(19)日本国特許庁(JP)

(12)特許公報(B2)

(11)特許番号
特許第7498813号
(P7498813)

(45)発行日　令和6年6月12日(2024.6.12)

(24)登録日　令和6年6月4日(2024.6.4)

(51)国際特許分類
Ｇ０６Ｆ　8/60　(2018.01)

ＦＩ
Ｇ０６Ｆ　8/60

請求項の数　19　（全27頁）

(21)出願番号　特願2023-12464(P2023-12464)
(22)出願日　令和5年1月31日(2023.1.31)
(65)公開番号　特開2024-44967(P2024-44967A)
(43)公開日　令和6年4月2日(2024.4.2)
審査請求日　令和5年1月31日(2023.1.31)
(31)優先権主張番号　17/949,556
(32)優先日　令和4年9月21日(2022.9.21)
(33)優先権主張国・地域又は機関
米国(US)

(73)特許権者　391007161
エヌ・シー・アール・コーポレイション
ＮＣＲ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国　３０３０８－１００７
ジョージア州　アトランタ　スプリング
ストリート　エヌダブリュー　８６４
(74)代理人　100098589
弁理士　西山　善章
(74)代理人　100147599
弁理士　丹羽　匡孝
(72)発明者　シモン　ウォーターマン
英国　サマセット　ＢＡ５　１ＰＦ　ウェル
ズ　ブリードニー　スプリングウィロー
審査官　石坂　知樹

最終頁に続く

(54)【発明の名称】　周辺デバイス通信

(57)【特許請求の範囲】

【請求項１】

周辺デバイス接続データと関連付けられたソフトウェアコンテナ要素を提供し、前記周辺デバイス接続データにより前記ソフトウェアコンテナ要素が複数の所定の周辺デバイスの各周辺デバイスにアクセスし、かつ前記各周辺デバイスと通信することを可能にするコンピュータ実装方法であって、

コンピューティングデバイスの１つ以上のプロセッサ上で実行可能である第１の実行可能なソフトウェアを含むモジュールから、前記コンピューティングデバイスに接続可能な少なくとも１つのクラスの周辺デバイスを示す第１のデータを受信し、それによって、各クラスが複数の所定の周辺デバイスと関連付けられるステップと、

前記コンピューティングデバイスの前記１つ以上のプロセッサ上で実行可能である第２の実行可能なソフトウェアを包含するソフトウェアコンテナ要素を提供するステップであって、それによって、前記ソフトウェアコンテナ要素が、周辺デバイス接続データと関連付けられ、前記周辺デバイス接続データにより、前記ソフトウェアコンテナ要素が、各クラスと関連付けられた前記複数の所定の周辺デバイスの各周辺デバイスにアクセスし、かつ前記各周辺デバイスと通信することを可能にするステップと、

新しい周辺デバイスが前記コンピューティングデバイスに接続され、又は周辺デバイスが前記コンピューティングデバイスに対して切断及び再接続されることに応答して、前記ソフトウェアコンテナ要素によって、前記新しい又は再接続された周辺デバイスが前記コンピューティングデバイスに接続されていると決定するステップと、

の各ステップを含む、方法。
【請求項２】
前記コンピューティングデバイスに接続可能な単一のクラスの周辺デバイスを示すデータとして前記第１のデータを受信することであって、それによって、前記単一のクラスが、セルフサービス端末又はポイントオブセール端末に接続可能な複数の所定の周辺デバイスと関連付けられる、請求項１に記載の方法。
【請求項３】
各クラスと関連付けられた前記複数の所定の周辺デバイスを示す第２のデータを、前記コンピューティングデバイスによってアクセス可能なメモリに記憶することを更に含む、請求項１に記載の方法。
【請求項４】
前記第１のデータを受信する前に、どの周辺デバイスが前記コンピューティングデバイスに接続されているかを問い合わせるクエリを前記モジュールに送信することを更に含む、請求項３に記載の方法。
【請求項５】
前記コンピューティングデバイス上のソフトウェアコンテナ要素を管理する責任を有するソフトウェアアプリケーションにおいて前記第１のデータを受信することを更に含む、請求項４に記載の方法。
【請求項６】
接続された周辺デバイスと通信するように構成されたモジュールとして、前記モジュールを提供することを更に含む、請求項５に記載の方法。
【請求項７】
前記第１の実行可能なソフトウェアを包含するソフトウェアコンテナ要素として、前記モジュールを提供することを更に備え、任意選択的に、前記モジュールが、デバイスプラグインモジュールである、請求項６に記載の方法。
【請求項８】
前記第１のデータを受信することに応答して、前記ソフトウェアコンテナ要素への割り当てのために、各クラスと関連付けられた前記複数の所定の周辺デバイスの各々を準備するように前記モジュールに指令する少なくとも１つのコマンドを前記モジュールに送信することを更に含む、請求項７に記載の方法。
【請求項９】
前記モジュールから、各クラスと関連付けられた前記複数の所定の周辺デバイスの各々についての少なくともそれぞれの周辺デバイス経路を示す第３のデータを受信することを更に含む、請求項８に記載の方法。
【請求項１０】
各クラスと関連付けられた前記複数の所定の周辺デバイスの各々についての少なくともそれぞれの周辺デバイス経路を表すデータとして、前記周辺デバイス接続データを提供することを更に含む、請求項９に記載の方法。
【請求項１１】
前記ソフトウェアコンテナ要素と関連付けられた制御グループに前記周辺デバイス接続データを記憶することを更に含む、請求項１０に記載の方法。
【請求項１２】
第２の実行可能な命令の実行時に、前記ソフトウェアコンテナ要素によって、各クラスと関連付けられた前記複数の所定の周辺デバイスのうちのどの周辺デバイスが前記コンピューティングデバイスに接続されているかを決定することを更に含む、請求項１１に記載の方法。
【請求項１３】
決定することに応答して、前記ソフトウェアコンテナ要素によって、前記コンピューティングデバイスに接続された各周辺デバイスを初期化することを更に含む、請求項１２に記載の方法。

【請求項１４】
初期化することに応答して、前記ソフトウェアコンテナ要素によって、前記接続された周辺デバイスのうちの少なくとも１つに１つ以上のコマンドを送信することを更に含む、請求項１３に記載の方法。
【請求項１５】
コンピューティング環境に関係なく実行可能であるソフトウェアとして、前記第１の実行可能なソフトウェア及び前記第２の実行可能なソフトウェアを提供することを更に含む、請求項１４に記載の方法。
【請求項１６】
前記モジュール及び前記ソフトウェアコンテナ要素をそれぞれ介して、隔離されたコンピューティング環境で前記第１の実行可能なソフトウェア及び前記第２の実行可能なソフトウェアを実行することを更に含む、請求項１５に記載の方法。
【請求項１７】
コンピューティングデバイスであって、前記コンピューティングデバイス上のソフトウェアコンテナ要素を管理する責任を有するソフトウェアアプリケーションを実行するように構成された１つ以上のプロセッサを備え、前記ソフトウェアアプリケーションが、実行されたときに、
コンピューティングデバイスの１つ以上のプロセッサ上で実行可能である第１の実行可能なソフトウェアを含むモジュールから、前記コンピューティングデバイスに接続可能な少なくとも１つのクラスの周辺デバイスを示す第１のデータを受信し、それによって、各クラスが、複数の所定の周辺デバイスと関連付けられることと、
前記コンピューティングデバイスの前記１つ以上のプロセッサ上で実行可能である第２の実行可能なソフトウェアを包含するソフトウェアコンテナ要素を提供することであって、それによって、前記ソフトウェアコンテナ要素が、周辺デバイス接続データと関連付けられ、前記周辺デバイス接続データにより、前記ソフトウェアコンテナ要素が、各クラスと関連付けられた前記複数の所定の周辺デバイスの各周辺デバイスにアクセスし、かつ前記各周辺デバイスと通信することを可能にすることと、
新しい周辺デバイスが前記コンピューティングデバイスに接続され、又は周辺デバイスが前記コンピューティングデバイスに対して切断及び再接続されることに応答して、前記ソフトウェアコンテナ要素によって、前記新しい又は再接続された周辺デバイスが前記コンピューティングデバイスに接続されていると決定することと、
を行うように構成されている、コンピューティングデバイス。
【請求項１８】
前記コンピューティングデバイスが、ポイントオブセール端末又はセルフサービス端末である、請求項１７に記載のコンピューティングデバイス。
【請求項１９】
命令を含むコンピュータプログラムであって、前記命令が、コンピューティングデバイスによって実行されたときに、前記コンピューティングデバイスに、請求項１７又は１８において定義される方法を実行させる、コンピュータプログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、コンテナ要素が後で実行されるか、又は接続されるときに所定の周辺デバイスが接続されているかどうかに関係なく、特定のクラスと関連付けられた周辺デバイスの所定のセットの各周辺デバイスへのアクセスを有するように構成されている、
ソフトウェアコンテナ要素を提供するための方法及びコンピューティングデバイスに関する。特に、しかし排他的ではないが、本発明は、デバイスプラグインソフトウェアコンテナが、そのリソースを、セルフサービス端末又はポイントオブセール端末に接続可能な周辺デバイスのクラスとしてアドバタイズする方法論に関する。
【０００２】

このクラスは、周辺デバイスの所定のセットと関連付けられている。次いで、ソフトウェアコンテナを、そのクラスの各周辺デバイスに対するデバイス経路を示すソフトウェアコンテナの周辺デバイス接続データと関連付けることによって、そのクラス内の全ての周辺デバイスに対するアクセスを有する端末上に提供することができる。

【背景技術】

【０００３】

セルフサービス端末（ＳＳＴ）及び／又はポイントオブセール（ＰＯＳ）端末は、小売業界で適宜、使用されることが知られている。ＳＳＴ及びＰＯＳ端末は、通常、顧客が小売施設と取引を行うことを可能にするために使用される。任意の小売施設の各ＳＳＴ又はＰＯＳ端末を、異なる周辺デバイスに接続することができる。

【０００４】

各ＳＳＴ又はＰＯＳ端末はまた、異なる基礎となるオペレーティングシステム（Ｌｉｎｕｘ、Ｗｉｎｄｏｗｓなど）及び異なるソフトウェアアプリケーションを有するなどの、異なるソフトウェアを実行し得る。これは通常、ＳＳＴ又はＰＯＳ端末が使用される小売施設、及び小売施設におけるＳＳＴ又はＰＯＳ端末の使用目的に依存する。適宜、ＳＳＴ又はＰＯＳ端末上で実行されているソフトウェアをアップグレードすることもでき、又は接続された周辺デバイスを変更することができる。

【０００５】

端末間で頻繁に存在するばらつきのため、ソフトウェア開発者は、ＳＳＴ又はＰＯＳ端末上で実行する必要のあるソフトウェアを包含するソフトウェアコンテナを使用し始めている。ソフトウェアコンテナは、走るソフトウェアを隔離するので、複雑なプラットフォーム依存性を回避するのに役立つ。

【０００６】

つまり、ソフトウェアコンテナは、端末のコンピューティング環境に関係なく、端末の１つ以上のプロセッサ上で実行することができる。これは、ソフトウェアコンテナがソフトウェア（アプリケーションコード及び任意のソフトウェア依存性）の全てを包含し、任意のコンピューティング環境で実行可能である必要があるためである。

【０００７】

更に、これらのコンテナは独自の隔離されたコンピューティング環境で動作するため（ソフトウェアコンテナの外側にある他のソフトウェア／ハードウェアとのある特定の事前定義された通信経路（特定のファイル、特定のポートへのアクセスなど）は例外）、これによりまた、当該コンテナが特に安全になる。このように、ソフトウェアコンテナは、ＳＳＴ及びＰＯＳ端末上で使用するためのソフトウェアをパッケージング及び配信する効果的な方法である。

【０００８】

ソフトウェア又はハードウェアのアップグレードが端末上で行われると、これらのアップグレードを構成する新しいソフトウェアコンテナが端末上で実行され得る。また、コンテナが事前構築され得るので、これは、あらゆる端末上での複雑な構築を回避するのに役立つ場合がある。

【０００９】

ソフトウェアコンテナは、コンピューティングデバイス上にハイパーバイザをインストールする必要がないという点において、仮想マシンとは異なることが留意される。ソフトウェアコンテナはまた、通常、仮想マシンよりも軽量であり、かつ仮想マシンよりも速く走る。更に、仮想マシンはコンピュータシステム全体を仮想化するのに対し、ソフトウェアコンテナはオペレーティングシステムを仮想化する。ソフトウェアコンテナはまた、単一のオペレーティングシステムを共有するが、各仮想マシンは独自のオペレーティングシステムを有する。

【００１０】

ソフトウェアコンテナを展開する場合、（ｄｏｃｋｅｒ、ＲＫＴ、ＣＲＩ－Ｏ、及びＬＸＤなどのような）コンテナエンジンが利用される。コンテナエンジンは、ユーザ要求、

又は（Ｋｕｂｅｒｎｅｔｅｓ、Ｓｗａｒｍ、Ｍｅｓｏｓなどのような）コンテナオーケストレータのＡＰＩサーバからの要求を受け入れ、レジストリから（特定の画像形式の）画像ファイルを引き出し、コンテナマウントポイントを準備し、ソフトウェアコンテナを実行するのに必要なメタデータを準備し、かつコンテナランタイムを呼び出すことができる。

【００１１】

コンテナランタイムは、コンテナエンジンの一部である。コンテナランタイム（ｒｕｎｃ、ｃｏｎｔａｉｎｅｒｄ、ｃｒｕｎ、ｒａｉｌｃａｒ、ｋａｔａｃｏｎｔａｉｎｅｒｓなどのような）は、コンテナ画像ファイルを実行し、かつそれによって、ソフトウェアコンテナを作成するために、マウントポイント、及びコンテナエンジンによって提供されるメタデータを使用し、コンピューティングデバイス上で走る（ホストＯＳの）カーネルと通信する。

【００１２】

複数のコンピューティングデバイスにわたって複数のコンテナを実装する場合、ソフトウェアコンテナのオーケストレータプラットフォームがよく使用される。これらのプラットフォームは、複数のコンピューティングデバイス（例えば、ＳＳＴ又はＰＯＳ端末）にわたってコンテナのワークロードをスケジュールすることができ、かつ、また、標準化されたアプリケーション定義ファイル（例えば、ｋｕｂｅ　ＹＡＭＬ、ｄｏｃｋｅｒ　ｃｏｍｐｏｓｅなど）を提供することができる。Ｋｕｂｅｒｎｅｔｅｓは、コンテナオーケストレーションプラットフォームの一例である。

【００１３】

Ｋｕｂｅｒｎｅｔｅｓコンテナオーケストレーションプラットフォームは、クラスタを管理する、Ｋｕｂｅｒｎｅｔｅｓマスタと呼ばれる制御ユニットを含む、ユニットのクラスタ、及びワークロード（コンテナ）を走らせる少なくとも１つのノード（又はワーカ）である。

【００１４】

Ｋｕｂｅｒｎｅｔｅｓオーケストレータプラットフォームの１つの部分は、ｋｕｂｅｌｅｔである。ｋｕｂｅｌｅｔは、Ｋｕｂｅｒｎｅｔｅｓシステムの一部である、あらゆるワーカ上で走るエージェントである。動作中、ｋｕｂｅｌｅｔは、どのコンテナをコンピューティングデバイス上で走らせるべきかをｋｕｂｅｌｅｔに通知する（Ｋｕｂｅｒｎｅｔｅｓマスタ上のＡＰＩサーバからの）コマンドを受信する。Ｋｕｂｅｒｎｅｔｅｓでは、これらのコンテナは、「ポッド」内に提供される。

【００１５】

通常、単一のポッドは、単一のコンテナを含むが、ポッド内に複数のコンテナを含むことが可能である。ｋｕｂｅｌｅｔは、コンテナエンジン内のコンテナランタイムを介してソフトウェアコンテナを実行するために、どのコンテナをコンピューティングデバイス上で走らせるべきかに関する情報を（例えば、コンテナランタイムインターフェース（ＣＲＩ）を介して）コンテナエンジンに渡す。

【００１６】

コンテナランタイムが、実行可能なコンテナ画像ファイルを実行すると、ソフトウェアコンテナが作成される。そのため、ソフトウェアコンテナは本質的に、関連付けられた実行可能なコンテナ画像ファイルのランタイムインスタンスである。この意味で、実行可能なコンテナ画像ファイルは、ソフトウェアコンテナ要素として少なくとも実行可能である必要がある、ソフトウェアの全てを有する画像ファイルである。

【００１７】

より詳細には、コンテナ画像ファイルは通常、ソフトウェアコンテナの必要性及び能力を記述する任意のメタデータに加えて、ソフトウェアコンテナを走らせるための全ての必要な要件を含むバイナリファイルである。コンテナ画像ファイル自体は、ソフトウェアコンテナを走らせるために必要な実行可能な命令を定義するいくつかの層から構成される。

【００１８】

例えば、コンテナ画像ファイルは、ソフトウェアアプリケーションの実行可能なコード

を定義するいくつかの層、ソフトウェアアプリケーションが依存する任意のソフトウェア依存性に対するコードを定義するいくつかの層、及び任意の必要な構成設定に対するコードを定義するいくつかの層を含み得る。

【００１９】

コンテナ画像ファイルは、多くの場合、コンテナ画像レジストリに記憶される。各コンテナ画像ファイルは、コンテナ画像ファイル内の層及びメタデータを定義する特定のコンテナ画像形式で記憶される。例えば、オープンコンテナイニシアチブ（ＯＣＩ）画像形式は、各層のｔａｒファイル、及び画像ファイルと関連付けられたメタデータを指定するＪＳＯＮ形式のマニフェストファイルとして、画像ファイルを定義する。

【００２０】

ＳＳＴ及びＰＯＳ端末上にソフトウェアコンテナを配置する場合、デバイスドライバ／取り扱いソフトウェアをコンテナ内で実行することが望ましい。これは、デバイスドライバ及びそれらの関連ソフトウェアを、コンテナ内の他の全てのソフトウェアと同様に配信、管理、及び維持することができることを意味する。

【００２１】

しかしながら、このアプローチは通常、特権ありのコンテナを必要とし、これは、コンテナが、基となるホストコンピューティングデバイスのほとんどの又は全てのルート能力を有し、それによって、非特権付きコンテナでは利用できないであろうリソースへのアクセスを可能にすることを意味する。

【００２２】

ゆえに特権アクセスを有するソフトウェアコンテナを提供することはセキュリティリスクであり、展開のベストプラクティスに反するものである。これは、ＳＳＴ及びＰＯＳ端末が堅牢なセキュリティを有する必要があることを考慮すると、特に小売において懸念される。

【００２３】

特権的な操作を回避するために、現在の手法では、通常、事前にアクセスする必要のあるデバイスのセットを正確に知るために、コンテナを必要とする。すなわち、コンテナを実行するときに、コンテナには、動作中にアクセスする必要のある周辺デバイスのみを示すデータが提供される。このアプローチが成功するためには、これらのデバイスも、コンテナが最初に実行されたときに存在する必要があることに留意されたい。

【００２４】

デバイスが存在しない場合（例えば、それらがプラグインされていない場合）、コンテナは、利用可能となった（例えば、一旦プラグインされると）これらのデバイスと通信できなくなる。したがって、このアプローチは、デバイスプラグアンドプレイ又はアンプラグ－リプラグのシナリオを防止することが理解されよう。

【発明の概要】

【００２５】

本発明の目的は、上述の問題のうちの１つ以上を少なくとも部分的に軽減することである。

【００２６】

本発明の特定の実施形態の目的は、どのデバイスが実際に接続されるかを予め指定することなく、ＳＳＴ又はＰＯＳ端末に接続可能な任意の所定の周辺デバイスと通信することができるソフトウェアコンテナを提供することを助けることである。

【００２７】

本発明の特定の実施形態の目的は、異なる端末上の異なるハードウェア設定（例えば、異なる周辺デバイス）に適合する、カスタムコンテナセットアップ（例えば、Ｋｕｂｅｒｎｅｔｅｓポッドセットアップ）を提供する必要性を回避することを助けることである。

【００２８】

本発明の特定の実施形態の目的は、端末上の異なるハードウェアセットアップ、又はソフトウェアコンテナの実行における特定のデバイスの存在若しくは不在に適応する必要が

ないコンテナ仕様を提供することを助けることである。
【００２９】
本発明の特定の実施形態の目的は、ソフトウェアコンテナの初期実行後にプラグインされるか、又はソフトウェアコンテナの初期実行後にアンプラグされ、かつリプラグされる周辺デバイスと通信することができるソフトウェアコンテナを実行する、ＳＳＴ又はＰＯＳ端末を提供することを助けることである。
【００３０】
本発明の特定の実施形態の目的は、ファームウェア更新により、検出されたデバイスの存在が一時的に消失し、その後再び現れる場合でも、ソフトウェアコンテナと周辺デバイスとの間の通信を維持することを助けることである。
【００３１】
本発明の特定の実施形態の目的は、デバイスドライバソフトウェアを含むソフトウェアコンテナを提供することを助けることであり、ここで、コンテナは、特権アクセスを必要とせず、セキュリティを損なわずに配信及び管理され得る。
【００３２】
本発明の特定の実施形態の目的は、ホットプラグ機能を有するソフトウェアコンテナを提供することを助けることであり、ここで、デバイスは、コンテナの起動時に存在する必要はなく、それらが後で追加又は除去される場合、システムは適合する。
【００３３】
本発明の第１の態様によれば、ソフトウェアコンテナ要素が複数の所定の周辺デバイスの各周辺デバイスにアクセスし、かつそれと通信することを可能にする、周辺デバイス接続データと関連付けられたソフトウェアコンテナ要素を提供するためのコンピュータ実装方法が提供されている。
このような実装方法は、コンピューティングデバイスの１つ以上のプロセッサ上で実行可能である第１の実行可能なソフトウェアを含むモジュールから、コンピューティングデバイスに接続可能な周辺デバイスの少なくとも１つのクラスを示す第１のデータを受信するステップであって、各クラスが、複数の所定の周辺デバイスと関連付けられている。
【００３４】
受信するステップと、コンピューティングデバイスのその１つ以上のプロセッサ上で実行可能である第２の実行可能なソフトウェアを含む、ソフトウェアコンテナ要素を提供するステップであって、ソフトウェアコンテナ要素が、ソフトウェアコンテナ要素が各クラスと関連付けられた複数の所定の周辺デバイスの各周辺デバイスにアクセスし、かつそれと通信することを可能にする、周辺デバイス接続データと関連付けられている、提供するステップと、を含む。
【００３５】
好適には、方法は、コンピューティングデバイスに接続可能な周辺デバイスの単一のクラスを示すデータとして第１のデータを受信することであって、単一のクラスが、セルフサービス端末又はポイントオブセール端末に接続可能な複数の所定の周辺デバイスと関連付けられている、受信することを更に含む。
【００３６】
好適には、方法は、コンピューティングデバイスに接続されている新しい周辺デバイス、又はコンピューティングデバイスに対して切断され、かつ再接続されている周辺デバイスに応答して、ソフトウェアコンテナ要素によって、新しい又は再接続された周辺デバイスがコンピューティングデバイスに接続されていると決定することを更に含む。
【００３７】
好適には、方法は、各クラスと関連付けられた複数の所定の周辺デバイスを示す第２のデータを、コンピューティングデバイスによってアクセス可能なメモリに記憶することを更に含む。
【００３８】
好適には、方法は、第１のデータを受信する前に、どの周辺デバイスがコンピューティ

ングデバイスに接続されているかを問い合わせるクエリを、モジュールに送信することを更に含む。

【００３９】

好適には、方法は、コンピューティングデバイス上のソフトウェアコンテナ要素を管理する責任を有するソフトウェアアプリケーションにおいて第１のデータを受信することを更に含む。

【００４０】

好適には、方法は、接続された周辺デバイスと通信するように構成されたモジュールとして、モジュールを提供することを更に含む。

【００４１】

好適には、方法は、第１の実行可能なソフトウェアを含むソフトウェアコンテナ要素として、モジュールを提供することであって、任意にモジュールがデバイスプラグインモジュールである、提供することを更に含む。

【００４２】

好適には、方法は、第１のデータを受信することに応答して、モジュールを指令する少なくとも１つのコマンドをモジュールに送信して、ソフトウェアコンテナ要素への割り当てのために、各クラスと関連付けられた複数の所定の周辺デバイスの各々を準備することを更に含む。

【００４３】

好適には、方法は、モジュールから、各クラスと関連付けられた複数の所定の周辺デバイスの各々についての少なくともそれぞれの周辺デバイス経路を示す第３のデータを受信することを更に含む。

【００４４】

好適には、方法は、各クラスと関連付けられた複数の所定の周辺デバイスの各々について、少なくともそれぞれの周辺デバイス経路を表すデータとして、周辺デバイス接続データを提供することを更に含む。

【００４５】

好適には、方法は、ソフトウェアコンテナ要素と関連付けられた制御グループに周辺デバイス接続データを記憶することを更に含む。

【００４６】

好適には、方法は、第２の実行可能な命令の実行時に、ソフトウェアコンテナ要素によって、各クラスと関連付けられた複数の所定の周辺デバイスのうちのどの周辺デバイスがコンピューティングデバイスに接続されているかを決定することを更に含む。

【００４７】

好適には、方法は、決定することに応答して、ソフトウェアコンテナ要素によって、コンピューティングデバイスに接続された各周辺デバイスを初期化することを更に含む。

【００４８】

好適には、方法は、初期化することに応答して、ソフトウェアコンテナ要素によって、接続された周辺デバイスのうちの少なくとも１つに１つ以上のコマンドを送信することを更に含む。

【００４９】

好適には、方法は、コンピューティング環境に関係なく実行可能であるソフトウェアとして、第１の実行可能なソフトウェア及び第２の実行可能なソフトウェアを提供することを更に含む。

【００５０】

好適には、方法は、モジュール及びソフトウェアコンテナ要素をそれぞれ介して、隔離されたコンピューティング環境で第１の実行可能なソフトウェア及び第２の実行可能なソフトウェアを実行することを更に含む。

【００５１】

本発明の第２の態様によれば、コンピューティングデバイス上のソフトウェアコンテナ

要素を管理する責任を有するソフトウェアアプリケーションを実行するように構成された１つ以上のプロセッサを備える、コンピューティングデバイスが提供されており、ソフトウェアアプリケーションは、コンピューティングデバイスの１つ以上のプロセッサ上で実行可能である第１の実行可能なソフトウェアを含むモジュールから、コンピューティングデバイスに接続可能な周辺デバイスの少なくとも１つのクラスを示す第１のデータを受信することであって、各クラスが、複数の所定の周辺デバイスと関連付けられている、受信することと、コンピューティングデバイスのその１つ以上のプロセッサ上で実行可能である第２の実行可能なソフトウェアを含む、ソフトウェアコンテナ要素を提供することであって、ソフトウェアコンテナ要素が、ソフトウェアコンテナ要素が各クラスと関連付けられた複数の所定の周辺デバイスの各周辺デバイスにアクセスし、かつそれと通信することを可能にする、周辺デバイス接続データと関連付けられている、提供することと、を行うように構成されている。

【００５２】

好適には、コンピューティングデバイスは、ポイントオブセール端末又はセルフサービス端末である。

【００５３】

本発明の第３の態様によれば、命令を含むコンピュータプログラムが提供されており、命令は、コンピューティングデバイスによって実行されたときに、コンピューティングデバイスに、本発明の第１の態様によって定義される方法のステップを行わせる。

【００５４】

本発明の特定の実施形態は、ソフトウェアコンテナ要素を、ソフトウェアコンテナ要素が、端末に接続され得る所定の周辺デバイスのセットのいずれかと通信することを可能にする、周辺デバイス接続データとともに提供する方法論を提供することを助けることである。

【００５５】

本発明の特定の実施形態は、ＳＳＴ又はＰＯＳ端末に接続可能な周辺デバイスのクラスをアドバタイズする、デバイスプラグインコンテナを提供することを助けることであり、端末は、このクラス内の全てのデバイスにアクセスできるソフトウェアコンテナを実行する。

【００５６】

本発明の特定の実施形態の目的は、アクセスが許可されている一連のダミーデバイスノードを表すデータを含む、ソフトウェアコンテナを提供することを助けることである。ダミーデバイスノードは、周辺デバイスの特定のクラスの現在及び将来の全てのデバイスに対して作成される。接続された周辺デバイスは、ダミーデバイスノードのうちの１つでアクセス可能であり、ソフトウェアコンテナは、どのノードが使用されているか、及びどのデバイスが接続されているかを決定する責任を有する。

【００５７】

本発明の特定の実施形態の目的は、ソフトウェアコンテナを実行し、かつプラグアンドプレイ及びアンプラグ－リプラグ機能を有する、コンピューティングデバイスを提供することを助けることである。

【００５８】

本発明の特定の実施形態の目的は、小売デバイスクラスと関連付けられた全ての周辺デバイスへのアクセスを有するように構成されたソフトウェアコンテナを提供することを助けることである。

【００５９】

本発明の特定の実施形態の目的は、周辺デバイスが接続されている、又は接続され得る一連のデバイス経路へのアクセスを有するように構成されたソフトウェアコンテナを実行するために必要なデータをコンテナエンジンに提供することを助けることである。

【００６０】

本発明の特定の実施形態の目的は、周辺デバイスの抽象的なセットへのアクセスを有す

るソフトウェアコンテナを提供することを助けることである。
【００６１】
ここで、本発明の実施形態を、例としてのみ、添付図面を参照して以降で説明する。
【００６２】
図面では、同様の参照番号は同様の部品を指す。
【図面の簡単な説明】
【００６３】
【図１】コンピューティングシステムを示す。
【図２】Ｋｕｂｅｒｎｅｔｅｓオーケストレーションプラットフォームの制御下にあるコンピューティングシステムを示す。
【図３】いくつかのソフトウェアコンテナを実行するセルフサービス端末のためのハードウェア及びソフトウェアアーキテクチャを示す。
【図４】セルフサービス端末上で実行されるソフトウェアコンテナ要素を示す。
【図５】複数のセルフサービス端末と通信するサーバ上で実行されるソフトウェアを示す。
【図６】実行可能な画像ファイルの層を示す。
【図７】周辺デバイス接続データを含むソフトウェアコンテナを提供する、コンピューティングシステムのいくつかの構成要素を示す。
【図８】あるクラスの周辺デバイスの各周辺デバイスへのアクセス及び通信を有するように構成されたソフトウェアコンテナがどのように提供されるかを説明する、フローチャートを示す。
【発明を実施するための形態】
【００６４】
図１は、コンピューティングシステム１００を示す。コンピューティングシステム１００には、３つのセルフサービス端末（ＳＳＴ）１１０１、１１０２、１１０３がある。ＳＳＴは、コンピューティングデバイスの一例である。
【００６５】
本発明のある特定の他の実施形態では、コンピューティングデバイスは、ポイントオブセール（ＰＯＳ）端末、現金自動預け払い機（ＡＴＭ）、パーソナルコンピュータ、ラップトップ、タブレットなどであってもよい。各ＳＳＴは、１つ以上のプロセッサ１１２、及び少なくとも１つのメモリ１１４を含む。メモリは、非一時的コンピュータ可読記憶媒体である。メモリ１１４は、ＳＳＴのプロセッサ１１２によって実行可能である実行可能なソフトウェアを記憶する。各ＳＳＴはまた、サーバと通信するための通信インターフェース（図示せず）、及び接続された周辺デバイスと通信するための１つ以上の通信インターフェース（図示せず）を含み得る。
【００６６】
図１に示されるシステムでは、スキャナ周辺デバイス１２０１及びスケール周辺デバイス１２０２が、第１のＳＳＴ１１０１に接続されている。また、プリンタ周辺デバイス１２０３及びスキャナ周辺デバイス１２０４が、第２のＳＳＴ１１０２に接続されている。また、スケール周辺デバイス１２０５、プリンタ周辺デバイス１２０６、及びスキャナ周辺デバイス１２０７が、第３のＳＳＴ１１０３に接続されている。
【００６７】
本発明のある特定の他の実施形態では、各ＳＳＴは、周辺デバイスの異なる組み合わせに接続され得ることが理解されよう。各周辺デバイスは、有線インターフェース１２２を介して接続されるＳＳＴと通信し得る。本発明のある特定の他の実施形態では、インターフェースは、無線、又は有線と無線との組み合わせであってもよいことが理解されよう。
【００６８】
各ＳＳＴは、ネットワーク１４０を介してサーバ１３０と通信する。サーバもまた、コンピューティングデバイスの一例である。ネットワーク１４０は、有線、無線、又は有線と無線との組み合わせであってもよい。サーバ１３０もまた、１つ以上のプロセッサ１３２、及び少なくとも１つのメモリ１３４を含む。メモリ１３４もまた、非一時的コンピュ

ータ可読記憶媒体である。メモリ１３４は、サーバのプロセッサによって実行可能である実行可能なソフトウェアを記憶する。ＳＳＴ及びサーバの実行可能なソフトウェアについて、以下でより詳細に説明する。

【００６９】

図２は、コンピューティングシステム２００を示す。コンピューティングシステムは、Ｋｕｂｅｒｎｅｔｅｓコンテナオーケストレーションプラットフォームの制御下にある、いくつかの構成要素を有する。そのため、本システムは、Ｋｕｂｅｒｎｅｔｅｓクラスタと称され得る。Ｋｕｂｅｒｎｅｔｅｓクラスタは、Ｋｕｂｅｒｎｅｔｅｓマスタ２１５を走らせるサーバ２１０と、それぞれのＫｕｂｅｒｎｅｔｅｓワーカ２３０１、２３０２を走らせるセルフサービス端末（ＳＳＴ）２２０１、２２０２と、を含む。

【００７０】

サーバ２１０は、物理サーバであっても、クラウドサーバであってもよいことが理解されよう。サーバ２１０及びＳＳＴは、ローカルエリアネットワーク又はインターネットなどの、ネットワーク２０５を経由して通信する。ネットワークは、有線及び／又は無線であってもよい。ＳＳＴ以外のデバイスをネットワークに接続して、Ｋｕｂｅｒｎｅｔｅｓワーカを走らせることができることが理解されよう。

【００７１】

サーバ２１０上で走るＫｕｂｅｒｎｅｔｅｓマスタ２１５は、Ｋｕｂｅｒｎｅｔｅｓクラスタを管理するＡＰＩサーバ２１６を含む。ＡＰＩサーバ２１６は、マスタ２１５の他の内部構成要素から受信する情報に基づいてコマンドを発行し、かつＫｕｂｅｒｎｅｔｅｓワーカ２３０１、２３０２上で走るｋｕｂｅｃｔｌ２１２及びｋｕｂｅｌｅｔ（ＳＳＴ２　２２０２上のｋｕｂｅｌｅｔ２３１など）などの外部構成要素とインターフェース接続する。Ｅｔｃｄ２１７は、クラスタの構成などの情報を記憶するＫｕｂｅｒｎｅｔｅｓクラスタのための配信データベースである。

【００７２】

Ｅｔｃｄ２１７はまた、Ｋｕｂｅｒｎｅｔｅｓワーカ２３０１、２３０２の望ましい状態、及びＫｕｂｅｒｎｅｔｅｓワーカ２３０１、２３０２の実際の状態を記憶する。ある状態は、クラスタ内の各Ｋｕｂｅｒｎｅｔｅｓワーカ２３０１、２３０２上で走っている、ポッド（ＳＳＴ２　２２０２上のポッド３　２３５など）及びポッドのコンテナ（ポッド２３５のコンテナ２３６など）の指標であると理解され得る。

【００７３】

スケジューラ２１８は、新しいポッドをＫｕｂｅｒｎｅｔｅｓワーカ上でいつ走らせるかを監視し、次いで、当該ポッドをどのＫｕｂｅｒｎｅｔｅｓワーカ上に展開するかを決定する。コントローラマネージャ２１９は、ｅｔｃｄ２１７に対して指定された望ましい状態に近づくようにＫｕｂｅｒｎｅｔｅｓワーカ２３０１、２３０２の実際の状態を移動させることを試みるコントローラプロセスを走らせる。

【００７４】

マスタ２１５はまた、ｋｕｂｅｃｔｌ２１２、ＡＰＩサーバ２１６を介してＫｕｂｅｒｎｅｔｅｓクラスタと通信するためのコマンドラインツール、及びオペレータインターフェース２１１を含む。

【００７５】

Ｋｕｂｅｒｎｅｔｅｓクラスタ内に位置する各Ｋｕｂｅｒｎｅｔｅｓワーカ２３０１、２３０２は、ＳＳＴ上で走る。

【００７６】

本発明のある特定の実施形態によれば、ワーカは、ＳＳＴの仮想マシン上で走らせることができる。ワーカ２３０は、ネットワーク２０５を介して、他のワーカ２３０及びマスタ２１５と通信することができる。各ワーカ２３０は、ワーカ２３０の動作を管理するｋｕｂｅｌｅｔを有する。ｋｕｂｅｌｅｔ（ＳＳＴ２２０２上のｋｕｂｅｌｅｔ２３１など）は、ワーカ２３０２の他の構成要素にコマンドを発行し、ワーカ（ポッド２３５など）及びワーカのコンテナ（コンテナ２３６など）上で走るポッドを監視し、かつＡＰＩサー

バ２１６と通信する。ｋｕｂｅｌｅｔ２３１は、展開ファイルを受信し、これらの展開ファイルに記述されたコンテナ２３６が走っており、かつ正常であることを保証する。

【００７７】

ｋｕｂｅ－ｐｒｏｘｙ（ｋｕｂｅ－ｐｒｏｘｙ２３２など）は、ポッドが、同じＫｕｂｅｒｎｅｔｅｓワーカ及び異なるワーカの両方で通信することを可能にする、ネットワークプロキシである。

【００７８】

コンテナエンジン（エンジン２３３など）は、コンテナを走らせ、かつ管理して、ｋｕｂｅｌｅｔからのコマンド、及びレジストリからのコンテナ画像を受信する。コンテナエンジンは、内部にランタイムが位置するＫｕｂｅｒｎｅｔｅｓワーカ内のコンテナを走らせることに関与するコンテナランタイム（コンテナランタイム２３４など）に渡されるコンテナメタデータを準備する。

【００７９】

Ｋｕｂｅｒｎｅｔｅｓマスタ２１５のＡＰＩサーバ２１６によってポッドがＫｕｂｅｒｎｅｔｅｓワーカに展開された後、任意のＫｕｂｅｒｎｅｔｅｓワーカ内にポッドが存在する。ポッドは一般に単一のコンテナを含むが、ポッドは、ストレージ及びネットワークリソースを共有することになる、類似の機能を有する複数のコンテナを含み得る。ポッドは、ｋｕｂｅｌｅｔを介して、ワーカが利用可能な特定のリソースへのアクセスを要求するか、又はｋｕｂｅ－ｐｒｏｘｙを使用することによって、他のポッドと通信することができる。

【００８０】

図３は、いくつかのソフトウェアコンテナ要素を実行するように構成されたセルフサービス端末のためのハードウェア及びソフトウェアアーキテクチャ３００を示す。図３では、基礎となるハードウェアは、ＳＳＴ３１０である。

【００８１】

これは、図１又は図２に関して記述されるＳＳＴのうちの１つであってもよい。上述のように、ＳＳＴは、１つ以上のプロセッサ、及び少なくとも１つのメモリを含む。メモリは、プロセッサによって実行可能である実行可能なソフトウェアを記憶する。

【００８２】

実行可能なソフトウェアには、（Ｕｎｉｘ又はＵｂｕｎｔｕなどのような）ホストオペレーティングシステムの一部であってもよい、Ｌｉｎｕｘカーネル３２０が含まれる。本発明のある特定の他の実施形態では、他のカーネル及び他のホストオペレーティングシステム（Ｗｉｎｄｏｗｓ、Ｍａｃなど）を利用することができることが理解されよう。実行可能なソフトウェアの一部として、コンテナエンジン３３０も含まれる。

【００８３】

コンテナエンジンは、ユーザ要求、又は（Ｋｕｂｅｒｎｅｔｅｓ、Ｓｗａｒｍ、Ｍｅｓｏｓなどのような）コンテナオーケストレータのＡＰＩサーバからの要求を受け入れ、レジストリから（特定の画像形式の）画像ファイルを引き出し、コンテナマウントポイントを準備し、ソフトウェアコンテナを実行するのに必要なメタデータを準備し、かつコンテナランタイムを呼び出すことに関与する。

【００８４】

コンテナランタイム（図示せず）は、コンテナエンジンの一部である。コンテナランタイム（ｒｕｎｃ、ｃｏｎｔａｉｎｅｒｄ、ｃｒｕｎ、ｒａｉｌｃａｒ、ｋａｔａｃｏｎｔａｉｎｅｒｓなどのような）は、いくつかのコンテナ画像ファイルを実行し、かつそれによって、いくつかのソフトウェアコンテナを作成するために、マウントポイント、及びコンテナエンジンによって提供されるメタデータを使用し、コンピューティングデバイス上で走っているＬｉｎｕｘカーネル３２０と通信する。

【００８５】

図３に示されるソフトウェアコンテナの各々についての実行可能な画像ファイルは、レジストリからダウンロードされ得る。図３では、４つのソフトウェアコンテナ要素が示さ

れている。第１のソフトウェアコンテナ要素３７０は、デバイスプラグインコンテナと称される。デバイスプラグインコンテナは、アプリケーションソフトウェア３７２、並びに関連付けられたバイナリ及びライブラリ３７４を含む（バイナリ及びライブラリは、ソフトウェア依存性と称され得る）。

【００８６】

デバイスプラグインコンテナ内で走るアプリケーションは、以下に説明するように、どの周辺デバイスがＳＳＴに接続されているかを知らせることに関与する。デバイスプラグインコンテナは、単一の層のみを含み得る。したがって、デバイスプラグインコンテナは、全ての依存性が単一の（多くの場合、Ｇｏｌａｎｇ）実行可能ファイルに組み込まれる、「ディストロレス」コンテナと呼んでもよい。

【００８７】

第２のソフトウェアコンテナ要素３４０は、デバイスサーバコンテナと称される。デバイスサーバコンテナは、アプリケーションソフトウェア３４２、並びに関連付けられたバイナリ及びライブラリ３４４を含む（バイナリ及びライブラリは、ソフトウェア依存性と称され得る）。

【００８８】

デバイスサーバコンテナ内で走るアプリケーションは、低いレベルで、ＳＳＴに接続された周辺デバイスのうちの１つ以上を制御すること、構成すること、又は別様に当該周辺デバイスにアクセスすること、並びにネットワークを横切って、ＳＳＴの他の構成要素に事業レベルの機能を公開することに関与する。例えば、デバイスサーバは、「ＵＳＢ」プロトコルを介して、スキャナ（低レベル）と通信し、スキャンされたバーコード（事業レベル）を他の構成要素に報告し得る。

【００８９】

デバイスサーバコンテナ内のソフトウェアは、周辺デバイスパスにアクセスし、ひいては、周辺デバイスを使用するか、又は周辺デバイスと相互作用することができる。第３のソフトウェアコンテナ要素３５０は、ＩＮＩＴコンテナと称される。ＩＮＩＴコンテナは、アプリケーションソフトウェア３５２、並びと関連付けられたバイナリ及びライブラリ３５４を含む（バイナリ及びライブラリは、ソフトウェア依存性と称され得る）。

【００９０】

ＩＮＩＴコンテナで実行されるアプリケーションは、そのポッド内で初期化されてから、メイン（ＩＮＩＴではない）コンテナを開始する。ＩＮＩＴコンテナは、Ｋｕｂｅｒｎｅｔｅｓシステムの概念であるが、最初に（すなわち、他のコンテナの前に）実行されるように構成されたコンテナもまた、他のコンテナオーケストレーションプラットフォームにおいて利用され得ることが理解されよう。

【００９１】

第４のソフトウェアコンテナ要素３６０は、エンドポイントコンテナと称される。エンドポイントコンテナは、アプリケーションソフトウェア３６２、並びに関連付けられたバイナリ及びライブラリ３６４を含む（バイナリ及びライブラリは、ソフトウェア依存性と称され得る）。エンドポイントコンテナ内で走るアプリケーションにより、マザーボードのユニバーサル固有識別子（ＵＵＩＤ）などの、ＳＳＴに関する情報が、Ｋｕｂｅｒｎｅｔｅｓクラスタの残りの部分に対して利用可能になる。

【００９２】

図３に見ることができるように、各ソフトウェアコンテナ要素は、独自のバイナリ及びライブラリ（ｂｉｎ／ｌｉｂ）を有する。しかしながら、本発明のある特定の他の実施形態によると、コンテナの任意の組み合わせは、ｂｉｎ／ｌｉｂを共有し得ることが理解されよう。

【００９３】

ここで図４を参照すると、セルフサービス端末４００、及びセルフサービス端末上で実行されるように構成された（ＳＳＴの１つ以上のプロセッサ上にある）ソフトウェアコンテナ要素が示されている。

【００９４】
ＳＳＴ上のソフトウェアコンテナの各々は、（画像ファイルによって定義される）実行可能なソフトウェアを包含する。実行可能なソフトウェアは、隔離されたコンピューティング環境でソフトウェアが実行されるような方法で、コンテナ内で実行される。ソフトウェアは、動作するためにＳＳＴ上で同様に実行される他のソフトウェアコンテナのいずれにも依存しないという意味において、隔離されている。
【００９５】
ソフトウェアは、独自のコンピューティング環境で効果的に実行され、かつ事前定義された通信経路を介して、この環境の外側にあるハードウェア／ソフトウェアと通信する。ソフトウェアコンテナ内に実行可能なソフトウェアを提供することは、コンピューティング環境に関係なくソフトウェアを実行することができることを意味する。
【００９６】
図４では、コンテナは、Ｋｕｂｅｒｎｅｔｅｓコンテナオーケストレーションプラットフォームを使用して管理されている。図４に示されるＳＳＴは、上記の図１～図３のいずれかを参照して説明されるＳＳＴであってもよい。ＳＳＴ４００は、ホストオペレーティングシステムの一部としてＬｉｎｕｘカーネル４０５を走らせる。
【００９７】
当然のことながら、本発明のある特定の他の実施形態に従って、他のオペレーティングシステムを使用することができる。Ｋｕｂｅｒｎｅｔｅｓシステムを使用して、セルフサービス端末は、Ｋｕｂｅｒｎｅｔｅｓワーカ４１０と称されるソフトウェアを実行する。
【００９８】
Ｋｕｂｅｒｎｅｔｅｓワーカはまた、ノードと称されることもある。第１のソフトウェアコンテナ要素４５０、第２のソフトウェアコンテナ要素４３０、第３のソフトウェアコンテナ要素４２０、及び第４のソフトウェアコンテナ要素４４０が、Ｋｕｂｅｒｎｅｔｅｓワーカ４１０内に含まれる。Ｋｕｂｅｒｎｅｔｅｓプラットフォームは、図２を参照して説明されるように、これらのコンテナを管理することに関与する。
【００９９】
第１のソフトウェアコンテナ、第２のソフトウェアコンテナ、第３のソフトウェアコンテナ、及び第４のソフトウェアコンテナは、図３を参照して説明されるのと同じコンテナであってもよい。セルフサービス端末４００はまた、Ｋｕｂｅｒｎｅｔｅｓワーカ４１０の外部を実行する追加のソフトウェア（図示せず）を含む。
【０１００】
Ｋｕｂｅｒｎｅｔｅｓワーカ４１０では、第２のソフトウェアコンテナ４３０及び第３のソフトウェアコンテナ４２０は、デバイスサーバポッド４２５と称される単一のポッド内で実行される。第３のソフトウェアコンテナ４２０はＩＮＩＴタイプのものであるため、デバイスサーバポッド内で実行される（すなわち、デバイスサーバコンテナの前に実行される）第１のコンテナである。
【０１０１】
第４のソフトウェアコンテナ４４０は、エンドポイントポッド４４５と称される単一のポッド内で実行される。第１のソフトウェアコンテナ４５０は、デバイスプラグインポッド４５５と称される単一のポッド内で実行される。これらのポッドの各々の作成は、当業者によって理解されるように、３つの異なるポッド仕様ファイル（すなわち、ｄｅｐｌｏｙｍｅｎｔ．ＹＡＭＬファイル）によって定義される。
【０１０２】
ポッドは、共有ストレージ及びネットワークリソースを備えたコンテナ、並びにポッド内でコンテナをどのように走らせるかについての仕様を提供するために、Ｋｕｂｅｒｎｅｔｅｓで使用される。ＳＳＴ４００の動作中、これらのポッド／コンテナの各々は、コンテナエンジン（図示せず）のコンテナランタイム（図示せず）によって実行される。これらのコンテナの各々と関連付けられた画像ファイルは、レジストリからダウンロードされ得る。

【０１０３】
図５は、Ｋｕｂｅｒｎｅｔｅｓマスタ及びＫｕｂｅｒｎｅｔｅｓワーカを隔離して、走らせているサーバ５００を示す。サーバは、複数のＳＳＴ（図５には示されていない）と通信し得る。サーバは、１つ以上のプロセッサ（図示せず）、及び少なくとも１つのメモリ（図示せず）を有する。メモリは、ランタイム時にプロセッサによって実行される実行可能なソフトウェアを記憶する。
【０１０４】
実行可能なソフトウェアは、（ホストＯＳの）Ｌｉｎｕｘカーネル５１０を含む。本発明のある特定の他の実施形態では、他のオペレーティングシステムが使用され得ることが理解されよう。
【０１０５】
実行可能なソフトウェアはまた、Ｋｕｂｅｒｎｅｔｅｓマスタ５２０を含む。Ｋｕｂｅｒｎｅｔｅｓマスタは、図２を参照して上述したのと同様の構成要素を含む。これらは、ＡＰＩサーバ５２２、スケジューラ５２４、コントローラマネージャ５２６、及びｅｔｃｄデータベース５２８である。
【０１０６】
Ｋｕｂｅｒｎｅｔｅｓワーカ５３０もまた、サーバ上で実行される。Ｋｕｂｅｒｎｅｔｅｓワーカ５３０は、３つのポッドを含み、ポッド自体は、ソフトウェアコンテナ要素を含む。サーバ上の第１のポッドは、動的コンテナプロキシポッド５３２である。
【０１０７】
このポッドは、対応する動的コンテナプロキシソフトウェアコンテナを含む。動的プロキシソフトウェアコンテナは、実行可能な画像ファイルに対するコンテナエンジンからの要求を受信することに関与する。
【０１０８】
サーバ上の第２のポッドは、上流のコンテナレジストリポッド５３４である。このポッドは、対応する上流のコンテナレジストリソフトウェアコンテナを含む。レジストリコンテナは、画像ファイルを記憶し、かつ要求に応じて、これらの画像ファイルをコンテナエンジンに提供することに関与する。サーバ上の第３のポッドは、事業ロジックポッド５３６である。このポッドは、対応する事業ロジックソフトウェアコンテナを含む。
【０１０９】
図６は、ソフトウェアコンテナとして実行するための実行可能な画像ファイル６００の概略図を示す。図６に示される画像ファイルは、図４のデバイスサーバコンテナを提供するために実行可能である。
【０１１０】
図６に見ることができるように、実行可能な画像ファイルは、ベースオペレーティングシステム層６０５、Ｊａｖａ層６１０、ドライバ層６１５、共通層６２０、ユーティリティ層６２５、スキャナ層６３０、スケール層６３５、プリンタ層６４０、及びデバイスサーバ層６４５を有する。これらの層を有するこの画像ファイルは、レジストリに記憶され、かつ端末のコンテナエンジンからの要求に応じてダウンロードされ得る。ベースＯＳ層は、ソフトウェアコンテナ（例えば、Ｕｂｕｎｔｕ）内で利用されるオペレーティングシステムを定義する。
【０１１１】
しかしながら、ベースＯＳ層は、本発明のある特定の他の実施形態で必要とされない場合がある。Ｊａｖａ層は、Ｊａｖａプログラミング言語を解釈するための実行可能な命令を定義する。しかしながら、本発明のある特定の他の実施形態では、他のプログラミング言語が使用されている場合、Ｊａｖａ層は必要とされない場合があることが理解されよう。
【０１１２】
ドライバ層は、画像にユーザ空間ドライバをインストールすることに関与する実行可能な命令を定義する、任意選択の層である。ユーザ空間ドライバの一例は、低レベルのＵＳＢヘルパであってもよい。

【０１１３】
一部の画像は、いずれのユーザ空間ドライバも必要としない場合があることが理解されよう。共通層は、他の層によって共有又は使用されるフレームワーク構成要素を包含する実行可能な命令を定義する。かかるフレームワークの一例は、他の構成要素からのログを制御及びキャプチャするために使用されるロギングフレームワークであってもよい。
【０１１４】
ユーティリティ層は、ユーザ向けユーティリティ及びツールを含む実行可能な命令を定義する、任意選択の層である。かかるツールの一例は、ＳＳＴ上にインストールされているデバイスのリストを閲覧し、かつ診断チェックを走らせるなどの動作により、これらのデバイスと相互作用するために使用され得る、システムメンテナンスユーティリティであってもよい。
【０１１５】
ドライバ層、共通層、及びユーティリティ層は、ソフトウェア依存性と集合的に称され得る。ドライバ層及び／又は共通層及び／又はユーティリティ層は、本発明のある特定の実施形態において必要とされない場合があることが理解されよう。
【０１１６】
スキャナ層は、スキャナ周辺デバイスと通信するための実行可能な命令を定義し、スケール層は、スケール周辺デバイスと通信するための実行可能な命令を定義し、プリンタ層は、プリンタ周辺デバイスと通信するための実行可能な命令を定義する。
【０１１７】
これらの層は、周辺デバイスドライバ画像層と称され得る。スキャナ、スケール、及びプリンタの層は、特定のスキャナ、スケール、又はプリンタと関連付けられた層であることが理解されよう。一例として、スキャナは、ＮＣＲ２３５６ハンドヘルドスキャナであってもよく、スケールは、ＮＣＲ（本願出願人の登録商標）８７８９スケールであってもよく、プリンタは、ＮＣＲ７１６９サーマルレシートプリンタであってもよい。
【０１１８】
本発明のある特定の他の実施形態では、他の周辺デバイス（バーコードリーダ、カメラなど）と関連付けられた層が必要となる場合があることも理解されよう。
【０１１９】
図６に示される周辺デバイス画像層は、必ずしも必要ではない場合があることも理解されよう。デバイスサーバ層は、低いレベルで、ＳＳＴに接続された周辺デバイスのうちの１つ以上を制御し、構成し、又は別様に当該周辺デバイスにアクセスし、かつネットワークを横切って、ＳＳＴの他の構成要素に事業レベルの機能を公開するための実行可能な命令を定義する。
【０１２０】
図７は、コンピューティングシステム７００の特定の構成要素を示す。図７に示す構成要素は、図１～図４を参照して説明したＳＳＴなどのＳＳＴ又はＰＯＳ端末上に存在する。より詳細には、図７は、ｋｕｂｅｌｅｔ７１０、リソースマネージャ７２０、デバイスプラグイン要素７３０、コンテナ化されたランタイム要素７４０、及び特定の接続された周辺デバイス７５０を示す。ｋｕｂｅｌｅｔは、Ｋｕｂｅｒｎｅｔｅｓオーケストレーションプラットフォームの一部としてＳＳＴで使用されるソフトウェアアプリケーションである。
【０１２１】
このソフトウェアアプリケーションは、ＳＳＴ上で動作するソフトウェアコンテナの管理に使用される。リソースマネージャ（明確にするために別個に表示）は、ｋｕｂｅｌｅｔの一部である。リソースマネージャは、ノード上で利用可能なリソースを列挙し、Ｋｕｂｅｒｎｅｔｅｓシステムの他の構成要素、特にＫｕｂｅｒｎｅｔｅｓスケジューラへのＫｕｂｅｌｅｔを介してそれらを識別する責任を有する。リソースには、ＲＡＭ、ＣＰＵ、ローカル永続ストレージなどのリソースを含めることができ、カスタムリソースで拡張することができる。

【０１２２】
カスタムリソースは、低レベル又は汎用リソースとすることができる。例えば、カスタムリソースは、一般的に、バス２上のポート１０にあるＵＳＢデバイスであってもよく、又は、より機能的に明示的であってもよく、例えば、「バーコードスキャナ」であってもよい。リソースマネージャは、利用可能である特定のタイプのリソース数など、リソース容量の管理する責任をも有する。
【０１２３】
リソースマネージャはまた、コンテナに取り付けるためのデバイスを準備するか、コンテナから取り外した後にデバイスをきれいにする責任を有する。例えば、デバイスが、コンテナに取り付けられる前に、工場出荷時の状態にリセットされるときなどである。デバイスプラグインは、ソフトウェアコンテナとして実行されてもよい。
【０１２４】
しかしながら、本発明の特定の他の実施形態によれば、デバイスプラグインは、ソフトウェアコンテナとして実行される必要はないことが理解されよう。デバイスプラグイン要素は、ＳＳＴ又はＰＯＳ端末のプロセッサ（又は複数のプロセッサ）上で実行される実行可能な命令を含む、
モジュールである。デバイスプラグインは、カスタムリソースを管理する責任を有する。デバイスプラグインモジュールは、接続された周辺デバイス、及びｋｕｂｅｌｅｔのリソースマネージャと通信する。デバイスプラグインは、クラス内の全ての周辺デバイスにアクセスできるソフトウェアコンテナを提供できる情報を、Ｋｕｂｅｌｅｔに提供する。
【０１２５】
これについては以下で詳しく説明する。コンテナ化されたランタイム要素は、コンテナエンジンの一部である、コンテナランタイムである。ランタイムは、画像ファイルをソフトウェアコンテナとして実行する。ランタイムには、いくつかの必要なソフトウェアコンテナを実行するために、ｋｕｂｅｌｅｔからの情報が提供される。ここで、これらの構成要素がどのように相互作用するかについて説明する。
【０１２６】
ＳＳＴの起動時、及び図７に示すソフトウェア構成要素が実行され、ＳＳＴ上で行われた後、Ｋｕｂｅｌｅｔ７１０は、リソースマネージャを呼び出し、利用可能なリソースを見つける。
【０１２７】
リソースマネージャ７２０は、標準リソース（ＲＡＭ、ＣＰＵなど）を報告し、デバイスプラグインを更に呼び出して、追加のカスタムリソースを検出する。１つ以上のデバイスプラグインが存在し得ることに留意されたい。
【０１２８】
リソースマネージャは、どのデバイスがＳＳＴに物理的に接続されているかを問い合わせるクエリをデバイスプラグインに送信することで、これを行う。このクエリはデバイスプラグインで受信され、このクエリに対する応答として、デバイスプラグインは、カスタムリソースを列挙し、カスタムリソースが何であり、各リソースのうちいくつがＫｕｂｅｌｅｔに戻って利用可能であるかを報告する。
【０１２９】
特に、デバイスプラグイン７３０は、ＳＳＴに接続可能な全ての既知のプリンタ、スキャナ、計量器、ディスプレイ、及び小売デバイスなどの、１つ以上の、おそらくは重複した抽象的なクラスのデバイスを識別するデータを、リソースマネージャに提供する。この例では、小売デバイスクラスは、プリンタ、スキャナ、及び計量器を含む、単一のクラスであり得る。
【０１３０】
このクラスには、ディスプレイなどの特定のタイプのデバイスが含まれていない場合がある。ｋｕｂｅｌｅｔは、Ｋｕｂｅｒｎｅｔｅｓ ＡＰＩサーバデータモデル内のＫｕｂｅｒｎｅｔｅｓワーカオブジェクトに対するリソース（標準及びカスタム）を記録する。

【０１３１】
そこから、利用可能なリソースに関する情報を、Ｋｕｂｅｒｎｅｔｅｓスケジューラなどの構成要素が使用して、作業負荷（ポッド）スケジューリングに関する決定を行うことができる。例えば、作業負荷がバーコードスキャナが必要であると識別した場合、そのクラスに利用可能な少なくとも１つのデバイスがあるＫｕｂｅｒｎｅｔｅｓワーカにスケジュールすることができる。
【０１３２】
クラス（小売デバイスクラスなど）と関連付けられた小売周辺デバイスの定義は、ＳＳＴ上のメモリに記憶される。これは、例えば、構成マップ（ＣｏｎｆｉｇＭａｐ）に記憶された、特定のグループ（又はクラス）へのデバイスのメンバーシップを定義するルールの形態であり得る。構成マップ内のルールは、そのデバイス経路に基づいてクラスに入るデバイスを識別するための「正規表現」を含み得る。構成マップは、追加的又は代替的に、例えば、デバイスベンダー、クラス及びサブクラス（ＵＳＢ概念）によるフィルタリングをサポートしてもよい。
【０１３３】
本発明の特定の他の実施形態では、クラスと関連付けられた小売周辺デバイスのセットは、他の場所（例えば、サーバ上）に記憶され得ることが理解されよう。
【０１３４】
本発明の特定の他の実施形態では、デバイスプラグインは、複数のクラス（例えば、プリンタデバイスクラス、スキャナデバイスクラス、計量器デバイスクラスなど）を示すデータを、クエリに応答してリソースマネージャに送信し得ることがまた理解されよう。
【０１３５】
このような場合、リソースマネージャは、クラスの潜在的な重複の性質のために複雑であり得る特定のクラスにわたってリソースの可用性を管理する。例えば、ある作業負荷がすでに「スキャナ」に割り当てられており、別の作業負荷が「小売デバイス」を要求している場合、スキャナではない小売デバイスのサブセットを新しいコンテナに割り当てることのみが可能である。
【０１３６】
Ｋｕｂｅｒｎｅｔｅｓのスケジューラが、特定のＫｕｂｅｒｎｅｔｅｓワーカに関する特定のリソースを必要とするワークロード（ポッド）をスケジュールすることを決定したら、ｋｕｂｅｒｎｅｔｅｓは、再びリソースマネージャに話し、ポッドに標準リソースとカスタムリソースを割り当てる。カスタムリソースの場合、リソースマネージャは、デバイスプラグインと通信し、（以前はリソースマネージャにアドバタイズされていた）周辺デバイスのクラスを作業負荷に割り当てる準備が整うように要求する。
【０１３７】
これを行うために、リソースマネージャは、デバイスプラグインにコマンドを送信する。送信されるコマンドによって、デバイスプラグインに指令し、小売デバイスクラスと関連付けられた周辺デバイスの各々をポッド／コンテナに割り当てる準備をする。要求の受信に応答して、デバイスプラグインはまた、ポッドに割り当てられるクラス内の各デバイスに対する経路（すなわち、周辺デバイスがアクセスすることができる経路）を識別するさらなるデータを送信し、この情報をリソースマネージャに渡す。
【０１３８】
したがって、プラグインコンテナは、多数のダミーデバイス経路又はノード（多くの経路には接続される周辺デバイスが存在しないため）をリソースマネージャに効果的に送信する。これらは、ｋｕｂｅｌｅｔ及びコンテナランタイムと共有される。
【０１３９】
ランタイムは、制御グループ（ｃグループ）内の可能性のあるデバイス経路の各々を示すデータを記憶し、かつこのｃグループを、実行されるコンテナと関連付ける。これらの経路又はｃグループは、周辺デバイス接続データと呼ばれ得る。経路は、例えば、形式１８９：０、１８９：１、１８９：２などのＬｉｎｕｘのデバイスメジャー：マイナー番号

形式のｃグループに記憶されてもよい。次に、コンテナランタイムはこの情報を使用して、ポッド内のコンテナが、コンテナの実行時にデバイスにアクセスするために必要な権限を持っていることを確実にする。

【０１４０】

この一例として、デバイスプラグインは、各々が、‘／ｄｅｖ／ｄｕｍｍｙ／１’、‘／ｄｅｖ／ｄｕｍｍｙ／２’などで終わる経路を提供してもよく、これらの経路は、デバイスメジャー及びマイナー番号形式（１８９：０）、（１８９：１）などのｃグループに記憶されてもよい。

【０１４１】

これは、クラス内の必要な数の周辺デバイスに対して実行され得る。したがって、コンテナｃｇｒｏｕｐは、ダミーデバイス経路を記憶し、これは、接続されている実際のデバイスによって必要とされるときにのみ、コンテナから見えるようになる。言い換えれば、対応するＵＳＢポートに物理的に接続されているデバイスがある場合、コンテナは、依然として、デバイス‘／ｄｅｖ／ｂｕｓ／ｕｓｂ／００１／００２’が（１８９：１）に等しいことだけを見ることになる。

【０１４２】

これらのステップが実行されると、ｋｕｂｅｌｅｔは、特定のポッド仕様（展開ファイル）に従って、（スケジューラによって決定されるように）ソフトウェアコンテナ要素を実行するように、コンテナ化されたランタイム要素に指示する。

【０１４３】

次に、コンテナランタイムは、展開ファイルで指定された画像ファイルをダウンロードして実行することにより、ソフトウェアコンテナ要素を提供する。ソフトウェアコンテナ要素は、実行時に、コンテナが、ソフトウェアコンテナ要素が各クラスと関連付けられた複数の所定の周辺デバイスの各周辺デバイスにアクセスし、かつそれと通信することを可能にする、周辺デバイス接続データと関連付けられるように、ｃグループと関連付けられる。

【０１４４】

展開ファイルには、ソフトウェアコンテナ及び必要なリソースの実行可能画像ファイルの表示が含まれる。これには、小売デバイスクラスなどのカスタムリソースが含まれる場合がある。当然のことながら、ソフトウェアコンテナの開始後にデバイスが接続されても、新しいデバイスが位置する経路がコンテナのｃグループに定義されるため、デバイスはコンテナからアクセス可能となる。

【０１４５】

ソフトウェアコンテナが実行された後は、各露出したデバイス経路が、使用したいデバイスを表すかどうかを決定する責任を有する。これは、デバイス検出プロセスを実装するソフトウェアを使用することによって行うことができる。

【０１４６】

このようにして、実行されたソフトウェアコンテナは、クラスと関連付けられているどのデバイスがＳＳＴに実際に接続されているかを決定する。次に、ソフトウェアコンテナは、接続された周辺デバイスの各々を初期化する。接続された周辺デバイスが初期化されると、ソフトウェアコンテナ要素は次に、周辺デバイスの各々にコマンドを送信し得る。

【０１４７】

デバイス検出プロセスを実装するソフトウェアは、所定の間隔でソフトウェアコンテナ内で実行されるように構成されている。しかしながら、本発明の特定の他の実施形態では、デバイス検出プロセスがまた連続的に実行され得ることが理解されよう。

【０１４８】

したがって、新しい周辺デバイスがプラグインされる実施例では、デバイス検出プロセスを使用して、この新しいデバイスが存在していることを決定し、その後、デバイスを初期化する責任を有する。次に、新しいデバイスは、上述したように、コマンドを送信し得る。

【０１４９】
当然のことながら、上述のように、周辺デバイス接続データと関連付けられたソフトウェアコンテナ要素を提供することによって、ソフトウェアコンテナは、所定のクラスと関連付けられた周辺デバイスのセットの各々にアクセスすることができる。
【０１５０】
したがって、コンテナが最初に実行されたときに、経路がコンテナのｃグループに既に定義されているためにコンテナは依然としてこれらのデバイスにアクセスできるため、周辺デバイスが接続されているかどうかは重要ではない。
【０１５１】
このアプローチを取ることによって、ソフトウェアコンテナはまた、後日（ソフトウェアコンテナの実行後）にプラグインされる、又は（ソフトウェアコンテナの実行後に）アンプラグされ、かつリプラグされる、新しい周辺デバイスと通信することができる。このアンプラグ－リプラグイベントは、ソフトウェアアップデートの結果として仮想イベントとなる場合がある。
【０１５２】
したがって、ソフトウェアコンテナは、特権ありのアクセスを必要とせず、かつセキュリティを犠牲にすることなく、所定のクラスと関連付けられた全ての周辺デバイスといつでも通信する柔軟性を有する。
【０１５３】
図８は、周辺デバイス接続データと関連付けられたソフトウェアコンテナ要素が、セルフサービス端末（ＳＳＴ）上で初期化されたときに行われる、ある特定のステップのブロック図８００を示す。
【０１５４】
ＳＳＴは、図１～図７に示されるＳＳＴのいずれかであってもよい。周辺デバイス接続データと関連付けられたソフトウェアコンテナ要素は、第２のソフトウェアコンテナ要素と呼んでもよい。第２のソフトウェアコンテナ要素は、ＳＳＴのプロセッサ（又は複数のプロセッサ）上で実行される実行可能な命令を含む。
【０１５５】
ＳＳＴが最初に展開されるとき、第１のステップＳ８０５は、ＳＳＴのスタートアップ時に初期化されるべきコンテナのインスタンスの構成を指定する１つ以上の展開ファイルを作成することを伴う。展開ファイルは、他のものの中でも、ソフトウェアコンテナを提供するためにダウンロード及び実行される画像ファイルの名前を指定する。
【０１５６】
展開ファイルは、また、どのリソースがコンテナによってアクセス可能になるかを指定する。展開ファイルが作成されると、展開ファイルは、ＳＳＴと通信するサーバ上で走っているＫｕｂｅｒｎｅｔｅｓマスタのＡＰＩサーバにアップロードされる。
【０１５７】
Ｋｕｂｅｒｎｅｔｅｓマスタは、図２又は図５に関して示されるのと同じマスタであってもよい。ＡＰＩサーバは、展開ファイルを受信し、かつ展開ファイルを、Ｋｕｂｅｒｎｅｔｅｓマスタのｅｔｃｄデータベースに記憶する。ＳＳＴが最初に展開されたときに実行される別のステップＳ８１０は、ＳＳＴによってアクセス可能なメモリにデータを記憶することを伴う。データは、例えば、ＳＳＴ上又はＳＳＴと通信するサーバ上に記憶されてもよい。データは、あるクラスの周辺デバイスと関連付けられた特定の所定の周辺デバイスを示す。データは、構成マップとして記憶されてもよい。
【０１５８】
次のステップＳ８１５は、ＳＳＴの電源を入れることである。これは、店舗の開店時又は端末の再始動時など、毎日行われる場合がある。次いで、ホストＯＳ及びＫｕｂｅｒｎｅｔｅｓワーカ構成を含む、ＳＳＴ上のソフトウェアが、ＳＳＴのメモリからロードされ、ＳＳＴのプロセッサによって実行される。Ｋｕｂｅｒｎｅｔｅｓワーカ構成がロードされると、最初、走っているポッドはない。

【０１５９】
次のステップＳ８２０は、Ｋｕｂｅｒｎｅｔｅｓマスタのコントローラマネージャによって、ｅｔｃｄに記憶されている展開ファイルで指定されるようにＳＳＴ上で走らせる必要があるポッドと、ＳＳＴ上で現在走っているポッドとの間の差異を検出することを伴う。
【０１６０】
ポッドによって必要とされるリソースがＫｕｂｅｒｎｅｔｅｓワーカで利用可能であることを決定するために、Ｋｕｂｅｒｎｅｔｅｓワーカで利用可能なリソースもチェックされることになる。これには、ＲＡＭやＣＰＵなどの標準リソース、及びデバイスプラグインによって処理されるカスタムリソースのチェックが含まれる。
【０１６１】
ＳＳＴ上で実行する必要があるポッドがないこと、及び好適なリソースが利用可能であることを検出することに応答して、ＫｕｂｅｒｎｅｔｅｓマスタのＡＰＩサーバは、この不一致を解決するためにＳＳＴ上のｋｕｂｅｌｅｔに情報を送信する。この情報は、ＳＳＴ上で実行される各ポッドのための展開ファイルを含む。Ｋｕｂｅｒｎｅｔｅｓオーケストレーションプラットフォームを使用しない、本発明のある特定の他の実施形態によれば、コンテナは、ポッド内で走らせる必要なしに実行され得る（これは、Ｋｕｂｅｒｎｅｔｅｓシステムの特定の特徴である）ことが理解されよう。
【０１６２】
次のステップＳ８２５は、ｋｕｂｅｌｅｔによって、ＡＰＩサーバからの情報を受信し、かつ展開ファイルをＳＳＴ上のコンテナエンジン要素に渡すことを伴う。
【０１６３】
展開ファイルを受信することに応答して、次のステップＳ８３０は、コンテナエンジンによって、展開ファイルを読み取り、かつコンテナ画像レジストリから、展開ファイルで指定された実行可能な画像ファイルを要求することを伴う。
【０１６４】
次のステップＳ８３５は、次に、コンテナエンジンによってＳＳＴ上でデバイスプラグインポッドを実行することを伴う。デバイスプラグインポッドには、デバイスプラグインコンテナが含まれる。デバイスプラグインコンテナは、第１のソフトウェアコンテナ要素と称されてもよい。第１のソフトウェアコンテナ要素は、ＳＳＴのプロセッサ（又は複数のプロセッサ）上で実行され得る実行可能なソフトウェアを含む。
【０１６５】
デバイスプラグインコンテナは、１つ以上の抽象的なクラスのデバイスをカスタムリソースとして露出し、これらのクラス内の全てのデバイスのデバイス経路情報を提供し、プラグアンドプレイデバイスサポートを有効にする責任を有する。実行後、デバイスプラグインコンテナが（リソースマネージャによって）呼び出され、そのカスタムリソースが報告される。リソースマネージャは、デバイスプラグインコンテナがＫｕｂｅｒｎｅｔｅｓワーカ上で利用できるカスタムリソースを追跡する。
【０１６６】
次のステップＳ８３５は、ｋｕｂｅｌｅｔのリソースマネージャによって、どの周辺デバイスがＳＳＴに接続されているか、及びいくつの周辺デバイスがＳＳＴに接続されているかに関するクエリをデバイスプラグインコンテナに送信することを伴う。
【０１６７】
本発明の特定の他の実施形態によれば、デバイスプラグインコンテナに利用可能なリソースは、リソースマネージャに自動的にアドバタイズされ得ることが理解されよう。
【０１６８】
このクエリに対する応答として、次のステップＳ８４０は、ＳＳＴに接続可能な周辺デバイスのクラスを示すデータを、ｋｕｂｅｌｅｔのリソースマネージャに送信することを伴う。
【０１６９】
データは、デバイスプラグインコンテナから送信される。このクラスは、周辺デバイス

の所定のセットと関連付けられている。このクラスは、＜ｒｅｔａｉｌ－ｄｅｖｉｃｅｓ＞と呼ばれ得るリソースとしてアドバタイズされる。

【０１７０】

上述のように、クラス内のデバイスは、ＳＳＴ上のメモリ内の構成マップ内に定義されてもよい。リソースマネージャがこのデータを受信すると、次のステップＳ８４５は、リソースマネージャが、（ＳＳＴで実行されるようにリソースがスケジュールされている必要があるポッドに応答して）周辺デバイスの抽象的なクラスをコンテナ／ポッドに割り当てる準備ができるように要求することを伴う。

【０１７１】

この要求は、リソースマネージャによって、プラグインコンテナによって解釈されるコマンドとして、デバイスプラグインコンテナに送信される。このコマンドに応答して、次のステップＳ８５０は、デバイスプラグインコンテナによって、クラス内の各周辺デバイスのデバイス経路を示すデータを送信することを伴う。

【０１７２】

言い換えれば、デバイスプラグインコンテナは、デバイスの抽象的なセット／クラスに対する要求を、抽象的なセット／クラス内のデバイスを表す経路のセットに変換する。

【０１７３】

デバイス経路は、ＯＳによって事前に決定される。このデータは、リソースマネージャによって受信される。このデータは、クラスと関連付けられた各所定の周辺デバイスに対して１つである、全ての可能な（既存及び将来の）デバイス経路のダミーリストを効果的に表す。このデータは、周辺デバイス接続データと呼ばれてもよく、かつソフトウェアコンテナと関連付けられてもよい。

【０１７４】

次のステップ８５５は、制御グループ（ｃグループ）内の特定の形式で周辺デバイス接続データを記憶することを伴い得る。したがって、制御グループ内のデータは、周辺デバイス接続データとも呼ばれ得る。

【０１７５】

リソースマネージャがデバイス経路データを受信すると、次のステップＳ８６０は、Ｋｕｂｅｌｅｔが、クラス内のデバイスを、Ｋｕｂｅｒｎｅｔｅｓクラスタ上のコンテナが要求することができる利用可能なデバイスリソースであるとアドバタイズすることを伴う。次のステップＳ８６５は、Ｋｕｂｅｌｅｔが、（コンテナエンジンの）コンテナランタイムに、第２のソフトウェアコンテナ要素を実行するように指示することを伴う。

【０１７６】

これは、図４のデバイスサーバコンテナであり得る。ステップ８６５は、周辺デバイス接続データを、実行されたソフトウェアコンテナと関連付けるステップ（図示せず）を更に含む。これは、制御グループ（周辺デバイス経路を示すデータを含む）をコンテナと関連付けることによって達成され得る。

【０１７７】

ソフトウェアコンテナは、展開ファイルをコンテナエンジンに渡すことによって実行される。コンテナエンジンは、展開ファイルを解釈し、展開ファイルで指定された画像ファイルをダウンロードし、かつ実行のために画像ファイルをコンテナランタイムに渡す。コンテナランタイムが画像ファイルを実行すると、それによって、第２のソフトウェアコンテナ要素が提供される。第２のソフトウェアコンテナ要素が提供されるとき、それは、コンテナエンジンにＫｕｂｅｌｅｔによって提供される周辺デバイス接続データと関連付けられる。

【０１７８】

上述のように、これは、コンテナをｃグループと関連付けることによって達成され得る。当然のことながら、周辺デバイス接続データは、クラスと関連付けられた所定の周辺デバイスの各々に対する経路を表し、これは、ｋｕｂｅｌｅｔによってアクセス可能なリソースとしてアドバタイズされている。

【０１７９】
次のステップＳ８７０は、第２のソフトウェアコンテナ要素が、クラスと関連付けられたどの周辺デバイスがデバイス経路リスト上の場所の各々で物理デバイスとして接続されているかを決定することを伴う。
【０１８０】
次のステップＳ８７５は、第２のソフトウェアコンテナ要素が第２のソフトウェアコンテナと周辺デバイスとの間の通信リンクを確立することを含む。リンクが確立される場合、次のステップＳ８８０は、ＳＳＴに接続された周辺デバイスの初期化を伴う。これは、対応するデバイス経路に沿って周辺デバイスと通信する第２のソフトウェアコンテナによって達成され得る。したがって、第２のソフトウェアコンテナは、物理デバイスを検出して管理することができる。
【０１８１】
次のステップＳ８８５では、デバイスが再接続されるか、又はＳＳＴが動作中に新しいデバイスがＳＳＴに接続される場合、第２のソフトウェアコンテナ要素は、ステップＳ８７０～Ｓ８８０に記載されるものと同様のステップに従うことによって、新しい又は再接続されたデバイスを検出する。
【０１８２】
その後、第２のソフトウェアコンテナは、新しいデバイスが第２のソフトウェアコンテナの初期化時に存在しないか、又は第２のソフトウェアコンテナの実行中に取り外されているにもかかわらず、関連するデバイス経路を使用して、新たに接続（又は再接続）されたデバイスを初期化し、かつそれと通信することができる。
【０１８３】
展開ファイルは、Ｋｕｂｅｒｎｅｔｅｓマスタ又はＳＳＴ上で動作するコンテナエンジンによって利用されて、図８で論じたステップの一部を促すことができる。すなわち、コンテナエンジンは、所定のクラスと関連付けられた各周辺デバイスへのアクセス、及びそれとの通信を有するように構成されたソフトウェアコンテナ要素を提供するステップを開始するために、この展開ファイルを処理する。
【０１８４】
展開ファイルは、Ｋｕｂｅｒｎｅｔｅｓクラスタ内のコンテナを走らせるポッドの数及び構成を指定する。ポッド内の各コンテナは、展開ファイルで定義された画像のランタイムインスタンスである。コンテナの構成の一部は、実行されると、展開ファイルで定義された「リソース」によって決定される。一例として、コンテナは、実行されると、「スキャナデバイス」及び「表示デバイス」クラス内の全てのデバイスにアクセスできる場合がある。
【０１８５】
本明細書の記述及び特許請求の範囲全体を通して、「備える（ｃｏｍｐｒｉｓｅ）」及び「包含する（ｃｏｎｔａｉｎ）」という言い回し及びこれらの変形例は、「を含むが、これらに限定されない」ことを意味し、他の部分、添加物、構成要素、整数、又はステップを除外する（及び除外しない）ことを意図するものではない。本明細書の記述及び特許請求の範囲全体を通して、文脈上別段の解釈が要求されない限り、単数形は複数形を包含する。特に、不定冠詞が使用される場合、文脈上別段の要求がない限り、本明細書は、複数及び単数を企図するものとして理解されるべきである。
【０１８６】
好ましい実施形態及びこれらの様々な態様を参照して、本開示を具体的に示し、説明してきたが、当業者は、本開示の趣旨及び範囲から逸脱することなく、様々な変更及び修正がなされ得ることを理解されよう。添付の特許請求の範囲は、本明細書に記載される実施形態、上述の代替案、及びこれら全ての等価物を含むものとして解釈されることが意図される。
【０１８７】
本発明の特定の態様、実施形態、又は実施例と併せて記載される特徴、整数、特性、又

は群は、これらとの互換性がない場合を除き、本明細書に記載される任意の他の態様、実施形態、又は実施例に適用可能であると理解されるべきである。

【０１８８】

本明細書（任意の添付の特許請求の範囲、要約、及び図面を含む）に開示される全ての特徴、並びに／又はそのように開示される任意の方法若しくはプロセスの全てのステップは、特徴及び／又はステップのうちの少なくともいくつかが相互に排他的である組み合わせを除いて、任意の組み合わせで組み合わせられてもよい。

【０１８９】

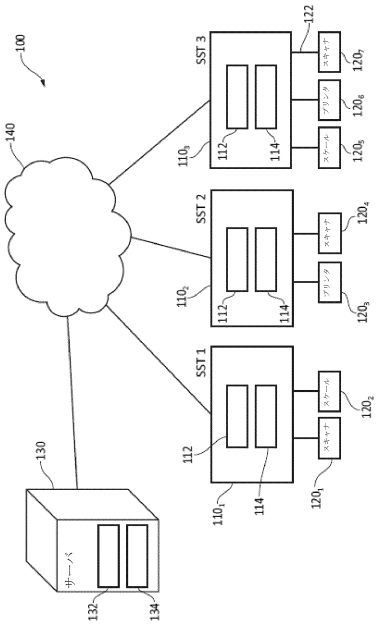
本発明は、任意の前述の実施形態のいかなる詳細にも限定されない。本発明は、本明細書（任意の添付の特許請求の範囲、要約、及び図面を含む）に開示される特徴の任意の新しい１つ、又は新しい組み合わせ、あるいはそのように開示される任意の方法又はプロセスのステップの任意の新しい１つ、若しくは任意の新しい組み合わせに及ぶ。

【０１９０】

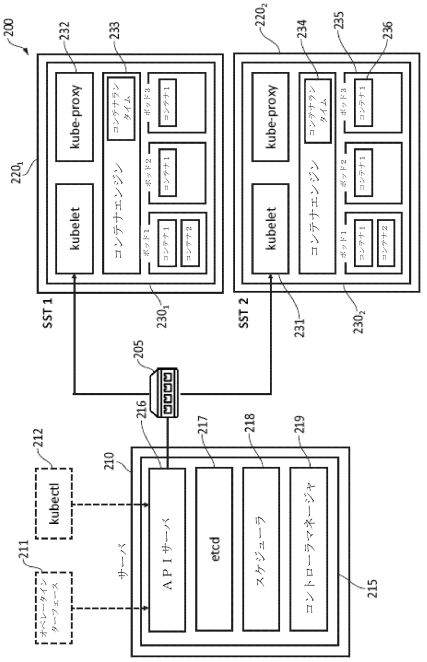
尚、本出願に関連して本明細書と同時に、又は本明細書の前に出願され、かつ本明細書に関して一般に閲覧される全ての論文及び文献に向けられ、かかる全ての論文及び文献の内容は、参照により本明細書に組み込まれる。

【図面】

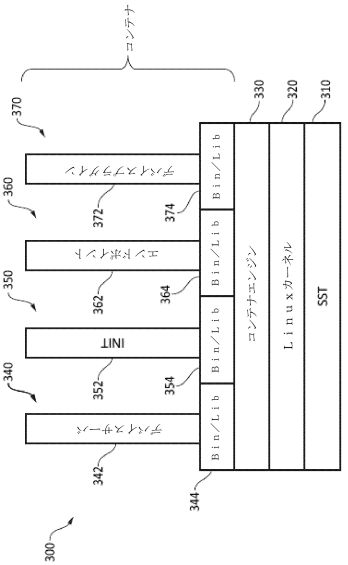
【図１】


100
140
SST 3
SST 2
SST 1
110
112
114
120
130
サーバ
132
134

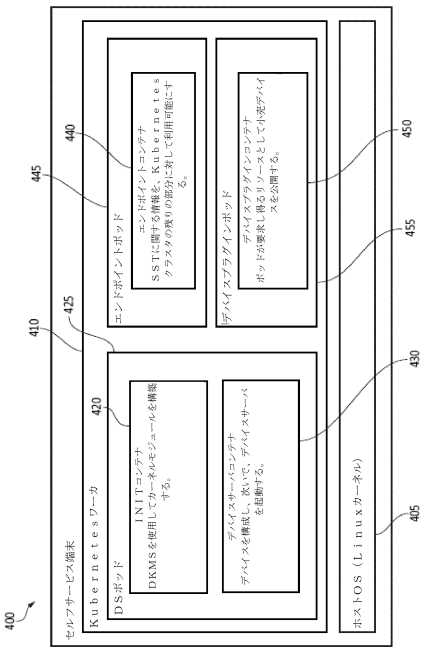
【図２】


200
220
232
233
234
235
236
kube-proxy
kubelet
コンテナランタイム
ポッド
コンテナ
コンテナエンジン
SST 1
SST 2
230
231
205
216
217
218
219
212
210
211
kubectl
サーバ
APIサーバ
etcd
スケジューラ
コントローラマネージャ
215

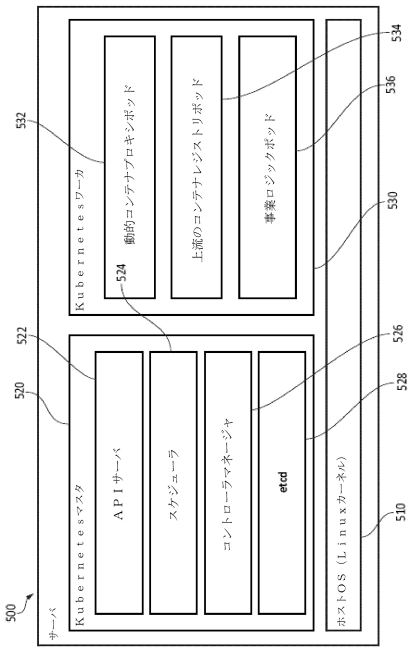
【図３】


コンテナ
370
330
320
310
360
372
374
350
362
364
340
352
354
342
344
300
Bin/Lib
コンテナエンジン
Linuxカーネル
SST
INIT

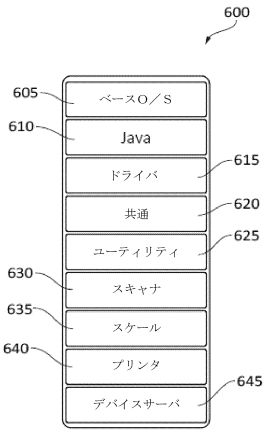
【図４】


400
セルフサービス端末
Kubernetesワーカ
DSポッド
INITコンテナ
DKMSを使用してカーネルモジュールを構築する。
420
デバイスサーバコンテナ
デバイスを構成し、次いで、デバイスサーバを起動する。
410
425
エンドポイントポッド
エンドポイントコンテナ
SSTに関する情報を、Kubernetesクラスタの残りの部分に対して利用可能にする。
440
445
デバイスプラグインポッド
デバイスプラグインコンテナ
ポッドが要求し得るリソースとして小売デバイスを公開する。
450
455
ホストOS（Linuxカーネル）
430
405

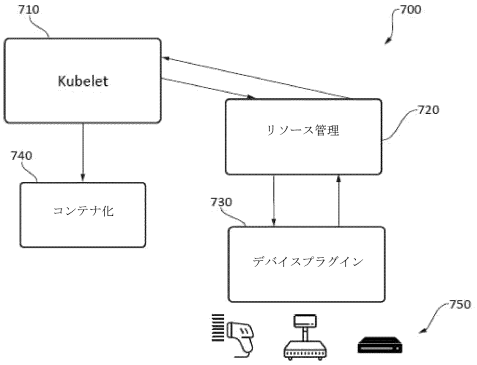
【図５】


500
サーバ
Kubernetesマスタ
APIサーバ
スケジューラ
コントローラマネージャ
etcd
ホストOS（Linuxカーネル）
Kubernetesワーカ
動的コンテナプロキシポッド
上流のコンテナレジストリポッド
事業ロジックポッド
510
520
522
524
526
528
530
532
534
536

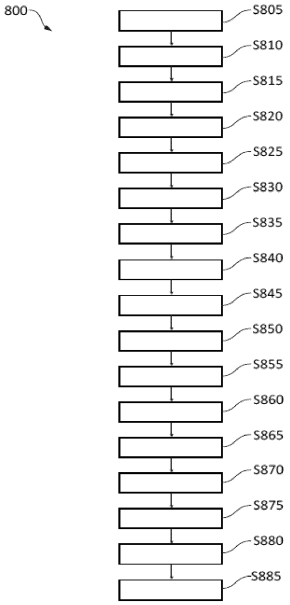
【図６】


600
605
ベースO／S
610
Java
615
ドライバ
620
共通
625
ユーティリティ
630
スキャナ
635
スケール
640
プリンタ
645
デバイスサーバ

【図７】


710
700
Kubelet
720
リソース管理
740
コンテナ化
730
デバイスプラグイン
750

【図８】


800
S805
S810
S815
S820
S825
S830
S835
S840
S845
S850
S855
S860
S865
S870
S875
S880
S885

フロントページの続き

(56)参考文献　米国特許出願公開第２０１７／０２２８１８２（ＵＳ，Ａ１）
特表２０２４－５０９７３９（ＪＰ，Ａ）
特開２００４－３０３２５２（ＪＰ，Ａ）
特開平０６－３０１５５８（ＪＰ，Ａ）
米国特許出願公開第２０１３／００３６４３１（ＵＳ，Ａ１）
米国特許出願公開第２０１９／０１５６０４７（ＵＳ，Ａ１）

(58)調査した分野　(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　８／６０