


Change Application
3817
Yes
REMOVE?
3800
No
3801
Yes
Is service registered?
No
3803
Perform Service ID lookup to obtain application name and initial data
3805
3809
Is application running?
No
Get application
Yes
3811
Yes
3819
Insert?
Start application
No
3807
Send application GAINING_FOCUS
Notify event manager of XID of application
3813
Yes
Is there a previously front
3815
3812
No
Send LOSING_FOCUS to previous front
DONE

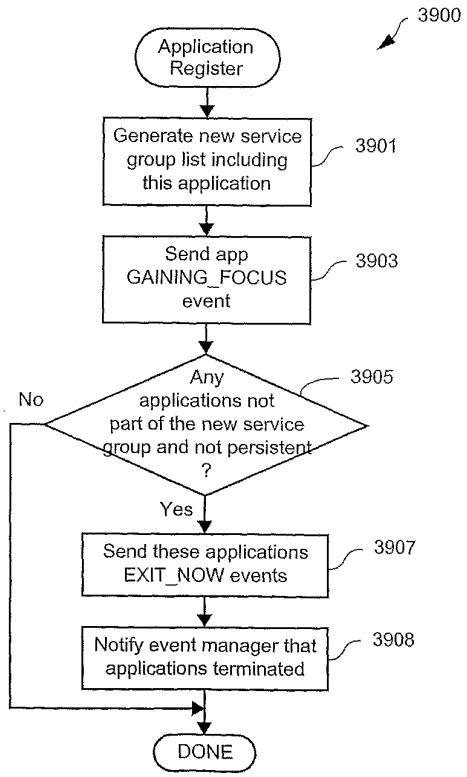

3900
Application Register
Generate new service group list including this application
3901
Send app GAINING_FOCUS event
3903
Any applications not part of the new service group and not persistent ?
3905
No
Yes
Send these applications EXIT_NOW events
3907
Notify event manager that applications terminated
3908
DONE

Fig. 38

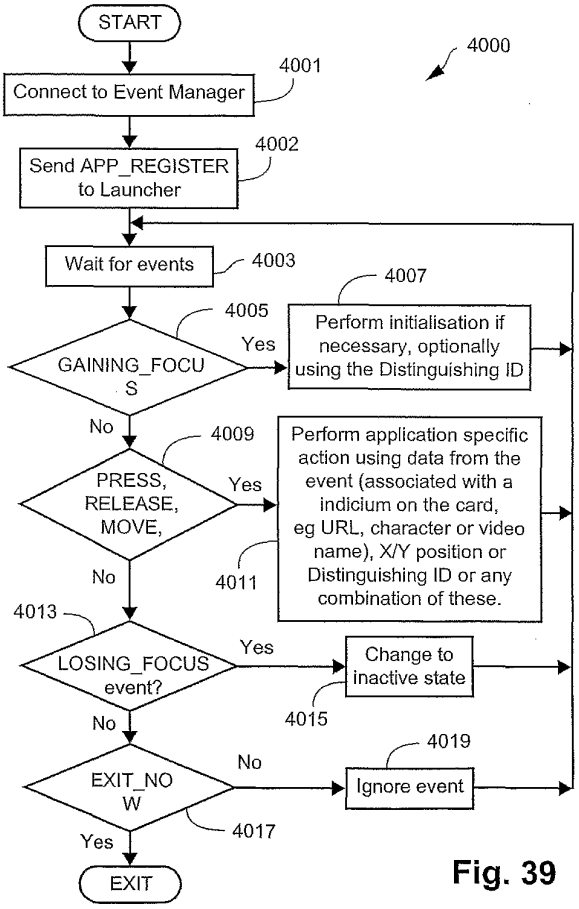

START
4000
Connect to Event Manager
4001
Send APP_REGISTER to Launcher
4002
Wait for events
4003
4007
4005
Perform initialisation if necessary, optionally using the Distinguishing ID
GAINING_FOCUS
Yes
No
4009
PRESS, RELEASE, MOVE,
Yes
Perform application specific action using data from the event (associated with a indicium on the card, eg URL, character or video name), X/Y position or Distinguishing ID or any combination of these.
4011
No
4013
LOSING_FOCUS event?
Yes
Change to inactive state
4015
No
4019
EXIT_NOW
No
Ignore event
4017
Yes
EXIT

Fig. 39

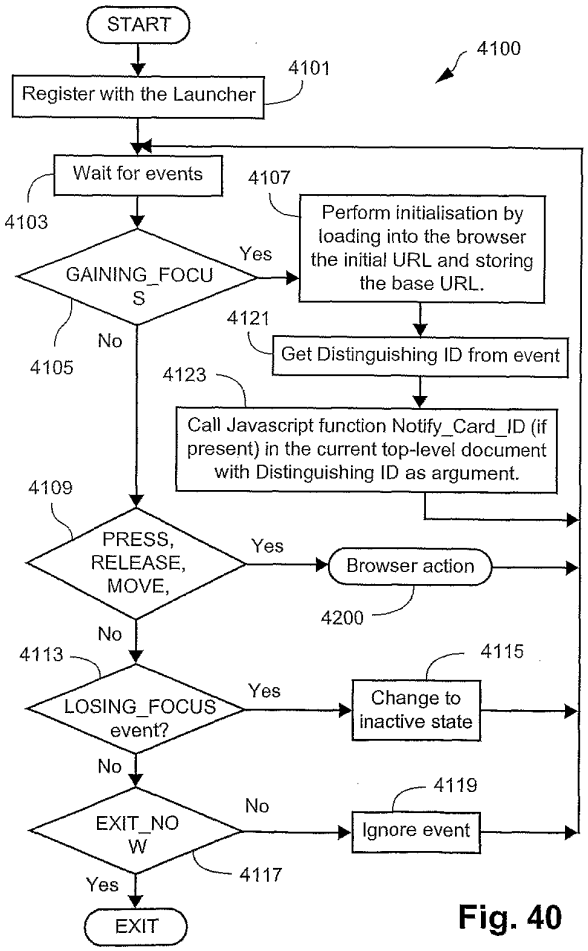
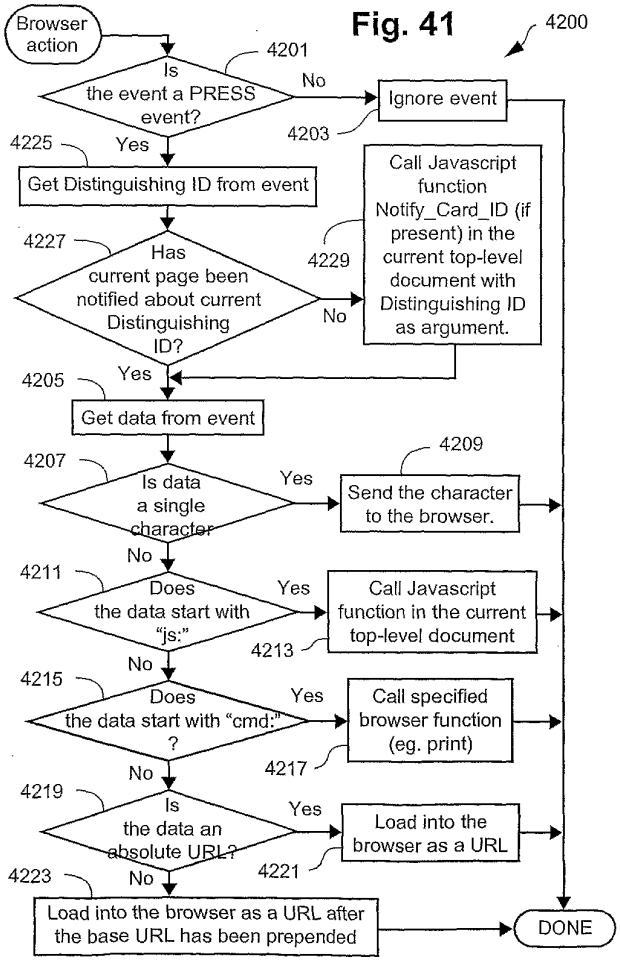

START
4100
4101
Register with the Launcher
Wait for events
4103
4107
Perform initialisation by loading into the browser the initial URL and storing the base URL.
GAINING_FOCUS
Yes
No
4105
4121
Get Distinguishing ID from event
4123
Call Javascript function Notify_Card_ID (if present) in the current top-level document with Distinguishing ID as argument.
4109
PRESS, RELEASE, MOVE,
Yes
Browser action
4200
No
4113
4115
LOSING_FOCUS event?
Yes
Change to inactive state
No
4119
EXIT_NOW
No
Ignore event
4117
Yes
EXIT

Fig. 40

Fig. 41


Browser action
4200
4201
Is the event a PRESS event?
No
Ignore event
4203
Yes
4225
Get Distinguishing ID from event
4227
Has current page been notified about current Distinguishing ID?
4229
Call Javascript function Notify_Card_ID (if present) in the current top-level document with Distinguishing ID as argument.
No
Yes
4205
Get data from event
4207
Is data a single character
Yes
4209
Send the character to the browser.
No
4211
Does the data start with "js:"
Yes
Call Javascript function in the current top-level document
4213
No
4215
Does the data start with "cmd:" ?
Yes
Call specified browser function (eg. print)
4217
No
4219
Is the data an absolute URL?
Yes
Load into the browser as a URL
4221
No
4223
Load into the browser as a URL after the base URL has been prepended
DONE

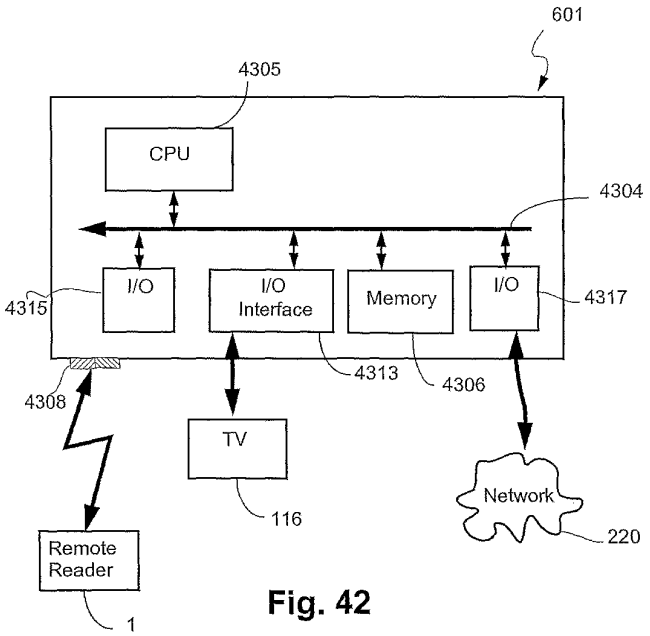

601
4305
CPU
4304
I/O
I/O Interface
Memory
I/O
4315
4317
4313
4306
4308
TV
Network
116
220
Remote Reader
1

Fig. 42

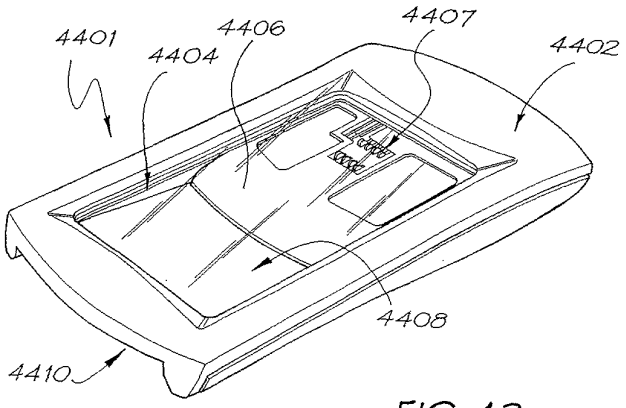

4401
4404
4406
4407
4402
4408
4410

FIG. 43

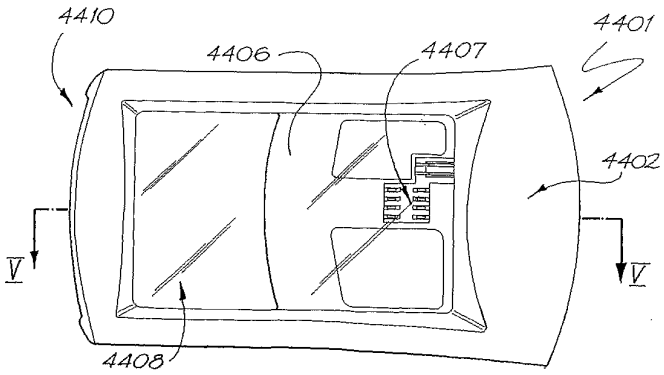

4410
4406
4407
4401
4402
V
V
4408

FIG. 44

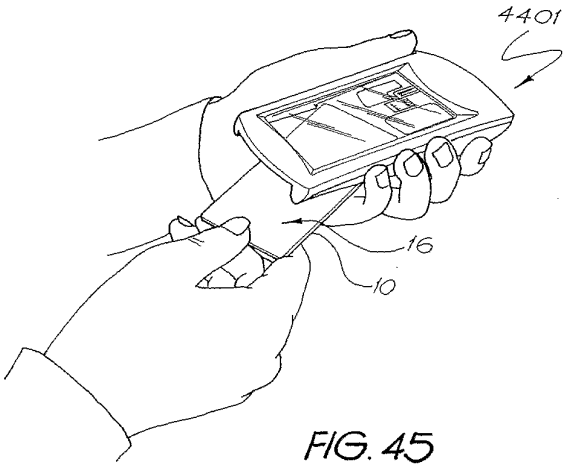


FIG. 45

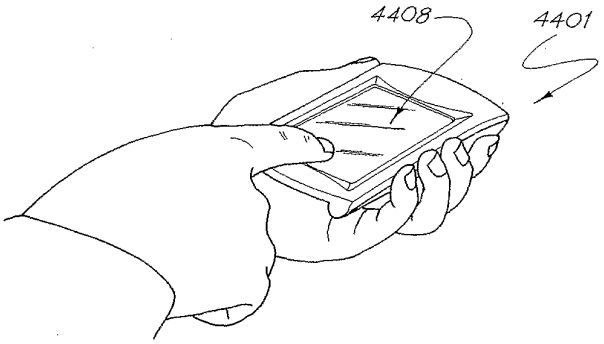


FIG. 46

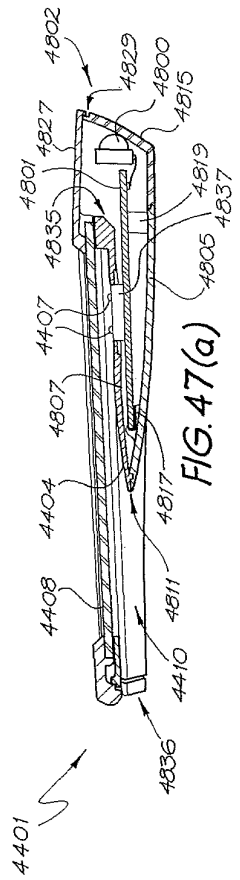


FIG. 47(a)

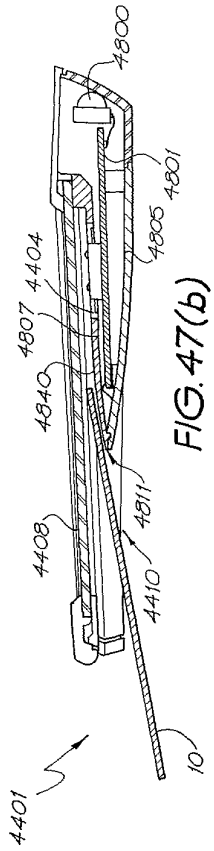


FIG. 47(b)

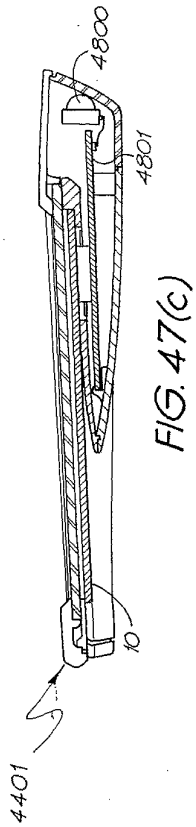


FIG. 47(c)

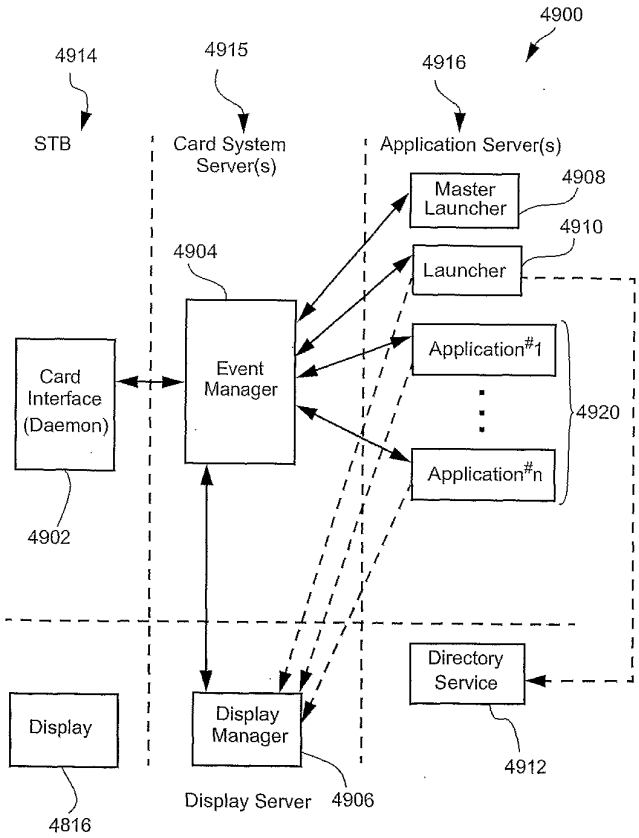

4900
4914
4915
4916
STB
Card System Server(s)
Application Server(s)
Master Launcher
4908
4910
Launcher
4904
Event Manager
Card Interface (Daemon)
Application#1
Application#n
4920
4902
Directory Service
Display
Display Manager
4912
Display Server
4906
4816

Fig. 48

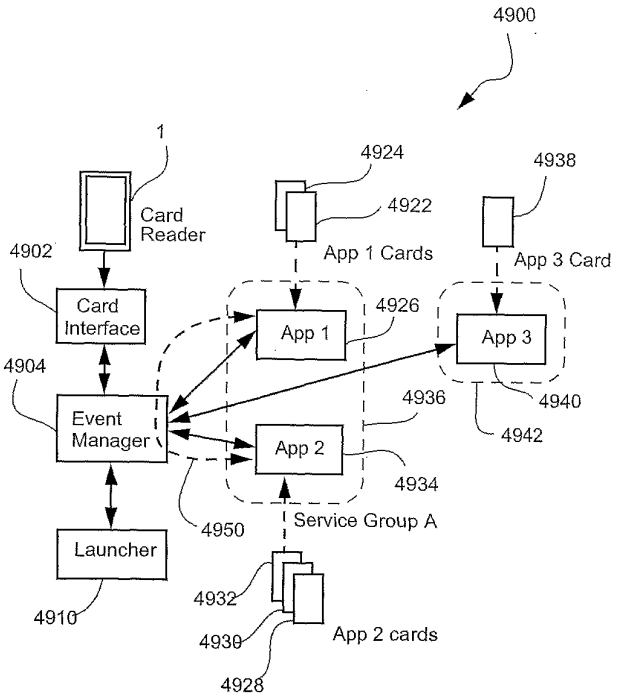

4900
1
Card Reader
4902
Card Interface
4904
Event Manager
Launcher
4910
4950
4924
4922
App 1 Cards
4938
App 3 Card
App 1
4926
App 3
4936
4940
4942
App 2
4934
Service Group A
4932
4930
4928
App 2 cards

Fig. 49

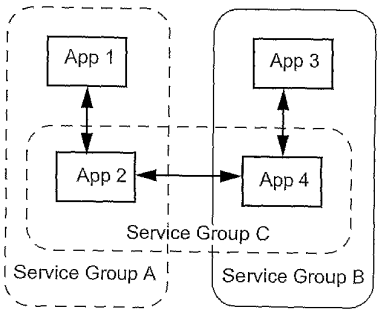

App 1
App 3
App 2
App 4
Service Group C
Service Group A
Service Group B

Fig. 50

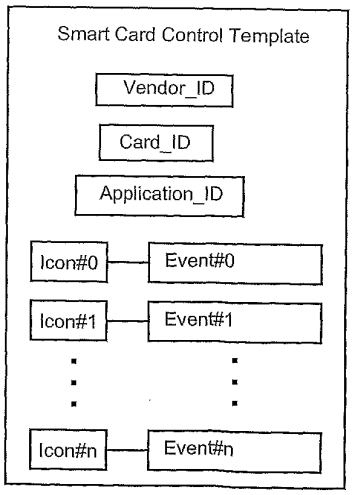

Smart Card Control Template
Vendor_ID
Card_ID
Application_ID
Icon#0
Event#0
Icon#1
Event#1
Icon#n
Event#n

Fig. 52

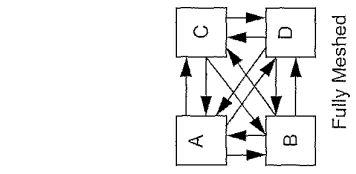


Fig. 51C

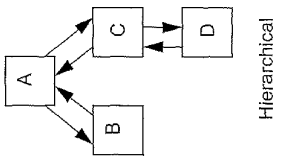


Fig. 51B

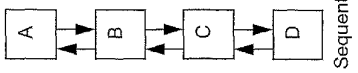
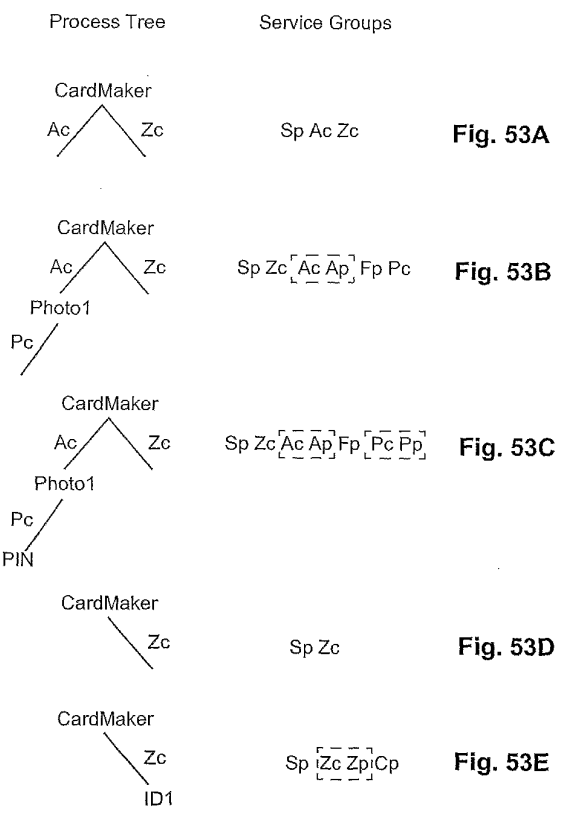


Fig. 51A


Process Tree
Service Groups
CardMaker
Ac
Zc
Sp Ac Zc
Fig. 53A
CardMaker
Ac
Zc
Photo1
Pc
Sp Zc Ac Ap Fp Pc
Fig. 53B
CardMaker
Ac
Zc
Photo1
Pc
PIN
Sp Zc Ac Ap Fp Pc Pp
Fig. 53C
CardMaker
Zc
Sp Zc
Fig. 53D
CardMaker
Zc
ID1
Sp Zc Zp Cp
Fig. 53E

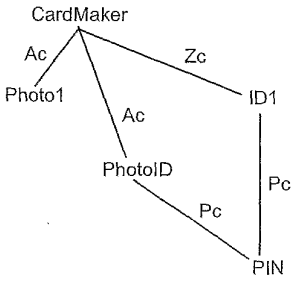

CardMaker
Ac
Zc
Photo1
ID1
Ac
PhotoID
Pc
Pc
PIN

Fig. 54

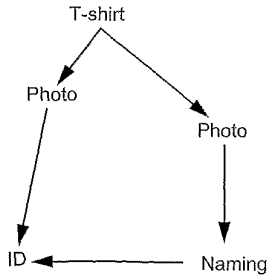

T-shirt
Photo
Photo
ID
Naming

Fig. 55

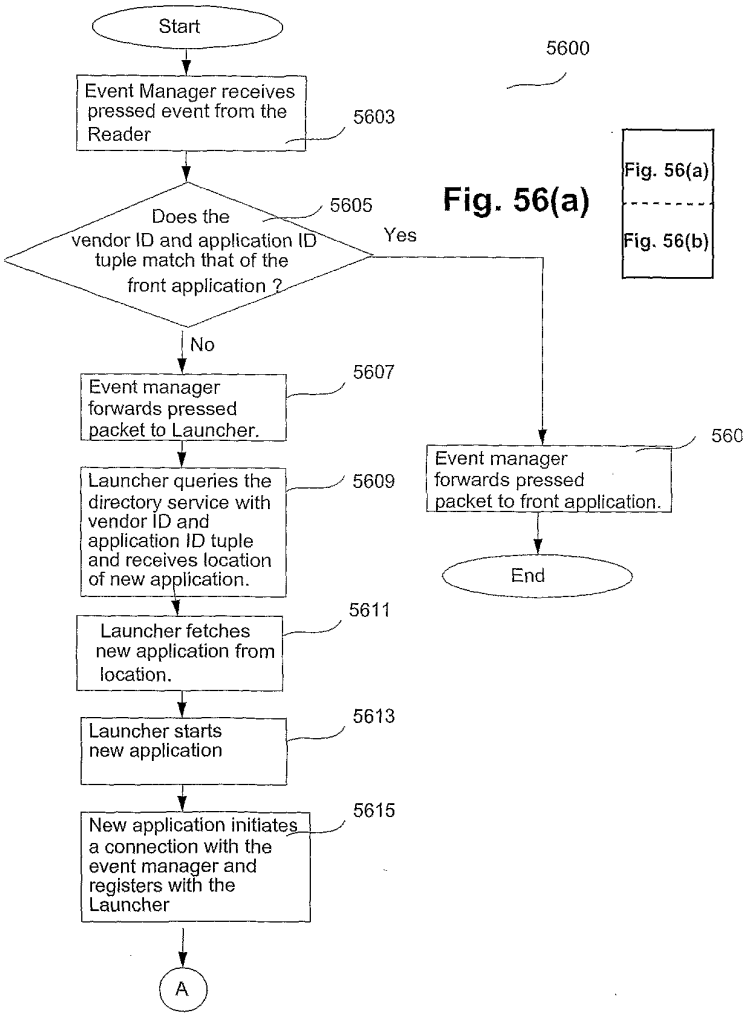


Fig. 56(a)

Fig. 56(a)
Fig. 56(b)

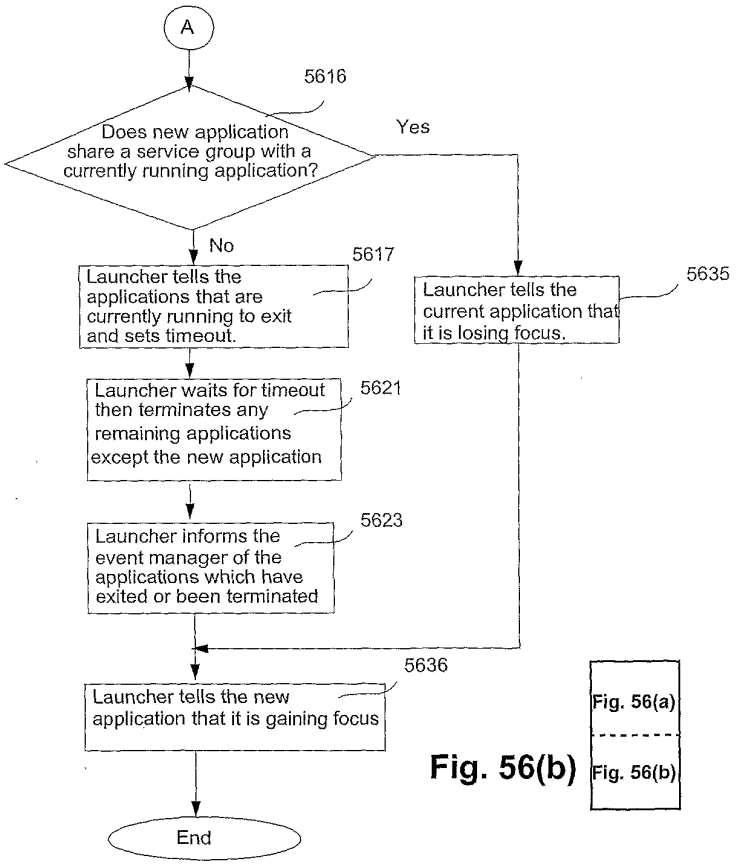


Fig. 56(b)

Fig. 56(a)
Fig. 56(b)

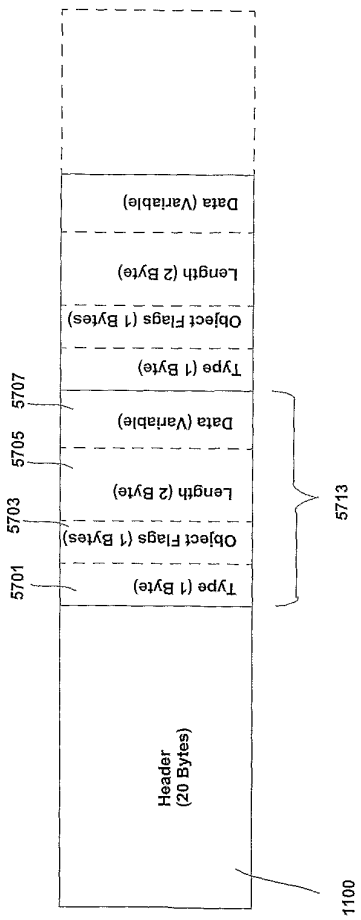

Header
(20 Bytes)
1100
Type (1 Byte)
Object Flags (1 Bytes)
Length (2 Byte)
Data (Variable)
5701
5703
5705
5707
5713
Type (1 Byte)
Object Flags (1 Bytes)
Length (2 Byte)
Data (Variable)

Fig. 57

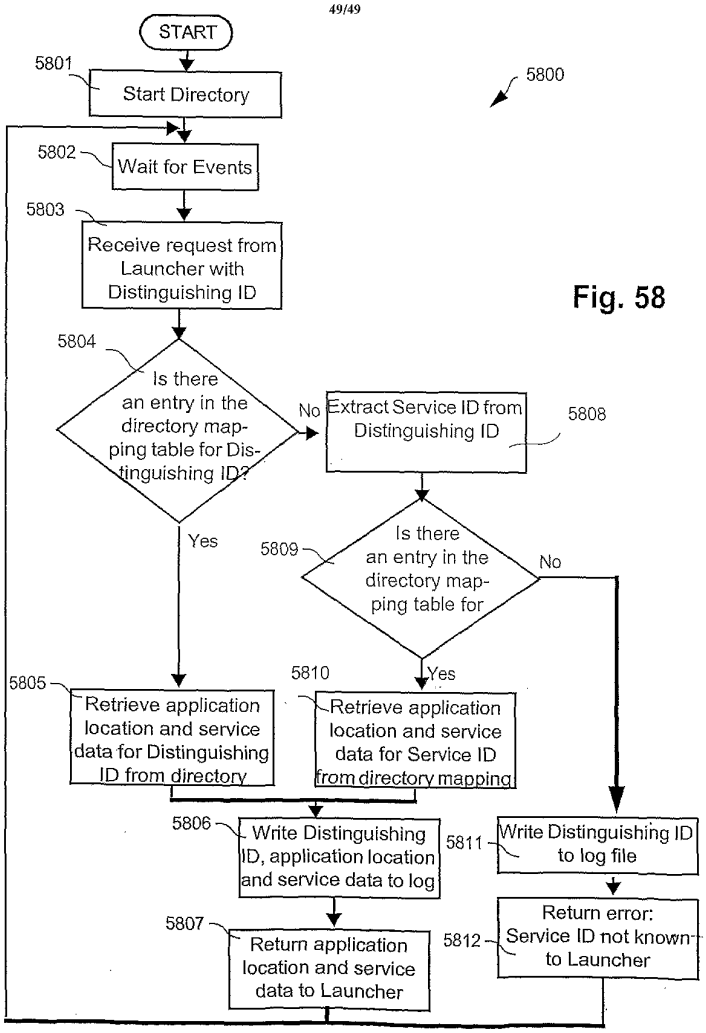

START
5801
Start Directory
5800
5802
Wait for Events
5803
Receive request from Launcher with Distinguishing ID
5804
Is there an entry in the directory mapping table for Distinguishing ID?
No
Extract Service ID from Distinguishing ID
5808
Yes
5809
Is there an entry in the directory mapping table for
No
Yes
5805
Retrieve application location and service data for Distinguishing ID from directory
5810
Retrieve application location and service data for Service ID from directory mapping
5806
Write Distinguishing ID, application location and service data to log
5811
Write Distinguishing ID to log file
5807
Return application location and service data to Launcher
5812
Return error: Service ID not known to Launcher

Fig. 58

【国際公開パンフレット（コレクトバージョン）】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
21 March 2002 (21.03.2002)

PCT

(10) International Publication Number
WO 02/023320 A1

(51) International Patent Classification⁷: G06F 3/023, G06K 19/07

(21) International Application Number: PCT/AU01/01142

(22) International Filing Date:
12 September 2001 (12.09.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
PR 0073 12 September 2000 (12.09.2000) AU
PR 5593 8 June 2001 (08.06.2001) AU

(71) Applicant *(for all designated States except US)*: **CANON KABUSHIKI KAISHA** [JP/JP]; 30-2, Shimomaruko 3-chome, Ohta-ku, Tokyo 146 (JP).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **YAP, Sue-Ken** [AU/AU]; 19/9 Burley Street, Lane Cove, NSW 2066 (AU). **SMEALLIE, Robert** [AU/AU]; 16 Cypress Street, Normanhurst, NSW 2076 (AU). **SIMPSON-YOUNG, William** [NZ/AU]; 118 Balaclava Road, Eastwood, NSW 2122 (AU). **NEWMAN, Andrew, Timothy, Robert** [AU/AU]; 16 Burton Street, Glebe, NSW 2037 (AU).

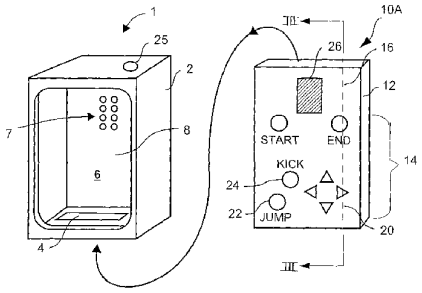
(74) Agent: **SPRUSON & FERGUSON**; GPO BOX 3898, Sydney, NSW 2001 (AU).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European

[Continued on next page]

(54) Title: CARD READING DEVICE FOR SERVICE ACCESS



(57) Abstract: A read device (1) for reading an interface card (10) is disclosed. The card (10) is configured for insertion into the read device (1). The card (10) comprises indicia (14) formed thereon and a memory (19) having data stored therein for communicating with an external device. The read device (1) comprises a substantially transparent touch sensitive membrane (8) arranged to overlay the interface card (10) upon receipt of the card (10) in the read device (1). The read device (1) also comprises a central processing unit for sending a service identifier, and a specific portion of the data to the external device. The specific data is related to a user selected indicia (14), wherein the external device provides a service identified by the service identifier upon receipt of the data.

WO 02/023320 A1

WO 02/023320 A1

patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
with international search report

(48) Date of publication of this corrected version:
8 May 2003

(15) Information about Correction:
see PCT Gazette No. 19/2003 of 8 May 2003, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

CARD READING DEVICE FOR SERVICE ACCESS

Technical Field of the Invention

The present invention relates to a control template or smart card for use with a remote reader device and, in particular, to a card interface system for providing a service. The invention also relates to a computer program product including a computer readable medium having recorded thereon a computer program for a card interface system.

Background Art

Control pads of various types are known and used across a relatively wide variety of fields. Typically, such pads include one or more keys, buttons or pressure responsive areas which, upon application of suitable pressure by a user, generate a signal which is supplied to associated control circuitry.

Unfortunately, prior art control pads are somewhat limited, in that they only allow for a single arrangement of keys, buttons or pressure sensitive areas. Standard layouts rarely exist in a given field, and so a user is frequently compelled to learn a new layout with each control pad they use. For example many automatic teller machines ("ATMs") and electronic funds transfer at point of sale ("EFTPOS") devices use different layouts, notwithstanding their relatively similar data entry requirements. This can be potentially confusing for a user who must determine, for each control pad, the location of buttons required to be depressed. The problem is exacerbated by the fact that such control pads frequently offer more options than the user is interested in, or even able to use.

Overlay templates for computer keyboards and the like are known. However these are relatively inflexible in terms of design and require a user to correctly configure a system with which the keyboard is associated, each time the overlay is to be used.

One known system involves a smart card reading device intended for the remote control of equipment. Such, for example, allows a television manufacturer, to manufacture a card and supply same together with a remote control housing and a television receiver. A customer is then able to utilise the housing, in conjunction with the card, as a remote control device for the television receiver. In this way the television manufacturer or the radio manufacturer need not manufacture a specific remote control device for their product, but can utilise the remote control housing in conjunction with their specific card. However, the above described concept suffers from the disadvantage in that control data (e.g. PLAY, RECORD, REWIND commands etc.,) stored upon the card, and to be used for controlling an associated apparatus, comes from the manufacturer of the apparatus and is thus limited in its application.

Another known system involves an operating card reading device known as a 'remote commander' used for remote-controlling a video device, audio device etc. The operating card of this known system includes a card identification mechanism for identifying which mode the remote commander is operating in and as such what control data is to be transmitted from the remote commander. The operating card identification mechanism can be in the form of either electrodes/notches formed on side surfaces of the cards or identification information stored within the operating cards. The operating card identification mechanism can be configured in order to enable the remote commander to send commands for either a video tape recorder or for a television receiver, depending on the configuration of the identification mechanism. Again, this known system suffers from the disadvantage in that control data (e.g. PLAY, RECORD, REWIND commands etc.,) to be used for controlling the video tape recorder or television, comes from the manufacturer of the apparatus and is thus limited in its application. Further, the operating card identification mechanism must be configured each time the user wishes to change the

apparatus to be controlled and is restricted to the operating card such that the identification mechanism can not be used to interact with the video device, audio device etc., to be controlled.

Still another known smart card system includes optics for receiving information from a television channel and a modem for providing real-time two way communication with an application running on a remote service provider. This known smart card system is used for remote service transactions such as an existing home shopping application. In accordance with this known system, information including home shopping program information, an item name, an item description, an item price and item commercial and programming re-run times, can be down-loaded to a smart card. The smart card can then use the access information along with the modem of the smart card to automatically dial a home shopping program automated service computer to place an order. However, again this system is limited in its application since the access information must be down-loaded to the smart card each time the smart card is to be used to purchase an item and can only be used to purchase the item specified by the item name and description.

The above-described systems all lack flexibility and are all limited in their respective applications. These systems are all used with pre-running applications and there is no interaction with the application other than that specified by the manufacturer.

Summary of the Invention

It is an object of the present invention to substantially overcome, or at least ameliorate, one or more disadvantages of existing arrangements.

According to one aspect of the present invention there is provided a read device for reading an interface card, said card being configured for insertion into said read device, and wherein said card comprises indicia formed thereon and a memory having

data stored therein for communicating with an external device, said read device comprising:

a substantially transparent touch sensitive membrane arranged to overlay said interface card upon receipt of said card in said read device,

a central processing unit for sending a service identifier, and a specific portion of said data to said external device, said specific data being related to a user selected indicia, wherein said external device provides a service identified by the service identifier upon receipt of said data.

According to another aspect of the present invention there is provided a program to be executed in a read device having a receptacle to receive an interface card, said card comprising a substrate and indicia formed on said substrate, said program comprising:

code for reading a service identifier and a specific portion of data stored within a memory of said card, said specific data being related to a user selected indicia; and

code for a sending said service identifier and said specific data to an external device, whereby a service is provided by said external device, said service being identified by the service identifier.

According to still another aspect of the present invention there is provided a data processing method for a read device that has a receptacle to receive an interface card, said card comprising a substrate with indicia formed thereon, said method comprising the steps of:

reading a service identifier and a specific portion of data stored within a memory of said card, said specific data being related to a user selected indicia; and

sending said service identifier and said specific data to an external device, whereby a service is provided by said external device, said service being identified by the service identifier.

According to still another aspect of the present invention there is provided a read device for reading an interface card, said card comprising indicia formed thereon and a memory having data stored therein, and wherein said card is configured for insertion into said read device, said read device comprising:

read means for reading a service identifier and a specific portion of data stored in the card, said specific data being related to user selected indicia; and

send means for sending said service identifier and said data to an external device, wherein said external device provides a service, said service being identified by the service identifier.

According to still another aspect of the present invention there is provided a card interface system comprising an interface card and a read device, said card being configured for insertion into said read device, and said card comprising at least a substrate with indicia formed thereon, said read device comprising a receptacle to receive said interface card said system comprising:

a memory on said card for storing a service identifier and data related to a user selected indicia; and

a central processing unit integrally formed within said read device for sending said service identifier and said data to an external device in order that a user of said card receives a service from said external device, said service being identified by the service identifier.

According to still another aspect of the present invention there is provided an electronic card reader for reading an electronic card, said electronic card having at least one indicia formed thereon and an electronic memory having data stored therein for controlling data controlled equipment, said electronic reader comprising:

a touch sensitive substantially transparent membrane having an upper surface configured to be depressible by a user of said reader;

a receptacle shaped to receive said electronic card, wherein said electronic card received therein and said indicia can be viewed through said touch sensitive membrane; and

an electronic circuit coupled to said membrane to read a specific portion of said data from said memory according to depression of said membrane associated with a specific one of said indicia, and for sending said specific data together with a service identifier to said data controlled equipment, said data controlled equipment providing a function controlled by receipt of said specific data, wherein said function is associated with said service identifier.

According to still another aspect of the present invention there is provided a read device for a interface card, said data comprising a substrate having at least one indicia formed thereon, said read device comprising:

a receptacle shaped to receive said interface card;

a substantially transparent touch sensitive membrane arranged to overlay said interface card upon receipt of said interface card in said receptacle such that said indicia can be viewed through said touch sensitive membrane; and

an electronic circuit for coupling to a memory component of said interface card and for reading data related to one of said indicia selected by a user of said read device via said membrane, wherein said electronic circuit is configured to send said data together with a service identifier to an external device, whereby said external device provides a service associated with said service identifier upon receipt of said read data.

According to still another aspect of the present invention there is provided a read device having a receptacle to receive an interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for determining a validity of said card that was inserted into said read device and sending to an external device a service identifier and data stored in the card, said data being related to a user pressed indicia, in order that a card user receives a service identified by the service identifier according to said pressed indicia.

According to still another aspect of the present invention there is provided a read device having a substantially transparent touch sensitive membrane arranged to overlay a detachable interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for determining a validity of said card that was inserted into said read device and sending to an external device a service identifier and user press coordinates pressed on said touch sensitive membrane to select said indicia, in order that a card user receives a service identified by the service identifier according to said pressed indicia.

According to still another aspect of the present invention there is provided a read device having a substantially transparent touch sensitive membrane arranged to overlay a detachable interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for distinguishing a identifier stored in said card and sending to an external device a service identifier and user press coordinates pressed on said touch sensitive membrane to select said indicia or a service identifier and user press coordinates pressed on said touch sensitive membrane to select said indicia, based on said

result of said distinction, in order that a card user receives a service identified by the service identifier according to said pressed indicia.

According to still another aspect of the present invention there is provided a read device having a substantially transparent touch sensitive membrane arranged to overlay a detachable interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for detecting a user press touch on said on said touch sensitive membrane and sending a touch coordinates of said user press touch to an external device, wherein said touch coordinates is used as a cursor information.

According to still another aspect of the present invention there is provided a read device having a receptacle to receive an interface card , said device comprising:

a central processing unit for reading an information that affects functions that said card perform in said read device, performing the functions based on said information.

According to still another aspect of the present invention there is provided a read device having a receptacle to receive an interface card, said device comprising:

a central processing unit for generating a session identifier identifying a current session of a card insertion, which is a number that is incremented each time a card is inserted into said read device and sending said session identifier to a external device, in order to determining a validity of said inserted card in said external device.

Other aspects of the invention are also disclosed.

Brief Description of the Drawings

One or more embodiments of the present invention will now be described with reference to the drawings, in which:

Fig. 1 is a perspective view of a read device and an associated card;

Fig. 2 is a perspective view of an opposite side of the card shown in Fig. 1;

Fig. 3 is a longitudinal cross-sectional view of the card shown in Fig. 1 taken along the line III - III;

Figs. 4 and 5 are perspective views of the rear face of alternative arrangements of cards to the card shown in Fig. 1;

Fig. 6(a) shows a hardware architecture of a card interface system;

Fig. 6(b) shows another hardware architecture of a card interface system ;

Fig. 7 is a schematic block diagram of the general-purpose computer of Figs. 6(a) and 6(b);

Fig. 8 is a schematic block diagram representation of a card interface system architecture;

Fig. 9 is a schematic block diagram representation of a card interface system;

Fig. 10 is a schematic block diagram showing the internal configuration of the reader of Fig. 1;

Fig. 11 shows the data structure of a card header as stored in the card of Fig. 1;

Fig.12 shows a description of each of the fields of the header of Fig. 11;

Fig. 13 shows a description of each of the flags contained in the header of Fig. 11;

Fig. 14 shows a description of each of the fields of the object structure for the card of Fig. 1;

Fig. 15 shows a description of the flag for the object structure of Fig. 14;

Fig. 16 shows a description of each of the object types for the object structure of Fig. 14;

Fig. 17 shows a description of each of the fields for a user Interface Object Structure according to the object structure of Fig. 14;

Fig. 18 shows a description for each of the user Interface object flags according to the object structure of Fig. 14;

Fig. 19 shows the format of a message header that is sent from the reader of Fig. 1;

Fig. 20 shows a table listing message event types for the header of Fig. 19;

Fig. 21 shows the format of a simple message;

Fig. 21(a) shows the format of an INSERT message;

Fig. 22 shows the format of a MOVE message;

Fig. 23 shows the format of PRESS and RELEASE messages;

Fig. 24 is a data flow diagram showing the flow of messages within the system of Fig. 6;

Fig. 25 is a flow diagram showing a read method performed by the reader of Fig. 1;

Fig. 26 is a flow diagram showing a method of initialising the system of Fig. 6, performed during the method of Fig. 25;

Fig. 27 is a flow diagram showing a method of checking the card of Fig. 1, performed during the method of Fig. 25;

Fig. 28 is a flow diagram showing a method of scanning the touch panel of the reader of Fig.1, performed during the method of Fig. 25;

Fig. 29 is a flow diagram showing a wait 10ms method, performed during the method of Fig. 25;

Fig. 30 is a flow diagram showing an overview of events of the system of Fig. 6;

Fig. 31 is a flow diagram showing processes performed by the event manager during the process of Fig. 30;

Fig. 32 is a flow diagram showing a method for starting a new application, performed during the process of Fig. 30;

Fig. 33 is a flow diagram showing a method of ending an application performed during the process of Fig. 30;

Fig. 34 is a flow diagram showing a method of closing a current session for a persistent application;

Fig. 35 is a flow diagram showing a method for performing a focus change;

Fig. 36 is a flow diagram showing an overview of a method performed by the launcher;

Fig. 37 is a flow diagram showing a method of changing an application, performed during the method of Fig. 36;

Fig. 38 is a flow diagram showing a method of registering a new application, performed during the method of Fig. 36;

Fig. 39 is a flow diagram showing a method performed by an application when receiving events from the launcher;

Fig. 40 is a flow diagram showing a method performed by the browser controller application when receiving events from the launcher;

Fig. 41 is a flow diagram showing a browser application method;

Fig. 42 is schematic block diagram showing the set top box of the system 600 in more detail;

Fig. 43 is a perspective view of a "bottom-entry" reader;

Fig. 44 is a plan view of the reader of Fig. 43;

Fig. 45 shows a user inserting a card into the reader of Fig. 43;

Fig. 46 shows a user operating the reader of Fig. 43 after a card has been fully inserted;

Fig. 47(a) is a longitudinal cross-sectional view along the line V-V of Fig. 44;

Fig. 47(b) is a view similar to Fig. 47(a), with a card partially inserted into the receptacle of the reader;

Fig. 47(c) is a view similar to Fig. 47(a), with a card fully inserted into the template receptacle of the reader.

Fig. 48 is a schematic block diagram representation of a further card interface system architecture;

Fig. 49 is a schematic block diagram representation showing the relationships between cards and applications;

Fig. 50 illustrates the relationships between applications and service groups;

Figs. 51A to 51C illustrates different types of card orderings within the architecture of Fig. 48;

Fig. 52 illustrates the control template data stored in the smart card for the architecture of Fig. 48;

Figs. 53A to 53E illustrate an example of a multi-card application structure;

Fig. 54 shows an alternative approach to achieve the end of Figs. 53A to 53E;

Fig. 55 shows a directed graph representation of a multi-application method;

Fig. 56 shows a method of starting an application;

Fig. 57 shows one or more object structures following the card header of Fig. 11; and

Fig. 58 is a flow diagram, showing an overview of the process performed by the directory service of Fig. 8.

Detailed Description including Best Mode

Where reference is made in any one or more of the accompanying drawings to steps and/or features, which have the same reference numerals, those steps and/or features

have for the purposes of this description the same function(s) or operation(s), unless the contrary intention appears.

The embodiments disclosed herein have been developed primarily for use with remote control systems, automatic tellers, video game controllers and network access, and will be described hereinafter with reference to these and other applications. However, it will be appreciated that the invention is not limited to these fields of use.

For ease of explanation the following description has been divided into Sections 1.0 to 13.0, each section having associated subsections.

1.0 CARD INTERFACE SYSTEM OVERVIEW

Fig. 1 shows a remote reader 1, having a housing 2, which defines a card receptacle 4 and a viewing area 6. Data reading means are provided in the form of exposed electrical contacts 7 and associated control circuitry (not shown). The remote reader 1 also includes sensor means in the form of a substantially transparent pressure sensitive membrane forming a touch panel 8 covering the viewing area 6. The remote reader 1 disclosed herein has been described with a substantially transparent pressure sensitive membrane forming the touch panel 8, however it will be appreciated by one skilled in the art that alternative technology can be used as a substantially transparent touch panel. For example, the touch panel can be resistive or temperature sensitive. The remote reader 1 is configured for use with a user interface card, which, in the cards shown in Figs. 1 to 3, takes the form of an electronic smart card 10A. The smart card 10A includes a laminar substrate 12 with various control indicia 14 in the form of a four way directional controller 20, a "jump button" 22, a "kick button" 24, a "start button" and an "end button" printed on an upper face 16 thereof. Other non-control indicia, such as promotional or instructional material, can be printed alongside the control indicia. For

example, advertising material 26 can be printed on the front face of the smart card 10A or on a reverse face 27 of the card 10A, as seen in Fig. 2.

As seen in Fig. 3, the smart card 10A includes storage means in the form of an on-board memory chip 19 for data associated with the control indicia. The smart card 10A also includes electrical data contacts 18 connected to the on-board memory chip 19 corresponding with the exposed contacts 7 on the remote reader 1.

As seen in Fig. 3, the upper face 16 may be formed by an adhesive label 60 upon which are printed control indicia 14, in this case corresponding to the "End Button" and the Right arrow "button" of the directional controller 20. The label 60 is affixed to the laminar substrate 12. A home user can print a suitable label for use with a particular smart card 10A by using a printer, such as a colour BUBBLEJET ™ printer manufactured by Canon, Inc. Alternatively, the control indicia 14 can be printed directly onto the laminar substrate or separate adhesive labels can be used for each of the control indicia.

In use, the smart card 10A is inserted into the card receptacle 4, such that the pressure sensitive touch panel 8 covers the upper face 16 of the smart card 10A. In this position, the control indicia are visible within the viewing area 6 through the transparent pressure sensitive touch panel 8.

The exposed contacts 7 and associated circuitry of the reader 1 are configured to read the stored data associated with the control indicia 14 from the memory chip 19, either automatically upon insertion of the smart card 10A into the control template receptacle 4, or selectively in response to a signal from the remote reader 1. The signal can, for example, be transmitted to the smart card 10A via the exposed contacts 7 and data contacts 18.

Once the data associated with the control indicia 14 has been read, a user can press areas of the pressure sensitive touch panel 8 on or over the underlying control

indicia 14. By sensing the pressure on the pressure sensitive touch panel 8 and referring to the stored data, the remote reader 1 can deduce which of the control indicia 14 the user has selected. For example, if the user places pressure on the pressure sensitive touch panel 8 adjacent the "kick button" 24, the remote reader 1 is configured to assess the position at which the pressure was applied, refer to the stored data, and determine that the "kick" button 24 was selected. This information can then be used to control an external device, for example, an associated video game console (of conventional construction and not shown). It will be appreciated from above that the control indicia 14 are not, in fact buttons. Rather, the control indicia 14 are user selectable features which, by virtue of their corresponding association with the mapping data and the function of the touch panel 8, operate to emulate buttons traditionally associated with remote control devices.

In one advantageous implementation, the remote reader 1 includes a transmitter (of conventional type and not shown), such as an infra-red (IR) transmitter or radio frequency (RF) transmitter, for transmitting information in relation to indicia selected by the user. As seen in Fig. 1, the remote reader 1 incorporates an IR transmitter having an IR light emitting diode (LED) 25. Upon selection of one of the control indicia 14, the remote reader 1 causes information related to the selection to be transmitted to a remote console (not shown in Fig.1) where a corresponding IR or RF receiver can detect and decode the information for use in controlling some function, such as a game being played by a user of the reader 1.

Any suitable transmission method can be used to communicate information from the remote reader 1 to the remote console, including direct hard wiring. Moreover, the remote console itself can incorporate a transmitter, and the remote reader 1 a receiver, for communication in an opposite direction to that already described. The communication from the remote console to the remote reader 1 can include, for example, handshaking

data, setup information, or any other form of information desired to be transferred from the remote console to the remote reader 1.

Turning to Fig. 4, there is shown a control card 10B. The control card 10B includes a laminar substrate 12, which bears control indicia (not illustrated). In the control card 10B the storage means takes the form of a magnetic strip 29 formed along an edge 28 of the reverse face 27 of the control card 10B. The stored data associated with the control indicia may be stored on the magnetic strip 29 in a conventional manner. A corresponding reader (not shown) for this arrangement includes a magnetic read head positioned at or adjacent an entrance to the corresponding control template receptacle. As the control card 10B is slid into the card receptacle, the stored data is automatically read from the magnetic strip 29 by the magnetic read head. The reader 1 may then be operated in a manner corresponding to the card 10A of Fig. 1.

Fig. 5 shows another card in the form of a control card 10C, in which the storage means takes the form of machine-readable indicia. In the card 10C of Fig. 5, the machine readable indicia takes the form of a barcode 36 formed along an edge 38 of the reverse face 27 of the card 10C. The stored data is suitably encoded, and then printed in the position shown. A corresponding controller (not shown) for the card 10C of Fig. 5 includes an optical read head positioned at or adjacent an entrance to the associated control template receptacle. As the card 10C is slid into the control receptacle, the stored data is automatically read from the barcode 36 by the optical read head. Alternatively, the barcode can be scanned using a barcode reader associated with the reader immediately prior to inserting the card 10C, or scanned by an internal barcode reader scanner once the card 10C has completely been inserted. The card 10C may then be operated in a manner again corresponding to the card 10A of Fig. 1. It will be appreciated that the position, orientation and encoding of the barcode can be altered to suit a particular application.

Moreover, any other form of machine readable indicia can be used, including embossed machine-readable figures, printed alpha-numeric characters, punched or otherwise formed cut outs, optical or magneto optical indicia, two dimensional bar codes. Further, the storage means can be situated on the same side of the card 10A or 10B or 10C as the control indicia.

Fig. 6(a) shows a hardware architecture of a card interface system 600A. In the system 600A, the remote reader 1 is hard wired to a personal computer system 100 via a communications cable 3. Alternatively, instead of being hardwired, a radio frequency or IR transceiver 106 can be used to communicate with the remote reader 1. The personal computer system 100 includes a screen 101 and a computer module 102. The computer system 100 will be explained in more detail below with reference to Fig. 7. A keyboard 104 and mouse 203 are also provided.

The system 600A includes a smart card 10D which is of similar configuration to the smart card 10A described above. The smart card 10D is programmable and can be created or customised by a third party, which in this case can be a party other than the manufacturer of the card 10D and/or card reader 1. The third party can be the ultimate user of the smart card 10D itself, or may be an intermediary between the manufacturer and user. In accordance with the system 600A of Fig. 6(a), the smart card 10D can be programmed and customised for one touch operation to communicate with the computer 100 and obtain a service over a network 220, such as the Internet. The computer 100 operates to interpret signals sent via the communications cable 3 from the remote reader 1, according to a specific protocol, which will be described in detail below. The computer 100 performs the selected function according to touched control indicia, and can be configured to communicate data over the network 220. In this manner the

computer 100 can permit access to applications and/or data stored on remote server computers 150, 152 and appropriate reproduction on the display device 101.

Fig. 6(b) shows a hardware architecture of a card interface system 600B. In the system 600B, the remote reader 1 can be programmed for obtaining a service locally at a set top box 601, that couples to an output interface, which in this example takes the form of an audio-visual output device 116, such as a digital television set. The set-top box 601 operates to interpret signals 112 received from the remote reader 1, which may be electrical, radio frequency, or infra-red (IR), and according to a specific protocol which will be described in detail below. The set top box 601 can be configured to perform the selected function according to touched control indicia and permit appropriate reproduction on the output device 116. Alternatively, the set top box 601 can be configured to convert the signals 112 to a form suitable for communication and cause appropriate transmission to the computer 100. The computer 100 can then perform the selected function according to the control indicia, and provide data to the set-top box 601 to permit appropriate reproduction on the output device 116. The set top box 601 will be explained in more detail below with reference to Fig. 42.

In one application of the system 600B, the smart card 10D can be programmed for obtaining a service both remotely and locally. For instance, the smart card 10D can be programmed to retrieve an application and/or data stored on remote server computers 150, 152, via the network 220, and to load the application or data on to the set top box 601. The latter card can be alternatively programmed to obtain a service from the loaded application on the set top box 601.

Unless referred to specifically, the systems 600A and 600B will be hereinafter generically referred to as the system 600. Further, unless referred to specifically, the

smart cards 10A, 10B, 10C and 10D will be hereinafter generically referred to as the smart card 10.

Fig. 7 shows the general-purpose computer system 100 of the system 600, which can be used to run the card interface system and to run software applications for programming the smart card 10. The computer system 100 includes a computer module 102, input devices such as a keyboard 104 and mouse 203, output devices including the printer (not shown) and the display device 101. A Modulator-Demodulator (Modem) transceiver device 216 is used by the computer module 102 for communicating to and from the communications network 220, for example connectable via a telephone line 221 or other functional medium. The modem 216 can be used to obtain access to the Internet, and other network systems, such as a Local Area Network (LAN) or a Wide Area Network (WAN).

The computer module 102 typically includes at least one central processing unit (CPU) 205, a memory unit 206, for example formed from semiconductor random access memory (RAM) and read only memory (ROM), input/output (I/O) interfaces including a video interface 207, and an I/O interface 213 for the keyboard 104 and mouse 203, a write device 215, and an interface 208 for the modem 216. A storage device 209 is provided and typically includes a hard disk drive 210 and a floppy disk drive 211. A magnetic tape drive (not illustrated) is also able to be used. A CD-ROM drive 212 is typically provided as a non-volatile source of data. The components 205 to 213 of the computer module 201, typically communicate via an interconnected bus 204 and in a manner, which results in a conventional mode of operation of the computer system 102 known to those in the relevant art. Examples of computers on which the arrangement described herein can be practised include IBM-computers and compatibles, Sun Sparcstations or alike computer system evolved therefrom.

Typically, the software programs of the system 600 are resident on the hard disk drive 210 and read and controlled in their execution by the CPU 205. Intermediate storage of the software application programs and any data fetched from the network 220 may be accomplished using the semiconductor memory 206, possibly in concert with the hard disk drive 210. In some instances, the application programs can be supplied to the user encoded on a CD-ROM or floppy disk and read via the corresponding drive 212 or 211, or alternatively may be read by the user from the network 220 via the modem device 216. Still further, the software can also be loaded into the computer system 102 from other computer readable medium including magnetic tape, ROM or integrated circuits, a magneto-optical disk, a radio or infra-red transmission channel between the computer module 102 and another device, a computer readable card such as a smart card, a computer PCMCIA card, and the Internet and Intranets including email transmissions and information recorded on Websites and the like. The foregoing is merely exemplary of relevant computer readable media. Other computer readable media are able to be practised without departing from the scope of the invention defined by the appended claims.

The smart card 10 can be programmed by means of a write device 215 coupled to the I/O interface 213 of the computer module 102. The write device 215 can have the capability of writing data to the memory on the smart card 10. Preferably, the write device 215 also has the capability of printing graphics on the top surface of the smart card 10. The write device 215 can also have a function reading data from the memory on the smart card 10. Initially, the user inserts the smart card 10 into the write device 215. The user then enters the required data via the keyboard 104 of the general purpose computer 102 and a software application writes this data to the smart card memory via the write

device 215. If the stored data is encoded for optical decoding such as using a barcode, the write device can print the encoded data onto the smart card 10.

Fig. 42 shows the set top box 601 of the system 600, which can be used to interpret signals 112 received from the remote reader 1. The set top box 601 in some implementations essentially is a scaled version of the computer module 102. The set top box 601 typically includes at least one CPU unit 4305, a memory unit 4306, for example formed from semiconductor random access memory (RAM) and read only memory (ROM), and input/output (I/O) interfaces including at least an I/O interface 4313 for the digital television 116, an I/O interface 4315 having an IR transceiver 4308 for receiving and transmitting the signals 112, and an interface 4317 for coupling to the network 220. The components 4305, 4306, 4313, 4315 and 4317 of the set top box 601, typically communicate via an interconnected bus 4304 and in a manner which results in a conventional mode of operation. Intermediate storage of any data received from the remote reader 1 or network 220 may be accomplished using the semiconductor memory 4306. Alternatively, the set top box can include a storage device (not shown) similar to the storage device 209.

The card interface system 600 will now be explained in more detail in the following paragraphs.

2.0 CARD INTERFACE SYSTEM SOFTWARE ARCHITECTURE

2.1 Software Architecture Layout

A software architecture 200 for the hardware architectures depicted by the system 600, is generally illustrated in Fig. 8. The architecture 200 can be divided into several distinct process components and one class of process. The distinct processes include an I/O interface 300, which may be colloquially called an "I/O daemon" 300, an event manager 301, a display manager 306, an (application) launcher 303 and a directory

service 311. The class of process is formed by one or more applications 304. In the architecture 200 described herein , there exists one I/O daemon 300, one event manager 301, one display manager 306 and one launcher 303 for every smart card remote connection, usually formed by the set-top box 601, and one master launcher (not shown) for each computer 100 (e.g. the server computers 150, 152) that is running the launchers 303, and at least one directory service 311 for all systems. The Directory service 311, is queried by the launcher 303 to translate service data into a Resource Locater (eg. URL) that indicates a name or location of a service or the location or name of an application 304 to be used for the service.

In this form, the architecture 200 can be physically separated into six distinct parts 101, 307, 309, 312, 313 and 601 as shown by the dashed lines in Fig. 8, each of which can be run on physically separate computing devices. Communication between each of the parts of the system 600 is performed using Transport Control Protocol/ Internet Protocol (TCP/IP) streams. Alternatively, each of the parts 101, 307, 309, 312, 313 and 601 can be run on the same machine.

In the system 600A of Fig. 6(a), all of the process components 300, 301, 303, 304 and 306 can be run on the computer 100. The event manager 301, the launcher 303 and display manager 306 are preferably all integrated into one executable program which is stored in the hard disk 209 of the computer 100 and can be read and controlled in its execution by the CPU 205. The directory service 311 runs on the same computer 100 or on a different computer (e.g. server 150) connected to the computer 100 via the network 220.

In the system 600B of Fig. 6(b), all of components 300 to 304 and 306 can run from the set-top-box 601. In this instance, the components 300 to 304 and 306 can be stored in the memory 4306 of the set top box 601 and can be read and controlled in their execution

by the CPU 4305. The directory service 311 can run on the computer 100 and can be stored in the memory 206 of the computer 100 and be read and controlled in its execution by the CPU 205. Alternatively, the directory service 311 can be run on the set top box 601 or its function performed by the launcher 303.

Alternatively, if the set-top-box 601 is not powerful enough to run the system 600 locally, only the I/O daemon 300 need run on the set-top-box 601 and the remainder of the architecture 200 (i.e. process components 301, 303, 304, 306 and 311) can run remotely on the other servers (150, 152) which can be accessed via the network 220. In this instance, the I/O daemon 300 can be stored in the memory 4306 of the set top box 601 and can be read and controlled in its execution by the CPU 4305. Again, the functional parts of such a system can be divided as shown in Fig. 8.

2.1.1 I/O Daemon

The I/O daemon 300 is a process component that converts datagrams received from the remote reader 1 into a TCP/IP stream that can be sent to the event manager 301 and vice versa (e.g. when using a two-way protocol). Any suitable data format can used by the remote reader 1. The I/O daemon 300 is preferably independent of any changes to the remote reader 1 data format, and can work with multiple arrangements of the remote reader 1. In one advantageous implementation of the system 600, the I/O daemon 300 is integrated into the event manager 301.

In the system 600A, the I/O daemon 300 is started when a user starts the smart card system 600 by powering up the computer 100 and the event manager 301 has been started. Alternatively, the I/O daemon 300 is started when a user starts the system 600 by turning on the set-top box 601.

The I/O daemon 300 will be explained in more detail below with reference to section 9.0.

2.1.2 Event Manager

The event manager 301 forms a central part of the architecture 200 in that all communications are routed through the event manager 301. The event manager 301 is configured to gather all events that are generated by the remote reader 1 and relayed by the I/O daemon 300. These events are then redistributed to the various process components 300 to 304, and 306 and running applications. The event manager 301 is also configured to check that an event has a valid header, correct data length, but is typically not configured to check that an event is in the correct format. An "event" in this regard represents a single data transaction from the I/O daemon 300 or the launcher 303 or applications 304.

Any changes in protocol between different systems can be dealt with by the event manager 301. Where possible, events can be rewritten to conform to the data format understood by any presently running application 304. If such is not possible, then the event manager 301 reports an error to the originating application 304. When different data formats are being used, for example with a system running multiple smart cards, the event manager 301 preferably ensures that the smallest disruption possible occurs.

The event manager 301 does not have any presence on the display screen or other output device 116. However, the event manager 301 can be configured to instruct the display manager 306 as to which application is presently required (i.e. the "front" application) and should currently be displayed on the display 101. The event manager 301 infers this information from messages passed to the applications 304 from the launcher 303 as will be explained in more detail below with reference to section 10.0.

The event manager 301 can be configured to always listen for incoming I/O daemon connections or alternatively, can start the system 600. The method used is dependent on the overall configuration of the system 600. In this connection, the event

manager 301 can start the system 600 or the set top box 601 can use the incoming connection of the I/O daemon 300 to start the system 600. The event manager 301 will be described in more detail below with reference to section 7.0.

2.1.3 Master Launcher

Where a thin client computer is being utilised and multiple launchers 303 are running with each launcher 303 being responsible for one set top box, a master launcher (not shown) which communicates directly with the event manager 301 can be used. The master launcher is used to start the launcher 303 corresponding to each of the event managers 301 if more than one event manager is running on the system 600. Initially, when the I/O daemon 300 connects to the event manager 301, the event manager 301 requests that the master launcher start a first process for the event manager 301. This first process is generally the launcher 303 for any smart card application 304. The master launcher can also be configured to shut down the launcher 303 of an application 304 when the event manager 301 so requests, and for informing the event manager 301 that the launcher 303 has exited.

There is preferably one master launcher running for each physically separate server (e.g. 150, 152) that is running an associated smart card application 304. This one master launcher handles the requests for all event managers that request launchers on a particular server. When run on a computer 100, as seen in Fig. 7, the master launcher commences operation either before or no later than at the same time as the rest of the system 600. In this instance, the master launcher is started first.

The master launcher can be integrated into the event manager 301, for example, when an associated launcher is running on the same computer as the event manager 301.

2.1.4 Launcher/First Application

In one advantageous implementation of the system 600, the first process started by the insertion of a smart card 10 into the remote reader 1 is the launcher 303. In specific systems, specific applications may be commenced, for example an automatic teller machine can start a banking application. Another example includes the use of restricted launchers that only start a specified sub-set of applications. The launcher 303 is an application that starts other applications for a specific event manager 301. The launcher 303 starts and ends applications and can also start and end sessions. The launcher 303 also informs the event manager 301 when applications are starting and ending, and tells the applications 304 when they are receiving or losing focus, or when they need to exit. In this regard, where a number of applications 304 are operating simultaneously, the application 304 that is currently on-screen is the application having focus, also known as the "front application". When another application is about to take precedence, the launcher 303 tells the front application that it is losing focus, thereby enabling the current application to complete its immediate tasks. The launcher 303 also tells the new application 304 that it is gaining focus, and that the new application 304 shall soon be changing state. The launcher 303 is also configured to force an application to exit.

The launcher 303 may receive certain events such as "no-card", "low battery" and "bad card" events generated by the remote reader 1. The launcher 303 also receives events that are intended for applications that are not currently the front application, and the launcher 303 operates to correctly interpret these events.

The launcher 303 is preferably only started when a request is generated by the event manager 301 to request the launcher 303 to be started. The launcher 303 can also be told to exit and forced to exit by the event manager 301.

The launcher 303 is preferably the only process component that needs to communicate with the directory service 311. When the launcher 303 is required to start a

new application 304, the launcher 303 queries the directory service 311 with service data, and the directory service 311 returns a location of the application 304 and service data associated with the new application 304. The service data is sent to the new application 304 as initialisation data in an event, referred to herein as the EM_GAINING_FOCUS event. The application location specifies the location of the application 304 to be run. This may be local, for implementations with a local computer, or networked. If the application location is empty, then the launcher 303 has to decide which application to start based on the service data.

The launcher 303 can also be configured to start any applications, for example browser controllers that will generally always be running while the system 600 is operating. Such applications are referred to as persistent applications. Applications can also be started by the launcher 303 either as a response to the first user selection on a corresponding smart card 10, or at the request of another one of the applications 304.

The launcher 303 can be integrated into the event manager 301 in some implementations of the system 600.

The launcher 303 will be explained in more detail below with reference to section 10.0.

2.1.5 Display Manager

The display manager 306 selects which smart card application 304 is currently able to display output on the display screen 101. The display manager 306 is told which application 304 can be displayed by an EM_GAINING_FOCUS event originating from the launcher 303. This event can be sent to the display manager 306 directly, or the event manager 301 can send copies of the event to the display manager 306 and the intended recipient.

Generally, the only application 304 that should be attempting to display output should be the front application. The display manager 306 can provide consistent output during the transfer between applications having control of the display. The display manager 306 may need to use extrapolated data during changeovers of applications as the front application.

The architecture 200 can be configured such that the display manager 306 is not needed or the role of the display manager 306 may be assumed by the other parts 301or 303, of the architecture 200.

2.1.6 Directory Service

The directory service 311 is configured to translate service identifiers that are stored on smart cards 10, into resource locators (e.g. a URL) that indicate the location of the services or the location of an application associated with a service. The directory service 311 is also configured to translate optional service data. The directory service 311 allows the launcher 303 associated with a particular card 10 to decide what to do with a resource locator, for example, download and run the associated application 304 or load the resource locator into a browser application. The translation by the directory service can be performed using a distributed lookup system.

2.1.7 Applications

The applications 304 associated with a particular smart card 10 can be started by the launcher 303 associated with that smart card 10 in a response to a first button press on a corresponding card. Each application 304 can be a member of one or more service groups, described in detail later in this specification. An application 304 can be specified to not be part of any service group in which case the application will never be run with other applications. An application can become part of a service group once the

application is running and can remove itself from a service group when the application is the currently front application.

Some applications can be started when the system 600 is started and these applications, for example a browser control application or a media playing application can be always running. These persistent applications can be system specific or more generally applicable.

Fig. 9 is a schematic block diagram representation of a card interface system, including the process components 301 to 306 described above. In the system of Fig. 9, the remote reader 1 communicates with a computer 100 via an IR link in conjunction with an I/O daemon 300 for controlling the IR link. Further, the computer 100 is configured for communicating to and from a communications network in this case represented by the Internet 400 to a Web server 410. In this instance, some of the applications 304 accessible utilising the smart cards 10 and remote reader 1 can be Web pages 406 associated with different smart cards 10. The Web libraries 407 contain functions (e.g. JavaScript functions) and classes (e.g. Java classes) that can be included with web pages for use with the smart card 10. The Web pages 406 can be accessed with a running application called the Web browser 403. In the system of Fig. 9, the event manager 301 is configured to receive an event from the remote reader 1. The event is then sent to the launcher 303, which can be configured to send a message to the browser controller 402, which controls the Web browser 403. The process for starting an application or browser session will be explained in more detail below. The launcher 303 can also be configured to download applications 408 as well as running applications from a file server 411 which is also connected to the computer 100 via the Internet 400.

3.0 READER

The remote reader 1 is preferably a hand-held, battery-powered unit that interfaces with a smart card 10 to provide a customisable user interface. As described above, the remote reader 1 is intended for use with a digital television, a set top box, computer, or cable television equipment to provide a simple, intuitive interface to on-line consumer services in the home environment.

Figs. 43 and 44 show a reader 4401 similar to the reader 1 described above. The reader 4401 is configured for the reading of the card 10. The reader 4401 is formed of a housing 4402 incorporating a card receptacle 4404 and a viewing area 4406. The receptacle 4404 includes an access opening 4410 through which a smart card 10, seen in Fig. 1, is insertable.

An upper boundary of the viewing area 4406 is defined by sensor means in the form of a substantially transparent pressure sensitive membrane 4408 similar to the membrane 8 described above. Arranged beneath the membrane 4408 is data reading means provided in the form of an arrangement of exposed electrical contacts 4407 configured to contact complementary contacts of the smart card 10.

The card 10 is inserted into the reader 4401 via the access opening 4410 as shown in Fig. 45. The configuration of the reader 4401 allows a user to hold the reader 4401 in one hand and easily insert the smart card 10 into the reader 4401 with the user's other hand. When the smart card 10 is fully inserted into the reader 4401, the pressure sensitive membrane 4408 fully covers the upper face 16 of the smart card 10. The viewing area 4406 preferably has substantially the same dimensions as the upper face 16 of the card 10 such that the upper face 16 is, for all intents and purposes, fully visible within the viewing area 4406 through the transparent pressure sensitive membrane 4408.

Fig. 46 shows a user operating the reader 4401 after a card has been fully inserted.

Referring to Figs. 47(a) to 47(c), the housing 4402 is formed of a substantially two part outer shell defined by a top section 4827 that surrounds the membrane 4408, and a base section 4805 which extends from a connection 4829 with the top section 4827 to a location 4811 below and proximate the transverse centre of the membrane 4408. The base section 4805 incorporates a facing end 4815 formed from infrared (IR) transparent material thereby permitting IR communications being emitted by the reader 4401.

The location 4811 defines a point of connection between the base section 4805 a card support surface 4807 which extends through a plane in which the contacts 4407 lie to an interior join 4835 that sandwiches the membrane 4408 between the surface 4807 and the top section 4827. The access opening 4410 is substantially defined by the space between the location 4811 and a periphery 4836 of the housing 4402, seen in Fig. 47(a).

The contacts 4407 extend from a connector block 4837 mounted upon a printed circuit board (PCB) 4801, the PCB 4801 being positioned between the base section 4805 and the support surface 4807 by way of the two mountings 4817 and 4819. Arranged on an opposite side of the PCB 4801 to the connector block 4837 is electronic circuitry (not shown), electrically connected to the connectors 4407 and the touch sensitive membrane 4408 and configured for reading data from the card 10 according to depression of the membrane 4408. Also mounted from the PCB 4801 is an infrared light emitting diode (LED) 4800 positioned adjacent the end 4815 which acts as an IR window for communications with a device (e.g. the set top box 601) to be controlled.

Fig. 47(b) shows a similar view to Fig. 47(a), with the smart card 10 partially inserted through the access opening 4410 into the receptacle 4404. As can be seen in Fig. 47(b), the support surface 4807 has an integrally formed curve contour 4840 that leads downward from the plane of the contacts 4407 towards the join 4811. This configuration allows the reader 4401 to receive the smart card 10 such that the smart card 10 may be

initially angled to the plane of the receptacle 4404, as seen in Fig. 47(b). The configuration of the curve contour portion 4840 of the support surface 4807 guides the smart card 10 into a fully inserted position under the force of the user's hand. Specifically, as the card 10 is further inserted, the curvature of the support surface 4807 guides the card 10 into the plane of the contacts 4407 and receptacle 4404.

Fig. 47(c) shows a similar view to Fig. 47(a), with the smart card 10 fully inserted into the receptacle 4404. In this position, the card 10 lies in the plane of the receptacle 4404 and the contacts 4407 which touch an associated one of the data contacts (not seen) of the smart card 10, and the smart card 10 is covered by the pressure sensitive membrane 4408. Further, the contacts 4407 are preferably spring contacts that act to provide a force against the card 10 and associated with the membrane 4408, sufficient for the card 10 to be held within the receptacle by a neat interference fit.

In the following description references to the reader 1 can be construed as references to a reader implemented as the reader 1 of Fig. 1 or the reader 4401 of Figs. 43 to 47(c).

Fig. 10 is a schematic block diagram showing the internal configuration of the remote reader 1 in more detail. The remote reader 1 includes a microcontroller 44 for controlling the remote reader 1, coordinating communications between the remote reader 1 and a set top box 601, for example, and for storing mapping information. The microcontroller 44 includes random access memory (RAM) 47 and flash (ROM) memory 46. The microcontroller 44 also includes a central processing unit (CPU) 45. The microcontroller 44 is connected to a clock source 48 and a clock controller 43 for coordinating the timing of events within the microcontroller 44. The CPU 45 is supplied with electrical power from a 5 volt battery 53, the operation of the former being

controlled by a power controller 50. The microcontroller 44 is also connected to a beeper 51 for giving audible feedback about card entry status and for "button" presses.

Infra-red (IR) communications are preferably implemented using two circuits connected to the microcontroller 44, an IR transmitter (transmitter) 49 for IR transmission and an IR receiver (receiver) 40 for IR reception.

The pressure sensitive touch panel 8 of the remote reader 1 communicates with the microcontroller 44 via a touch panel interface 41. A smart card interface 42 connects to the electrical contacts 7.

An in-system programming interface 52 is also connected to the microcontroller 44, to enable programming of the microcontroller 44 by way of the microcontroller FLASH memory 46 with firmware. The firmware will be explained in further detail later in this document with reference to section 6.0.

The internal configuration of the remote reader 1 will now be described in further detail.

3.1 Low Power Mode Lifetime

The power controller 50 is operable to provide two power modes, one being a low-power "sleep" mode, and another being an active mode. The low power mode lifetime is the lifetime of the battery 53 expressed in years. When the remote reader 1 is not functioning and is in the low power mode, the lifetime can be between greater than 2 years.

If the reader 1 is in sleep mode and a user presses the touch panel 8, the remote reader 1 then comes out of sleep mode, and the CPU 45 calculates the touch co-ordinates and sends a serial message by infra-red transmission. The battery 53 should preferably remain serviceable for the current supply requirements of more than 100,000 button presses.

3.2 Service Life

The service life is defined as the period of time that the remote reader 1 can be expected to remain serviceable, not including battery replacement. The service life is related to the Mean Time Between Failures (MTBF) figure and is usually derived statistically using accelerated life testing. The service life of the remote reader 1 can thus be greater than 5 years.

3.3 Microcontroller

The microcontroller 44 of the remote reader 1 has an 8 bit central CPU with 4096 bytes of FLASH memory 46 and 128 bytes of random access memory 47. The microcontroller 44 preferably operates on a supply voltage from 3 to 5 Volts and has flexible on-board timers, interrupt sources, 8 bit analog to digital converters (ADC), clock watchdog and low voltage reset circuits. Preferably, the microcontroller 44 also has high current output pins and can be programmed in circuit with only a few external connections.

3.4 Clock Source

The main clock source 48 for the remote reader 1 is preferably a 3 pin 4.91MHz ceramic resonator with integral balance capacitors. The frequency tolerance is 0.3%. While such tolerance is not as good as a crystal, such is however adequate for serial communications and is much smaller and cheaper than a crystal.

3.5 Beeper

The beeper 51 is included with the remote reader 1 to give audible feedback about card entry status and for button presses. The beeper 51 is preferably a piezo-ceramic disk type.

3.6 Infra-red Communications

As described above, infra-red (IR) communications are preferably implemented using two circuits, an IR transmitter 49 for IR transmission and an IR receiver 40 for IR reception. The two circuits 40 and 49 are preferably combined on a printed circuit board (e.g the PCB 4801 of Fig. 47) within the remote reader 1. The printed circuit board 4801 can be connected to the microcontroller 44 by a 4 way flat printed cable. Large bulk decoupling capacitors (not shown) are required on the PCB 4801 to provide surge currents, which are required when transmitting.

3.7.1 Infra-red Transmission

IR transmission is preferably by means of an infra-red Light Emitting Diode (LED) (e.g. the LED 4800 of Fig 47(a)) forming part of the IR transmitter 49.

3.7.2 Infra-red Reception

The IR receiver 40 is preferably integrated with an infra-red filter, a PIN diode, an amplifier and discriminator circuitry into a single device. Received serial information passes directly from this device to an input port of the microcontroller 44. This port can be programmed to generate an interrupt on receiving data allowing speedy storage and processing of incoming signals.

3.8 CPU / Memory Card Interface

The remote reader 1 can preferably support smart cards 10 as defined by the International Standards Organisation (ISO) standards 7816-3 and ISO 7810. Three and five volt CPU cards (i.e. cards with an embedded microprocessor) with T=0 and T=1 protocols can also be supported as are 3 and 5V memory cards.

The electrical contacts 7 used to make contact between the card 10 and the microcontroller 44 are preferably a surface mount connector with 8 sliding contacts and a "card in" switch. In accordance with the ISO requirements the following signals must be provided:

- Pin 1 - VCC - Supply voltage;
- Pin 2 - RST - Reset signal. Binary output to card;
- Pin 3 - CLK - Clock signal, Binary output to card;
- Pin 4 - RFU - Reserved, leave unconnected;
- Pin 5 - GND – Ground;
- Pin 6 - VPP - Programming voltage, not required, link to GND, VCC or open;
- Pin 7 - I/O - Data I/O, bi-directional signal; and
- Pin 8 - RFU - Reserved, leave unconnected.

The RST and I/O pins are preferably connected directly to the microcontroller 44. All pins except the power supplies are equipped with series termination and transient voltage suppressor diodes to prevent electrostatic discharge problems.

3.9 CPU Card Power Supply

As described above, the microcontroller 44 requires a 3 - 5 Volt power supply for operation. The 5 Volt supply can be generated from a 3V Lithium coin cell operating as the battery 53 by means of the power controller 50 in the form of a regulated 5V charge-pump DC-DC converter chip.

3.10 Touch Sensitive Interface

As described above, the pressure sensitive touch panel 8 of the remote reader 1 communicates with the microcontroller 44 via a touch panel interface 41. The touch panel interface 41 provides an analog signal according to the position of the touch on the touch panel 8. This analog signal is then communicated to the microcontroller 44.

The calculation of touch co-ordinates requires bottom and left touch panel 8 contacts (not shown) to be connected to the inputs of an analog to digital converter on the microcontroller 44.

A touch on the touch panel 8 can preferably be used to wake up the remote reader 1 from sleep mode. A resistive connection from the left screen contact to a sleep WAKE UP port as illustrated provides this feature. Note that during in-system programming, up to 8 volts may be applied to a pin on the microcontroller 44 referred to as the Interrupt Request Pin (IRQ) so a clamping diode needs to be fitted to this pin to prevent device damage. In this instance, it is the internal pull up on the IRQ pin that actually provides the bias required to detect touch panel 8 presses.

3.11 Battery

As described above, the remote reader 1 uses a battery 53. A 5 Volt lithium coin cell can be used as the battery 53 to power all the circuitry of the remote reader 1.

3.12 In System Programming

The microcontroller supports in-system programming (ISP) options. The in-system programming interface 52 is used in the remote reader 1 to perform programming of the microcontroller 44 such as programming of the microcontroller FLASH ROM memory 46 with firmware.

3.13 Printed Circuit Boards and Interconnection

The remote reader 1 can include two printed circuit boards (PCB), instead of the one PCB 4801 of the reader 4401, as follows:

(i) an infra-red (IR) PCB which holds the infra-red diode, drive FET and receiver; and

(ii) a main PCB (e.g. the PCB 4801 of Fig. 47(a)) which holds all the other components 40 to 53 mentioned above.

Both of the PCB boards described above are preferably double sided designs using standard grade FR4, 1.6mm PCB material. The main PCB preferably utilises surface

mount components since the thickness of the finished PCB is critical and preferably components are restricted to a height of approximately 3mm max.

The IR PCB can use through hole parts but again there are preferably stringent component height restrictions imposed. The interconnection of the two PCBs is via a custom designed 4 way flat printed cable (FCA). This interfaces to the two PCBs via a surface mount FCA connector that is the same part used to interface to the touch panel 8.

3.14 Low Power Mode

When the remote reader 1 has not been used for a short period of time, pre-programmed firmware preferably puts the unit into the low-power mode to conserve battery life. In low-power mode, the supply voltage is switched off to all current consuming components, the ports of the microcontroller 44 are set into a safe sleep state and the clock 48 is stopped. In this state the current consumption of the remote reader 1 is less than 5 μA. A P-channel FET can be used to control the supply of power to the current consuming components.

There are three alternative preferred methods to wake the remote reader 1 up from low power mode as follows:

- touch the touch panel 8;
- insert a card into the card receptacle 4; and
- remove and re-insert the battery 53.

The card insert wake up enables the remote reader 1 to always beep when a card is inserted, regardless of whether the unit is in low power mode or not. The touch and card insert wake ups are handled by the IRQ pin as illustrated on the microcontroller 44. It is important that this pin is set to "edge trigger" only so that only a new touch or card insert wakes the microcontroller up. If IRQ sensitivity is set to "level" trigger then

inadvertently leaving the touch panel 8 pressed, for example when the remote reader 1 is packed in luggage, would prevent the remote reader 1 from entering low power mode.

3.15 Interrupts and Resets

The microcontroller 44 firmware for the remote reader 1 uses two external and one internal interrupt sources. External interrupts come from the IRQ pin for low power mode wake up. The internal interrupt is triggered by a timer overflow and is used to time various external interfaces. These interrupts are serviced by pre-programmed firmware procedures.

There are four possible reset sources for the microcontroller as follows:

- low supply voltage reset at 2.4 Volts;
- illegal firmware op-code reset;
- Computer Operating Properly (COP) reset if firmware gets stuck in a loop; and
- ISP reset forced onto a RESET pin when in-system programming (ISP) starts.

4.0 CARD DATA FORMAT

The format of data for the card 10 described above will be described in the following paragraphs. For memory cards such as the control card 10B as described in relation to Fig. 4, data conforming to the format to be described will be copied directly onto the card. For the CPU cards described above, data conforming to the format to be described can be loaded as a file into the file system of the CPU of the card.

The card 10 described above preferably stores a data structure that describes various card properties and any user- interface indicia printed on the card. The cards 10 can also include global properties that specify attributes such as information about the card, vendor and a service. User-interface objects, if present, specify data to associate with areas of the surface of the card 10.

The user-interface objects as described herein, represent mapping data, which relate predetermined areas, or iconic representations directly imprinted on a surface of the card 10, to commands or addresses (eg: Uniform Resource Locators (URLs)). The mapping data includes coordinates which typically define the size and location of user Interface Elements (eg: predetermined areas) on the card 10. In this connection, the term user interface element typically refers to indicia on the card 10, whilst the term user interface object typically refers to the data related to a particular indicia. However, these terms are used interchangeably throughout the following description.

The user-interface objects are preferably stored directly on the card 10. Alternatively, the user-interface objects can be stored not on the card 10 itself, but in the system 600. For example, the card 10 can store, via an on-card memory, a barcode or a magnetic strip, a unique identifier, which is unique to cards 10 having substantially similar user interface elements and layout. The unique identifier together with the coordinates determined from the touch panel 8, as a result of a user press, can be transmitted by the reader 1 to the computer 100 or to the set top box 601, of the system 600. The system 600 having the user-interface objects stored on the computer 100, set top box 601 or a server 150, can perform the mapping from the determined coordinates to a corresponding command, address or data relevant to a service associated with the card 10 and the user press, in order to provide a desired function represented by the user interface element on the card 10. In this instance, the data related to the user selected indicia as described above takes the form of coordinates determined by the reader 1 as a result of a user press on a portion of the touch panel 8 which overlays the desired indicia.

In the cards (e.g. 10) described above, data stored by the card 10 includes a card header followed by zero or more objects as described in the following sections.

4.1 Card Header

Fig. 11 shows the data structure of a card header 1100 as stored in the smart card 10. The header 1100 includes a number of rows 1101, each of which represent four bytes of data. The data is preferably in 'big endian' format. The complete header is 20 bytes long and includes the following fields (described in more detail in Fig. 12):

(i) magic number field 1102, which includes a constant specifying a card as being a valid memory card. For example, the magic number field 1102 can be used to check or verify that a propriety card belonging to a particular manufacture is being used.

(ii) versions field 1103, which includes each version increment that specifies a change in the card layout that can not be read by a reader which is compatible with lower versions of the layout;

(iii) reserved field 1104, this field is reserved for future use;

(iv) flags field 1105, which includes flags for a card (see Fig. 13);

(v) distinguishing identifier field 1110, which includes two fields – a service 1106 and a service specific field 1107. The service field 1106 identifies the service of a corresponding card and the service specific field 1107 optionally contains a service-specific value;

(vi) a number of objects field 1108, which includes a number value representing how many objects follow the header. This field can be set to zero; and

(vii) a checksum field 1109, which includes a card checksum of all data on the card excluding the checksum itself.

Fig. 12 provides a description of the content of the various (number) fields described with reference to Fig. 11. In particular, the distinguishing ID number field 1110 comprises an eight byte distinguishing identifier. The distinguishing identifier includes two portions, unit pieces of data, namely, a service identifier and a service-specific identifier.

Preferably, the distinguishing identifier is arranged so that the service identifier occupies five bytes and the service-specific identifier occupies three bytes of the total distinguishing identifier value.

The service identifier contained in the field 1106 distinguishes one service from another or distinguishes one vendor from another. That is, for example, a service can be associated with an application that provides the service to a card user as distinct from a vendor who can provide multiple services to the card user by providing multiple applications.

The service identifier can be an identifier to identify the application to be used or application location (e.g. URL). Also, generic cards may be added to the System 600A or 600B and they are a special use of the Service identifier. The Generic cards are cards with a special Service identifier that can be used to provide input to a current application already running. The special value for the service 0x0000000001 is known as "the generic service identifier " and is used on "generic cards". A generic card can be used to send data to the front application already running. These are used, for example, for keypads that can be used to send text input to any application or a card with personal details that also may be used to submit this information to any application.

The service – specific identifier contained in the field 1107 can be optionally used by the vendor of a particular service to provide predetermined functions associated with that particular service. The use of the service-specific identifier is substantially dependent upon the application 304 run on the system 600. For example, the service identifier together with the service-specific identifier can be used as a unique identifier for a card 10. This unique identifier can be used to gain or deny access to a specific feature associated with a particular service, to reproduce a specific-service identifier in a log file in order to confirm or verify that a particular card 10 having that value was used to access

a service, and to provide a unique identifier that can be matched up with a corresponding value in a database in order to retrieve information about the user of the service (eg: name, address, credit card number etc).

Another example of a use for the service-specific identifier can include providing information about a mechanism or mode of distribution of the cards 10 (e.g. by mail, bus terminal kiosks, handed out on a train etc). Further, the service-specific identifier, can identify what data should be loaded into the system 600 when a service is accessed.

The foregoing is not intended to be an exhaustive list of possible uses or applications of the service-specific identifier but a small sample of possible applications and there are many other applications of the service-specific identifier of field 1107.

4.1.1 Card Flags

The flags field 1105 of the header 1100 of Fig. 11 may include three flags as follows:

(i) Don't beep;

(ii) No move events; and

(iii) No event co-ordinates.

Fig. 13 shows a description of each of the above flags. The above flags affect the functions that a smart card 10 can perform in a remote reader 1, as is defined by the description of each flag. An example, of a user interface element as referred to in Fig. 13 is a "button" on the card 10. user interface elements will be explained in further detail later in this document.

4.2 Objects

As shown in Fig. 57, immediately following the card header 1100 of Fig. 11 can be zero or more object structures 5713 defining the objects of a particular card 10 and

forming part of the data stored on the card 10. Each object structure 5713 comprises four fields as follows:

(i) a type field 5701;

(ii) an object flags field 5703;

(iii) a length field 5705; and

(iv) a data field 5707.

The structure of the data field 5707 depends on the object type as will be described below.

Fig. 14 shows a description of each of the fields 5701, 5703, 5705 and 5707 of the object structure 5713. The flags field 5703 of the object structure 5713, preferably includes an inactive flag. Fig. 15 shows a description of the inactive flag.

There are preferably five object types provided for the cards 10A, 10B, 10C and 10D described above, as follows:

(i) user Interface objects (i.e. data defining a button on the card 10);

(ii) Card Data;

(iii) Fixed Length Data;

(iv) Reader Insert;

(v) No operation; and

(vi) No operation (single byte).

Fig. 16 shows a description of each of the above object types (i) to (vi).

4.2.1 user Interface Object

Each user interface object defines a rectangular area on the card 10 and some quantity of associated data that is transmitted when the user touches an area of the panel 8 over the corresponding rectangular area of the card 10. The origin for the co-ordinate mapping system is the top left of the smart card 10 as if it was an ISO standard memory

smart card held in a portrait view with the chip contacts 18 facing away from the viewer and towards the bottom of the card 10. For any reader 1 that does not use this card orientation, the values of the corner points must be adjusted by the reader 1 so as to report a correct "button" press.

The user interface (element) object structure preferably has six fields as follows:

(i) a flags field;

(ii) an X1 field;

(iii) an Y1 field;

(iv) an X2 field;

(v) a Y2 field; and

(vi) a data field which typically includes data associated with the user interface element for example, a URL, a command, a character or name.

Fig. 17 shows a description of each of the above fields for the described user interface object structure. A press on the pressure sensitive touch panel 8 is defined to be inside a particular user interface object if:

(i) the X value of the press location is greater than or equal to the X1 value of the associated user interface object and is strictly less than the X2 value for that particular user interface object; and

(ii) the press Y value for the press location is greater than or equal to the Y1 value of the particular user interface element and strictly less than the Y2 value.

Overlapping user interface elements is allowed. If a press is within the bounds of more than one user interface element then the object sent is determined by a Z order. The order of the user interface elements on the card defines the Z ordering for all of the user interface elements on that particular card. The top user interface element is the first user

interface element for a particular card 10. The bottom user interface element is the last user interface element for that particular card 10. This allows for non-rectangular areas to be defined. For example, to define an "L" shaped user interface element, a first user interface object would be defined with zero bytes in the data field, and a second user interface object would be defined to the left and below the first user interface object but overlapping the user interface object.

The location of a press is to be reported in "fingels", which represent finger elements (analogous to "pixels" which represent picture elements). The height of a fingel is defined to be 1/256th of the length of an ISO memory smart card and the width is defined to be 1/128th of the width of an ISO memory smart card. The behaviour associated with each element may be modified with one or more flags. Each user interface element preferably has four associated flags as follows:

(i) Invert Beep Enable;

(ii) Auto repeats;

(iii) Do Not Send Data on Press; and

(iv) Do Not Send Data on Release.

Fig. 18 shows a description for each of the user interface element flags.

4.2.2 Card Data

The Card Data object is used to store data which is specific to a particular card. The data layout for this object has no fixed form. The contents of the Card Data object are sent from the reader 1 as part of the INSERT message when the card 10 is inserted into the reader 1.

4.2.3 Fixed Length Data

The fixed length data object is used to define a fixed length block on the card that can be written to by the computer 100, for example.

4.2.4 Reader Insert

The reader insert object is used to store instructions for the remote reader 1 when a particular card is inserted. This can be used, for example, to instruct the reader 1 to use a specific configuration of IR commands to allow communication with a specific set top box or TV.

4.2.5 No Operation

The No Operation object is used to fill in unused sections between other objects on a particular card. Any data stored in the no operation object is ignored by the remote reader 1. Any unused space at the end of the card 10 does not need to be filled in with a no operation object.

4.2.6 No Operation (One Byte)

The No Operation (One Byte) object is used to fill gaps between objects that are too small for a full object structure. These objects are only one byte long in total.

5.0 READER PROTOCOL

The remote reader 1 uses a datagram protocol that supports both uni-directional and bi-directional communication between the remote reader 1 and the set top box 601 or computer 100, for example. The format used for messages from the remote reader 1 as a result of user interactions with the remote reader 1 are of a different format than those that are sent to the remote reader 1.

5.1 Message Types

There are at least seven message event types that can be sent by the remote reader 1. These events are as follows:

- INSERT: When a card 10 is inserted into the remote reader 1, and the card 10 is validated, an INSERT event is generated by the remote reader 1 and an associated message is transmitted. This message announces the card 10 to a receiver (e.g. the set

top box 601). The INSERT message preferably includes the particular distinguishing identifier and allows applications to be started or fetched immediately upon card 10 insertion rather than waiting until the first interaction takes place. The INSERT message preferably includes the contents of the card data object from the card 10 inserted into the reader 1 if an object of this type is present on the card 10.

- REMOVE: When a card 10 is removed from the remote reader 1, a corresponding REMOVE event is generated and a REMOVE message is sent to the particular receiver associated with the remote reader 1. Like the INSERT message, the associated distinguishing identifier is transmitted along with the message. As the distinguishing identifier cannot be read from the now removed card 10, the distinguishing identifier is stored in the memory 47 of the remote reader 1. This is a useful optimisation as the distinguishing identifier is required for all other messages and reading the distinguishing identifier from the card 10 each time the distinguishing identifier is required can be too slow. INSERT and REMOVE messages are not relied upon by the system 600 to control processing. The system 600 is configured to infer missing messages if a message is received and is not immediately expected. For example, if an application detects two INSERT messages in a row, then an application can assume that it has missed the REMOVE message associated with the card of the first INSERT message as it is not possible to have two cards inserted at one time in present arrangement. The application can then take whatever action is required prior to processing the second INSERT message.

- Another example of where a missing message can occur is where a hand-held, infrared connected reader 1, as compared with a wired reader, is being used. Often a user does not point the reader 1 directly at a receiver when inserting or removing cards. This problem can be corrected by the system 600 inferring the INSERT or

REMOVE operations based on differing distinguishing identifiers in consecutive PRESS and RELEASE pairs.

- BAD CARD: If an invalid card is inserted, then the remote reader 1 is preferably configured to generate a BAD CARD event and to send a BAD CARD message. This message allows an associated receiver to take some action to alert the user to the invalid card.
- PRESS: When a touch is detected by the remote reader 1, a PRESS event is generated and a PRESS message is sent to an associated receiver. The PRESS message contains details of the associated card, the position of the press and the data associated with the user-interface element at that particular position. If there is no user interface element defined for that position (including if there is no user interface elements defined on the card 10 at all) a PRESS message is sent containing details of the associated card and the position of the press. If there is no card present in the remote reader 1 when a PRESS event is generated then a PRESS message is sent containing the special "NO_CARD" identifier (i.e. eight bytes of zero - 0x00) and the position of the press.
- RELEASE: A RELEASE event complements the PRESS event and a RELEASE message can be sent in order to inform the application program of the system 600 that a PRESS has been lifted. Every PRESS event preferably has a corresponding RELEASE event. Readers can allow multiple presses to be registered or provide other events that may occur between PRESS and RELEASE messages.
- MOVE: If, after processing a PRESS event, the touch position changes by a certain amount then the finger (or whatever is being used to touch the card) is assumed to be moving. MOVE EVENTS are generated and MOVE messages are sent until the touch is lifted. MOVE events auto-repeat by re-sending the last MOVE messages when the touch position remains stationary. The repeated sending finishes when the

touch is lifted and a corresponding RELEASE message is sent. Unlike PRESS and RELEASE events there is no user-interface object involved with MOVE events.

- LOW BATT: A LOW BATT event is generated and a LOW BATT message is sent when the battery 53 in the remote reader 1 is getting low. This message is sent after user interactions to increase the chance that the message will be received by the rest of the system 600. The sending of the LOW BATT message does not prevent the remote reader 1 from entering a low power state.

5.2 Data Formats

The preferred data format of the reader protocol used in the system 600 is a fixed size header followed by a variable length data field which can be zero bytes or more in length, followed by an eight bit check-sum and complement.

5.2.1 Message Header

The message header is preferably of a fixed length and is prepended (i.e. appended to, but in front of) to all messages sent from the remote reader 1. It is necessary to keep the message header as small as possible due to any bandwidth restrictions that may be imposed. Fig. 19 shows the format of the message header that is sent from a remote reader 1.

Service and service-specific identifiers can be assigned, by a smart card identification authority, to a vendor when the vendor registers a particular service. The service and service-specific identifier are the same for every message from a given card. A service specific identifier is preferably set by a vendor for use with their application. The Reader identifier is also in the header of each message. This identifier can be used by an application 304 to distinguish different users, for example, in a multi-player game.

Fig. 20 shows a table listing the message event types that have been described above.

5.2.2 Simple Messages

A number of message types are considered simple in that they consist solely of the message header described above followed by the message checksum byte and its complement. For example, a BADCARD message, a LOW_BATT message and a REMOVE message are simple messages.

Fig. 21 shows the format of a simple message.

5.2.3 MOVE Messages

MOVE messages are formed of the message header described above followed by two fields defining the co-ordinates of the touch position on the touch panel 8 of the remote reader 1. Fig. 22 shows the format of a MOVE message.

5.2.4 PRESS and RELEASE Messages

Fig. 23 shows the format of PRESS and RELEASE messages. PRESS and RELEASE messages, like MOVE messages contain the message header and touch co-ordinates. In addition, PRESS and RELEASE messages send data associated with the user-interface element if the touch position matches a user-interface element defined on the card. This data is of variable length, the actual size being defined by a corresponding card 10. If the touched position does not match a user-interface element defined on the card (including if no user-interface elements are defined on the card), zero bytes of data associated with user interface elements are sent. If there is no card 10 in the reader 1 then the service identifiers are all set to zero (ie 0x00) and zero bytes of data associated with the user-interface elements are sent. The data associated with the user interface element normally corresponds to the data associated with the user interface element defined on the card but may be modified or generated by processing on the card 10 or reader 1.

Fig. 24 is a data flow diagram showing the flow of the above-described messages within the system 600. As seen in Fig. 24, the card header 1100 and object structure 5713

are read by the CPU 45 of the remote reader 1 which sends a corresponding INSERT, REMOVE, PRESS, RELEASE, MOVE, BADCARD or LOW BAT message to the event manager 301 via the I/O daemon 300. As will be described in more detail below, the event manager 301 has twenty-one core messages, which are sent to and received from the ML 302, launcher 303 and applications 304.

5.2.5 INSERT Messages

INSERT messages are formed of the message header described above and the contents of the card data object from the inserted card 10. Fig. 21A shows the format of an INSERT message.

6.0 READER FIRMWARE

6.1 Overview

The microcontroller 44 has non-volatile memory 46 embedded within which can be programmed with the firmware to be described in detail below. The firmware working in concert with the microcontroller 44 and peripheral hardware (e.g. the computer 100) can thus dictate the functional requirements of the remote reader 1.

6.2 Code Type

In an attempt to minimise the cost of the remote reader 1 to a user, memory on the remote reader 1 is preferably minimised. As a result the application program written for the remote reader 1(i.e. the firmware) must be as compact and fast as is possible.

6.3 Resource Constraints

The microcontroller 44 has the following characteristics:

6.3.1 Non-volatile memory

The flash memory 46 is configured with 4096 bytes of FLASH ROM and can be utilised for firmware storage. The FLASH ROM is re-programmable but in the case of mass production a MASK ROM part can be utilised.

6.3.2 Random Access Memory (RAM)

The RAM 47 is configured as 128 bytes of RAM for use by the firmware.

6.4 Interrupts

The remote reader 1 uses two of the numerous interrupt sources supported by the microcontroller 44. These interrupts can be described as follows:

6.4.1 Received Data Interrupt

An infrared (IR) serial data receiver generally generates a falling edge when incoming data is received. This data has to be sampled and buffered as quickly as possible. One port of the microcontroller 44 doubles as an input timing capture pin which can initiate an interrupt on the falling edge.

6.4.2 Timer Overflow Interrupt

The microcontroller 44 has a free-running 16-bit timer which can be programmed to generate an interrupt when it overflows. In conjunction with the 4.91MHz clock source and pre-scale factor of 64, this equates to an interrupt every 3.41 seconds. An interrupt service routine increments a counter which triggers the suspension to low power mode preferably after about one minute of inactivity.

6.5 Resets

The microcontroller 44 supports five reset sources and the remote reader 1 is preferably configured to use all of reset sources. These reset sources can be described as follows:

6.5.1 Power On Reset (POR)

The POR reset is initiated when a new battery is fitted to the remote reader 1. The microcontroller 44 includes a circuit that detects the power on condition and generates a reset.

6.5.2 Low Voltage Inhibit (LVI) Reset

The LVI reset is initiated when a circuit (not shown) within the microcontroller 44 detects that the supply voltage has fallen below 2.4 Volts. When this kind of reset occurs a flag is set in a Reset Status Register (RSR) and an initialisation routine can deduce that the battery 53 is becoming depleted. For example, when infrared data is being transmitted, the infrared LED consumes high current as it is being pulsed. If the battery 53 is depleted, the supply voltage can dip under the 2.4 Volt threshold during transmission causing an LVI reset. After reset the battery 53 voltage recovers and the LVI reset does not occur until the next high current drain. This gives the remote reader 1 a chance to flag the failing of the battery 53 to an associated set-top box or remote equipment so that the user can be prompted to replace the battery 53.

6.5.3 Computer Operating Properly (COP) Reset

The COP reset is configured to reset the microcontroller 44 if the microcontroller 44 gets stuck doing a particular operation for an inordinate amount of time. The COP circuit takes the form of a counter that generates a reset if the counter is allowed to overflow. The COP register must be written at predetermined time intervals to avoid a COP reset.

6.5.4 Illegal Address / Opcode Reset

An Illegal Address/Opcode Reset is generated by the microcontroller 44 if it encounters either an address out of a predetermined range or an opcode that does not conform to predefined conditions. This reset cannot be turned off but should only be in evidence during code debugging.

6.5.5 Hardware Reset

A hardware reset is generated by driving a 'Reset' pin on the microcontroller 44 low during normal operation. Additionally, if the microcontroller 44 is in low power mode, a falling edge on the Interrupt Request (IRQ) pin also generates a hardware reset. This reset is the mechanism used to wake the microcontroller 44 out of low power mode in the firmware. The IRQ pin is preferable for this function since it can be configured to be edge sensitive only, not level sensitive as the reset pin is.

6.6 Memory Card / CPU Card Interface

The firmware preferably supports only memory card peripherals using an Integrated Circuit Protocol (e.g. the I^2C protocol) . Alternatively, the firmware can support CPU card formats.

6.7 Power Consumption

The firmware plays a critical role in conserving the life of the battery 53. All operations performed by the microcontroller 44 are optimised so as to be performed as quickly as possible while wasting as little power as possible. As soon as the remote reader 1 has been inactive for a time (e.g. 1 minute) the microcontroller 44 suspends to low power mode to conserve battery life still further. Low power mode consumes about 1000 times less current than normal operating mode so efficient suspension to this mode is very desirable. The firmware controls the state of the microcontroller 44 ports during low power mode.

6.8 Device Programming

The microcontroller 44 is able to be programmed using an In-System program (ISP) function supported by an embedded monitor within the microcontroller 44. Monitor code is typically factory set by a manufacturer and can not be altered.

Programming of the microcontroller 44 for specific hardware can be performed using an In-Circuit Simulator (ICS) kit and a monitor-mode download cable. This cable uses the VCC, GND, RST, IRQ and PTB0 pins on the microcontroller 44. Source code to be programmed can be delivered, for example, from a Windows™ 95 development environment via a computer serial port to the ICS hardware and from there via the download cable to the microcontroller 44 pins. This programming method is ideal for firmware development and testing, but may be altered for mass production. A monitor-mode programming model is preferred in the microcontroller and an embedded programming jig for production can be used. Test points for programming signals can be provided to allow for production ISP. If the firmware is mask programmed into the microcontroller 44 then device programming will not be required.

6.9 Firmware Programming Sequence

The programming of the firmware will be described with reference to the reader 1 being operative coupled to a local computer 100.

6.9.1 The Main Loop

Fig. 25 is a flow diagram showing the read method 2500 performed by the remote reader 1 of the system 600 incorporating the software architecture 200. The method 2500 begins after a reset event, as described above, has been generated and the method 2500 is executed by the CPU 45. The method of Fig. 25 is configured in a "paced loop" manner. That is, the method 2500 is paced by a routine, which generates a 10ms delay. This delay gives adequate service to the necessary routines while providing good latency for the handling of interrupts.

At the first step 2600, an initialisation routine is performed by the CPU 45. The initialisation routine is performed in order to initialise configuration registers and will be explained below with reference to the flow diagram of Fig. 26. The method 2500

continues at the next step 2501, where the computer operating properly (COP) register is cleared indicating that the firmware is not stuck in any recurring loops. At the next step 2700 a check card process is performed by the CPU 45, in order to check for any changes in the presence and validity of a particular smart card 10. The check card process will be explained in more detail below with reference to the flow diagram of Fig. 27. The method 2500 continues at the next step 2800, where a scan touch panel process is performed by the CPU 45 to check for any touches on the touch panel 8 by the user. At the next step 2900, a wait 10 ms routine is performed by the CPU 45, and the method 2500 then returns to step 2501.

6.9.1 The Initialisation Process

After a reset from any one of the five sources described above all configuration registers require correct initialisation. If an LVI reset was received then a "possibly depleted battery" flag is set. Fig. 26 is a flow diagram showing a method 2600 of initialising the system 600 incorporating the software architecture 200. The method 2600 is executed by the CPU 45 and begins at step 2601 where all registers are initialised to a predetermined default state. At the next step 2602, a check is performed by the CPU 45 to determine if the reset was an LVI reset. If the reset was not an LVI reset at step 2602, then the method 2600 concludes. Otherwise the method 2600 proceeds to step 2603 where the possibly depleted battery flag is set and then the method 2600 concludes.

6.9.2 The Check Card Process

Fig. 27 is a flow diagram showing a method 2700 of checking the card 10 of the system 600 incorporating the software architecture 200. As described above, the method 2700 checks for changes in the presence and validity of a smart card 10 in the remote reader 1 and responds accordingly. The method 2700 is performed by the CPU 45 and begins at step 701 where if a smart card 10 is inserted in the remote reader 1, then the

method 2700 proceeds to step 702. At step 702, if the card 10 is a new card (i.e. in the previous state there was no card in the reader 1), then the method 2700 proceeds to step 703. Otherwise, the method 2700 concludes. At the next step 703, the "magic number" and "checksum" fields are read from the card header stored in the memory 19 of the card 10, and are checked for correctness. If the "magic number" and "checksum" are correct, then the method 2700 proceeds to step 704. The method 2700 continues at step 704, where the distinguishing identifier is read from the card header and the "No MOVE events" and "No Event Co-ordinates" flags are set. The Card Data, if present, is also read from the card at this step 704. At the next step 705, an INSERT message, including the Card Data if present, is sent to computer 100, and the INSERT message is processed by the CPU 205. Then at step 706, a "BEEP" is sounded and the method 2700 concludes.

If the "magic number" and "checksum" fields are not correct (ie: the card 10 is not valid) at step 703, then the method 2700 proceeds to step 710 where the don't beep, no move events and event co-ordinate flags are set. At the next step 711, a BAD CARD message is sent to the computer 100, and the BAD CARD message is processed by the CPU 205. Then at step 712, a "BOOP" is sounded and the method 2700 concludes.

If a smart card 10 is not inserted in the remote reader 1 at step 701, then the method 2700 proceeds to step 707. At step 707, if this is the first operation of the reader 1 after the reset then the method 2700 concludes. Otherwise, the method 2700 proceeds to step 708 where the "Don't beep", "No MOVE Events" and "No Event Co-ordinates" flags are set and the distinguishing identifier stored in memory 47 is set to "NO_CARD". At the next step 709, a REMOVE message is sent to the computer 100, and the REMOVE message is processed by the CPU 205. The method 2700 concludes after step 709.

6.9.3 The Scan Touch Panel Routine

Fig. 28 is a flow diagram showing a method 2800 of scanning the touch panel 8 of the reader 1 of the system 600 incorporating the software architecture 200. As described above, the scan touch panel routine checks for touch panel touches that equate with card button presses and responds accordingly. The method 2800 is executed by the CPU 45 and begins at step 801 where if the panel 8 is being touched, then the method 2800 proceeds to step 802. Otherwise, the method 2800 proceeds to step 812, where if the panel 8 has been touched previously then the method 2800 proceeds to step 813. Otherwise, the method 2800 concludes.

At step 813, the "don't beep", "no move events" and "event co-ordinate" flags are set. Then at step 814, the message type is set to RELEASE and the method 2800 proceeds to step 805.

The method 2800 continues at step 802, where if this is the first time that the touch has been noticed since there was no touch, then the method 2800 proceeds to step 803. At the next step 803, the CPU45 determines if a bad card has been inserted into the reader 1 by checking the result of step 703, then in the case that a bad card has been inserted into the reader 1, the method 2800 proceeds to step 815. Then at step 815, a BAD Card message is sent to the computer 100, the BAD CARD message is stored in memory 206, and the method 2800 concludes. If it was determined at step 803 that the card 10 was valid, by checking the result of step 703, or that no card was inserted into the reader 1, by the checking of step 701, then the method 2800 proceeds to step 804, where the type of message is set to PRESS in the message header of Fig. 19. At the next step 805, the CPU45 determines the touch coordinates (i.e. X, Y coordinates of user press location) via the touch panel interface 41. Then at the next step 807, the offset and scale functions are applied to the coordinates. The offset and scale functions map the coordinate space of the touch panel 8 to the coordinate space of the card 10. The method

2800 continues at the next step 807, where if the CPU45 determines that the sent message was a MOVE and/or no card was inserted into the reader 1, by checking step 701, then the method 2800 proceeds directly to step 809. Otherwise, the method 2800 proceeds to step 808 and the memory 19 of the card 10 is searched in order to find the first user interface element whose X1, Y1, X2, Y2 values form a range within which the touch coordinates fall and data associated with matched user interface element is read from the card 10. At the step 809, the message is sent along with any data to the associated computer 100, and the CPU 205 in the computer 100 processes the message. The method 2800 continues at the next step 811, where a BEEP sound is sounded and the method 2800 concludes.

If this is not the first time that a touch has been noticed since there was no touch, at step 802, then the method 2800 proceeds to step 816. At step 816, if the touch detected at step 801 was a move, then the method 2800 proceeds to step 817. Otherwise the method 2800 concludes. At step 817, the message type is set to MOVE and the method 2800 proceeds to step 805. For example, a MOVE message can be sent along with the X, Y coordinates of a touch position as defined by Figs. 19 and 22, a PRESS and RELEASE message can be sent along with X, Y coordinates of a touch position and data associated with a user interface object (i.e. one of Indicia 14) as defined by Figs. 19 and 23. If it was determined at step 807 that the message was a MOVE, at step 809, then the CPU 45 sends a MOVE message to the computer 100. The CPU205 processes X, Y coordinates as cursor information and moves a cursor that is displayed on the Video Display 101. In this case, the next RELEASE message can be interpreted as a command to select the displayed object at the cursor position (eg to execute a program, select an item or load a URL). Further, if NO Event Coordinates (see Fig. 13) have been set in the card 10, then the reader 1 may send the data associated with a user interface object to the event

manager 301in the computer 100 or STB 601 without sending the X, Y coordinates of the touch position.

In addition, if the application 304 has a user interface Object structure such as that shown in Fig. 17, and a matching function such as at step 808, then the reader 1 may send X, Y coordinates of a touch position to the application 304. As a result, the CPU 205 executes the same matching function to read data associated with the user interface object from the event manager 301 and provides the card user, a service (e.g. game) identified by a service identifier 1106 associated with the read data. For example, at step 4205 of Fig. 41, the CPU205 determines if data is in the data field of a message. If data is in the data field, then CPU205 reads the data and processes the data at the next steps in Fig. 41. If data is not in the data field, then the CPU205 reads the X, Y coordinates from the message and executes the matching function for the coordinates to get data associated with user pressed indicia. Alternatively, the event manager 301, using the user interface object structure available to the event manager 301, can perform this function.

Therefore, if a card user uses the reader 1 (without inserting a card 10) as a mouse by moving his or her finger on the touch pane l 8, the user can select one of the STB services on a STB menu displayed on the TV display. Also, if the card user uses the reader 1 with an inserted card 10 and selects some indicia 14, the user receives a service (e.g. game) from the computer 100 or STB 601. In particular, if the user selects a START indicia, a desired game can be executed in the computer 100 or STB 601 and an object in the game kicks a ball according to the selection of a KICK indicia 14.

By defining per-card flag values in advance for the card 10, various types of cards 10 can be provided to a user. For example, if a flag (i.e. information) of "NO Move Events" has been set in a card 10 in advance, the reader 1 can be configured to not perform as a mouse based on the flag. On the other hand, if a flag of "NO Move Events"

has not been set in the card 10 in advance, then the reader 1 can be configured to perform as a mouse based on the flag.

As shown in Fig.13, the reader 1 has a default condition in which the reader 1 provides audio feedback, acts as a mouse and sends coordinates for press, release and more events. Alternatively, the reader 1 can provide a default condition in which the reader 1 does not provide audio feedback, act as a mouse and send coordinates.

If the reader 1 is configured to perform the 'beep function' using the per-card flag values, the reader 1 sounds a "beep" and executes a method in accordance with the flow diagrams shown in Figs. 27 and 28. Further, if the reader 1 is configured to perform the 'mouse function' using the per-card flag values, then the reader 1 acts as a mouse and executes a method in accordance with the flow diagrams of Figs. 27 and 28. Still further, if the reader 1 is configured to perform the 'matching function' using the per-card flag values, then the reader 1 sends coordinates for press, release and move events and executes a method in accordance with the flow diagrams of Figs. 27 and 28.

The matching function is also executed in the EM301 as at step 808 of Fig. 28. The card 10 can also be configured as a card having only the mouse function and/or a basic function (e.g. sending the EM301 data associated with indicia selected by a user). Therefore, by combining each per-card flag value randomly, various types of cards 10 can be provided to a user.

As described herein , the service identifier 1106 is an indispensable identifier for the system 600. By sending at least a service identifier 1106 in the distinguishing identifier 1110, to the EM 301, a service can be provided to a user.

The service specific identifier 1107 described above is preferably set by a vendor for use with a particular application. Therefore, if the vendor defines a unique service specific identifier 1107 for each card 10, then the card 10 would be unique. If the service

specific identifier 1107 is being used to provide information about a means by which particular cards have been distributed (e.g. by mail, handed out on a train), then the service specific identifier 1107 can be added to a file which gives a record of which cards have been used to access the service for later use in determining how effective different distribution means have been used.

6.9.4 The Wait 10ms Process

Fig. 29 is a flow diagram showing a wait 10ms routine 2900. The wait 10 ms routine 2900 loops so as to consume CPU cycles until 10ms has elapsed. The delay process 2900 is executed by the CPU 45 and begins at step 901 where a predefined process counter is cleared. At the next step 902, the counter is incremented. Then at the step 903, if 10ms has not elapsed, then the method 2900 returns to step 902. Otherwise the delay process 2900 concludes.

7.0 EVENT MANAGER

The event manager 301 is one of the process components of the software architecture 200. The event manager 301 enforces the rules of the architecture 200 and ensures consistent behaviour between the other process components.

7.1 Role in the System

Most communications pass through the event manager 301 and the event manager 301 is the only component of the architecture 200 that all process components except the directory service 311 components need to be able to directly communicate with. The event manager 301 acts as the enforcer of the rules of the architecture 200, and the event manager 301 does not necessarily have to be configured as one distinct program. The event manager 301 can also be formed of trusted relays or other separate process components that perform part of the event manager role. This can be done for efficiency or security reasons for example.

The event manager 301 may incorporate various other parts of the software architecture 200 such as the I/O daemon 300 and the launcher 303. The event manager 310 may even incorporate an application such as a browser controller.

The event manager 301 can communicate with every process component of the system 600 except the directory service 311 either directly or through a trusted relay. These components include the I/O daemon 300, launcher 303 and any of the applications 304. The event manager 301 can use any suitable communications method to communicate with the other process components. The preferred communication method is Transmission Control Protocol / Internet Protocol (TCP/IP) due to it's nearly universal implementation but other OS specific methods, such as Unix™ sockets, etc can also be used. When the process components are integrated together the method used to communicate can be internal data passing between separate threads.

The event manager 301 is preferably configured to be immune to interference from other process components which includes other processes being able to kill the event manager 301 or being able to starve the event manager 301 of CPU time or network bandwidth. This ensures that the event manager 301 can remain in ultimate control of the system 600.

7.2 Internal Requirements

The event manager 301 performs non-blocking I/O to all the other process components 300, 303, 304 and 306 of the architecture 200 by methods such as polling (NB: polling is not recommended due to the CPU load), interrupt driven I/O, having a separate thread reading and writing from each component or any other suitable method that achieves the same goal. This ensures that one component is not starved out by another component and also reduces user wait time.

The event manager 301 is also configured to check all incoming data for validity and to repair the data if possible before output. This includes data from trusted components. The event manager 301 is preferably also fail safe. If the event manager 301 receives unexpected data from one of the components 300, 303, 304, or 306, then the event manager 301 is configured to deal with the data and not exit unless it is absolutely unavoidable.

The event manager 301 can be required to be running for a considerable length of time and it is configured so as to ensure that performance does not degrade over time. The event manager 301 is preferably configured to assume that the transmission mechanism is reliable for communication with any component that is using a predetermined event manager protocol (i.e. EM-protocol) but assumes that the transmission mechanism used to communicate with the remote reader 1, via the I/O daemon 300, is unreliable and parts of the incoming data may be incorrect or missing.

7.3 Procedures

The event manager 301 is a direct participant in some of the operations of the system 600 but also transparently takes part in many of the other operations of the architecture 200. The event manager 301 is transparent in that it uses data packets as they pass through it without modifying them. The procedures will be explained in more detail below particularly with reference to section 8.0.

Fig. 30 is a flow diagram showing an overview process 3010 of events performed by the system 600 incorporating the software architecture 200. The process 3010, is executed by the CPU 205 depending on the configuration of the system 600. The process 3010 begins at step 3000 where a system initialisation routine is performed, with the initialisation routine including starting the event manager 301. At step 3000 the I/O daemon is typically also started with the event manager 301.

At the next step 3700 the event manager 301 starts the launcher 303. Then at the step 3300, the event manager 301 passes a message to the launcher 303, enabling the launcher 303 to determine which application 304 to execute, and the launcher 303 then starts the corresponding application 304. The process 3010 continues at the next step 3400, where once the currently running application 304 is no longer needed, for instance, when a new card 10 is inserted into the reader 1, the launcher 303 provides an exit message to the running application in order to end the execution of the running application. All applications are terminated when the system 600 is powered down (or switched off).

Fig. 31 is a flow diagram showing a method 3000 of receiving an event performed by the event manager 301. The method 3000 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3000 can be executed by the CPU 4305 in set top box implementations. The method 3000 begins at step 3101, where the launcher 303 is started. At the next step 3103, the event manager 301 receives an event. If the event received at step 3103 is not from the remote reader 1 at the next step 3105, then the method 3000 proceeds to step 3107 where the component identifier (XID) is checked and corrected if necessary. The method 3000 continues at the next step 3109, where if the new application sending an event is allowed to send the event, then the method 3000 proceeds to step 3111. At step 3111, the event is sent to a destination process component and the method 3000 returns to step 3103. If the sending application is not allowed to send the event at step 3109, then the method 3000 proceeds to step 3113, where the event is dropped and the method 3000 returns to step 3103.

If the event is from the remote reader 1 at step 3105, then the method 3000 proceeds to step 3115. If the event is a BADCARD, LOWBAT, INSERT or REMOVE event at step 3115 then the method 3000 proceeds to step 3117. Otherwise the method 3000 proceeds to step 3119. At step 3117, the event is passed to the launcher 303 and the

method 3000 returns to step 3103. If the distinguishing identifier is the NO_CARD identifier at step 3119, then the corresponding message is passed to the launcher 303 at step 3117. Otherwise the method 3000 proceeds to step 3121, where the service identifier portion of the distinguishing identifier is compared with the service identifier used in determining the current front application. If the service identifier is not the same as that which has been used to determine the front application and the service identifier portion of the distinguishing identifier is not the special generic service identifier, then the method 3000 proceeds to step 3117 where this message is passed to launcher 303 . Otherwise, the method 3000 proceeds to step 3123, where the event is sent to the front application and the method 3000 returns to step 3103.

7.4 Focus Change

The event manager 301 can safely ignore any EM_LOSING_FOCUS events that are not for the current front application. The event manager 301 needs to watch for EM_GAINING_FOCUS messages for which applications becoming the front application as well as the service identifiers that are associated with that application. The event manager 301 can safely ignore multiple EM_GAINING_FOCUS events that are to the same application with the same service identifier as well as any EM_LOSING_FOCUS events to applications that are not the currently front application. Messages that are ignored are passed on as normal.

7.5 Reader Messages

The event manager 301 is also responsible for distributing the messages to the correct component. The event manager 301 is configured to follow certain predetermined protocol rules, which will be described in detail below.

7.6 Restrictions on Sending Messages

A further role of the event manager 301 is to enforce predetermined restrictions on the transmitting of messages.

8.0 EVENT MANAGER PROTOCOL

The event manager protocol (EM-protocol) is the protocol used to communicate between all components of the architecture 200 except for the directory service 311. Generally all messages are configured to go through the event manager 301 before being passed onto an intended recipient. The EM-protocol is a datagram based protocol that is implemented on top of a reliable communications protocol, for example, Transport Control Protocol/Internet Protocol (TCP/IP). The event manager 301 is configured to assume that all data being sent will arrive unchanged and in the correct order. The event manager 301 does not assume that there is a reliable method of synchronisation between the process components of the architecture 200.

All multi-byte values are sent in Internet byte order (i.e. big-endian). The exception to this is the 'distinguishing identifier' values representing services, which are sent as blocks of several single bytes and are always treated as such (i.e. the distinguishing identifier values are never stored as a number typically because of the byte ordering issues).

8.1 Communication Methods

The event manager protocol is preferably configured to assume a TCP/IP like method of communication between the components of the architecture 200 and the system 600 hardware components. Alternatively, any known method of communication that ensures reliable transport can be used. For example, an operating system specific method such as 'Unix sockets' can be used. The data can be passed between the process components 301, 303, 304 and 306 directly via internal data structures in a multi-threaded application, for example.

In the case of architectures where an alternative method of communication between the components is being used, the problem of byte-ordering must be taken into account. If it is possible that applications can run on a machine that has different byte orderings or is required to communicate with components that expect the data in network byte order, which all components assume by default, then all affected communications can be done in network byte order.

8.2 Data Format

8.2.1 Basic data types

Some abbreviations that are used in the following paragraphs to refer to data types are as follows:

- int8: An eight bit signed value;
- uint8: An eight bit unsigned value;
- int16: A 16 bit signed value;
- uint16: A 16 bit unsigned value;
- int32: A 32 bit signed value;
- uint32: A 32 bit unsigned value; and
- xid_t: A 32 bit unsigned value.

8.2.2 Component Addressing

Every addressable process component in the architecture 200 is assigned a 32 bit unsigned value referred to as an 'xid' (or component identifier). This number is unique within the boundaries of each individual system 600 instance. Some xids of the process components are always the same. These are:

- Event Manager 301: EM_EVENT_MANGER_XID
- Master Launcher : EM_MASTER_LAUNCHER_XID
- Launcher 303: EM_FIRST_APP_XID

- Display Manager 306: EM_DISPLAY_MANAGER_XID

The xid value is divided up into a one byte type field and a three byte identifier. The different types are shown in Table 1 below.

Table 1

Value	Type
Internal xid's	These xid values are not routable and can be used internally by all components. They are dropped if seen by the EM
Core System xid's	These identify the core system components of a user interface Card system. These components include the EM, Launcher and Master Launcher.
Standard Application	These identify standard applications that are started and ended by the Launcher as needed.
Special application	These identify special applications that aren't controlled by the standard rules for starting and ending applications. They are applications that are written to provide the user interface card system with functionality that can be controlled by other applications such as a video on demand player or a browser controller.
Readers	Readers are assigned xids by the EM. These xids are unique to each reader that is used to access the system for the duration of the EM. If the event manager and therefore the system is restarted then the reader xids will change.

8.3 Message types

There are twenty-two core messages in the EM-protocol, which preferably have the following labels:

- EM_NEW_LAUNCHER
- EM_KILL_LAUNCHER
- EM_APP_REGISTER
- EM_EXIT_NOW
- EM_CLOSE
- EM_APP_STARTING
- EM_APP_DYING
- EM_GAINING_FOCUS
- EM_LOSING_FOCUS
- EM_LIST_MESSAGES
- EM_LIST_APPS
- EM_SEND_MESSAGE
- EM_POST_MESSAGE
- EM_GET_MESSAGE
- EM_DELETE_MESSAGE
- EM_READER_INSERT
- EM_READER_REMOVE
- EM_READER_BADCARD
- EM_READER_MOVE
- EM_READER_PRESS
- EM_READER_RELEASE
- EM_READER_LOW_BATT

These messages will be explained in more detail in the following paragraphs.

8.3.1 Message Header

The messages sent within the system 600 have a header portion preferably including the following information:

version: This represents the version number of the protocol being used by the component. This should always be set to EM_PROTOCOL_VERSION, which is defined in library headers to be the version used by the library.

type: This represents the type of message that a header proceeds and is set to one of the message types listed above and described below. The length of the messages is assigned the label dataLength .

reserved: This represents that the value in these two bytes is reserved and should be set to zero.

timestamp: This represents the timestamp of a data packet.

to_xid: This represents the destination xid of a particular packet. This is the final destination of the packet and should only be set to the event manager if that is the intended final recipient.

from_xid: This represents the source xid of the packet.

dataLength: This represents the length of the data that follows a header. This value can be zero. Different types of messages impose different requirements on the data following the message header. Components should not assume the length of a message from the type. The number of bytes in the dataLength field is always read even if this is different to the correct size of the message to insure that the stream can only be corrupted by an incorrect dataLength.

8.3.2 EM_NEW_LAUNCHER

The EM_NEW_LAUNCHER message is sent when the event manager 301 requires a new launcher 303. This message is only sent between the event manager 301 and the

Master Launcher if the software architecture 200 includes such a Master Launcher. The packet containing this message also contains information that a new launcher needs to connect to the event manager 301. The EM_NEW_LAUNCHER message preferably includes the following information:

port: This represents the port number that the event manager 301 is listening for new connection on.

host: This represents the host name of the machine running the event manager 301.

8.3.3 EM_KILL_LAUNCHER

The EM_KILL_LAUNCHER message is sent when the event manager 301 wants the Master Launcher to kill the current launcher 303. The EM_KILL_LAUNCHER message has no data associated with it.

8.3.4 EM_APP_REGISTER

The EM_APP_REGISTER message is sent when an application is starting up to the launcher 303 and informs the rest of the components of the architecture 200 that it is now ready to receive messages. Any messages that an application 304 sends before it has registered will be discarded by the event manager 301.

The EM_APP_REGISTER message preferably includes the following information:

xid: This represents the component identifier that was assigned to the application by the associated launcher 303. The remainder of the information sent cannot be represented by the structure as the remaining fields are of variable length. The data following the xid is a series of null terminated strings with a maximum length of 256 characters not including the terminating null, consisting of the lower and upper case characters a-z, the numbers 0-9 and the characters (.,-_). If the strings are longer than 256 characters they will be truncated at 256 characters.

Application Name: this represents a name that is used to identify the present application to other applications.

Service Group: this represents one or more names of service groups that the application wishes to be a part of.

An application that is persistent, such as a browser controller, only needs to register once. Such a persistent application does not need to register every time it gets an EM_GAINING_FOCUS event.

8.3.5 EM_EXIT_NOW

The EM_EXIT_NOW message is sent by the launcher 303 to an application when the application is about to be forced to exit. The EM_EXIT_NOW message has no data associated with it.

8.3.6 EM_CLOSE

The EM_CLOSE message is sent to persistent applications to indicate that the current session is closed and to return the application to its startup state. Once this message is received by an application, the application is required to treat the next EM_GAINING_FOCUS event as the start of a new session rather than as a change in input/output focus. The EM_CLOSE message has no associated data.

8.3.7 EM_APP_STARTING

The EM_APP_STARTING message is sent by the launcher 303 to the event manager 301 when an application is about to start. The EM_APP_STARTING message preferably includes the following information:

xid: This represents the component identifier of the application that is about to start.

8.3.8 EM_APP_DYING

The EM_APP_DYING message is sent by the launcher 303 to the event manager 301 when an application has exited. The EM_APP_DYING message is sent only after the launcher 303 is certain that the application has finished. The EM_APP_DYING message preferably includes the following information:

xid: This represents the component identifier of the application that has exited.

8.3.9 EM_GAINING_FOCUS

The EM_GAINING_FOCUS message is sent to an application by the launcher 303 when the application 304 is about to start receiving input from the remote reader 1. The EM_GAINING_FOCUS message preferably includes the following information:

id: This represents the distinguishing identifier of the remote reader 1 messages that will be sent to an application.

Data: This represents extra data that is to be sent to the application when it is about to receive focus. This is specific to each service and it is up to the application to interpret the data. The extra data is not checked for byte ordering issues and this should be dealt with by the application. Any multi-byte data is sent by applications in network byte order and assumed to be in this order by the receiving application. An example of this data, when the receiving application is a browser controller, is a URL which the browser controller is being instructed to load.

8.3.10 EM_LOSING_FOCUS

The EM_LOSING_FOCUS message is sent when an application 304 is about to lose input/output focus from the remote reader 1 and the display 101. The EM_LOSING_FOCUS message has no extra data.

8.3.11 EM_LIST_APPS

The EM_LIST_APPS message is sent when an application wishes to know what other applications are also running at a point in time. The EM_LIST_APPS message is returned to the application with the data field containing the application list. This message does not need to be addressed to any of the process components 301 to 306. The event manager 301 ensures that the EM_LIST_APPS message is sent to the correct component, which is usually the launcher 303, regardless of the to_xid field of the header. It is the role of the receiving component to decide which applications to list.

When used as a reply, the EM_LIST_APPS message has two formats. The first is the format used when the EM_LIST_APPS is sent as a request and the second is the format when it is sent as a reply. The request has no extra data associated with it.

The EM_LIST_APPS message preferably includes the following information:

app_xid: This represents the xid of the application being described.

app_desc: This represents the name string given to the launcher 303 when the application first registers.

8.3.12 EM_SEND_MESSAGE

The EM_SEND_MESSAGE message can be sent between any two concurrently running applications in the system 600. There is no structure imposed on this message by the architecture 200 but communicating applications need to agree on a common data structure.

8.3.13 EM_LIST_MESSAGES

The EM_LIST_MESSAGES message is used to get a list of all messages currently on a message board, which is used in the architecture 200. The message board will be described in more detail below with reference to section 8.4.7.1. The EM_LIST_MESSAGES message should be sent to the launcher 303. The

EM_LIST_MESSAGES message has a request and reply format. The request format has no data associated with it. The reply preferably includes the following information:

message_count: This represents the number of messages currently on the message board and can be equal to zero.

Messages: This represents a variable number (i.e. equal to message_count) of variable sized structures that have the following structure:

Each message preferably includes the following information:

message_id: This represents the message identifier of this message.

poster_id: This represent the xid (component identifier) of the component that posted this message.

mime_type: This represents the Multipurpose Internet Mail Extention-type (MIME-type) of the data associated with this message and is a null terminated string which can be of zero length in which case the terminating zero is still present.

message_desc: This represents the description of this message that was assigned when the message was posted by the posting application. This is a null terminated string that is at most 255 characters long not including the terminating zero. The length of this string can be zero in which case the terminating zero is still present.

8.3.14 EM_POST_MESSAGE

The EM_POST_MESSAGE message is used to post some data to the message board used in the architecture 200. These messages last until there is a service group change and can be accessed by any application that is running. The EM_POST_MESSAGE messages can also be deleted by any currently running

application and are not assumed to be totally reliable. Once the message has been posted it is returned to the application that posted it to inform the application of the message identifier of the message. These messages are sent to the launcher 303 by the application. The message from the application (i.e. the application that posted the message) includes the following information:

message_desc: This represents a description of the message and is a null terminated string that can be at most 255 characters long not including the terminating zero. The description can be zero bytes in length but must still have a terminating zero.

mime_type: This represents the MIME type of the message data that is being posted. The MIME type is not required but there must still be a terminating zero.

message_data: This represents the data to be posted to the message board.

The message returned to the application preferably includes the following information:

message_id: This represents the message identifier by which this message can be retrieved or deleted .

8.3.15 EM_GET_MESSAGE

The EM_GET_MESSAGE message is used to retrieve a message from the message board. It is sent containing the message identifier of the message that the component wishes to retrieve and it is returned to the component either containing the message or an error that there is no message with that identifier. These messages are sent to the launcher 303 by an application 304.

The information included when requesting the message is as follows:

message_id: This represents the message identifier of the message the application wishes to retrieve.

flags: This is a flags word. All unused bits should be set to zero. The flag description is shown in Table 2 below:

Table 2

Flag	Description	Value
EM_GM_DELETE	Delete the message from the message board after it has been sent	0x01

The reply has the following information:

error: If an error occurred then this will be set to one of the values in Table 3 below.

Table 3

Value	Description
EM_GM_NO_ERROR	No error occurred. The message is in the message field.
EM_GM_NO_SUCH_MESSAGE	No message exists with that message identifier on the message board.

message_id: This represents the message identifier of the message that was retrieved.

mime_type: This represents the MIME type of the message that was retrieved. This is a null terminated string. If this message has no MIME type associated with it then the string is zero length but the terminating zero is still present.

message: If no error occurred then this field will contain the data posted on the message board. The length is determined by the dataLength value in the header minus the size of the error field

8.3.16 EM_DELETE_MESSAGE

The EM_DELETE_MESSAGE message is used to delete messages from the message board. It is not an error to delete a message that does not exist. These messages are sent to the launcher 303 by the front application. The EM_DELETE_MESSAGE preferably includes the following information:

message_id: This represents the message identifier of the message that is to be deleted.

8.3.17 user Interface Card Reader Messages

The user interface card reader messages are generated by the remote reader 1 and are encapsulated by the event manager 301 so that they conform with the event manager protocol. There are three types of messages that are generated by the remote reader 1. These messages are "simple" messages, "move" messages and "press/release" messages. Move messages are simple messages with co-ordinates added, and press/release messages are simple messages with data and coordinates added.

8.3.17.1 Simple Messages

The following messages are simple messages:

- EM_READER_INSERT
- EM_READER_REMOVE
- EM_READER_BADCARD
- EM_READER_LOW_BATT

These simple messages preferably include the following information:

id: This represents the distinguishing identifier that was sent by the remote reader 1 and has no meaning for BADCARD messages.

8.3.17.2 Move Messages

The EM_READER_MOVE messages preferably include the following information:

id: This represents the distinguishing identifier that was sent by the remote reader 1, and is set to all zeros for no card messages.

X: This represents the x value.

Y: This represents the y value.

8.3.17.3 Press/Release Messages

EM_READER_PRESS and EM_READER_RELEASE messages preferably includes the following information:

id: This represents the distinguishing identifier that was sent by the remote reader 1.

x: This represents the x value.

y: This represents the y value.

data: This represents any data that was associated with the press or release (associated with the user interface-element data).

8.4 Procedures

The following paragraphs describe the main procedures that each process component of the architecture 200 follow.

8.4.1 Starting a new Application

Fig. 32 is a flow diagram showing detail of the method 3300 of starting a new application and performed whenever the launcher 303 starts a new application. The method 3300 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3300 can be executed by the CPU 4305 in set top box implementations. The method 3300 begins at the first step 3301 where the launcher 303 performs a mapping to translate the service identifier into a URL. At the next step 3303, the launcher 303 fetches and starts the application informing it of an event manager host-name and port number. The method 3300 continues at the next step 3305, where the launcher 303 sends the event manager 301 an EM_APP_STARTING message informing

the event manager 301 of the xid of the starting application. At the next step 3307, the new application connects to the event manager 301 and sends the launcher 303 an EM_APP_REGISTER message. Further, there is normally a focus change to the new application.

8.4.2 Ending an Application

Fig. 33 is a flow diagram showing a method 3400 of ending an application in the system 600 incorporating the software architecture 200. The method 3400 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3400 can be executed by the CPU 4305 in set top box implementations. This method is used whenever the launcher 303 terminates a running application. The method 3400 begins at step 3401, where the launcher 303 sends the running application an EM_EXIT_NOW message. The launcher 303 sets a time out at this point to give the application a chance to exit cleanly. At the next step 3403, the running application cleans up and exits. Alternatively, the application ignores the EM_EXIT_NOW message and the launcher 303 times out and forces the application to quit. Then at step 3405, the launcher 303 sends the event manager 301 an EM_APP_DYING to tell it that the application has exited and that the launcher 303 should discard any waiting data and close the connection to the application if the connection is still open, and the method 3400 concludes.

8.4.3 Closing a persistent application's session

Fig. 34 is a flow diagram showing a method 3500 of closing the current session of a persistent application on the system 600 incorporating the software architecture 200. The method 3500 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3500 can be executed by the CPU 4305 in set top box implementations. The method 3500 is analogous to the application ending but the application does not actually close. The method 3500 begins at step 3501, where the

launcher 303 sends the persistent application an EM_CLOSE message. At the next step 3503, the persistent application resets to its initial state, and the method 3500 concludes. This may involve closing connections to outside servers, loading a default web page etc. The next EM_GAINING_FOCUS event that the persistent application receives is assumed to be the start of a new session.

8.4.4 Focus Change

Fig. 35 is a flow diagram showing a method 3600 of performing a focus change on the system 600 incorporating the software architecture 200. The method 3600 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3600 can be executed by the CPU 4305 in set top box implementations. The method 3600 is used to tell an application that it is about to gain or lose input/output focus, which is not a signal for the application to exit. At the first step 3601, the launcher 303 makes the decision to change the application that currently has input/output focus and sends the application that is to receive input focus an EM_GAINING_FOCUS event typically based on a card change. The sending of this event is used by the event manager 301 to decide which application should receive input/output focus based on predetermined conditions. Then at the step 3603, the launcher 303 sends the previous front application an EM_LOSING_FOCUS event, and the method 3600 concludes. This message is less critical and is not sent when the current front application remains the same, but still needs the EM_GAINING_FOCUS (i.e. in the case of a browser controller where the EM_GAINING_FOCUS events are used to tell the browser controller 402 the base URL).

8.4.5 Message Passing

There are two distinct types of message passing between applications supported by the architecture 200. Through the message board that is as persistent as the current

service group, and a direct message method where two components communicate with each other directly as described below.

8.4.5.1 Message Board

One component of the architecture 200, typically the launcher 303, maintains a message board and the event manager 301 knows which component does this. The message board is formed of a list of messages that are assigned a 32 bit unsigned number as an identifier by the process component managing the message board. The messages are formed of a text description, an optional MIME type for the message data and the message itself. An application can request a list of all messages currently on the message board by sending an EM_LIST_MESSAGES message. This will return with the text descriptions of all messages currently on the message board with their associated message identifiers. The application can then request a specific message by sending a EM_GET_MESSAGE with the message identifier of the message that it requires. It is possible that a message could be deleted between getting a listing of the message board and actually requesting a message. The error field of the EM_GET_MESSAGE message reply is configured to indicate this.

8.4.5.2 Direct Communication

Two applications can send each other arbitrary data directly, by using direct communication. This is performed by one application sending the other application the data by using an EM_SEND_MESSAGE message. The two applications need to agree on a data format for these messages and byte ordering issues need to be taken into account. To get the component identifier of the other application, an application can request to be sent a list of all running applications by sending a EM_LIST_APPS message. This message returns a list of all publicly visible applications that are currently running.

8.5 Reader Messages

This section outlines the rules used by the event manager 301 to route the EM_READER_* messages. The following messages are always sent to the launcher 303 regardless of which application currently has focus.

- EM_READER_INSERT
- EM_READER_REMOVE
- EM_READER_BADCARD
- EM_READER_LOW-BATT

The following messages are sent to the currently front application if the messages are from cards 10 that have the same service identifier in their corresponding fields1106 as the currently front application. A service-specific identifier is not taken into account in this comparison. If the service identifier is different to the currently front application or the distinguishing identifier is the NO_CARD present value (i.e. all zeroes) then the message is sent to the launcher 303 as previously described.

- EM_READER_PRESS
- EM_READER_RELEASE
- EM_READER_MOVE

8.6 Restrictions on Sending Messages

To improve the security and stability of the system 600, there are preferably restrictions placed on the sending of messages. Any messages that breach these rules will be discarded by the event manager 301.

8.6.1 Restrictions for all components

No component except the remote reader 1 will be allowed to send EM_READER_* messages.

8.6.2 Restrictions on the event manager

The event manager 301 is the enforcer of the rules and as such can send any messages necessary. The event manager 301 is configured to only need to generate EM_KILL_LAUNCHER and EM_NEW_LAUNCHER messages but it can copy messages and send the copies to process components that are not the target component. The event manager 301 also handles all transmissions between components.

8.6.3 Restrictions on the Launcher

The launcher 303 sends messages to all components 301 to 306 of the architecture 200. The messages that the launcher 303 can not send are as follows:

- EM_KILL_LAUNCHER
- EM_NEW_LAUNCHER

8.6.4 Restrictions on Applications

Applications only send the following messages to other applications (which includes the launcher 303):

- EM_APP_REGISTER
- EM_SEND_MESSAGE
- EM_LIST_APPS
- EM_POST_MESSAGE
- EM_GET_MESSAGE
- EM_DELETE_MESSAGE
- EM_LIST_MESSAGES

8.7 Component Procedure Lists

This section lists the functions, which each component of architecture 200 is involved in.

8.7.1 Event Manager

The event manager 301 is a direct participant in the following procedures:

- System Initialisation

- System Startup
- Starting a new Application
- Ending an Application
- Focus Change
- Message Passing
- Reader Messages

8.7.2 Launcher

The Launcher 303 is a participant in the following procedures:

- System Initialisation
- System Startup
- Starting a new Application
- Ending an Application
- Focus Change
- Message Passing (in some instances)
- Reader Messages (in some instances)

8.7.3 Applications

The Applications 304 are participants in the following procedures:

- Starting a new Application
- Ending an Application
- Closing a session if the application is persistent.
- Focus Change
- Message Passing
- Reader Messages (in some instances)

9.0 I/O DAEMON

The I/O daemon 300 is responsible for transporting the data being sent from the remote reader 1 to the event manager 301, and vice versa for a two-way protocol. The I/O daemon 300 is configured to be able to read from the hardware of the system 600 either directly or through operating system drivers that are interface with the remote reader 1, for example, an IR link or standard serial hardware connection. The I/O daemon 300 is also required to listen on a TCP/IP port to wait for the event manager 301 to connect, at which point the I/O daemon 300 sends data from the remote reader 1 to the event manager 301 encapsulated in a TCP/IP stream.

The I/O daemon 300 does not communicate with the rest of the system 600 except to send the remote reader 1 data to the event manager 301, and vice versa in optional two way protocol arrangements between the I/O daemon 300 and the remote reader 1.

While the functionality of the I/O daemon 300 must be present in the system 600, the I/O daemon 300 does not have to be a separate component. For example, the I/O daemon 300 can be integrated into the event manager 301 if the event manager 301 is running on the same machine as the hardware used to interface with the remote reader 1.

The I/O daemon 300 is configured to run on minimum hardware for the instance where the rest of the system 600 is running remotely.

9.1 Requirements

9.1.1 General Requirements

The platform upon which the I/O daemon 300 is implemented must be configured be able to receive signals from (and optionally transmit signals to) a remote reader 1. The platform also preferably has a TCP/IP stack or other reliable communications method implemented on it to communicate with the other parts of the system (i.e. the event manager (EM) 301). The I/O daemon 300 can be required to do multiplexed I/O, and the I/O system of the architecture 200 is preferably configured to support multiplexed I/O.

The architecture 200 is preferably configured to assign a port that the I/O daemon 300 will be listening on, for example, as a command line argument.

9.1.2 Internal Requirements

The I/O daemon 300 is not required to understand the protocol used by the remote reader 1. The I/O daemon 300 is only required to forward all data that it receives to any listening EM (event manager). The I/O daemon 300 is not required to correct any errors of transmission from the remote reader 1 unless it is supported by the transport protocol of the communications link (i.e. through error correcting codes or similar). If the transport protocol being used supports error detection but not correction then any data that does not pass the error check can be passed onto the event manager 301.

9.1.3 External Interface Requirements

The I/O daemon 300 is preferably able to accept one or more TCP/IP connections. The data stream that is sent to the event manager 301 is the content of the data sent by the remote reader 1. All header and footer information that is transmitted as part of the communications protocol used is preferably stripped off and the byte ordering is big endian. If the communication method of the architecture 200 ever becomes unusable (e.g. due to an error arising) then the I/O daemon 300 closes all connections as soon as the error condition arises.

9.2 External Interface

The external interface (not shown) of the I/O daemon 300 is intentionally simplistic to allow it to be run on minimum hardware. The I/O daemon 300 is preferably configured in the following manner.

9.2.1 Start-up Procedure

The I/O daemon 300 listens on a TCP/IP port that is specified to it in some manner, for example, by command line arguments. The exact method of informing the I/O

daemon 300 of the TCP/IP port is implementation specific. The communications hardware used to communicate with the remote reader 1 is initialised if required and the method to read data that is sent from the remote reader 1 is configured to be ready to receive data. While the I/O daemon 300 is waiting for a connection, the I/O daemon 300 consumes the data that is being sent by the remote reader 1 so that when a connection is made, only new data is being sent. This new data is not required to start on a message boundary.

9.2.2 Connection from an event manager

If a connection arrives on the TCP/IP port then the I/O daemon 300 is configured to accept the connection and begin transmitting any data received from the remote reader 1 down the connection. If the I/O daemon 300 is already connected to an event manager (EM) 301 then the I/O daemon 300 has two options. Firstly, the I/O daemon can accept the connection and send all data down all currently connected event managers. This option is provided for system debugging purposes. The second method is to reject the second connection and continue to send the data to the already connected EM. Any encryption of the stream can be handled externally by some other method, such as port tunnelling.

9.2.3 Connection from an event manager closing

If at any time the connection to the event manager 301 is closed, then the I/O daemon 300 is configured to discard any data from the remote reader 1 that is waiting to be sent to that event manager 301. If this is the only event manager connected then the I/O daemon 300 is configured to return to an initial startup state whereby the I/O daemon 300 consumes data being sent by the remote reader 1 and waits for a connection.

9.2.4 Unrecoverable error is encountered

If the I/O daemon 300 detects an error that cannot be dealt with and will cause the I/O daemon 300 to exit, then the I/O daemon 300 is configured to close all connections to any EMs to inform the EMs that the I/O daemon 300 has detected an error. Examples of these errors include if the hardware that is being used to communicate with the remote reader 1 becomes unavailable or if the I/O daemon 300 receives a signal that would cause it to exit. The I/O daemon 300 is configured to close all connections as soon as an error is experienced.

10.0 LAUNCHER

The launcher 303 is the process component that enforces site specific rules such as allowed applications and basic application configuration rules. The launcher 303 allows the other component processes 300, 301, 304, 305 and 306 of the system architecture 200 to be used in a wide range of applications from a general home set top box 601 to a very specific application (e.g. an automatic teller machine (ATM)). A launcher 303 can be specifically written for each network or installation.

The launcher 303 is configured with special privileges. For example, the launcher 303 can be configured to be the first component to connect to the event manager 301 as the system 600 starts up. Further, the launcher 303 receives all "LOW_BATT", "BADCARD", "INSERT", and "REMOVE" messages sent by the remote reader 1 and also receives all "PRESS", "RELEASE" and "MOVE" messages that originate from a card other than the smart card 10 that the front application is associated with at any one point in time. The launcher 303 also receives PRESS, RELEASE and MOVE messages with a special "NO_CARD" distinguishing identifier. The launcher 303 also has control over which application is the front application via the EM_GAINING_FOCUS and EM_LOSING_FOCUS events.

The launcher 303 is configured to decide when applications need to be started and made to exit. The launcher 303 is also used to start and stop applications although this is not always the case. This role can be undertaken by another application at the instruction of the launcher 303, for instance, in the case where the applications 304 are run on separate machines to the rest of the components of the architecture 200.

The events that are sent to the launcher 303 instead of being sent to the current front application allow the launcher 303 to make decisions on which application(s) are to be running at the any moment in time and being configured to force applications to exit means that the launcher 303 can enforce which applications are to be currently running. The launcher 303 is also required to inform the event manager 301 when it is starting and stopping applications.

Fig. 36 is a flow diagram, showing an overview of the method 3700 performed by the launcher 303. The method 3700 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3700 can be executed by the CPU 4305 in set top box implementations or by the CPU of a remote server. The method 3700 begins at the first step 3701, where the launcher 303 connects to the event manager 301, and then continues to a next step 3702 where persistent applications are started. At the next step 3703, the launcher 303 waits for an event and when an event is received the launcher 303 proceeds to step 3705. If the event is the NO_CARD identifier at step 3705, then the process proceeds to step 3707. Otherwise the method 3700 proceeds to step 3709. At step 3707, the launcher 303 performs a predetermined system specific function (e.g. displays a message on the display 101) in response to the NO_CARD identifier and the method 3700 returns to step 3703.

If the event at decision step 3705 is determined not to be a NO_CARD identifier, another decision step 3709 is entered to determine whether or not the event is a PRESS,

RELEASE, REMOVE or MOVE. If this decision step 3709 returns a "yes", that is, the event is one of the aforementioned events, then the method 3700 proceeds to step 3800. Otherwise the method 3700 proceeds to a further decision step 3713. At step 3800, the launcher 303 changes the application in accordance with the process steps described with reference to the flow diagram Fig. 37. The method 3700 returns to step 3703.

If the event at step 3709 is not one of the PRESS, RELEASE, REMOVE or MOVE events, then a decision step 3713 is entered. This decision step 3713 makes a determination on a BADCARD or LOW_BATT event. If the event is a BADCARD or LOW_BATT event at step 3713, then the method 3700 proceeds to step 3715, otherwise the method 3700 proceeds to step 3717. At step 3715, the launcher 303 gives the user feedback on the event that has occurred (e.g. displaying a "Low Battery" message on the display 101 if the LOW_BATT event is determined or a "Incorrect Card" upon determination of a BADCARD event) and the method 3700 returns to step 3703. If the event at decision step 3713 is neither a BADCARD or LOW_BATT event, then step 3717 is entered.

If the event is an APP_REGISTER event at step 3717, then the method 3700 proceeds to step 3900, "Application Registering". Otherwise the method 3700 proceeds to step 3725. At step 3900, the application is registered as described herein with reference to Fig. 38 (i.e. the application informs the other components 301, 302 and 306 that it is now ready to receive messages, as described above with reference to section 8.3.4) and the method 3700 returns to step 3703. A method of registering an application in accordance with step 3900, will be described in more detail below with reference to the flow diagram of Fig. 38. At step 3725, the event is discarded and the method 3700 returns to step 3703.

Fig. 37 is a flow diagram showing the method 3800 of changing an application, which is performed by the launcher 303. The method 3800 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3800 can be executed by the CPU 4305 in set top box implementations or by the CPU of a remote server. The method 3800 begins at step 3817, where if a REMOVE message has been received by the launcher 303 then the process proceeds directly to step 3813. Otherwise, the method 3800 continues to decision step 3801. At decision step 3801, if the service represented by the event is associated with an application that is registered, then the method 3800 proceeds directly to step 3819. Otherwise, the method 3800 continues to step 3803, where a service identifier lookup is performed to determine the location and/or name of a new application and any initial data associated with the new application. For example, the initial data may be a URL to load into a browser 403 or a media file to be loaded into a media player application. At the next step 3805, if the application is already running the method 3800 proceeds to step 3819. Otherwise, the method 3800 proceeds to step 3809, where the new application is retrieved from applications 304. At the next step 3811, the new application is started as the front application, and at step 3812 the event manager 301 is notified of the component identifier (Xid) of this new front application.

Decision step 3819 is entered either from step 3801 if the service represented by the event is associated with an application that is registered or if the application is already running. At step 3819, if it is determined that an INSERT message is received by the launcher 303, then the method 3800 concludes. Otherwise, the method 3800 proceeds to step 3807, where the new application is sent a GAINING_FOCUS event indicating that the new application will soon be changing state. After the new application is sent a GAINING_FOCUS event, or as a result of a REMOVE event detected at decision step 3817, control is passed to decision step 3813. At step 3813 it is determined if there is an

existing front application, if there is no previously front application, then method 3800 concludes. Otherwise, a LOSING_FOCUS event is sent to the previous front application enabling the previous front application to complete immediate tasks, before the method 3800 concludes.

Fig. 38 is a flow diagram showing the method or process 3900 of registering a new application, which is performed by the launcher 303. The method 3900 can be executed by the CPU 205 for computer implementations. Alternatively, the method 3900 can be executed by the CPU 4305 in set top box implementations, or by the CPU of a remote server. The process 3900 begins at step 3901, where a new service group list, including the application, referred to with reference to step 3900 of Fig. 36, is generated. At the next step 3903, a GAINING_FOCUS event is sent to this application. Then at the step 3905, if any applications are not part of the new service group and are not persistent, the method 3900 proceeds to step 3907. Otherwise the method 3900 concludes. At step 3907, any applications which are not part of the service group are sent an EXIT_NOW event, and the method 3900 proceeds to a next step 3908 where the event manager 301 is notified that the applications, which were not part of the new service group, have been terminated. The method 3900 then concludes.

Fig. 39 is a flow diagram showing the process steps 4000 performed by an application when receiving events from the launcher 303. The method 4000 can be executed by the CPU 205 for computer implementations. Alternatively, the method 4000 can be executed by the CPU 4305 in set top box implementations or by the CPU of a remote server. The method steps 4000 begins at step 4001, where the launcher 303 connects to the event manager 301 and then the method 4000 proceeds to step 4002. At step 4002, the application is registered by sending an APP_REGISTER message to the launcher 303. Following the flowchart shown in Fig 39, to the next step 4003, the

application waits for events and when an event is received the process proceeds to step 4005. If the event is a GAINING_FOCUS event at step 4005, then the method 4000 proceeds to step 4007. Otherwise the method 4000 proceeds to step 4009. At step 4007, the application is initialised if necessary, optionally using the distinguishing identifier and optionally using the data field of the GAINING_FOCUS event. This data field used for initialisation may include a URL to load, a filename to load, etc. Control returns to waiting for events at step 4003.

If the event is a PRESS, RELEASE or MOVE event at step 4009, then the method 4000 proceeds to step 4011. Otherwise the method 4000 proceeds to step 4013. At step 4011, an application specific action is performed in response to the event. The application specific action is performed using data from the event (i.e. data associated with an indicium on the card 10, (eg URL, character or video name)), the X/Y position or distinguishing identifier or any combination of these.

The application specific action is typically associated with an indicium on the card 10. For example, an indicium can be associated with a particular URL and when the indicium is pressed the URL may be accessed. Therefore, for example, the computer 100 or STB 601 can download desired programs from a Web Page that was designated by the URL, and a card user can receive the service (i.e program download) from the system 600. Further, an indicium can be associated with a particular memory address and when the indicium is pressed the memory address can be used to data store at the memory address. Therefore, for example, the computer100 or STB 601 can download desired image data from memory or from a file server on a network, which was designated by the memory address, and a card 10 user can receive the service (e.g. image data download) from the system 600. After step 4011, the method 4000 returns to step 4003 as shown in Fig. 39.

The process steps 4000, according to the flowchart of Fig. 39 as described above, filters through to step 4013 if an event is not determined to be any one of a GAINING_FOCUS, PRESS, RELEASE or MOVE event at the corresponding decision steps 4005 or 4009. If the event is a LOSING_FOCUS event then at step 4013 the method 4000 proceeds to step 4015. Otherwise the method 4000 proceeds to decision step 4017. At step 4015, the application reverts to an inactive state and the method 4000 returns to step 4003. If the event is an EXIT_NOW event at step 4017, then the method 4000 concludes. Otherwise the method 4000 proceeds to step 4019, where the event is ignored and the method 4000 returns to step 4003.

Fig. 40 is a flow diagram showing the method 4100 performed by the browser controller 402 application when receiving events from the launcher 303. The method 4100 can be executed by the CPU 205 for computer implementations. Alternatively, the method 4100 can be executed by the CPU 4305 in set top box implementations, or by the CPU of a remote server. The method 4100 begins at step 4101, where the browser application sends an APP_REGISTER message to the launcher 303. At the next step 4103, the browser application waits for events and when an event is received the method 4100 proceeds to step 4105. If the event is a GAINING_FOCUS event at step 4105, then the method 4100 proceeds to step 4107. Otherwise the method 4100 proceeds to step 4109. At step 4107, the application is initialised if necessary. For example, the application reads the data field of the GAINING_FOCUS message and, if the data field represents a URL, the application loads that URL. Initialisation is performed on the browser controller 402, by loading an initial URL into the browser application 403 and storing the base of the URL. The method 4100 continues at the next step 4121, where the distinguishing identifier is determined from the event. At the next step 4123, a JavaScript call back function (preferably known as the Notify_Card_ID) is called in the

current top-level document with the distinguishing identifier 1110 as the argument, and then the method 4100 returns to step 4103.

If the event is a PRESS, RELEASE or MOVE event at step 4109, then the method 4100 proceeds to step 4200. Otherwise the method 4100 proceeds to step 4113. At step 4200, a browser application specific action is performed in response to the event. The browser application specific action will be described in more detail below with reference to the flow diagram of Fig. 41. After step 4200, the method 4100 returns to step 4103.

If the event is a LOSING_FOCUS event at step 4113, then the method 4100 proceeds to step 4115. Otherwise the method 4100 proceeds to step 4117. At step 4115, the browser application reverts to an inactive state and the method 4100 returns to step 4103.

If the event is an EXIT_NOW event at step 4117, then the method 4100 concludes. Otherwise the method 4100 proceeds to step 4119. At step 4119, the event is ignored and the method 4100 returns to step 4103.

Fig. 41 is a flow diagram showing a browser application method 4200 executing on the system 600 incorporating the software architecture 200. The method 4200 can be executed by the CPU 205 for computer implementations. Alternatively, the method 4200 can be executed by the CPU 4305 in set top box implementations or by the CPU of a remote server. The method 4200 begins at step 4201, where if the event is a PRESS event then the method 4200 proceeds to step 4225. Otherwise the method 4200 proceeds to step 4203, where the event is ignored and the method 4200 concludes. At step 4225, the distinguishing identifier is determined from the event. At the next step 4227, if the current page has been notified about the current distinguishing identifier then the method 4200 proceeds to step 4205. Otherwise, the method 4200 proceeds to step 4229, where the JavaScript call back function known as the Notify_Card_ID is called in the current

top-level document with the distinguishing identifier as the argument, and then the method 4200 proceeds to step 4205.

At step 4205, data is retrieved from the event. At the next step 4207, if the data is a single character then the method 4200 proceeds to step 4209. Otherwise the method 4200 proceeds to step 4211. At step 4209, the character is sent to the browser application 403, and the method 4200 concludes. This may be used to provide the same effect as a user pressing a key on a keyboard or a button on a conventional remote control. The current page may provide an action which is performed on receipt of a given keypress using existing methods such as those provided by Hyper Text Mark-up Language (HTML).

If the data starts with "js:" at step 4211, then the method 4200 proceeds to step 4213. Otherwise the method 4200 proceeds to step 4215. At step 4213, a JavaScript function in the current top-level document is called and the method 4200 concludes. The specified data may optionally include an argument for the JavaScript function. For example, the data "js:hello" would indicate that the browser controller is to call the JavaScript function "hello", and the data "js:hello(world)" would indicate that the browser controller is to call the JavaScript function "hello" with the argument "world".

If the data starts with "cmd:" at step 4215, then the method 4200 proceeds to step 4217. Otherwise the method 4200 proceeds to step 4219. At step 4217, a specified browser function is called and the method 4200 concludes. For example, the data "print" would result in the browser controller instructing the data "back" would result in the browser controller instructing the browser to return to the previously displayed page.

If the data is an absolute URL at step 4219, then the method 4200 proceeds to step 4221. Otherwise the method 4200 proceeds to step 4223. At step 4221, the data is loaded into the browser application 403 as a URL and the method 4200 concludes.

At step 4223, the data is loaded into the browser application 403 as a URL after the base URL has been appended, and the method 4200 concludes.

A variation on the browser controller application described above with reference to Fig. 40, is a program controller, which provides control of a software program. The software program can include any program, which is normally controlled with one or more keypress events (e.g. like a keyboard keypress event or the equivalent on a game controller). The program controller can be used to provide card-based control of an existing software program such as an interactive game. The program controller process behaves substantially as described with reference to Fig. 40 with the following exceptions. If the event at step 4105 is a GAINING_FOCUS event, then the program controller process proceeds to a step of getting a Resource Locator, for the software program to be controlled, from the GAINING_FOCUS message. The process then proceeds to a step of getting and starting the software program specified by the resource locator. The program controller process then proceeds to step 4103. Further, at step 4109, instead of testing for a PRESS, RELEASE or MOVE event, this particular variation in the method 4100 would substantially check for a PRESS event. If the event is a PRESS event, the process proceeds to the steps of getting the data from the event, taking the first character from that data, and effecting a keypress of that character resulting in the same effect as if a user had typed that character on a keyboard.

10.1 Special Routing Rules for the Launcher

The launcher 303 has a special set of routing rules and the launcher 303 always receives the following events:

- EM_REMOTE_INSERT
- EM_REMOTE_REMOVE
- EM_REMOTE_BADCARD

The launcher also receives EM_REMOTE_PRESS, EM_REMOTE_RELEASE and EM_REMOTE_MOVE messages if a service identifier does not match a currently front application or if the distinguishing identifier represents the NO_CARD present identifier (i.e. all zeroes). For the purposes of determining whether or not messages match, the service-specific identifier is ignored.

The launcher 303 can be configured to explicitly make itself the front application by sending itself a EM_GAINING_FOCUS event. In this instance, all messages will be sent to the launcher 303 regardless of the service identifier of the message. The launcher 303 is not required by the protocol to respond to any of these messages.

10.2 Sample Implementations

This section outlines several examples of launcher configuration.

10.2.1 Generic Launcher

A generic launcher can be used in an open set-top-box or computer environment with broad-band Internet connectivity. In accordance with this configuration, the launcher 303 assumes that there are applications that can be downloaded to a local machine or designated remote machine and run. A generic launcher can also be configured to accommodate the use of applications that use the browser 403 via the browser controller 402.

The generic launcher can be configured to download applications as well as support persistent applications. The computer 100 running the system 600 preferably has a reasonably fast Internet connection available. In this instance, some of the applications 304 can be web pages with JavaScript that is handled by a persistent application called the browser controller 402, as described above. Further some of the applications 304 can be designed to work together. The generic launcher preferably also assumes that the

communications link used by the remote reader 1 is unreliable (i.e. an IR link) so messages can be lost.

10.2.2 Rules for the generic launcher

The following rules are the rules that are preferably used by the launcher 303 to define the system 600.

- EM_REMOTE_PRESS and EM_REMOTE_RELEASE events that have the NO_CARD present identifier (i.e. all zeroes) are used as a cue that the user wishes to exit from the front application. This could result in the system 600 either generating a "Please insert a card" message on the display 101 or returning to an earlier application, depending on the configuration of the system 600.
- EM_REMOTE_BADCARD events cause the launcher 303 to provide the users with feedback indicating that the card is faulty.
- EM_REMOTE_INSERT, EM_REMOTE_REMOVE are not relied upon to provide the bounds of the session because of the assumed unreliable communications method from the remote reader 1 to the event manager 301.
- If the launcher 303 receives an EM_REMOTE_PRESS, EM_REMOTE_RELEASE or an EM_REMOTE_MOVE message then the launcher 303 does a service mapping, and if the service identifier resolves to a downloadable application then the corresponding application is downloaded and run. The mapping is done by querying the Directory Server 305 with the service information from cards. The values returned from the Directory Server 305 are an application location and associated service data. The application location specifies the location of the application or a value the launcher recognises as a local application. The service data is the initialisation data that is sent to the application in the EM GAINING FOCUS

message. If the application location is empty the launcher 303 is configured to decide which application to use based upon the service data which will be a URL.

- When a new application registers with an EM_APP_REGISTER message the specified service groups are compared with a currently running set of applications and if there is no overlap then all other currently running applications are told to exit. The new application is made the current front application (using an EM_GAINING_FOCUS event) and the previously front application is sent an EM_LOSING_FOCUS event. If this occurs and the service identifier resolves to a web page then the focus is changed, using an EM_GAINING_FOCUS message, to the browser controller 402 with the address (location) of the web page in the data field. The data field is returned in the query that told the launcher 303 that the service identifier resolved to that web page. In this situation, an EM_LOSING_FOCUS event is also sent to the current front application. All other applications are told to exit.

10.3 An Example Single Use System

The architecture 200 can be configured for use with a single specialised application. In this instance, the launcher 303 can be used where it is advantageous to have a physical token (e.g. a bank card) where part or all of the user interface can be printed onto the token. The example described below is in the form of an automatic teller machine, and whilst this example is described in terms of a specific specialised application it should not be read as being limited to automatic teller machines. Such a system can be configured to be able to use a single or at least very limited number of cards. In this system no other applications 304 are started regardless of the card that is entered. The launcher 303 takes the role of a single application 304 as well as that of a system controller. No modifications are made to the event manager 301.

A single use system can be used in an automatic teller machine for example. A bank can produce personalised bank cards with commonly used options on the cards that are used as the sole or supplementary interface for an automatic teller machine. In this instance, the automatic teller machine preferably contains an event manager 301 and other core process components of the architecture 200. In this specific example the communications link between the remote reader 1 and the event manager 301 must also be reliable.

10.3.1 Rules

The following rules can be used by a launcher 303 to define a single use system bank teller machine example :

- Any events that do not come from cards associated with a participating bank could cause the launcher to display an incompatible card screen on the terminal.
- EM_REMOTE_BADCARD events are ignored.
- EM_REMOTE_INSERT events are used to start the transaction.
- EM_REMOTE_REMOVE events are used to end the transaction.
- EM_REMOTE_PRESS, EM_REMOTE_RELEASE and EM_REMOTE_MOVE events are treated as a user interaction. These are preferably handled directly by a launcher as that is the one application that is running.
- Service mappings to an external Directory Server are never done. If the card is not one that a particular automatic teller machine (ATM) knows about then the card should be rejected.

These rules are examples of how a single use system can be configured to provide a specific application in the form of an ATM.

10.4 Directory service Operation

Fig. 58 is a flow diagram, showing an overview of the process 5800 performed by the Directory Service 311. The process 5800 is executed by the CPU 205 of a computer 100, which performs the role of a Directory Service 311. The software program as shown in Fig.58 is stored in a memory medium such as Memory206 or CD-ROM212 in the system 600A or Memory 4306 in the system 600B. The process 5800 begins at the first step 5801, where the Directory Service 311 is started. At the next step 5802, the CPU waits for incoming events from a Launcher 303. The events are sent from Read Device 1 to Launcher 303 via Event Manager 301. At the next step 5803, the CPU receives a request from a Launcher 303, which contains a Distinguishing identifier, which is to be mapped by the Directory Service 311. The connection between the Launcher 303 and the Directory Service 311 is shown in Fig. 8.

At the next step 5804, the CPU searches a directory-mapping table to check if the table has an entry corresponding to the Distinguishing identifier. The directory-mapping table typically contains relations between Service identifiers and corresponding application location (e.g. URL) and service data and additionally contains relations between Distinguishing identifiers and the corresponding application location and service data. Typically, the relation involving the Service identifier is used with respect to cards 10 for which the Directory Service 311 is intended to maintain service-level information for all cards 10 which can be used for that service (for example, the location of the application 304 which is to be executed to provide the service for the card 10). Typically, the relation involving the Distinguishing identifier is used with respect to cards 10 for which the Directory Service 311 is intended to maintain information specific to the actual cards 10 or groups of cards 10 which have identical service-specific identifiers (for example, the location of a media file which is to be played to provide the service for the card 10). The directory-mapping table is typically stored in hard disk 210 or in memory 206. At step

5804, if there is an entry for the Distinguishing identifier in the directory mapping table, at the next step 5805, the CPU retrieves the application location and service data from this entry and moves to step 5806. At step 5804, if there is not an entry for the Distinguishing identifier in the table, the CPU at step 5808 extracts the Service identifier from the Distinguishing identifier by taking the relevant portion of this value (typically the first 5 bytes as is indicated in Fig. 11). At the next step 5809, the CPU searches the directory-mapping table for an entry corresponding to the Service identifier. If one is found, the CPU retrieves the application location and service data from this entry at the next step 5810 and moves to step 5806. If one is not found, at step 5811, an entry is placed in a log file indicating that a request had been made for the specific Distinguishing identifier and, at step 5812, an error is returned to the Launcher 303 indicating that the Service identifier part of the Distinguishing identifier supplied is not known by this Directory Service 311. The flow then continues to step 5802.

At step 5806, where a Distinguishing identifier or a Service identifier has been successfully found, the Distinguishing identifier and corresponding application location and service data is written to a log file and the CPU returns the application location and service data to the Launcher 303 which made the request. Flow then continues to step 5802 to wait for another event.

11. GENERAL

Typically, applications 304 are resident on the hard disk drive 210 and read and controlled in their execution by the CPU 205. Intermediate storage of programs and any data fetched from the network 220 can be accomplished using the semiconductor memory 206, possibly in concert with the hard disk drive 210. In some instances, the applications

304 will be supplied to the user encoded on a CD-ROM or floppy disk and read via the corresponding drive 212 or 211, or alternatively may be read from the network 220 via the modem device 216. Other mechanisms for loading software application into a computer system 100 from other computer readable medium include magnetic tape, a ROM or integrated circuit, a magneto-optical disk, a radio or infra-red transmission channel between the computer module 102 and another device, a computer readable card such as a smart card, a computer PCMCIA card, and the Internet and/or Intranets including email transmissions and information recorded on Websites and the like. The foregoing is merely exemplary of relevant computer readable media. Other computer readable media are also possible including combinations of those described above .

Alternatively, the process components 301 to 306 described above can be implemented in dedicated hardware as one or more integrated circuits performing the described functions or sub-functions. Such dedicated hardware may include graphic CPUs, digital signal CPUs, or one or more microCPUs and associated memories. An examples of dedicated hardware is the set top box 601 for a television described with reference to Fig. 6(b) above.

12. OTHER VARIATIONS

12.1 A Session Identifier

As described above, the distinguishing identifier is included in every INSERT, REMOVE, PRESS, RELEASE and MOVE message sent from the reader 1 to the computer 100 or set-top box 601. As an alternatively, the distinguishing identifier can be sent in connection with an INSERT message only. In this instance, upon insertion of a new card 10, the reader 1 generates a session identifier (not illustrated). The session identifier identifies a current session of a card insertion. The session identifier, for example, can be a pseudo-random number (which can be represented with 2 bytes of data)

or the session identifier can be a number that is incremented each time a card is inserted (and reset to zero when a predetermined value is reached). The reader 1 sends an INSERT message to the computer 100 or the set-top box 601, which includes a distinguishing identifier as previously described above and a session identifier which is generated for each new insertion. All subsequent PRESS, RELEASE and MOVE messages need not include the distinguishing identifier but will include the session identifier and user interface object data or press coordinates previously described.

When using a session identifier, the system 600 performs as described above with reference to Figs. 6(a) and 6(b), except that the event manager 301, upon receiving an INSERT message from a reader 1, stores the session identifier as the current session identifier and a distinguishing identifier as the current distinguishing identifier. When the event manager 301 receives a PRESS, RELEASE or MOVE message, the event manager 301 checks that the session identifier is equal to the current session identifier. If so, the event manager 301 sets a distinguishing identifier used in all messages to the current distinguishing identifier. Otherwise, if the session identifier is not equal to the current session identifier, the event manager 301 informs the user, via the display manager 306, and the display device 101, that a message has been received without a corresponding INSERT message. The user, for example, is then requested to remove and reinsert the card 10.

12.2 Other Characteristics of a user Press

As described above, the sending of information relates to the pressing, moving and releasing of an object (typically with a finger or stylus) on the touch panel 8 of the reader 1. However, the reader 1 can send additional information pertaining to an interaction from the touch panel 8 to the computer 100 or set-top box 601 for use by the system 600. For example, the additional information can represent a length of time or an

amount of pressure exerted upon the touch panel 8 as a result of a press. This additional information can be incorporated in the PRESS messages sent from the reader 1 to the system 600 and with the EM_READER_PRESS messages sent within the system 600. In this instance, the information is passed to an application 304 corresponding to the card inserted in the reader 1. An application can make use of the additional information to provide, for example, an added effect on a particular action. For example, the application can use pressure information, when associated with a press on an indicium indicating an increase in (audio) volume, to determine an amount of increase in volume. That is, the harder the press on the selected indicium, the higher the rate of increase in the volume and conversely, the softer the press on the selected indicia the lower the rate of increase.

Another example of the use of additional information in relation to a length of time (or duration) of an interaction with a touch panel 8 is described below. If a press is of very short duration, the press can to be considered to be a "tap". On the other hand, a press of very long duration can be considered as a persistent "holding down" of a keypress. In this instance, additional information can add an extra dimension to a mode of interacting with an instant software application. For instance, a "tap" on the touch panel 8 can be an instruction to the software application to select an item displayed at a current (on-screen) cursor position.

12.3 No Coordinates

A PRESS and RELEASE message can be configured not to include coordinate data of a user's interaction with the touch panel 8. In this instance, coordinate data is only sent from the reader 1 to the system 600 in conjunction with a MOVE message. The advantage of not including coordinate data in a PRESS and RELEASE message is a size reduction of messages sent by a reader 1 to the system 600, where an applications 304

does not require coordinate information for mapping from coordinates to user interface element data.

12.4 Two-way protocol

A one-way or a two-way protocol can be used for communication between a reader 1 and a computer 100 or set-top box 601. The description of the reader 1 hardware with reference to Fig. 10, and the I/O Daemon described with reference to Figs. 8 and 9 included a sending of information from a reader 1 to computer 100 or set-top box 601 and vice versa. The sending of information back to a reader 1 from a computer 100 or set top box 601 can be used to change the data stored on a card 10. For example, changing user interface object data stored on the memory chip of a smart card 10.

A two-way protocol can also be used to enable hand-shaking in the protocol. For example, a two-way protocol between a reader 1 and a set-top box 601 or computer 100 can be used so that the system 600 can acknowledge the receipt of an INSERT message sent when a card is inserted in the reader 1. A system 600 which supports a two-way protocol should also provide an additional message in the event manager protocol, in order to allow an application to send a request in order to modify a portion of the stored data on a card 10, sent to the I/O daemon 300 via the event manager 301. The I/O daemon 300 can then send a message to the reader 1 to bring about a requested action. For example, if the system 600 uses a two-way protocol then the system 600 can provide a security mechanism to ensure that applications can not modify cards without the permission of a user or without a system-defined privilege. In one example of such a system, the event manager 301 can present a displayed message to a user asking if it is OK for the application to modify a currently inserted card. The user can assent to the proposal by pressing a first region of the touch panel 8 and dissent to the proposal by pressing a second region of the touch panel 8. If the user assents to the modification of

the card 10 then the event manager 301 can allow the request from the application 304 to be passed onto the I/O daemon 300 and then on to the reader 1. On the other hand, if the user dissents from the modification, the event manager 301 drops the message and the information is not sent to the reader 1.

12.5 Alternative Read Device

In the above system600A and 600B, the Read device 1 has a substantially transparent touch sensitive membrane arranged to overlay the card 10. To reduce a cost of the Read Device 1, instead of the touch sensitive membrane, the Read Device 1 may has a plurality of user operable switches positioned around the receptacle into which the smart card 10 is insertable for reading the data, the distinguishing identifier and relation information to associate the data with each switch. Therefore the user can select at least one of the switches that correspond to at least one indicia on the card, since the operable ones of the switches are associated with indicia on the smart card visually. In this case CPU45 reads the data corresponding to a switch pressed by the user based on the relation information and the distinguishing identifier from the card 10 and sends them to Event Manager 301.

13.0 ALTERNATIVE SOFTWARE ARCHITECTURE

A further software architecture 4900 for the hardware architecture depicted by the system 600, is generally illustrated in Fig. 48 and represents an alternate software architecture to that described in previous sections. The alternative architecture 4900 is configured to be scaled from very low hardware requirements at the users home (ie. a simple set-top box), up to a powerful home system, where for example the set-top box 601 functionality is implemented on personal computing system. Further, the alternative architecture 4900 is preferably implemented within the hardware system 600.

13.1 Structure

The architecture 4900 is divided into six distinct processes and one class of process. The distinct processes include a smart card interface 4902, referred to as an I/O daemon as in the architecture 200, an event manager 4904, a display manager 4906, a master launcher 4908, an (application) launcher 4910 and a directory service 4912. The class of process is formed by one or more smart card applications 4920. In the architecture 4900 there exists one card daemon 4902, one event manager 4904, one display manager 4906 and one launcher 4910 for every smart card remote connection, usually formed by the set-top box 601, but only one master launcher 4908 for each computer that is running the launchers 4910, and at least one directory service 4912 for all systems.

In this form, the architecture 4900 can be physically separated into three distinct parts 4914, 4915 and 4916, as shown by the dashed lines in Fig. 48, each of which can be run on physically separate computing devices. Communication between each of the parts of the system is performed using TCP/IP streams as with the architecture 200.

The I/O daemon 4902 is a process that converts datagrams received from the smart card remote reader 1 into a TCP/IP stream. The I/O daemon 4902 is not intended to understand the data format used by the reader 1, but to operate independent of any changes in the smart card remote data format, and thus provides the capability to work with multiple versions of the reader 1.

The I/O daemon 4902 is started when the user starts the system 600 which, in the case of the set-top box system 600B, is when the set-top box 601 is turned on. For the computer system 600A, the I/O daemon 4902 may be started when the user starts the smart card system after the event manager 4904 and master launcher 4908 have been started.

The event manager 4904 forms a central part of the architecture 4900 in that all communications are routed through the event manager 4904. The event manager 4904 is responsible for gathering all events that are generated by the smart card remote reader 1 and relayed by the I/O daemon 4902. These events are then redistributed to the various processes and running applications.

A further role of the event manager 4904 is to isolate misbehaving applications from other well-behaved applications. In this regard, any events passed through the event manager 4904 are guaranteed to be correct to the extent that the event manager 4904 can check the event. The event manager 4904 is required to check that an event has a valid header and the correct data length, but is typically not configured to check if the data is in the correct format.

Any changes to the protocol between different versions are also to be dealt with by the event manager 4904. If possible, the events are to be rewritten to conform with the version of the data format that the operating application 4920 understands. If such is not possible, then the event manager 4904 reports an error to the originating application 4920. When different data format versions are being used, the event manager 4904 ensures that the smallest disruption possible occurs.

The display manager 4906 operates in concert with those applications 4920 operating to control which operating application 4920 has priority with respect to the particular output device 116, typically a display (e.g. 116). It is the role of the display manager 4906 to select which video stream is sent to the display 116, this information being obtained from the respective launcher 4910 of the application 4920, via the event manager 4904. Generally only the front (ie. foreground) application will produce a video display stream. Further, the display manager 4906 may operate to maintain a constant

output stream from the inconsistent input streams and may fill-in some parts of the output stream with extrapolated data.

The event manager 4904 is not responsible for deciding when an application 4920 needs to be started/ended or for actually starting or terminating an application 4920. These operations are both the responsibility of the launchers 4908 and 4910, to be discussed below. Moreover, the event manager 4904 does not have any presence on the users screen or other output device 116. Any system related feedback, such as the display of the initial insert of a smart card, is performed by the launcher 4910.

For the system 600B of Fig. 6(b) incorporating the alternative architecture 4900, there will typically be an event manager 4904 running for every set-top box 601 that is allowed to connect to the system 600B. For the system 600A incorporating the architecture 4900, the event manager 4904 will be started when the smart card system 600A, is started after the master launcher 4908 has been started.

The role of the master launcher 4908 is to start the launchers 4910 at the request of any of the event managers 4904. When the I/O daemon 4902 connects to the event manager 4904, the event manager 4904 requests the master launcher 4908 to start a first process for the event manager 4904. This first process will generally be a launcher 4910 for any smart card application 4920. The master launcher 4908 is also responsible for shutting down the launcher 4910 of an application 4920 when the event manager 4904 so requests, and for informing the event manager 4904 that the correct launcher 4910 has exited.

For the system 600B of Fig 6(b) incorporating the alternative architecture 4900, there will always be one master launcher 4908 running for each physically separate server 150, 152 running smart card applications 4920. This one master launcher 4908 handles

the requests for all event managers 4904 that request launchers 4910 on that server. For the system 600A, the master launcher 4908 commences operation either before or no later than at the same time as the rest of the smart card system.

The card directory service 4912 is provided to translate vendor-application value (Service identifier value) stored within smart cards 10 into an application location such as Uniform Resource Locators (URLs) that each point to the application 4920 associated with a vendor-application pair (Service identifier) which will be described. The directory service 4912 can be split into a number of parts by changing the launcher 4910 so that applications 4920 can run on separate systems to the launcher 4910. The directory service 4912 performs this function using a distributed look-up system where the query is passed on to another directory server if the directory service currently in possession of the query does not know the answer. Such a distributed system allows each directory server to have a limited knowledge of the transition from vendor-application ID pairs to URL's, but to still be able to translate all ID's to URL's. This provides a number of advantages including a simpler database at each directory, is more robust and permits servers to become inoperable (i.e. crash or be removed from service) whilst still permitting queries.

Referring to Fig. 52, the control template customisation information that distinguishes the smart card 10 from traditional smart cards includes a tuple of data from by a vendor identifier, a card identifier and an application identifier. The vendor identifier and the application identifier pair are equivalent to the service identifier described above for the architecture 200. Also, the card identifier is equivalent to the service-specific identifier described above for the architecture 200. Further, associated with each of the icons 4804 is corresponding data that, when a user presses on the touch panel over the icon 4804, is sent as event data that, when passed to the particular application 4920, implements a particular operation within that application. Further

detection of user actions may be incorporated, for example to detect the release of an icon, as distinct from a depression of that icon, and also to detect moving depression, where the user may scribe a finger across the touch panel 8 to perform a particular function. On each such action, event data stored on the card can be sent, which may be read from a different location of memory on the card in each case. The service identifier implemented in this alternative architecture 4900 as a vendor-application identifier pair allows the vendor, of an application associated with a smart card, to be distinguished from other. For deployments of the architecture 4900 where there is no need to distinguish a vendor of the application associated with the smart card, the vendor identifier and the application identifier can be treated as a single value: a service identifier.

The first process started by the insertion of a smart card 10 into a reader 1 will be, in a generalised system (e.g. home), a launcher 4910. In specific systems, specific applications may be commenced. For example a banking teller would start a banking application. Another example includes the use of restricted launchers that only start a specified sub-set of applications. The launcher 4910 is a smart card application that starts other applications 4920 for a specific one event manager 4904. It is the decision of the launcher 4910 to start and end applications 4920 and to actually start and terminate applications 4920. The launcher 4910 informs the event manager 4904 when applications 4920 are starting and ending, and tells applications 4920 that they are receiving or losing focus, or when they need to exit. In this regard, where a number of applications 4920 are operating simultaneously, the application that is currently on-screen is the application having focus. When another application is about to take precedence, the launcher 4910 tells the current application that it is losing focus, thereby enabling the current application to complete its immediate tasks, and tells the new application it is

gaining focus, and that the new application shall soon be changing state. The launcher 4910 must be able to force a program to exit.

The first application 4920 started (ie. usually the launcher 4910) is given special privileges, and receives "NO_CARD", "Bad_CARD" and "POWER_OFF" events generated from the remote reader 1. The first application 4920 also receives events that are intended for applications 4920 that are not the current front application, and the launcher 4910 operates to correctly interpret these events. Such is related to the specific applications mentioned above, so that the launcher correctly interprets any changes. The launcher 4910 is an application 4920 but having special rights, including the right to start and shut down other applications.

The launcher 4910 is preferably only started when the event manager 4904 requests the launcher 4910 to be started. The launcher 4910 can also be told to exit by the event manager 4904.

Applications are started by the launcher 4910 either as a response to the first user selection on a corresponding smart card 10, or at the request of another one of the application 4920. In this regard, the architecture 4900 provides a substantial enhancement over conventional arrangements through each application 4920 being organised during its programming, as a member of one or more application service groups.

13.2 Application Service Groups

An application service group is comprised of a number of smart card applications 4920 that act co-operatively, as opposed to merely simultaneously, to provide a particular set of functions. Applications 4920 that form part of a service group are permitted to run simultaneously, and also share a communication means (ie. the event

manager 4904) by which data may be exchanged. Each such application 4920 is a process or sub-process that provides a set of functions corresponding to a particular user interface or set of user interfaces. Such an application 4920 may or may not have a visible display.

With reference to the example represented in Fig. 49, a service group is initiated once an application 4920 that forms part of that service group is started and registers the particular service group with the event manager 4904. As seen in Fig. 49, a first application 4926 has associated therewith two smart cards 4924 and 4926, and a second application 4934 is operable with smart cards 4928, 4930 and 4932. Accordingly, upon insertion of the card 4922 into the reader 1, the card daemon 4902 communicates that occurrence with the event manager 4904 which, via the launcher 4910 commences application 4926. The commencement of the application 4926 enables a service group 4936, this group also including application 4934. Applications that correspond to the currently established service group may be started by inserting the relevant smart cards. For example, removal of the card 4922 and insertion of the card 4932 operates to launch application 4934, maintaining the service group 4936 as being active. Further, the starting of applications that form part of the same service group does not cause other applications from the same service group to terminate. Rather, the other applications are kept running in the background.

Termination of a service group is initiated either by touching on an empty remote reader 1, or by inserting a smart card corresponding to a different service group, such as the card 4938, corresponding to application 4940 in service group 4942. Termination of a service group causes all the applications that are currently running as part of that service group to be similarly terminated.

Applications running under the same service group may communicate with each other via the event manager 4904 by way of a service-group defined protocol 4950 as seen in Fig.49. In the protocol 4950, the format and contents of data packets sent between applications (e.g. 4926 and 4934) should be defined by the authors of those applications that coexist within the same service group).

Seen in Fig. 50 is another feature of service groups within the architecture 4900, where a service group may contain one or more applications that may run as part of any other service group. These applications provide services that may be required across service groups. An example of such an application is a personal identification service that can provide the postal address and credit card details of a user (once the user has agreed to provide those details). In this respect, such a service may form a component of numerous other services or transactions that require a financial transaction, these including on-line shopping and banking.

The design of applications to support the architecture 4900 may or may not be the same as existing approaches to application design depending on whether applications developed require the new features provided by the architecture 4900. Existing applications will still function with some modification under the architecture 4900. An example of such a modification is where each application that runs under an existing architecture can be assumed to have a service group that has the same name as the running application (ie. each application forms its own service group having only one member), or some other method of choosing a group name that is unique, including not having a group name for existing applications or applications that do not work with other applications.

Applications within the same service group need not operate on the same physical hardware, and may not be able to communicate directly with each other by using operating system defined methods. Two methods of communication are preferably implemented in the event manager 4904 to provide a standard method of inter-application communication. These methods are:

(i) a datagram based protocol where a message is sent by one application to another; and

(ii) a protocol based on a message board, where messages are posted by applications 4920 to a common area from which any application 4920 in the same service group are able to read the messages.

The event manager 4904 imposes no structure on the data that is passed between applications 4920. All the messages are just blocks of data of known length. Any other structure that is imposed on the data only needs to be understood by the applications of the particular service group. The blocks of data may be given types (e.g. raw data, .wav, .doc, etc.) which are stored by the event manager 4904 by the posting application.

A datagram method is used to allow the sending of arbitrary length data from one application in a service group to another application in the same service group, and require that the sending application knows the identification (ID) number (also referred to above as the Xid) of the receiving application. The ID number is generated by the corresponding launcher 4910 when the application 4920 is started to uniquely identify that application 4920. The ID number is unique only in the context of the event manager 4904. In this fashion, many running applications can have the same ID number but every ID number will be unique amongst all the applications 4920 that are connected to the same event manager 4904 to which the particular application is connected. It is the responsibility of the corresponding launcher 4910 to ensure that this occurs, although the

event manager 4904 can detect when duplicate ID numbers are about to be used and prevent the new application from starting.

To send a message using the datagram method, the sending application retrieves the Xid of the destination application from the event manager 4904 and then sends the message via the event manager 4904 to the destination application using this Xid to address the message. The event manager 4904 does nothing to the packet that contains the message except to ensure that the data length and sender fields of the header are correct.

For the datagram method to be available, the event manager 4904 must provide the applications with some method of determining what other applications are running in their service group. This information must also include some method for applications to identify what other applications are capable of. Such is performed in the architecture 4900 using a list of function strings that the application lists when the application registers with the event manager 4904. This list of functions is service specific as the event manager 4904 does not need to understand them in any way. Only other applications in the service need to understand what each function string means.

The event manager 4904 may impose some upper limit on the size of messages that can be passed using this method.

In the architecture 4900, the message board mentioned above allows data to be broadcast to all applications in the service group at once and also allows the applications in the service group to store data in a central repository. This removes the need for any one application to be always present in a service group. The message board also allows smart cards, and therefore applications in a service group, to be inserted/run in an arbitrary order. Applications post the data they contribute to the service group onto the

message board and when an action needs to be taken by an application, the application can examine the message board for the data that is required.

To post data to the message board, the posting application sends to the event manager 4904 the data desired to be posted, a description string, and in some instances some form of typing information (e.g. a MIME type). If the application does not supply the type information, the event manager 4904 will assign the data a default type (e.g. default binary data, the MIME type application/octet-stream). The event manager 4904 then assigns this message a message identifier, which is used to identify the message in the message board. This message identifier is used to retrieve the message from the message board by other applications. The message identifier is also used by the posting application to remove the message from the message board. The message board, and any messages remaining on the message board corresponding to a service group are destroyed when a service group is terminated.

To retrieve a message from the message board, an application must find the message identifier of the message that is required. The application can obtain a listing of the messages on the message board, which will contain the message identifier, poster identifier and the message description of each of the messages on the board. The second method involves also obtaining a listing of running applications from the event manager 4904. This provides the application with the functions that each application provides for the service. The application requesting the message from the message board can then cross-reference the application identifier (Xid) of the application from which it needs the information, against the poster identifier on the message board, and then retrieve all messages posted by that application.

The format of both the messages and the message descriptions on the message board is decided by the service group and may be totally arbitrary. The event manager 4904 does not force any structure upon the data.

To support such a method of communication, the event manager 4904 is required to maintain the message board. To the event manager 4904, the message board appears simply as a list of known length data blocks. When an application posts a message to the message board, the event manager 4904 stores the data and its length. When an application reads a message from the message board, the event manager 4904 sends the data to the application. The event manager 4904 also creates a listing the contents of the message board for applications that request such a listing.

The event manager 4904 may limit the total size of messages that each application can post as well as the total size of all messages that can be posted by all applications in a service group, so that each application has a message size limit and each service group has a message size limit. The number of messages an application and service group may post may also be limited. The size of the descriptions of the messages may also be limited to a maximum length.

13.3 System Initialisation

This section describes the process of initially starting the system 600 incorporating the software architecture 4900 of Fig. 48. It is relevant to the computer system 600A as well as a distributed set-top box system 600B.

Firstly, the master launcher 4908 is started and listens over the network 220 for a reply over a communication port. The event manager 4904 is then started and makes a connection to the master launcher 4908.

This order of starting these two core parts of the architecture 4900 is arbitrary in the case of the system 600A, but has distinct advantages when used in a set-top box system 600B. In the system 600B the master launcher 4908 is already running when the event manager 4904 is started, it is possible to start more event managers when more users subscribe to the service, and to reduce the number of running event managers when users leave the service.

13.4 System Start-up

This section describes the process of starting a smart card system incorporating the hardware architecture of Fig. 6A or 6B and the alternative software architecture of Fig. 48. This description assumes that there is already an event manager 4904 and a master launcher running and they have an open connection.

(i) The I/O daemon 4902 is started and initiates a connection to the event manager 4904.

(ii) The event manager 4904 accepts the connection from the I/O daemon 4902. It is at this stage that any service accounting can be performed. For instance if the user hasn't paid the bill then the connection can be refused.

(iii) The event manager 4904 requests a new launcher 4910 from the master launcher 4908 informing the master launcher 4908 what port the event manager 4904 is listening on, and then waits for an incoming connection.

(iv) The master launcher 4908 starts a new launcher 4910 and gives the new launcher 4910 the address and port number of the event manager 4904.

(v) The new launcher 4910 initiates a connection with the event manager 4904.

(vi) The event manager 4904 accepts the connection.

The system 600 is now ready to start applications 4920 as the user inserts smart cards into the reader 1 and initiates a first button press.

13.5 Starting the First Smart Card Service

This section describes the process of starting a smart card service if no other service is running on the system 600 incorporating the software architecture of Fig. 48. This is the situation when the system is first initiated and can also occur if a service terminates, either though a time-out or because the user touched the remote 1 with no smart card 10 inserted.

(i) The user inserts the smart card 10 into the reader 1 and presses the touch panel 8.

(ii) The pressed event is sent to the event manager 4904 which reformats the packet and forwards it onto the launcher 4910.

(iii) The launcher 4910 receives the packet and recognises that no service is active and queries the directory service 4912 with the service identifier (the vendor identifier and the application identifier) and the service-specific identifier (the card identifier) of the smart card 10.

(iv) The query returns the location of the appropriate application 4920, which the launcher 4910 then fetches. The application 4920 will generally be sourced remotely from storage on a server computer somewhere in the network 220, but may need to be run locally to the

launcher 4910. In advanced systems, the application may be run remotely from the launcher.

(v) The launcher 4910 informs the event manager 4904 that a new application 4920 is starting.

(vi) When the application 4920 has finished downloading to the launcher where it is to be run, it is started by the launcher 4910.

(vii) The application 4920 initiates a connection with the event manager 4904 and when the event manager 4904 has accepted the connection, the application 4920 registers with the launcher 4910. This includes what service groups that application 4920 is part of and what functions the application is capable of performing.

(viii) The launcher 4910 tells the new application 4920 that it is gaining focus.

The application 4920 at this stage has started and capable of receiving events. PRESS, RELEASE and MOVE messages generated from the reader 1 are forwarded to the applications 4920 by the event manager 4904 so long as they are intended from that application. The application 4920 cannot interact with the event manager 4904 in any way until registered has been completed. Further, the event manager 4904 will not forward events to the application and any events that are not application registration events that the event manager 4904 receives from an application 4920 that has not registered, will be discarded.

13.6 Starting, controlling and stopping an Application

Fig. 56 (a) and (b) show a method 5600 of starting, controlling and stopping an application (a application #1- #n) of applications 4920 to provide a service to a user on

the system 600 incorporating the software architecture 4900. The process of method 5600 is executed by CPU such as CPU 205 in system 600A or CPU 4305 in system 600B. A software program indicating the method 5600 is stored in a memory medium such as CD-ROM212 in system 600A or Memory 4306 in system 600B. When a user inserts the smart card 10 into the reader 1 and presses the touch panel 8 to select desired indicia, CPU45 in the reader 1 reads Card Header 1100 and data associated with the selected indicia from the smart card 10 and sends the pressed event (e.g. Press Message) associated with the selected indicia to the event manager 4904 that reformats the packet. The event manager 4904 sends the pressed packet (e.g. EM-READER PRESS) to Launcher 4910. The software program is executed by the CPU that executes at least Card Interface (Demon) 4902, Event Manager 4904, Launcher 4910 and Applications 4920 in same computing device, when Card Interface (Demon) 4902 receives the pressed event from the reader 1 and sends it to Event Manager 4904. On the other hand, if the software program is executed by each CPU in a separate computing device, a first CPU in a first computing device executing Event Manager 4904 executes steps from 5603 to 5608 and second CPU in a second computing device executing at least Launcher 4910 and applications 4920 executes steps from 5609 to 5636.

At step 5603, by executing Event Manager 4904, the CPU receives the pressed event from the reader 1 via Card Interface 4902 and at the next step 5605 the CPU determines if the Service Identifier (the vendor identifier and application identifier) in the pressed event matches that of a front application (e.g. application #1) of applications 4920 already running. If it is determined that the Service identifier matches that of the front application (e.g. application #1) using a matching table at the next step 5605, by executing Event Manager 4904 at the next step 5608 the CPU forwards the pressed packet to the front application and the method 5600 concludes. The table having a relation between each

application of applications 4920 and corresponding service identifier is stored in a RAM in Memory 206 or Memory 4306. If it is determined that the service identifier does not match that of the front application at the step 5605, at the next step 5607 the CPU forwards the pressed packet from Event Manager 4904 to Launcher 4910. At the next step 5609, by executing Launcher the CPU queries the directory server 4912 with the service identifier and receives location of the new application (e.g. application #2) corresponding to the service identifier. At the next step 5611, by executing Launcher 4910, the CPU fetches the new application from the location. At the next step 5613, by executing Launcher 4910, the CPU executes the new application (e.g. application #2). At the next step 5615, the CPU initiates a connection between the new application and Event Manager 4306 and when Event Manager 4306 has accepted the connection, the CPU registers the new application with Launcher 4910 and also the application tells the Launcher 4910 which service groups it is part of. At the next step 5616, the CPU determines if the new application shares a service group with a currently running application using a service group table stored in a RAM in Memory 206 or Memory 4306. The table having a relation each service identifier and corresponding service group is stored in the RAM in Memory in Memory 206 or Memory 4306. For example, in the table, service identifier 1 (application #1) and service identifier 3 (application #3) correspond to a service group A and service identifier 2 (application #2) and service identifier 4 (application #4) correspond to a service group B. At the next step 5616 if it is determined that the new application shares the service group with the currently running application, at the next step 5635 by executing Launcher 4910 the CPU tells the current application (the front application) that it is losing focus. At the next step 5636 by executing Launcher 4910 the CPU tells the new application that it is gaining focus and the method 5600 concludes. In this case, the CPU is still executing the current application

(the front application) in the background but no longer receives any events from the reader 1. By executing the current application the CPU can still send broadcast messages and messages to specific applications but cannot remove itself from service groups.

At the step 5616 if it is determined that the new application does not share the service group with the currently running application, at step 5617 by executing Launcher 4910 the CPU tells the applications that are currently running to exit and sets time-out. At the next step 5621 by executing Launcher 4910 the CPU waits for time-out then terminates any remaining applications except the new application. At the next step 5623 by executing Launcher 4910 the CPU informs the Event Manager 4904 of the applications which have exited or been terminated. At the next step 5636 by executing Launcher 410 the CPU tells the new application that it is gaining focus and the method 5600 concludes. In this case the CPU is now executing the new application and receives pressed packet such as EM-READER PRESS, EM-READER-RELEASE and EM-READEF MOVE that are intended for it. The system 600A or 600B is now running a new service with only one application within the service.]

13.7 Passing Data Between Two Applications

This section describes the process of passing data between two applications 4920 (application #1) and 4920 (application #2) using the datagram protocol on the system 600 incorporating the software architecture of Fig. 48. This method requires that the sending application #1 know the application identifier (Xid) of the receiving application #2.

(i) The sending application #1 gathers the data that it wishes to send.

(ii) The sending application #1 asks the launcher 4910 for the list of applications that are running in the current service group.

(iii) The launcher 4910 sends the application #1 the list of all applications in the current service group. This list includes the functions that each

application has told the launcher 4910 that it can perform as well as the descriptive string the application provided. This list is order with the most recent application listed first.

(iv) The sending application #1 looks to see if there is a suitable recipient for the data. If there is not, then it is up to the application #1 to decide how to proceed. The application #1 could, for example, not bother sending the data, or possibly ask the user to insert another smart card 10, which will start the required application.

(v) If there is a suitable recipient then the sending application #1 sends the data to the receiving application #2 via the event manager 4904.

(vi) The event manager 4904 checks the message header to ensure that the sending application #1 has correctly filled out the data length and sender fields and then passes the message to the receiving application #2. If there is no such application #2 running, then the event manager 4904 discards the message and sends an error message back to the sending application #1.

13.8 Posting Data to a Message Board

This section describes the process of posting data to a common message board on the system 600 incorporating the software architecture 4900.

(i) The posting application 4920 gathers the data that it wishes to post on the message board.

(ii) The posting application 4920 sends the data to the event manager 4904 along with a short description of the data.

13.9 Retrieving Data From a Message Board

This section describes the process of retrieving data that has been previously been posted to the message board by another application on the system 600 incorporating the software architecture 4900.

(i) The requesting application #2 asks the event manager 4904 for a list of messages on the message board.

(ii) The event manager 4904 sends the application #2 the list of messages on the message board. This list will contain the short description of the data, the application identifier (Xid) for the application 4920 that posted the message to the message board and the message identifier for all messages on the message board.

(iii) The application #2 can then ask the event manager 4904 for a particular message by its message identifier, or the application #2 can request the list of all applications currently running from the launcher 4910.

(iv) If the application #2 has asked for the list of running applications the launcher 4910 will then send it to the application #2. This list will contain the application identifier (Xid) and the list of functions the corresponding application reported to the launcher 4910 that the corresponding application can perform.

(v) The requesting application #2 can then find all or some messages from the applications that perform the functions that it is looking for.

13.10 Removing Data From a Message Board

This section describes the process of removing data that has been previously posted to the message board by the same application, or another application on the system 600 incorporating the software architecture 4900.

(i) The requesting application #2 asks the event manager 4904 for a list of messages on the message board.

(ii) The event manager 4904 sends the application #2 the list of messages on the message board. This list will contain the short description of the data, the application identifier (Xid) of the posting application and the message identifier for all messages on the message board.

(iii) The application #2 can then ask the event manager 4904 to remove a particular message by specifying the specific message identifier.

13.11 Application Examples

Example A: Card Orderings

A number of potential application card orderings exist that may be implemented. The architecture 4900 places no restriction on which card ordering, or combination of card orderings is adopted for an application 4920.

Sequential card ordering in an service group, illustrated in Fig. 51A, requires that smart cards 10 for a particular set of applications to be inserted in a specified order. For example, card A followed by card B followed by card C, with removal and/or reinsertion following the same ordering.

Hierarchical card ordering in a service group and requires the cards for a particular set of applications to be inserted in a tree-like fashion as illustrated in Fig. 51B where if card A is inserted, only cards B or C may be then inserted. If card B is removed, card A must be reinserted. If card C is inserted, only card D may be inserted, and if card D is removed, only card C may be inserted.

A fully-meshed card ordering in a service group permits cards for a set of applications to be inserted and used in any order.

Example B: Pizza Ordering Service

With a prior art pizza ordering application, a number of choices for pizza type are presented (such as vegetarian, supreme and meat lovers), but no functionality is provided for customisation of the toppings or to make use of special offers.

An example set of applications that would make up a Joe's Pizzeria service group under the architecture 4900 could be as follows:

(i) Joe's Pizza Menu;

(ii) Topping Specialist;

(iii) Current Specials; and

(iv) Personal identifier.

Each of these applications can be made to work with the other applications to create a fully featured pizza ordering service. The Joe's Pizza Menu application provides a user interface that allows a customer to select a pizza type (vegetarian, supreme etc.), drinks (cola, lime etc.) and side orders (garlic bread, pasta, etc.). This application also keeps a shopping-basket style list of the current order, and provides buttons on the smart card for resetting the order, and completing the order.

The Topping Specialist application provides a user interface that allows a customer to move through a list of currently ordered pizzas, and to add/remove toppings to a selected pizza from a set of toppings printed on the surface of the card. The list of pizzas available is obtained from a running Joe's Pizza Menu application. Changes made to the toppings of a pizza will propagate back to the Joe's Pizza Menu application for modification of the pizza order.

The Current Specials application provides controls to navigate through a list of current special offers available from Joe's Pizzeria. Any specials selected are communicated to a Joe's Pizza Menu application for addition to an existing order.

The Personal identifier application provides a method of selectively communicating the home address and home phone number of the user to the Joe's Pizza Menu application depending on the details that a user wishes to supply.

Example C: Photo Lab Service

In prior art Photo Album and T-Shirt applications, a clipboard is shared (as a file) for communication of currently selected photographs. There is no facility however, for modification of a photograph (for example cropping, or increasing the brightness), or to have a number of linked cards that represent a full roll of film, with each card currently only containing a maximum of 20 photographs, each photograph being represented by an icon large enough to act as a button.

With the architecture 4900, a Photo Lab service may be designed that would have the following set of cards:

(i) Film_1a,

(ii) Film_1b;

(iii) T-Shirt printer; and

(iv) Photo Enhancer.

The Film_1a and Film_1b cards represent a complete roll of Advantix (trade mark of Kodak Corp. of USA) film containing 40 photographs each, and may be inserted with either card first. Once either card is inserted, access is provided to the complete set of photographs spanning both cards with direct access to photos that are printed on the surface of the inserted card. This means that a slideshow function would cycle through the photographs corresponding to both cards. Each card would also have buttons for adding a particular photograph reference to the service group clipboard for user with another application in the Photo Lab service group, and the application would also provide a function returning a reference to the photograph currently being viewed.

The T-Shirt printer application provides the ability to either instantly print a T-Shirt transfer using the most recently viewed photograph (a reference to which is obtained from the Film application), or to compose a T-Shirt transfer from the set of photos residing on the clipboard.

As part of a simple photo editing service, the Photo Enhancer application operates on the most recently viewed photograph (obtained either from the T-Shirt application, or the Film application - whichever was most recently in the foreground). The Photo Enhancer may provide such operations as automatic crop, sharpen, blur, lighten darken etc., with the changes able to be pushed back to the photo server and made permanent.

Example D: Video Email Service

Prior art video email applications provide a means to send video email messages to video email users appearing on the surface of the card. With some re-design it is possible to create a Video Email service according to the architecture 4900 in which an address book can be compiled of users that supply their smart card business cards to the owner of the address book. Applications forming the Video Email service are:

(i) Video Email Send;

(ii) Video Email Mailbox;

(iii) Video Email Address Book; and

(iv) Business Card.

The Video Email Send application operates in much the same way as the prior art application, with the exception that an address may be obtained from an inserted personal identification card, or an inserted Business Card.

The Video Email Mailbox application provides functions for retrieving video email messages from a remote server, and can also provide the address of senders for use as a reply address with the Video Email Send application.

Address book functionality is provided by the Video Email Address Book application. This application allows a user to build up a list of addresses from different Business Cards, personal identifier cards, or Video Email Mailbox cards that have been inserted. One or more entries from the list of addresses may be selected for use with a Video Email Send application.

Example E: Shopping Basket Service

With conventional software architectures, applications that provided online shopping needed to each maintain their own purchasing system, including a shopping basket, ordering, billing, and shipping means. A shopping basket service designed to make use of the features available as part of the architecture 4900 would allow these functions to be split out of each online shopping application, leaving more user interface area for other functions. Applications that would form part of such a Shopping Basket Service are:

(i) E-Deliver Shopping Basket;

(ii) Davy Jones Online; and

(iii) Pace Bros. Online.

The E-Deliver Shopping Basket application provides an overall shopping basket management facility, payment, and ordering facilities.

Davy Jones Online, and Pace Bros. Online applications provide facilities for browsing through a list of available items for purchase, with associated item descriptions, from corresponding department stores. When an item is found that a user wants to purchase, the item can be added to the shopping basket for future ordering and delivery by way of the E-Deliver Shopping Basket application.

It will be appreciated from the forgoing, that the architecture 4900 may be used to implement a card interface system that affords expanded flexibility through sectionalising management processes and through the judicious launching of applications.

This has permitted applications to be operated co-operatively to achieve a functional result. Further, such enables the various components of the architecture 4900 to be operated from hardware platforms of varying complexity through the capacity to operate procedures on platforms commensurate with their complexity. Such platforms range from low end set-top boxes with limited processing power, to home PC's, and remote server computers. Specifically, with a "dumb" set-top box, the card daemon 4902 would be run from within the set-top box and the balance of all processes from one or more remote server computers. Conversely, with a smart set-top box or home-style personal computer, all processes may be operated from within the one piece of hardware, excepting for where external communications via the network 220 is essential.

The architecture 4900 is also extensible to support security models appropriate to a particular application in order to protect both users and vendors from unauthorised data siphoning and fraud.

By virtue of the event manager 4904 acting as a conduit of event commands, the architecture is able to operate with applications developed over a range of versions of the communication protocol, as such would typically be developed over the course of time.

The architecture 4900 allows the card interface system 600 to continue to function even when card applications are not complying with expected modes of operation. This includes applications unexpectedly exiting, refusing to exit on command, and sending incorrect or excessive data to the system 600. The architecture 4900 supports multi-card applications by virtue of each card in the application belonging to the same service group, thereby ensuring that the application is maintained running when a card is removed and a new card inserted.

13.12 Application Management System

The architecture 4900 has been described above utilising the concept of service groups, their establishment, and their extinction, in order to permit multiple applications to operate simultaneously without overloading computing resources and ensuring adequate response.

An alternative approach in considering multiple applications arises from interpreting data flow between applications as being from producers of data to consumers of data. Fig. 55 shows a directed graph, with the graph direction flowing from consumers to producers for performing a collective function, in this case a T-shirt having a name and a photograph transferred to its surface, that data being derived from a number of other applications. The management of applications within such a graph structure depends upon the accessibility of nodes of the graph. Specifically, when a node becomes unreachable in the graph, the application at that node should be terminated, since, at that stage, that application is unable to perform a cognisant function. Further, links to a node should be removed when a consumer of that application's product de-registers for that service. When an application starts, the application is placed in the tree. If the application is a producer of a type that a consumer wants, the application is placed under that node in the tree.

As described above, the applications 4920 are referenced by their corresponding vendor identifier and application identifier which together are equivalent to the service identifier described above for the architecture 200. . The application identifier (or Xid) is used as a unique key for quick matching when starting-up an already running application. There are two application identifiers, the one stored on the card with the vendor identifier and the card identifier (A card identifier is equivalent to the service-specific identifier described above with reference to the architecture 200) , and one assigned by the system to applications when they start (the latter application identifier being referred to herein

also as the component identifier or Xid, the former application identifier being related to the service identifier as described above) .

Each application may register, using its Xid for identification, as a producer or consumer of a functionality on a needs basis. The application knows what it needs at a certain point in time by way of user interaction. For example, the user may navigate through the application to an "add photo" screen, at which point the application may register as a photo consumer. Registration in this regard is preferably be on the basis of a functionality, rather than a service group, as a service group approach would be too general for practical purposes. Further, such wouldn't allow an application to be linked to another in a consumer/producer relationship when the producer may not be able to provide the specific service that the producer requires unless all the applications in a service group support all functionality's offered by that service group.

Such a model presents two options for implementation, since an application may require two or more functions from any other applications:

1. Each node in the graph has only one connection to any other node. This means that the connection must also contain a list of the service included in the consumer/producer relationship. Each time a consumer de-registers for a service the list entry is removed. When the list of services for a connection becomes empty, the connection is removed. When a connection is removed, any producer that is linked by that connection is also checked. If the producer node is no longer connection to any other, that node may be removed.

2. This option is similar to (1) above except that instead of keeping a list of services, each specific service is a separate connection between the consumer/producer node. Thus, there may be multiple connections between two

applications. When a consumer de-registers for a service, that connection is removed. If the producer is no longer connected, the consumer is terminated.

Such proposals are problematic in that each allows the application associated with the smart card presently in the reader 1 to be terminated by an event other than a specific user action. This may be confusing from the user point of view. An alternative approach to termination of an application is therefore desired.

In such an alternative approach, the architecture 4900 may be operated without specific dependence upon any application 4920 being a member of a specific service group as described above, but through the transient formation of what is referred to herein as a "dominant" service group. A dominant service group arises from any transient functional relationship between two or more current applications being determined from whether any application 4920 is classed as either a producer, a consumer, both a producer and consumer, or neither a producer nor a consumer.

Such a management system for the applications 4920 revolves around the concept of the "dominant" service group being formed when a producer/consumer pair of applications, or a single application where that application meet both criteria, in the same service group are registered. For example simultaneous operation of applications Ac and Ap will cause service group A to be dominant and satisfies a producer-consumer pair, whereas AcBp or ApBc whilst satisfying a producer-consumer pair, will not create a dominant service group. According to the management system, when a dominant service group is formed, all applications not sharing that group are terminated. The dominant service group may exist in conjunction with a second dominant service group, provide both are registered simultaneously. For example, if Application#1 starts and registers ApBp and Application#2 starts and registers AcBc, A and B are then dominant. For two or more dominant service groups to exist, they must be formed when a new application

starting registers for each group establishing a producer-consumer pair. A producer/consumer pair of applications forming a service group registered after a dominant service group becomes a "subsidiary" of the dominant group. A subsidiary group of a subsidiary group may also be formed. A subsidiary of a subsidiary is formed when a producer of the subsidiary that was already registered as a consumer for the second subsidiary.

The net effect of such a management structure is the creation, and subsequent dismantling, of a tree or graph of interacting applications that pass data there between to achieve a final result desired by the user. Specifically, such a result may not be readily apparent from on the face of the applications being utilised, in contrast to Example B above for the pizza ordering service. This application management structure is best described with reference to the examples below.

The examples below make reference to a number of applications, details of which are described in Table 4 below.

Table 4

Card Application Name	**Description**	**Service Group Member** (p=producer, c=consumer, n=neither)
ID1	Identification detail card	Zp Cp
ID2	Identification detail card	Zp Qp
PhotoID	photograph identification card	Zp Qp Ap
Photo1	photograph card	Ap Fp
Photo2	photograph card	Mp Ap

PIN	personal identification number card	Pp
Bank	electronic banking card	Bn
Pizza	pizza ordering card	Rn
T-shirt	T-shirt manufacture	Tp
CardMaker	card used for making other cards	Sp

Example F:

In this example, it is desired by the user to create a greeting card having the recipient's name, a standard message, and a photograph on the card. A first step using the cards of Table 4 would be for the user to insert the CardMaker application card into the reader 1. Such an action commences that application and registers that application as a consumer of service groups A and Z. Applications may dynamically change their service group membership. For example, CardMaker may start and present the user with a screen display asking if the user wants to make a card identical to the card created on a previous occasion. Upon answering "NO", CardMaker registers as a consumer for ID1 and Photo1 since a new card will be made. A process tree for this stage appears as shown in Fig. 53A. Next, the user knows that a photograph is required, and provides that photograph by removing the CardMaker application and by inserting the Photo1 application. The CardMaker application remains in operation upon removal from the reader 1 since, its processes have yet to perform a function. The insertion of Photo1 application crates a dominant service group in Ac and Ap as illustrated, meaning that the CardMaker application requires a photograph and the Photo1 application can supply that photograph. The Photo1 application, requires a PIN to access the photograph and the arrangement is thus as represented in Fig. 53B. Not all photographs on the Photo1 card may require a

PIN to unlock them for use, so Photo1 only registers as Pc when it requires a PIN to proceed, such as in the present case. The PIN card is then provided according to Fig. 53C. As seen from Fig. 53C, a second producer-consumer pair is formed, and in this case the provision of the PIN, allows the Photo1 card to supply the photograph selected by the user to the CardMaker application. Those tasks having been completed, the left branch of the process tree is extinguished and those corresponding "performed" applications de-register from the event manger 4904, as shown in Fig. 53D. The next step to complete the process is to insert a card having the desired name, which in this case comes from the application ID1 as shown in Fig. 53E. This application supplies the required name and the CardMaker application is thus satisfied, thereby permitting all other applications to de-register and terminate. The CardMaker application can then output the required card without interaction with any other application.

In an alternative approach, the PIN application may be required to access both the photograph and the name. As such, the PIN application card need only be inserted the once only if the PIN for both photo cards is the same, and a process tree such as that shown in Fig. 54 may be formed. In this example PhotoID and Photo1 are used since PhotoID may have a picture of the recipient of the card being made, and Photo1 may have an attractive background picture to place over the photo.

Fig. 54 demonstrates that multiple links to nodes in the process tree are permitted, and that applications on unreachable nodes (being those with no links) are terminated.

Preferably, an upper limit on running applications is set to be seven (7). If this number is exceeded, termination of applications commences with the oldest leaf application in the process tree.

The foregoing describes only some arrangements and variations on those arrangements of the present invention, and modifications and/or changes can be made thereto without departing from the scope and spirit of the invention, the embodiments being illustrative and not restrictive.

The claims defining the invention are as follows:

1. A read device for reading an interface card, said card being configured for insertion into said read device, and wherein said card comprises indicia formed thereon and a memory having data stored therein for communicating with an external device, said read device comprising:

a substantially transparent touch sensitive membrane arranged to overlay said interface card upon receipt of said card in said read device,

a central processing unit for sending a service identifier, and a specific portion of said data to said external device, said specific data being related to a user selected indicia, wherein said external device provides a service identified by the service identifier upon receipt of said data.

2. A read device according to claim 1, wherein a service identifier is set by a vendor for use by an application.

3. A read device according to claim 2, wherein said service identifier is assigned to said vendor by a central authority.

4. A read device according to claim 1, wherein the central processing unit sends a read device identifier, read from the read device, to said external device.

5. A read device according to claim 1, wherein the central processing unit sends an insert message to the external device when said card is inserted into said read device.

6. A read device according to claim 1, wherein the central processing unit sends a remove message to said external device when said card is removed from said read device.

7. A read device according to claim 1, wherein the central processing unit sends a press message to the external device when at least one of said indicia is pressed via said touch sensitive membrane.

8. A read device according to claim 1, wherein the central processing unit sends a release message to the external device upon release of said at least one of said pressed indicia.

9. A read device according to claim 1, wherein the central processing unit sends a move message upon a press position being moved without being released.

10. A read device according to claim 1, wherein the central processing unit sends a bad card message when said an invalid card is inserted into said read device.

11. A read device according to claim 1, wherein the central processing unit sends a low battery message when a charge level of a battery powering said read device reaches a predetermined threshold.

12. A read device according to claim 1, wherein coordinates of a users press position on said touch sensitive membrane are sent to said read device.

13. A read device according to claim 12, wherein coordinates are calculated by an application which was provided by a vendor.

14. A read device according to claim 1, wherein the card stores said service identifier and said data in a memory chip.

15. A read device according to claim 1, wherein the external device is a set-top-box.

16. A read device according to claim 1, wherein the external device is a personal computer.

17. A program to be executed in a read device having a receptacle to receive an interface card , said card comprising a substrate and indicia formed on said substrate, said program comprising:

code for reading a service identifier and a specific portion of data stored within a memory of said card, said specific data being related to a user selected indicia; and

code for a sending said service identifier and said specific data to an external device, whereby a service is provided by said external device, said service being identified by the service identifier.

18. A program according to claim 17, wherein the program is stored on a computer-readable medium in said read device.

19. A data processing method for a read device that has a receptacle to receive an interface card, said card comprising a substrate with indicia formed thereon, said method comprising the steps of:

reading a service identifier and a specific portion of data stored within a memory of said card, said specific data being related to a user selected indicia; and

sending said service identifier and said specific data to an external device, whereby a service is provided by said external device, said service being identified by the service identifier.

20. A read device for reading an interface card, said card comprising indicia formed thereon and a memory having data stored therein, and wherein said card is configured for insertion into said read device, said read device comprising:

read means for reading a service identifier and a specific portion of data stored in the card, said specific data being related to user selected indicia; and

send means for sending said service identifier and said data to an external device, wherein said external device provides a service, said service being identified by the service identifier.

21. A card interface system comprising an interface card and a read device, said card being configured for insertion into said read device, and said card comprising at least a substrate with indicia formed thereon, said read device comprising a receptacle to receive said interface card said system comprising:

a memory on said card for storing a service identifier and data related to a user selected indicia; and

a central processing unit integrally formed within said read device for sending said service identifier and said data to an external device in order that a user of said card receives a service from said external device, said service being identified by the service identifier.

22. An electronic card reader for reading an electronic card, said electronic card having at least one indicia formed thereon and an electronic memory having data stored therein for controlling data controlled equipment, said electronic reader comprising:

a touch sensitive substantially transparent membrane having an upper surface configured to be depressible by a user of said reader;

a receptacle shaped to receive said electronic card, wherein said electronic card received therein and said indicia can be viewed through said touch sensitive membrane; and

an electronic circuit coupled to said membrane to read a specific portion of said data from said memory according to depression of said membrane associated with a specific one of said indicia, and for sending said specific data together with a service identifier to said data controlled equipment, said data controlled equipment providing a function controlled by receipt of said specific data, wherein said function is associated with said service identifier.

23. A read device for a interface card, said data comprising a substrate having at least one indicia formed thereon, said read device comprising:

a receptacle shaped to receive said interface card;

a substantially transparent touch sensitive membrane arranged to overlay said interface card upon receipt of said interface card in said receptacle such that said indicia can be viewed through said touch sensitive membrane; and

an electronic circuit for coupling to a memory component of said interface card and for reading data related to one of said indicia selected by a user of said read device via said membrane, wherein said electronic circuit is configured to send said data together with a service identifier to an external device, whereby said external device provides a service associated with said service identifier upon receipt of said read data.

24. A read device having a receptacle to receive an interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for determining a validity of said card that was inserted into said read device and sending to an external device a service identifier and data stored in the card, said data being related to a user pressed indicia, in order that a card user receives a service identified by the service identifier according to said pressed indicia.

25. A read device according to claim 24, wherein the service is identified by an application that is executed in the external device.

26. A read device having a substantially transparent touch sensitive membrane arranged to overlay a detachable interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for determining a validity of said card that was inserted into said read device and sending to an external device a service identifier and user press coordinates pressed on said touch sensitive membrane to select said indicia, in order that a card user receives a service identified by the service identifier according to said pressed indicia.

27. A read device according to claim 26, wherein the service is identified by an application that is executed in the external device.

28. A read device having a substantially transparent touch sensitive membrane arranged to overlay a detachable interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for distinguishing a identifier stored in said card and sending to an external device a service identifier and user press coordinates pressed on said touch sensitive membrane to select said indicia or a service identifier and user press coordinates pressed on said touch sensitive membrane to select said indicia, based on said result of said distinction, in order that a card user receives a service identified by the service identifier according to said pressed indicia.

29. A read device according to claim 28, wherein the service is identified by an application that is executed in the external device.

30. A read device having a substantially transparent touch sensitive membrane arranged to overlay a detachable interface card which comprises a substrate and indicia formed on said substrate, said device comprising:

a central processing unit for detecting a user press touch on said on said touch sensitive membrane and sending a touch coordinates of said user press touch to an external device, wherein said touch coordinates is used as a cursor information.

31. A read device having a receptacle to receive an interface card , said device comprising:

a central processing unit for reading an information that affects functions that said card perform in said read device, performing the functions based on said information.

32. A read device having a receptacle to receive an interface card , said device comprising:

a central processing unit for generating a session identifier identifying a current session of a card insertion, which is a number that is incremented each time a card is inserted into said read device and sending said session identifier to a external device, in order to determining a validity of said inserted card in said external device.

33. A read device for reading a user interface card, said card being configured for insertion into said read device, and wherein said card comprises a plurality of indicia formed thereon and a memory having stored therein a service identifier for communicating to an external device to provide an identified service, said read device comprising:

a receptacle for receiving said card and providing visibility of said plurality of indicia upon receipt of said card in said read device;

a selection mechanism for allowing a user to select at least one of said plurality of visible indicia;

a central processing unit coupled to said selection mechanism for generating data related to said user selected indicia and for reading said service identifier from said memory; and

a transmission unit for sending said indicia related data and said service identifier to said external device, wherein upon receipt of said data and service identifier said external device provides a service identified by the service identifier and said data.

34. A read device as claimed in claim 33, wherein the generating of data related to said user selected indicia includes reading data associated with said selected indicia previously stored in said memory.

35. A read device for reading a user interface card, said card being configured for insertion into said read device, and wherein said card comprises a plurality of indicia formed thereon , said read device comprising:

a receptacle for receiving said card and providing visibility of said plurality of indicia upon receipt of said card in said read device;

a selection mechanism for allowing a user to select at least one of said plurality of visible indicia;

a central processing unit coupled to said selection mechanism for generating data related to said user selected indicia and for reading from a memory of said card a service identifier for communicating to an external device to provide a service ; and

a transmission unit for sending said indicia related data and said service identifier to said external device, wherein upon receipt of said data and service identifier said external device provides a service identified by the service identifier and said data.

36. A read device as claimed in claim 35, wherein the generating of data related to said user selected indicia includes reading data associated with said selected indicia previously stored in said memory.

5

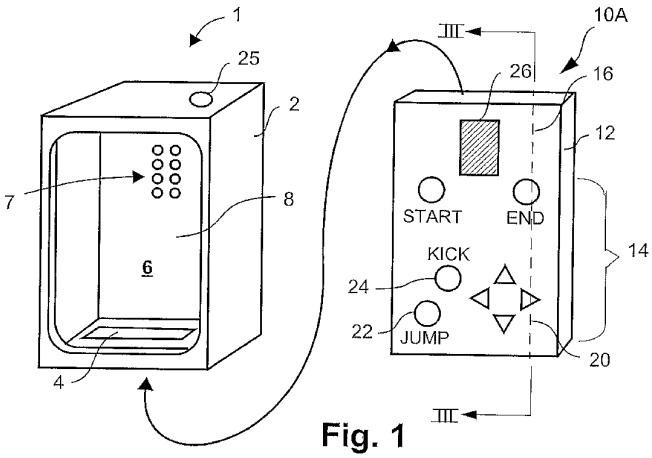

1
25
2
7
8
6
4
III
10A
26
16
12
START
END
KICK
14
24
22
JUMP
20
III

Fig. 1

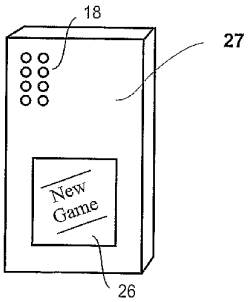

18
27
New Game
26

Fig. 2

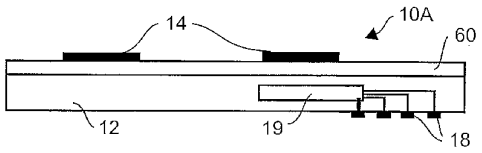


Fig. 3

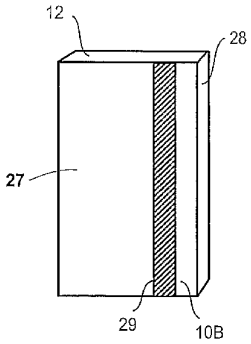


Fig. 4

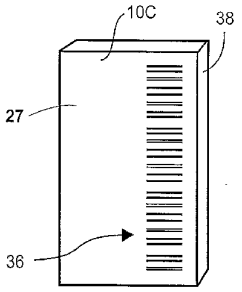


Fig. 5

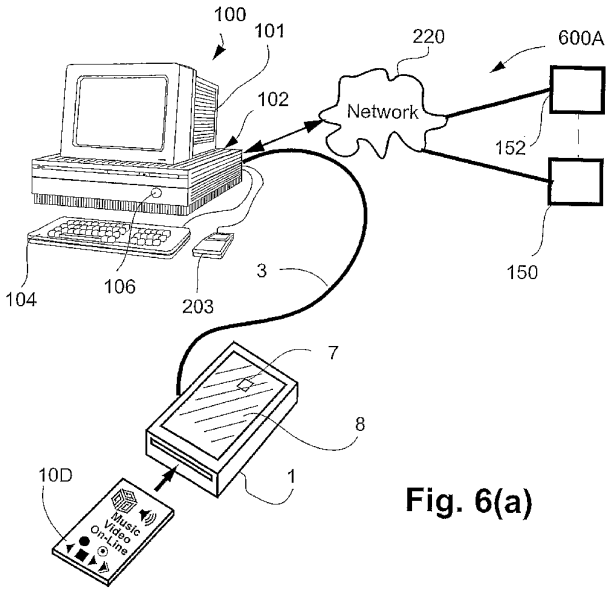

100
101
102
220
600A
Network
152
150
104
106
203
3
7
8
1
10D
Music
Video
On-Line

Fig. 6(a)

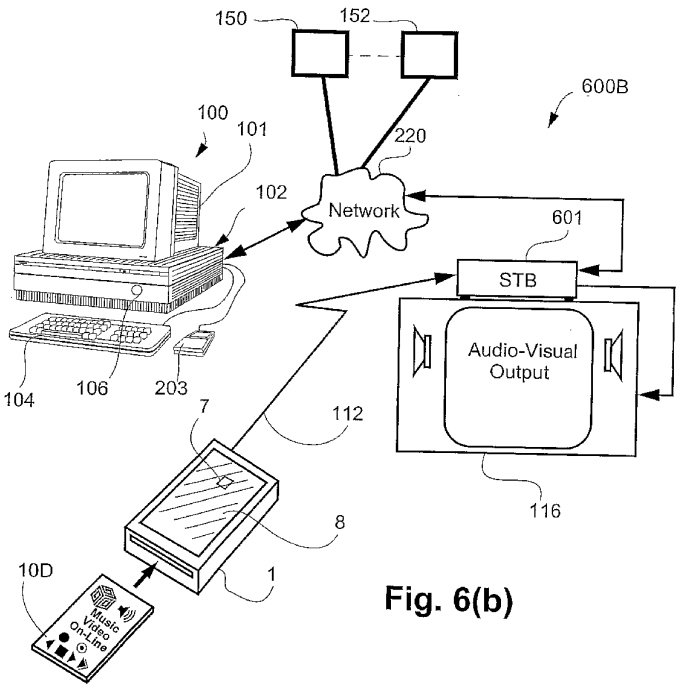

150
152
600B
100
101
220
102
Network
601
STB
Audio-Visual
Output
104
106
203
7
112
116
8
10D
1
Music
Video
On Line

Fig. 6(b)

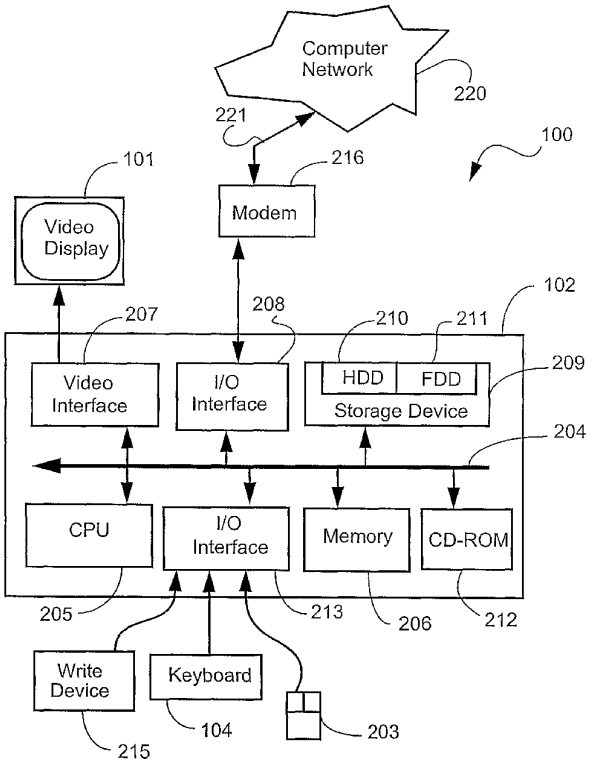

Computer Network
220
221
100
101
216
Modem
Video Display
102
207
208
210
211
209
Video Interface
I/O Interface
HDD
FDD
Storage Device
204
CPU
I/O Interface
Memory
CD-ROM
205
213
206
212
Write Device
Keyboard
104
203
215

Fig. 7

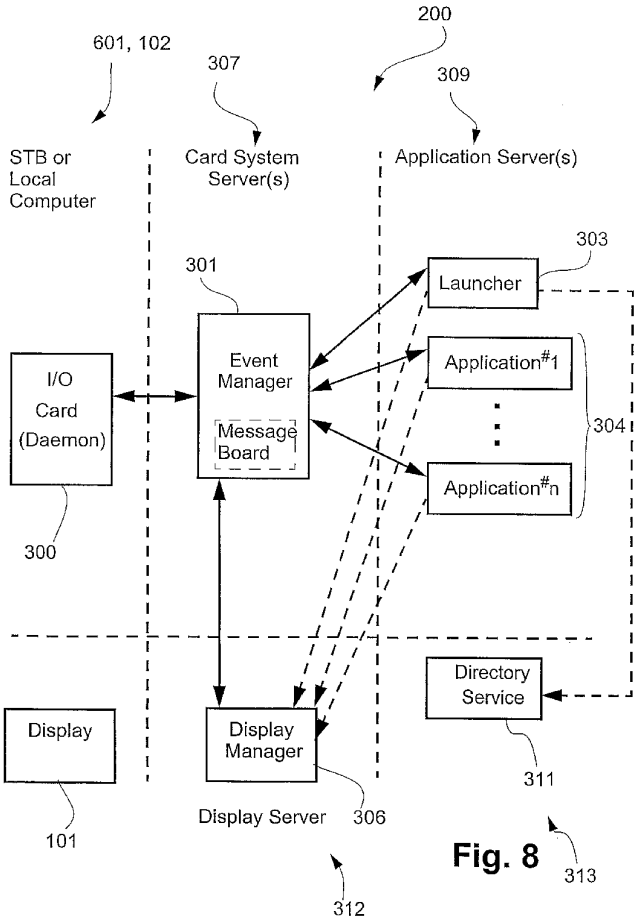

200
601, 102
307
309
STB or Local Computer
Card System Server(s)
Application Server(s)
303
301
Launcher
Event Manager
Application#1
I/O Card (Daemon)
Message Board
304
Application#n
300
Directory Service
Display
Display Manager
311
Display Server
306
101
312
313

Fig. 8

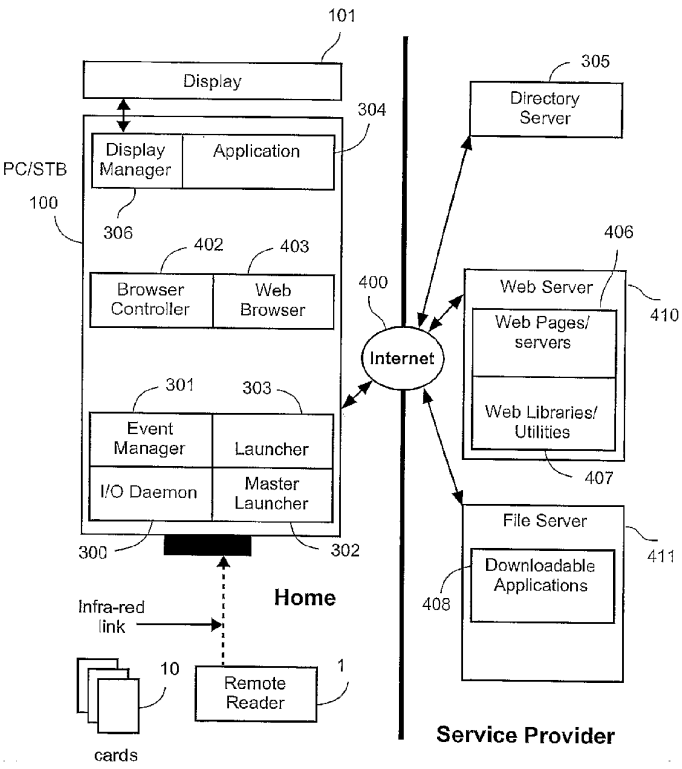

101
Display
304
Display Manager
Application
PC/STB
100
306
402
403
Browser Controller
Web Browser
400
301
303
Event Manager
Launcher
I/O Daemon
Master Launcher
300
302
Home
Infra-red link
10
Remote Reader
1
cards
305
Directory Server
406
Web Server
410
Web Pages/ servers
Web Libraries/ Utilities
407
Internet
File Server
411
Downloadable Applications
408
Service Provider

Fig. 9

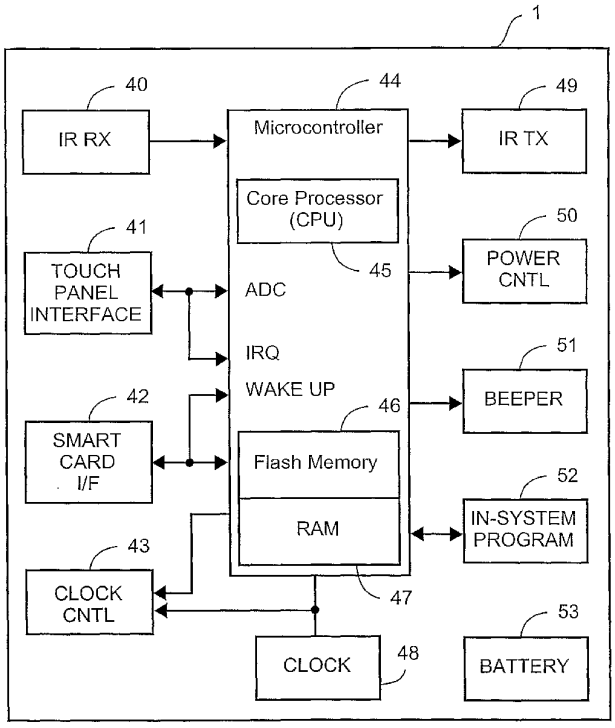

1
40
IR RX
44
Microcontroller
Core Processor (CPU)
45
ADC
IRQ
WAKE UP
46
Flash Memory
RAM
47
49
IR TX
50
POWER CNTL
51
BEEPER
52
IN-SYSTEM PROGRAM
53
BATTERY
41
TOUCH PANEL INTERFACE
42
SMART CARD I/F
43
CLOCK CNTL
CLOCK
48

Fig. 10

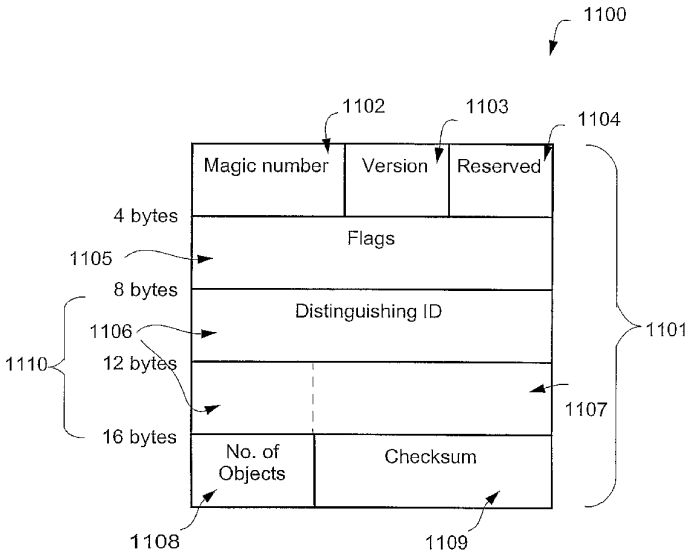

1100
1102
1103
1104
Magic number
Version
Reserved
4 bytes
Flags
1105
8 bytes
Distinguishing ID
1106
1110
12 bytes
1101
1107
16 bytes
No. of Objects
Checksum
1108
1109

Fig. 11

Field Number	Description (Card Header)
Magic Number	Two byte magic number. A constant that specifies this as being a valid card. Currently defined as the ASCII value for 'i' followed by the ASCII value for 'C'.
Version	One byte version number. Each version increment specifies a change in the card layout that can not be read by a reader that is compatible with lower versions of the layout. This document describes version 1(0x01) of the card format.
Reserved	This data is reserved for future use. Its value must be set to zero.
Flags	Four bytes of flags for this card. (See Fig. 13.) All non-assigned bits must be zero.
Distinguishing ID	Eight byte distinguishing identifier. Distinguishing identifiers include two fields - service identifier and service-specific identifier. The service identifier is five bytes and identifies the service associated with the card. The service-specific identifier is three bytes of service-specific value.
Number of Objects	One byte. The number of objects following this header. Can be zero.
Checksum	Card checksum, 2 bytes. The card checksum is sixteen bit, unsigned integer sum of all data bytes on the card excluding the checksum.

Fig. 12

Name	Description (Pre-Card Flag Values)	Value (hex)
Don't Beep	Stops the reader unit providing audio feedback by default. If this bit is set the reader will not issue any audio feedback when a UI element is pressed unless that element has the "INVERT BEEP" flag set in the UI Element object	0x0000 0001
No MOVE Events	Stops the reader unit from acting as a mouse when the user moves their finger around on the reader surface	0x0000 0002
No Event Co-ordinates	Stops the reader unit from sending co-ordinates for PRESS, RELEASE and MOVE events. X and Y values are sent with value zero.	0x0000 0004

Fig. 13

Name	Description (Object Structure)	Length
Type	The type of object (see Fig. 16).	1 byte
Object Flags	The general object flags that are associated with this object (see Fig. 15). Note: Additional flags specific to an object type are specified within the data field of the object.	1 byte
Length	The length of the data following this object. This value can be zero.	2 bytes
Data	The data associated with this object. The structure of this data is dependent on the type of object.	Variable

Fig. 14

Name	Description (Pre-Object Flag Values)	Value (hex)
Inactive	Indicates to the reader that the object is valid but is to be ignored regardless of it's type.	0x01

Fig. 15

Name	Description (Object Types)	Value (hex)
UI Object	A UICard button.	0x10
Card Data	Contains data that relates specifically to this card.	0x20
Fixed Length Data	An object that can be used to store fixed length blocks of data on the card.	0x30
Reader Insert	An object that can be used to give instructions to the reader when the card is inserted.	0x40
No Operation	An object that is used to fill blocks of empty space on the card.	0x01
No Operation (Single byte)	A single byte object that doesn't have a standard object header. Used to fill spaces on the card that are too small for a normal object header.	0x00

Fig. 16

Field	Description (User Interface Object Structure)	Size
Flags	Flags specific to this UI element on the card.	1 byte
X1	X value of the bottom-left hand corner co-ordinate of this object's rectangle.	1 byte
Y1	Y value of the bottom-left hand corner co-ordinate of this object's rectangle.	1 byte
X2	X value of the top-right hand corner co-ordinate of this object's rectangle.	1 byte
Y2	Y value of the top-right hand corner co-ordinate of this object's rectangle.	1 byte
Data	Zero or more bytes of data associated with this object. The size of this field is determined by the object data size minus the combined size of the above fields.	Variable

Fig. 17

Name	Description (Flags for UI Object)	Value
Invert Beep Enable	This flag causes this button to have the inverse of the don't beep flag in the card header. If the Don't Beep flag isn't set in the header, this flag causes this button not to beep and vice versa.	0x01
Auto-repeats	Messages associated with this button automatically repeat when the press is held on the button.	0x02
Don't Send Data on Press	This causes this button not to send the data associated with this button in the press event. The default is to send the data associated with the button in the press event.	0x04
Don't Send Data on Release	This causes this button not to send the data associated with this button in the release event. The default is to send the data associated with the button in the release event.	0x0a

Fig. 18

Field	Description (Message Header Format)	Bytes
Preamble	Preamble to the message. Value is always 0xAA 0x55 (bit sequence 10101010 01010101). This is to make it easier for the EM to find the beginning of a message.	2
Version	The version of the UICard IR message protocol this messages uses. This version of the protocol is version 1(0x01 in the version field.)	1
Type	Type of message. This is one of the values given in Fig. 20	1
Reader ID	The 16 bit id of the reader that sent the message. This number is a pseudorandom generated number that is changed when the battery is replaced in the reader. This is needed to distinguish readers when multiple readers are being used with applications.	2
Service	Service identifier as stored on the card.	5
Service-specific	Service-specific identifier as stored on the card.	3

Fig. 19

Name	Description (Message Type Codes)	Code
INSERT	A card has been inserted into the reader.	'I'
REMOVE	The card has been removed from the reader.	'E'
PRESS	The touch panel has been pressed.	'P'
RELEASE	The press on the touch panel has been released.	'R'
MOVE	The press position has moved but the press has not been released.	'M'
BADCARD	A card had been inserted however it has not passed validation	'B'
LOW_BATT	The battery in the reader is getting flat.	'L'

Fig. 20

Field	Description (Simple Message Format)	Bytes
Header	Message header as defined by Fig. 19.	14
Checksum	Message checksum. This is the sum of all the bytes in the message.	1
Checksum'	The 1's complement of the checksum.	1

Fig. 21

Field	Description (INSERT Message Format)	Bytes
Header	Message header as defined by Fig. 19.	14
Length	The number of bytes of data. Can be zero.	2
Data	The data from a Card Data object on the card.	Length
Checksum	Message checksum. This is the sum of all the bytes in the message.	1
Checksum'	The 1's complement of the checksum.	1

Fig. 21(a)

Field	Description (Move Message Format)	Bytes
Header	Message header as defined by Fig. 19.	14
X	The X co-ordinate of the touch position.	1
Y	The Y co-ordinate of the touch position.	1
Checksum	Message checksum. This is the sum of all the bytes in the message.	1
Checksum'	The 1's complement of the checksum.	1

Fig. 22

Field	Description (Press and Release Message Format)	Bytes
Header	Message header as defined by Fig. 19.	14
X	The X co-ordinate of the touch position.	1
Y	The Y co-ordinate of the touch position.	1
Length	The number of bytes of data. Can be zero.	2
Data	The data associated with the user interface element.	Length
Checksum	Message checksum. This is the sum of all the bytes in the message.	1
Checksum'	The 1's complement of the checksum.	1

Fig. 23

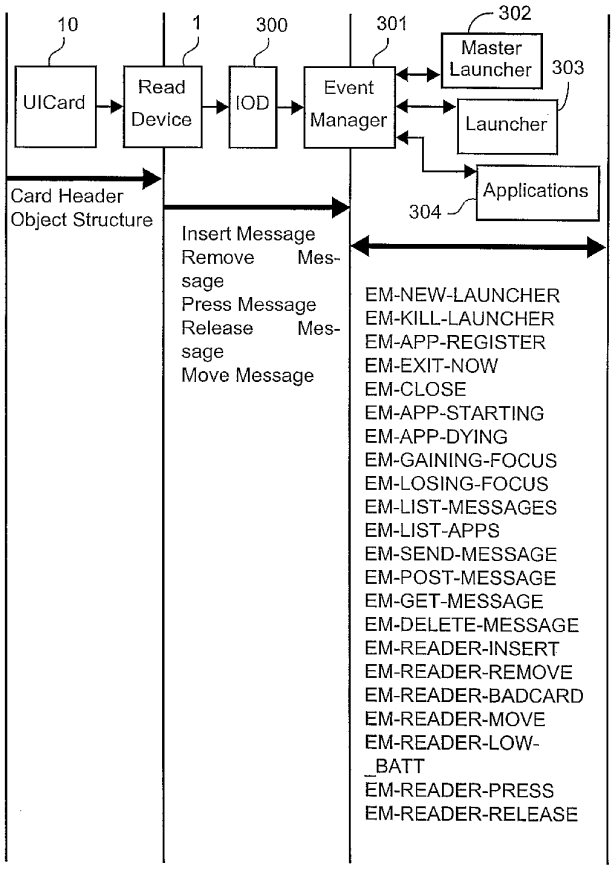

10
1
300
301
302
Master Launcher
303
UICard
Read Device
IOD
Event Manager
Launcher
Applications
304
Card Header Object Structure
Insert Message
Remove Message
Press Message
Release Message
Move Message
EM-NEW-LAUNCHER
EM-KILL-LAUNCHER
EM-APP-REGISTER
EM-EXIT-NOW
EM-CLOSE
EM-APP-STARTING
EM-APP-DYING
EM-GAINING-FOCUS
EM-LOSING-FOCUS
EM-LIST-MESSAGES
EM-LIST-APPS
EM-SEND-MESSAGE
EM-POST-MESSAGE
EM-GET-MESSAGE
EM-DELETE-MESSAGE
EM-READER-INSERT
EM-READER-REMOVE
EM-READER-BADCARD
EM-READER-MOVE
EM-READER-LOW-_BATT
EM-READER-PRESS
EM-READER-RELEASE

Fig. 24

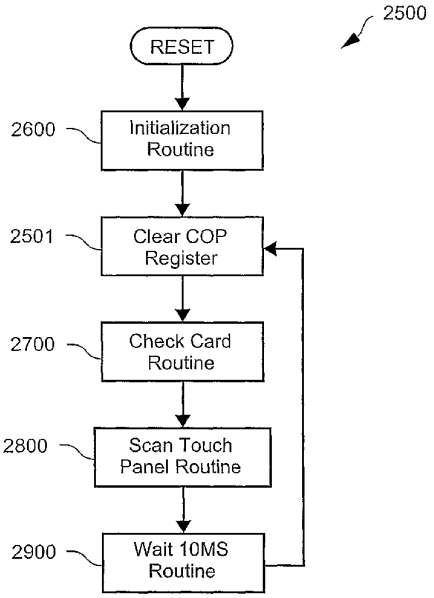

2500
RESET
2600
Initialization Routine
2501
Clear COP Register
2700
Check Card Routine
2800
Scan Touch Panel Routine
2900
Wait 10MS Routine

Fig. 25

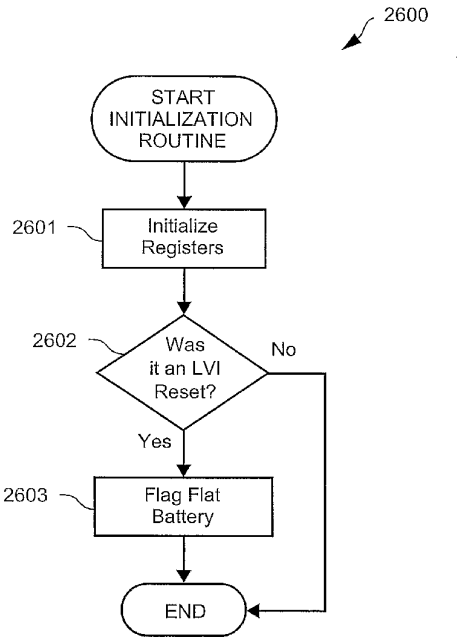

2600
START INITIALIZATION ROUTINE
2601
Initialize Registers
2602
Was it an LVI Reset?
No
Yes
2603
Flag Flat Battery
END

Fig. 26

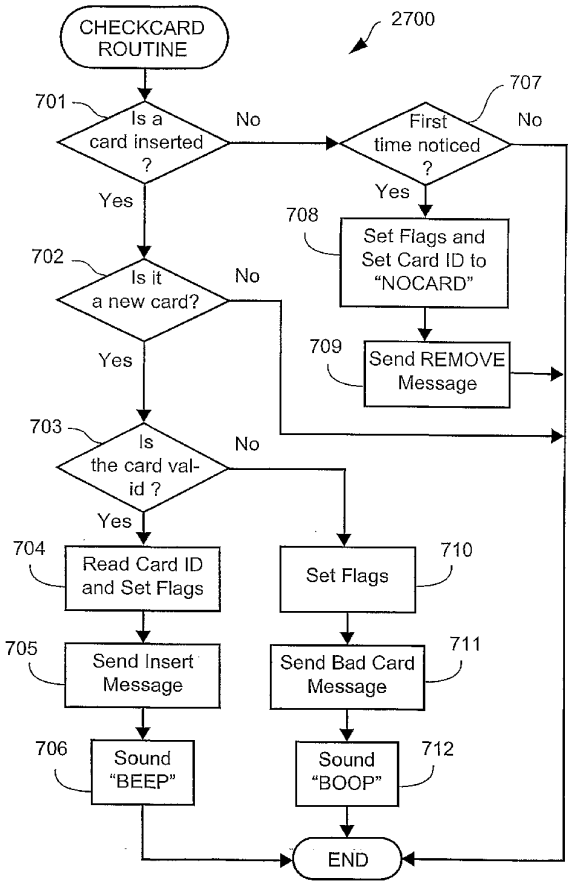

CHECKCARD ROUTINE
2700
701
Is a card inserted ?
No
707
First time noticed ?
No
Yes
Yes
708
Set Flags and Set Card ID to "NOCARD"
702
Is it a new card?
No
Yes
709
Send REMOVE Message
703
Is the card valid ?
No
Yes
704
Read Card ID and Set Flags
710
Set Flags
705
Send Insert Message
711
Send Bad Card Message
706
Sound "BEEP"
712
Sound "BOOP"
END

Fig. 27

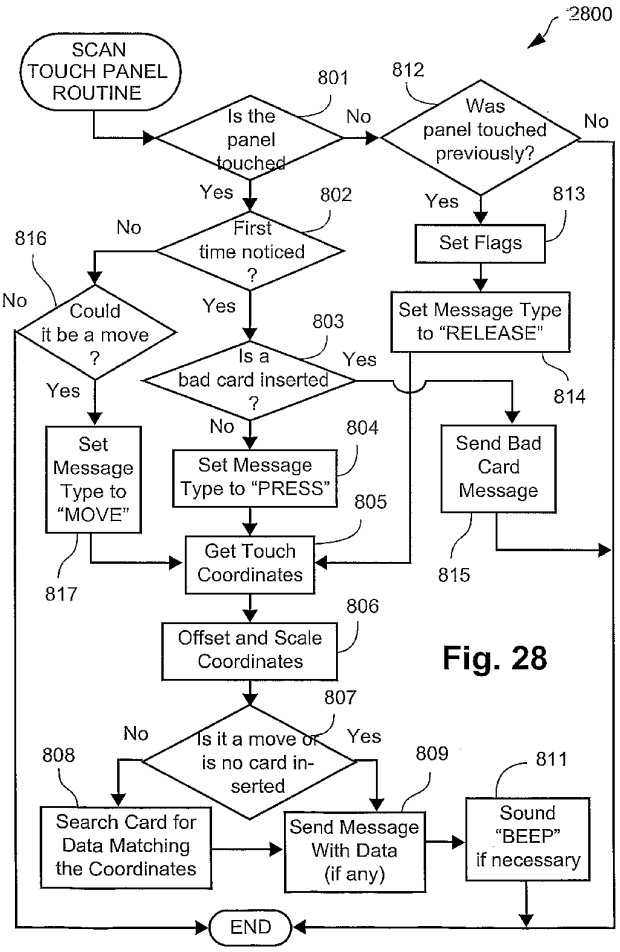

2800
SCAN TOUCH PANEL ROUTINE
801
Is the panel touched
No
812
Was panel touched previously?
No
Yes
802
First time noticed ?
No
816
No
Could it be a move ?
Yes
Yes
803
Is a bad card inserted ?
Yes
No
804
Set Message Type to "PRESS"
Set Message Type to "MOVE"
817
805
Get Touch Coordinates
806
Offset and Scale Coordinates
807
No
Is it a move or is no card in-serted
Yes
808
Search Card for Data Matching the Coordinates
809
Send Message With Data (if any)
811
Sound "BEEP" if necessary
END
Yes
813
Set Flags
Set Message Type to "RELEASE"
814
Send Bad Card Message
815

Fig. 28

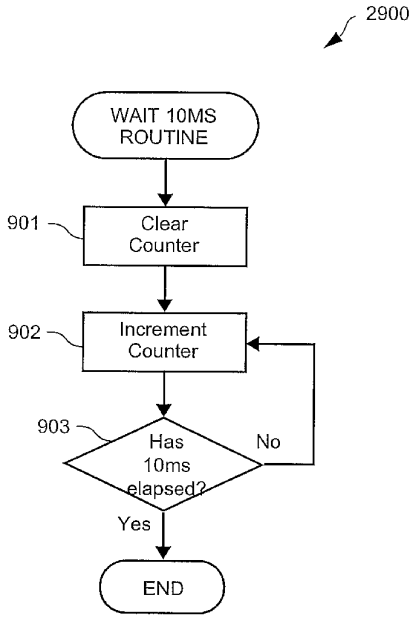

2900
WAIT 10MS ROUTINE
901
Clear Counter
902
Increment Counter
903
Has 10ms elapsed?
No
Yes
END

Fig. 29

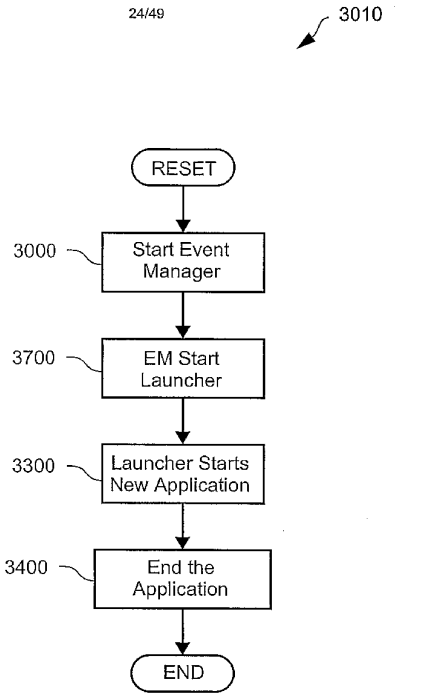

3010
RESET
3000
Start Event Manager
3700
EM Start Launcher
3300
Launcher Starts New Application
3400
End the Application
END

Fig. 30

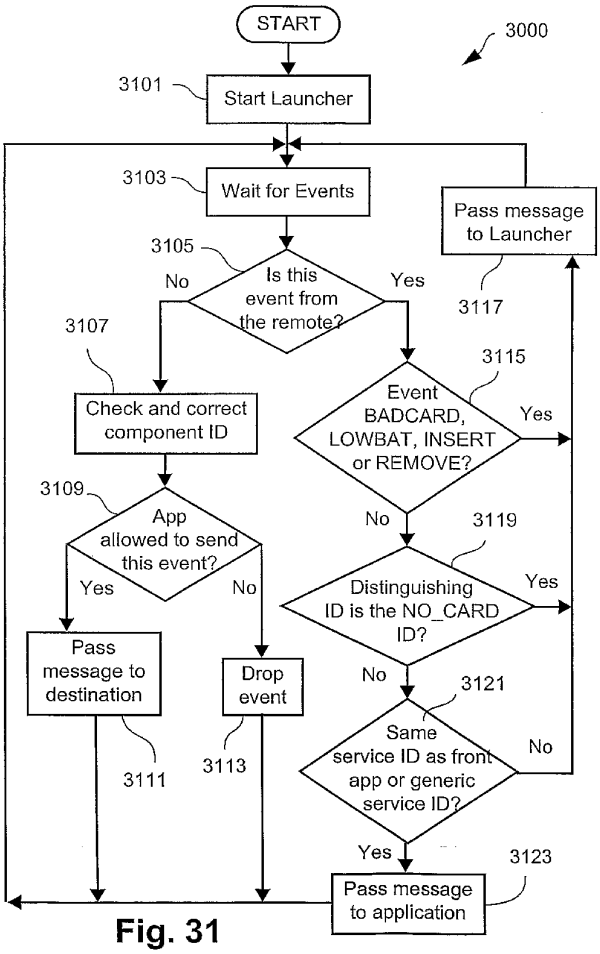

START
3000
3101
Start Launcher
3103
Wait for Events
Pass message to Launcher
3117
3105
Is this event from the remote?
No
Yes
3107
Check and correct component ID
3115
Event BADCARD, LOWBAT, INSERT or REMOVE?
Yes
3109
App allowed to send this event?
Yes
No
No
3119
Distinguishing ID is the NO_CARD ID?
Yes
Pass message to destination
Drop event
No
3121
Same service ID as front app or generic service ID?
No
3111
3113
Yes
3123
Pass message to application

Fig. 31

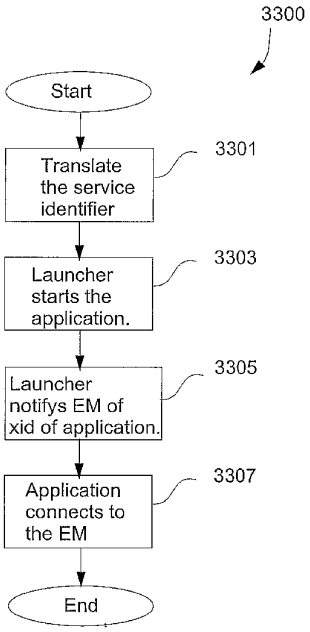

3300
Start
Translate the service identifier
3301
Launcher starts the application.
3303
Launcher notifys EM of xid of application.
3305
Application connects to the EM
3307
End

Fig. 32

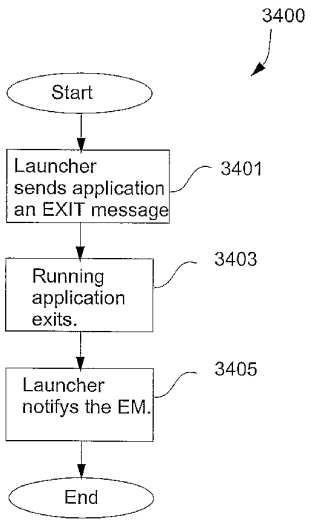

3400
Start
Launcher sends application an EXIT message
3401
Running application exits.
3403
Launcher notifys the EM.
3405
End

Fig. 33

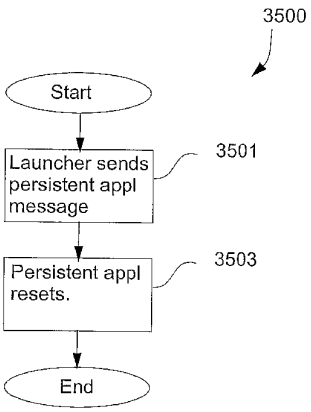

3500
Start
Launcher sends persistent appl message
3501
Persistent appl resets.
3503
End

Fig. 34

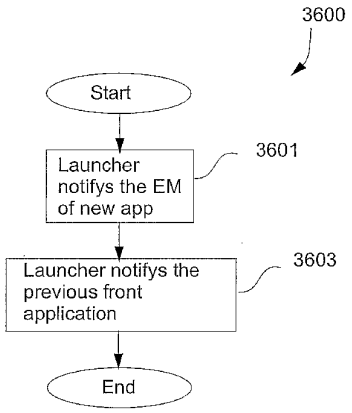

3600
Start
Launcher notifys the EM of new app
3601
Launcher notifys the previous front application
3603
End

Fig. 35

Fig. 36

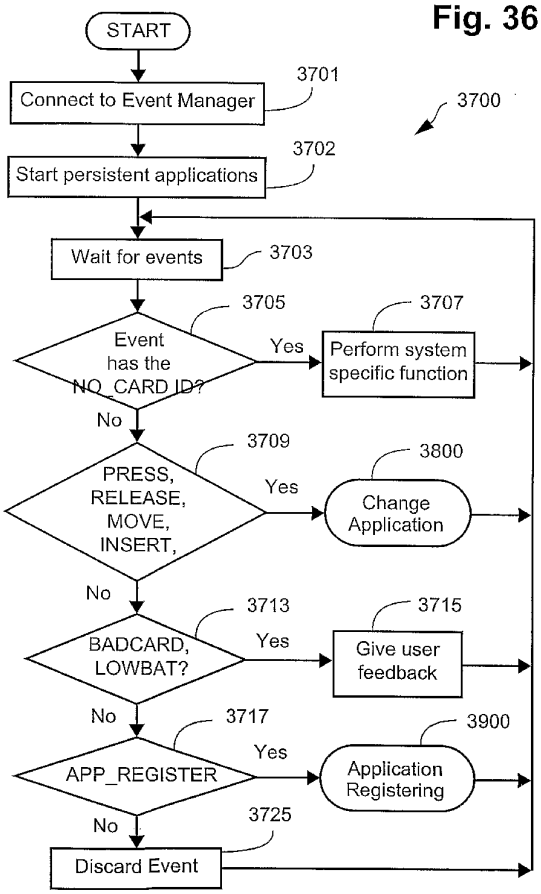
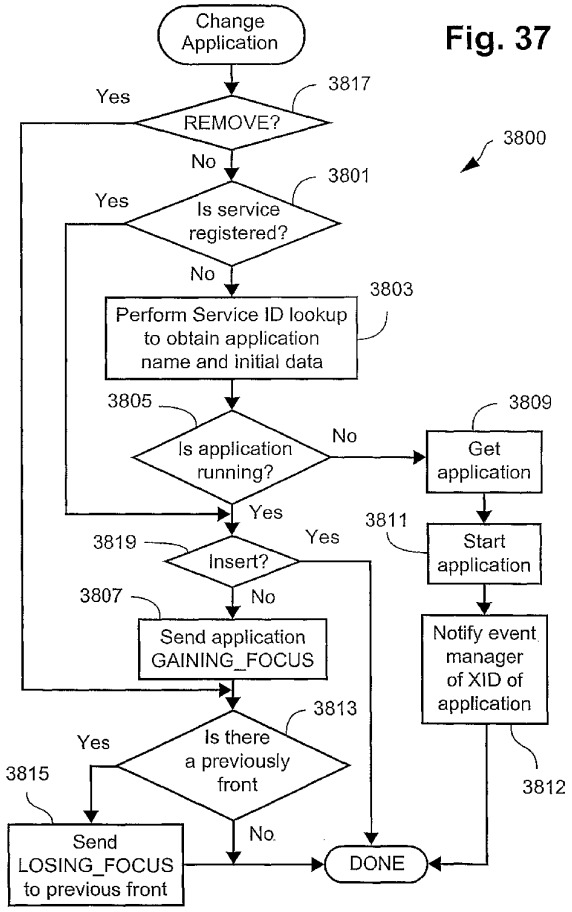

START
Connect to Event Manager
3701
3700
Start persistent applications
3702
Wait for events
3703
Event has the NO_CARD ID?
3705
Yes
Perform system specific function
3707
No
PRESS, RELEASE, MOVE, INSERT,
3709
Yes
Change Application
3800
No
BADCARD, LOWBAT?
3713
Yes
Give user feedback
3715
No
APP_REGISTER
3717
Yes
Application Registering
3900
No
Discard Event
3725

Fig. 37


Change Application
3817
Yes
REMOVE?
No
3800
3801
Yes
Is service registered?
No
3803
Perform Service ID lookup to obtain application name and initial data
3805
3809
Is application running?
No
Get application
Yes
3811
3819
Yes
Insert?
Start application
3807
No
Send application GAINING_FOCUS
Notify event manager of XID of application
3813
Yes
Is there a previously front
3815
3812
No
Send LOSING_FOCUS to previous front
DONE

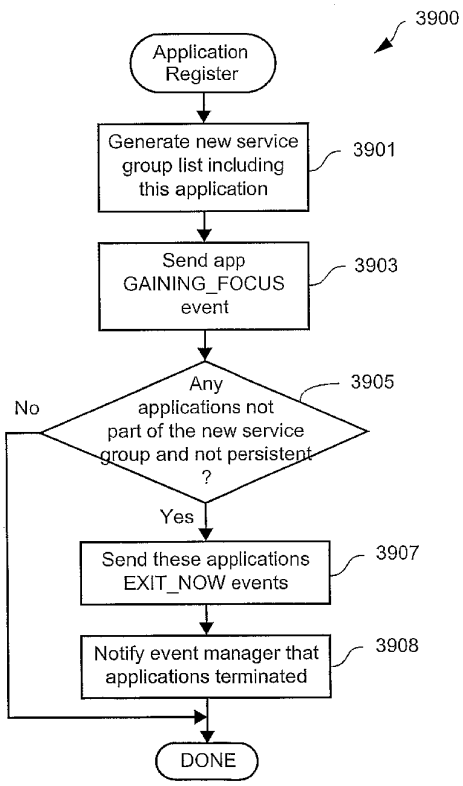

3900
Application Register
Generate new service group list including this application
3901
Send app GAINING_FOCUS event
3903
Any applications not part of the new service group and not persistent ?
3905
No
Yes
Send these applications EXIT_NOW events
3907
Notify event manager that applications terminated
3908
DONE

Fig. 38

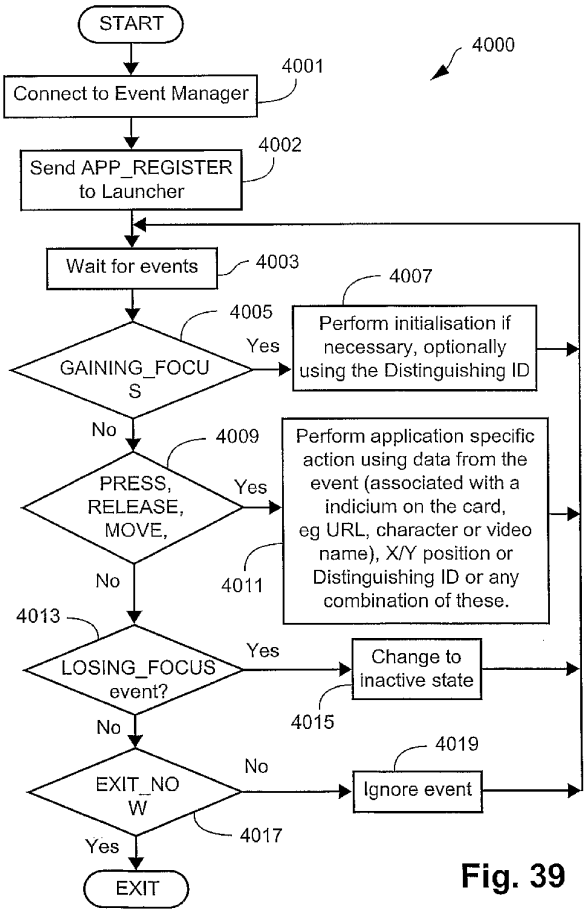

START
4000
4001
Connect to Event Manager
4002
Send APP_REGISTER to Launcher
Wait for events
4003
4007
4005
Yes
GAINING_FOCUS
Perform initialisation if necessary, optionally using the Distinguishing ID
No
4009
PRESS, RELEASE, MOVE,
Yes
Perform application specific action using data from the event (associated with a indicium on the card, eg URL, character or video name), X/Y position or Distinguishing ID or any combination of these.
4011
No
4013
LOSING_FOCUS event?
Yes
Change to inactive state
4015
No
4019
EXIT_NOW
No
Ignore event
4017
Yes
EXIT

Fig. 39

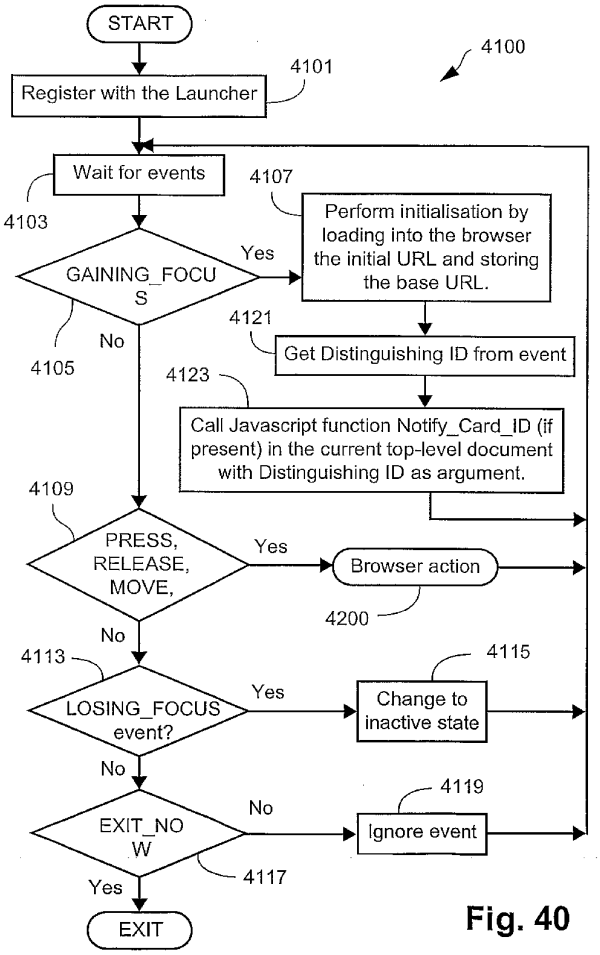
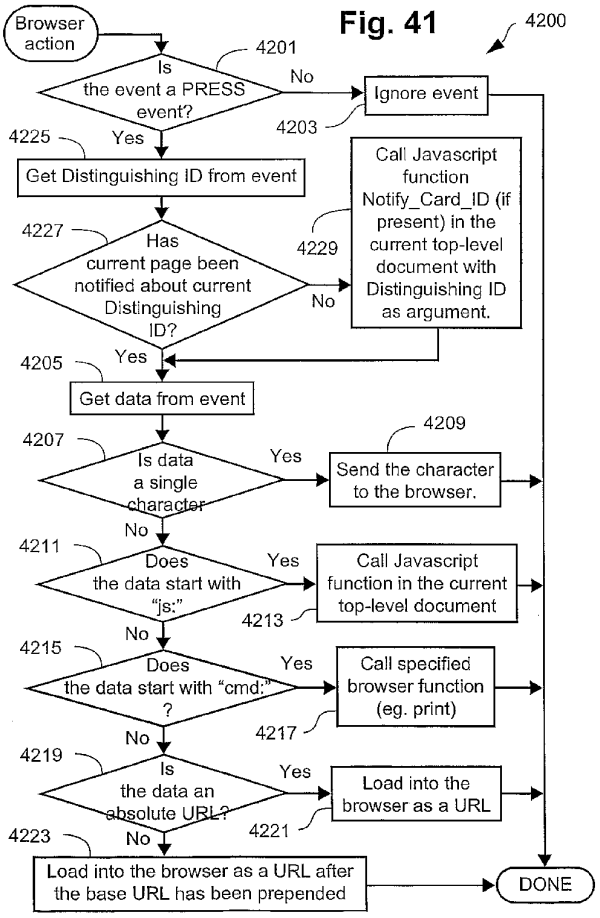

START
4100
4101
Register with the Launcher
Wait for events
4107
4103
Perform initialisation by loading into the browser the initial URL and storing the base URL.
GAINING_FOCUS
Yes
No
4105
4121
Get Distinguishing ID from event
4123
Call Javascript function Notify_Card_ID (if present) in the current top-level document with Distinguishing ID as argument.
4109
PRESS, RELEASE, MOVE,
Yes
Browser action
4200
No
4113
4115
LOSING_FOCUS event?
Yes
Change to inactive state
No
4119
EXIT_NOW
No
Ignore event
4117
Yes
EXIT

Fig. 40

Fig. 41


Browser action
4200
4201
Is the event a PRESS event?
No
Ignore event
4203
4225
Yes
Get Distinguishing ID from event
Call Javascript function Notify_Card_ID (if present) in the current top-level document with Distinguishing ID as argument.
4227
Has current page been notified about current Distinguishing ID?
4229
No
4205
Yes
Get data from event
4207
Is data a single character
Yes
4209
Send the character to the browser.
No
4211
Does the data start with "js:"
Yes
Call Javascript function in the current top-level document
4213
No
4215
Does the data start with "cmd:" ?
Yes
Call specified browser function (eg. print)
4217
No
4219
Is the data an absolute URL?
Yes
Load into the browser as a URL
4221
4223
No
Load into the browser as a URL after the base URL has been prepended
DONE

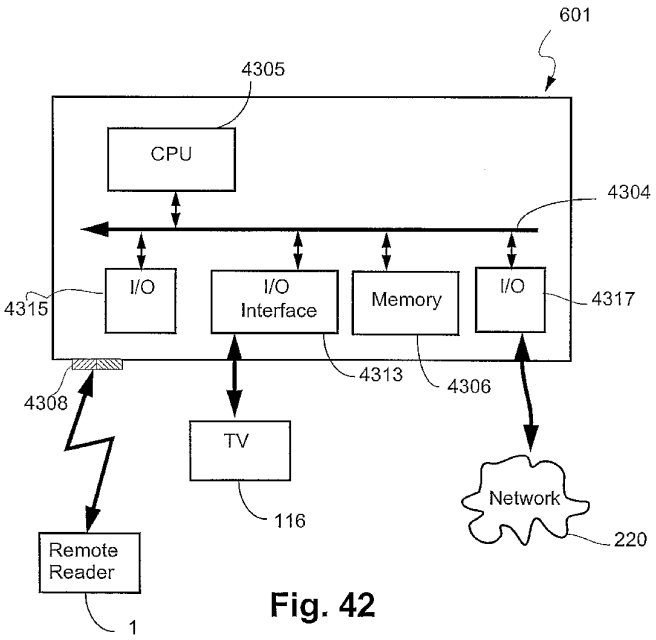

601
4305
CPU
4304
I/O
I/O Interface
Memory
I/O
4315
4317
4313
4306
4308
TV
Network
116
220
Remote Reader
1

Fig. 42

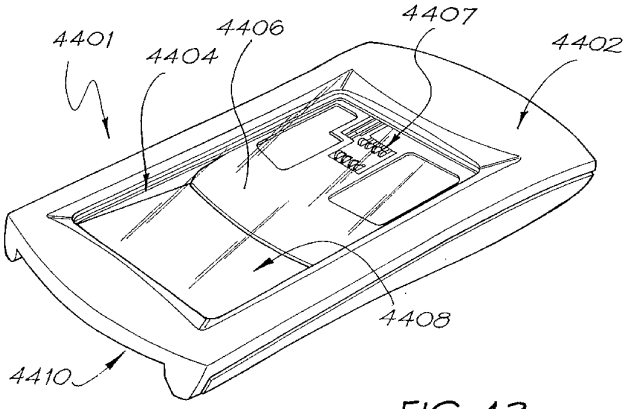

4401
4404
4406
4407
4402
4408
4410

FIG. 43

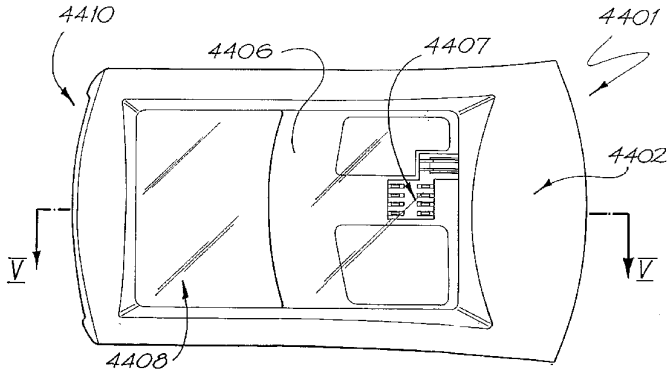

4410
4406
4407
4401
4402
V
V
4408

FIG. 44

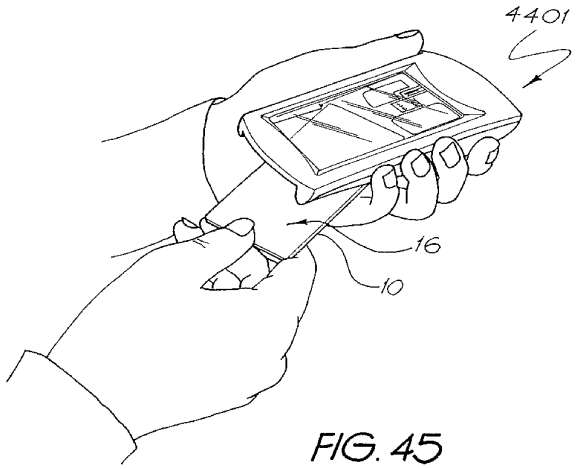


FIG. 45

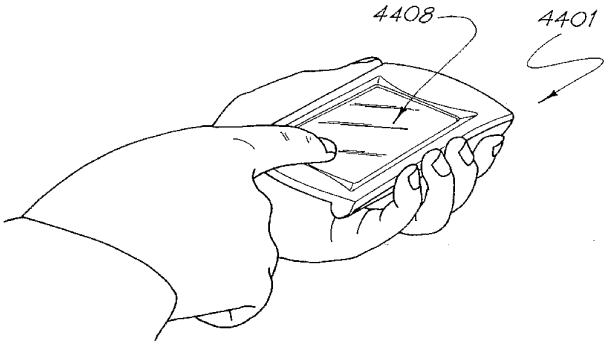


FIG. 46

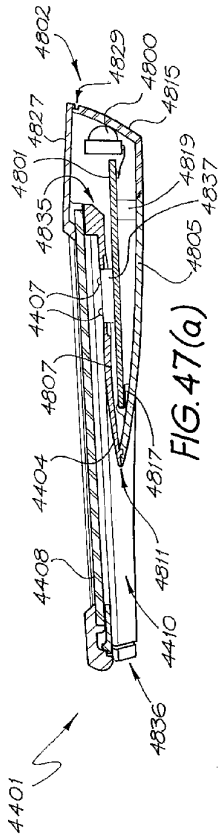


FIG. 47(a)

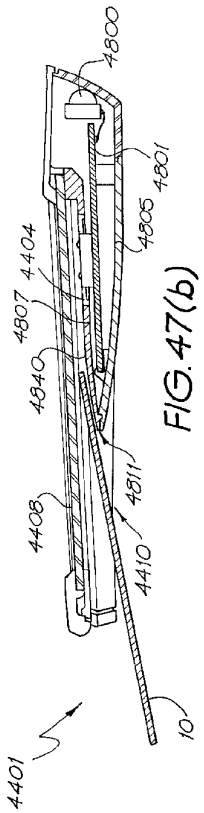


FIG. 47(b)

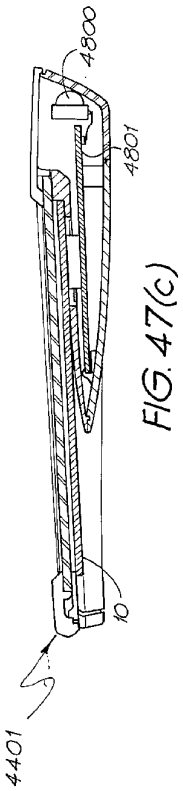


FIG. 47(c)

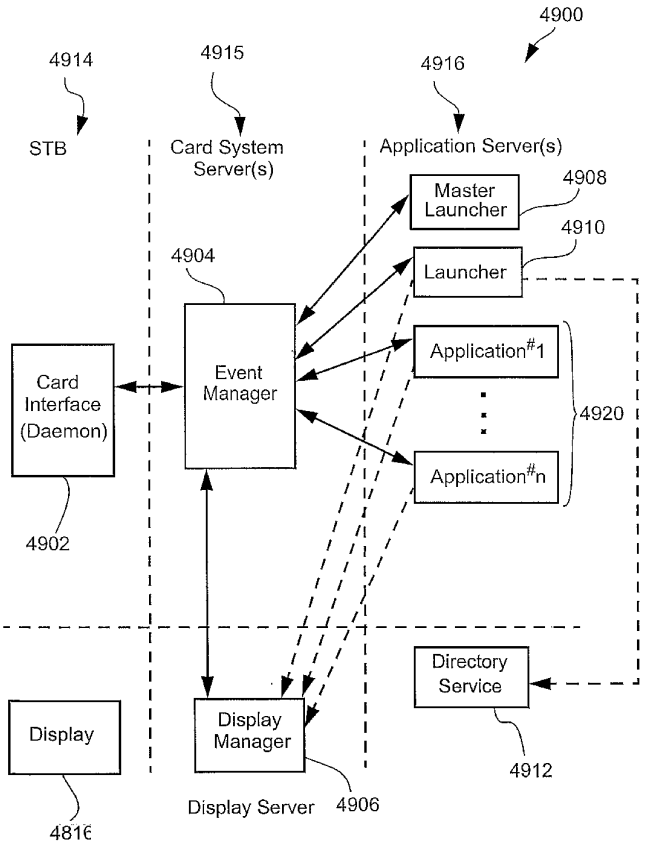

4900
4914
4915
4916
STB
Card System Server(s)
Application Server(s)
Master Launcher
4908
4910
Launcher
4904
Event Manager
Card Interface (Daemon)
Application#1
Application#n
4920
4902
Directory Service
4912
Display
Display Manager
Display Server
4906
4916

Fig. 48

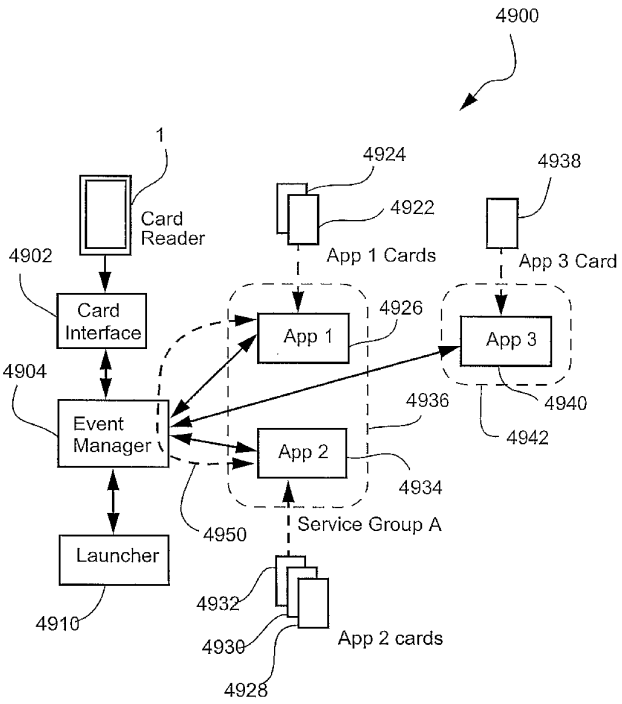

4900
1
Card Reader
4902
Card Interface
4904
Event Manager
Launcher
4910
4950
4924
4922
App 1 Cards
4938
App 3 Card
App 1
4926
App 3
4940
4936
4942
App 2
4934
Service Group A
4932
4930
4928
App 2 cards

Fig. 49

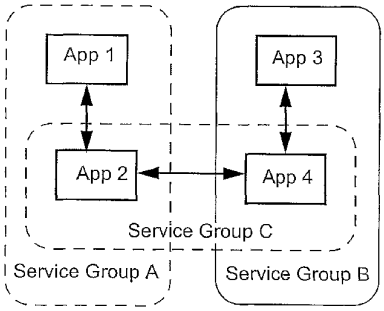

App 1
App 3
App 2
App 4
Service Group C
Service Group A
Service Group B

Fig. 50

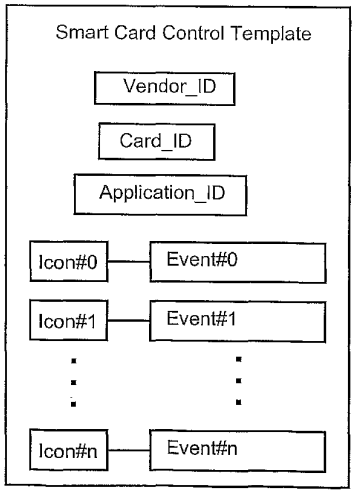

Smart Card Control Template
Vendor_ID
Card_ID
Application_ID
Icon#0
Event#0
Icon#1
Event#1
Icon#n
Event#n

Fig. 52

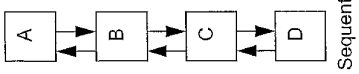


Fig. 51A

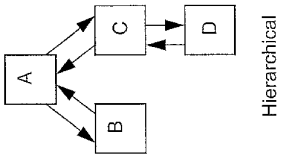


Fig. 51B

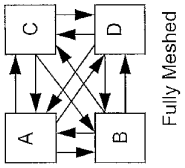
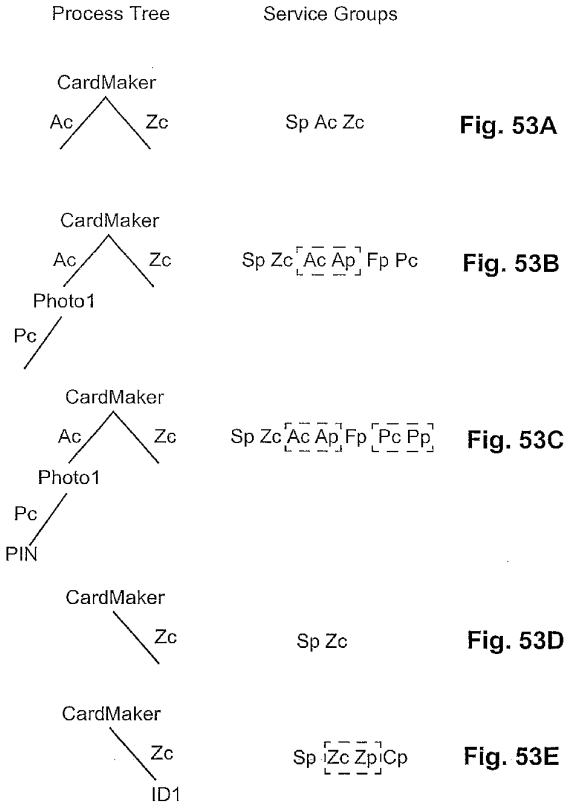


Fig. 51C


Process Tree
Service Groups
CardMaker
Ac
Zc
Sp Ac Zc
Fig. 53A
CardMaker
Ac
Zc
Photo1
Pc
Sp Zc Ac Ap Fp Pc
Fig. 53B
CardMaker
Ac
Zc
Photo1
Pc
PIN
Sp Zc Ac Ap Fp Pc Pp
Fig. 53C
CardMaker
Zc
Sp Zc
Fig. 53D
CardMaker
Zc
ID1
Sp Zc Zp Cp
Fig. 53E

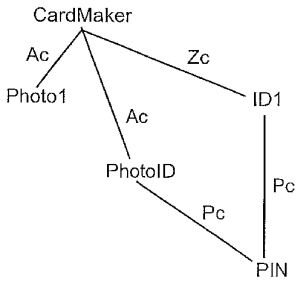

CardMaker
Ac
Zc
Photo1
ID1
Ac
PhotoID
Pc
Pc
PIN

Fig. 54

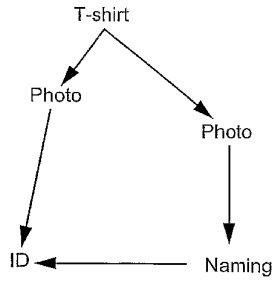

T-shirt
Photo
Photo
ID
Naming

Fig. 55

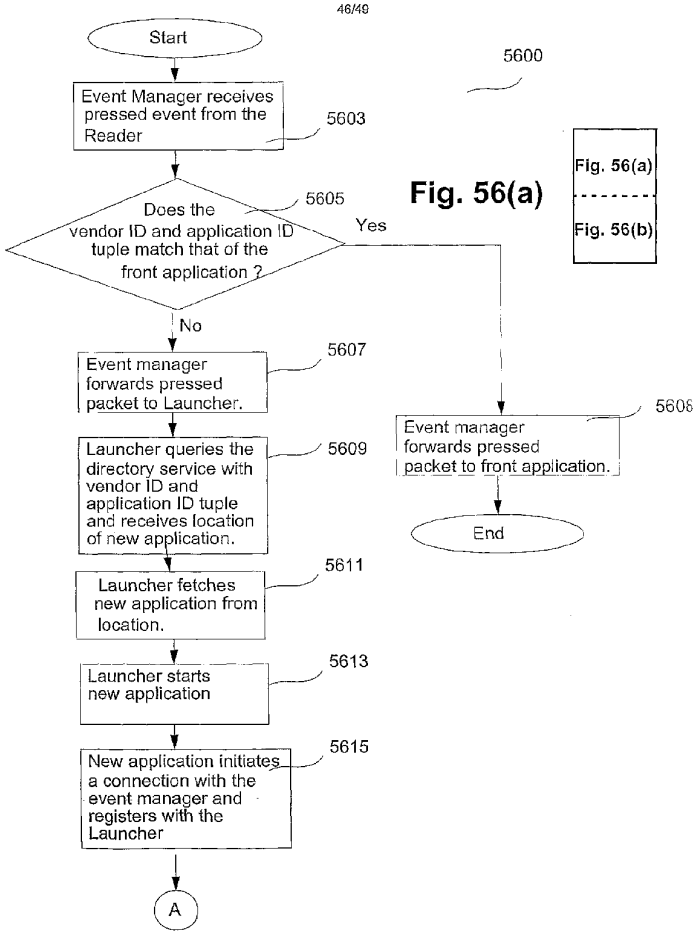

Start
5600
Event Manager receives pressed event from the Reader
5603
Does the vendor ID and application ID tuple match that of the front application ?
5605
Yes
No
Event manager forwards pressed packet to Launcher.
5607
Launcher queries the directory service with vendor ID and application ID tuple and receives location of new application.
5609
Launcher fetches new application from location.
5611
Launcher starts new application
5613
New application initiates a connection with the event manager and registers with the Launcher
5615
A
Event manager forwards pressed packet to front application.
5608
End
Fig. 56(a)
Fig. 56(b)

Fig. 56(a)

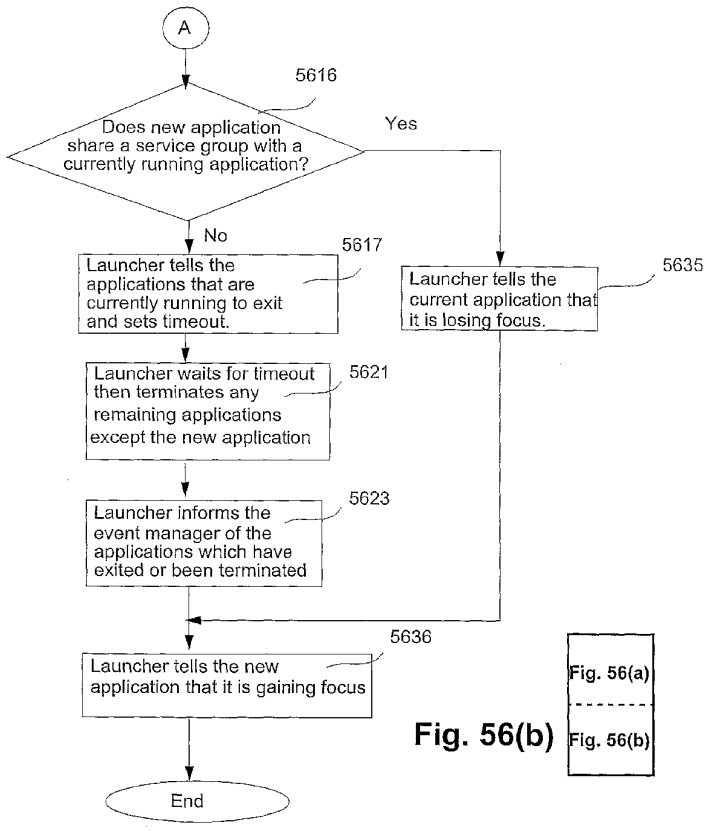


Fig. 56(b)

Fig. 56(a)
Fig. 56(b)

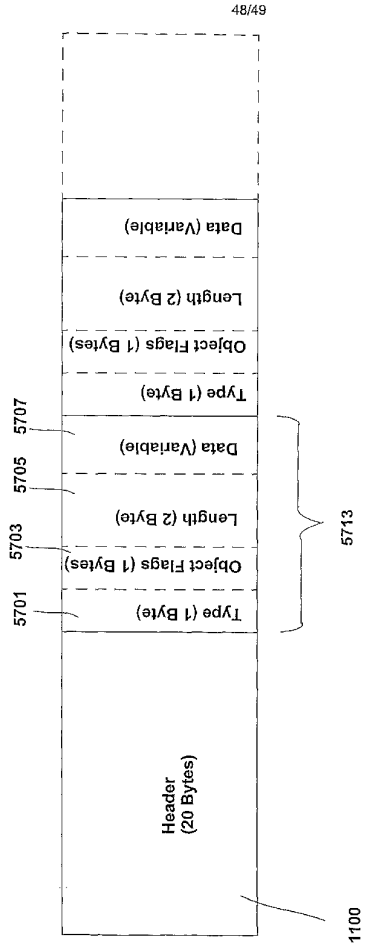

Header
(20 Bytes)
Type (1 Byte)
Object Flags (1 Bytes)
Length (2 Byte)
Data (Variable)
Type (1 Byte)
Object Flags (1 Bytes)
Length (2 Byte)
Data (Variable)
5701
5703
5705
5707
5713
1100

Fig. 57

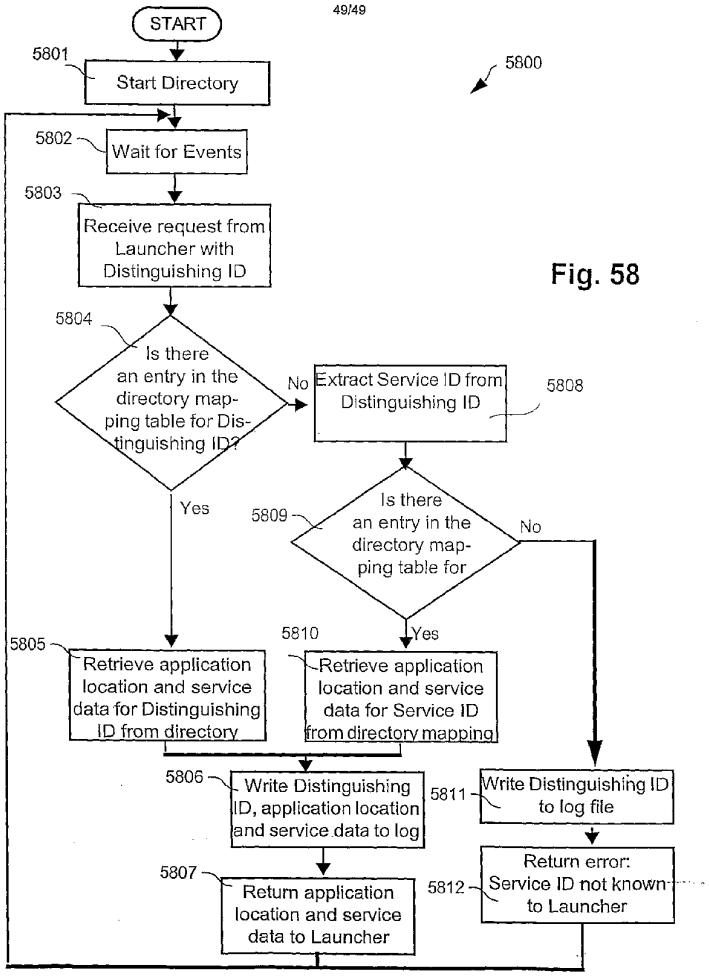

START
5801
Start Directory
5800
5802
Wait for Events
5803
Receive request from Launcher with Distinguishing ID
5804
Is there an entry in the directory mapping table for Distinguishing ID?
No
Extract Service ID from Distinguishing ID
5808
Yes
5809
Is there an entry in the directory mapping table for
No
Yes
5805
Retrieve application location and service data for Distinguishing ID from directory
5810
Retrieve application location and service data for Service ID from directory mapping
5806
Write Distinguishing ID, application location and service data to log
5811
Write Distinguishing ID to log file
5807
Return application location and service data to Launcher
5812
Return error: Service ID not known to Launcher

Fig. 58

【国際調査報告】

WO 02/23320 PCT/AU01/01142

INTERNATIONAL SEARCH REPORT

International application No.
PCT/AU01/01142

A. CLASSIFICATION OF SUBJECT MATTER

Int. Cl. [7]: G06F 3/023, G06K 19/07

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC G06F, G06K

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
AU:IPC AS ABOVE

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
WPAT, USPTO

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	AU 53527/99A, CANON KABUSHIKI KAISHA, 13 April 2000	1-36
A	WO 96/32702A, SMART TV COMPANY, 17 October 1996	
P,A	US 6145740A, MOLANO et al, 14 November 2000	

☐ Further documents are listed in the continuation of Box C ☒ See patent family annex

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search
14 December 2001

Date of mailing of the international search report
20 DEC 2001

Name and mailing address of the ISA/AU
AUSTRALIAN PATENT OFFICE
PO BOX 200, WODEN ACT 2606, AUSTRALIA
E-mail address: pct@ipaustralia.gov.au
Facsimile No. (02) 6285 3929

Authorized officer
S KAUL
Telephone No : (02) 6283 2182

Form PCT/ISA/210 (second sheet) (July 1998)

フロントページの続き

(81)指定国 AP(GH,GM,KE,LS,MW,MZ,SD,SL,SZ,TZ,UG,ZW),EA(AM,AZ,BY,KG,KZ,MD,RU,TJ,TM),EP(AT,BE,CH,CY,DE,DK,ES,FI,FR,GB,GR,IE,IT,LU,MC,NL,PT,SE,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AT,AU,AZ,BA,BB,BG,BR,BY,BZ,CA,CH,CN,CO,CR,CU,CZ,DE,DK,DM,DZ,EC,EE,ES,FI,GB,GD,GE,GH,GM,HR,HU,ID,IL,IN,IS,JP,KE,KG,KP,KR,KZ,LC,LK,LR,LS,LT,LU,LV,MA,MD,MG,MK,MN,MW,MX,MZ,NO,NZ,PH,PL,PT,RO,RU,SD,SE,SG,SI,SK,SL,TJ,TM,TR,TT,TZ,UA,UG,US,UZ,VN,YU,ZA,ZW

（特許庁注：以下のものは登録商標）
フロッピー
ＵＮＩＸ

(72)発明者 ヤップ, スーーケン
オーストラリア国 ２１１３ ニュー サウス ウェールズ州, ノース ライド, トーマス ホルト ドライブ １ キヤノン インフォメーション システムズ リサーチ オーストラリア プロプライエタリー リミテツド 内
(72)発明者 スミーリー, ロバート
オーストラリア国 ２１１３ ニュー サウス ウェールズ州, ノース ライド, トーマス ホルト ドライブ １ キヤノン インフォメーション システムズ リサーチ オーストラリア プロプライエタリー リミテツド 内
(72)発明者 シンプソンーヤング, ウイリアム
オーストラリア国 ２１１３ ニュー サウス ウェールズ州, ノース ライド, トーマス ホルト ドライブ １ キヤノン インフォメーション システムズ リサーチ オーストラリア プロプライエタリー リミテツド 内
(72)発明者 ニューマン, アンドリュー, ティモシー, ロバート
オーストラリア国 ２１１３ ニュー サウス ウェールズ州, ノース ライド, トーマス ホルト ドライブ １ キヤノン インフォメーション システムズ リサーチ オーストラリア プロプライエタリー リミテツド 内

Ｆターム(参考) 5B058 CA01 KA40