(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2004-272909
(P2004-272909A)
(43)公開日　平成16年9月30日(2004.9.30)

(51)Int.Cl.[7]	F I		テーマコード（参考）
G06F 11/36	G06F 9/06	620M	5B076
G06F 17/60	G06F 17/60	132	

審査請求 未請求　請求項の数 43　ＯＬ　外国語出願　（全 143 頁）

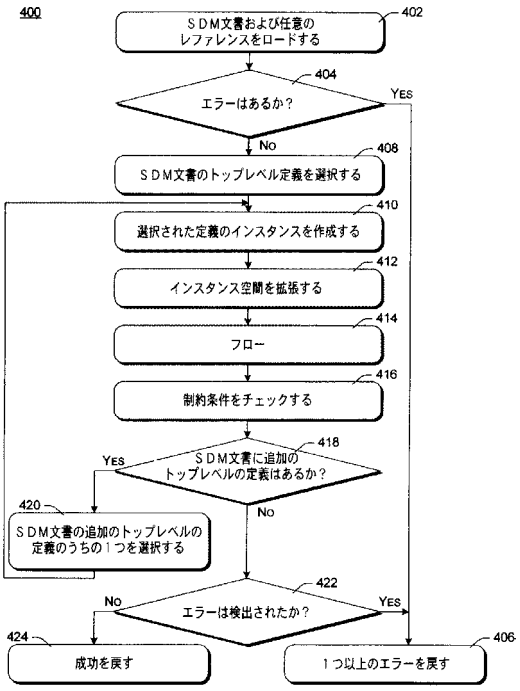
(21)出願番号　特願2004-63227 (P2004-63227)
(22)出願日　平成16年3月5日 (2004.3.5)
(31)優先権主張番号　60/452,736
(32)優先日　平成15年3月6日 (2003.3.6)
(33)優先権主張国　米国 (US)
(31)優先権主張番号　10/791,222
(32)優先日　平成16年3月1日 (2004.3.1)
(33)優先権主張国　米国 (US)

(71)出願人　500046438
マイクロソフト　コーポレーション
アメリカ合衆国　ワシントン州　９８０５２－６３９９　レッドモンド　ワン　マイクロソフト　ウェイ
(74)代理人　100077481
弁理士　谷　義一
(74)代理人　100088915
弁理士　阿部　和夫
(72)発明者　ジェフリー　アウスレッド
アメリカ合衆国　９８１０３　ワシントン州　シアトル　バグリー　アベニュー　ノース　４０２８

最終頁に続く

(54)【発明の名称】システムの設計時検証

(57)【要約】
【課題】　システムの設計時検証を提供すること。
【解決手段】　システムの設計時検証のある態様によれば、設計中のシステムの記述および環境の記述が受け取られる。システムを設計中、およびシステムを配置するよう試みる前に、環境に対してシステムを検証するために、受け取った記述の両方が使用される。
【選択図】　図４



【特許請求の範囲】
【請求項１】
設計中のシステムの記述を受け取るステップと、
環境の記述を受け取るステップと、
前記システムを設計中、および前記システムの配置を試みる前に、前記環境に対して前記システムを検証するために、前記受け取った記述の両方を使用するステップと
を含むことを特徴とする方法。
【請求項２】
前記システムの前記記述がＳＤＭ文書を含むことを特徴とする請求項１に記載の方法。
【請求項３】
前記環境の前記記述がＬＩＭ文書を含むことを特徴とする請求項１に記載の方法。
【請求項４】
前記システムがソフトウェアアプリケーションを含み、前記環境がデータセンタを含むことを特徴とする請求項１に記載の方法。
【請求項５】
前記環境が、前記システムが配置されることが予想される環境を含むことを特徴とする請求項１に記載の方法。
【請求項６】
１つまたは複数のプロセッサによって実行されると、前記１つまたは複数のプロセッサに、
前記１つまたは複数のプロセッサで実行されているプログラムによって設計中のプロセスでシステムを記述するシステム記述にアクセスさせ、
シミュレーションされた環境に対して前記システム記述を使用して前記システムを検証させる複数の命令を記憶していることを特徴とする１つまたは複数のコンピュータ可読媒体。
【請求項７】
前記複数の命令が、前記プロセッサにさらに、
前記システムを検証する要求をリクエスタから受け取らせ、
前記検証の結果を前記リクエスタに戻させることを特徴とする請求項６に記載の１つまたは複数のコンピュータ可読媒体。
【請求項８】
前記１つまたは複数のプロセッサに、前記シミュレーションされた環境に対して前記システムを検証させる前記命令が、前記１つまたは複数のプロセッサにさらに、
前記システム記述からトップレベルの定義を選択させ、
インスタンス空間に対して前記トップレベルの定義によって記述されているように、適切なインスタンスを生成させ、
前記トップレベルの定義内にネストされている追加の定義を選択させ、
前記選択された定義がオブジェクトを定義するか関係を定義するかに基づいて前記インスタンス空間に対して、前記追加の定義によって記述されているように、適切なインスタンスを生成させ、
前記トップレベルの定義内にネストされている前記定義のすべてに対するインスタンスが前記インスタンス空間に対して生成されるまで、前記追加の定義によって記述されているように、追加の定義の前記選択と適切なインスタンスの前記生成とを続行させることを特徴とする請求項６に記載の１つまたは複数のコンピュータ可読媒体。
【請求項９】
前記１つまたは複数のプロセッサに、前記シミュレーションされた環境に対して前記システムを検証させる前記命令が、前記１つまたは複数のプロセッサにさらに、
前記システムを記述するインスタンス空間の１つまたは複数のフローを特定させ、
前記１つまたは複数のフローの少なくとも１つのそれぞれに対して、
前記フローに対して前記インスタンス空間の他のインスタンスから取得した１つまたは

複数の入力値を特定させ、
少なくとも一部には前記入力値に基づいて前記フローに対する出力値を生成させることを特徴とする請求項６に記載の１つまたは複数のコンピュータ可読媒体。
【請求項１０】
前記１つまたは複数のプロセッサに、前記シミュレーションされた環境に対して前記システムを検証させる前記命令が、前記１つまたは複数のプロセッサにさらに、
前記システムを記述するインスタンス空間の１つまたは複数の制約条件を特定させ、
前記１つまたは複数の制約条件が満たされているか否かをチェックさせ、
前記１つまたは複数の制約条件のそれぞれに対して、前記制約条件が満たされているか否かを示す値を戻させることを特徴とする請求項６に記載の１つまたは複数のコンピュータ可読媒体。
【請求項１１】
１つまたは複数の文書がロードされるときに設計中のシステムを記述する１つまたは複数の文書をロードするよう構成されたローダと、
データセンタの環境をシミュレーションし、前記環境に対して前記システムを検証するよう構成されたシミュレータと
を含み、
前記データセンタから分離されていることを特徴とする装置。
【請求項１２】
トップレベルの定義内にネストされたメンバをインスタンス生成することによってインスタンス空間をポピュレートするために、前記１つまたは複数の文書の１つから前記トップレベルの定義を特定し、前記トップレベルの定義を拡張する拡張エンジンをさらに含むことを特徴とする請求項１１に記載の装置。
【請求項１３】
前記インスタンス空間のフローを特定し、前記フローへの入力値を特定し、前記フローへの前記入力に基づいて前記フローの出力を設定するフローエンジンをさらに含むことを特徴とする請求項１２に記載の装置。
【請求項１４】
前記インスタンス空間で制約条件を特定して、評価する制約条件エンジンをさらに含むことを特徴とする請求項１３に記載の装置。
【請求項１５】
１つまたは複数のプロセッサによって実行されると、前記１つまたは複数のプロセッサに、
データセンタの環境で使用されるように設計中のシステムを記述する文書にアクセスさせ、
前記文書からトップレベルの定義を選択させ、
インスタンス空間に対し前記トップレベルの定義によって記述されたように適切なインスタンスを生成し、
前記トップレベルの定義内にネストされている追加の定義を選択させ、
前記選択された定義がオブジェクトを定義するか関係を定義するかに基づいて前記インスタンス空間に対して、前記追加の定義によって記述されているように、適切なインスタンスを生成させ、
前記トップレベルの定義内にネストされている前記定義のすべてに対するインスタンスが前記インスタンス空間に対して生成されるまで、前記追加の定義によって記述されているように、追加の定義の前記選択と適切なインスタンスの前記生成とを続行させる複数の命令を記憶していることを特徴とする１つまたは複数のコンピュータ可読媒体。
【請求項１６】
前記選択された定義が関係を定義する場合に、前記追加の定義によって記述されているように、前記１つまたは複数のプロセッサに適切なインスタンスを生成させる前記命令が、前記１つまたは複数のプロセッサにさらに、

前記定義された関係に関与するソースインスタンス数とターゲットインスタンス数とに基づいて、作成すべき関係インスタンス数を特定させ、
前記特定された数の関係インスタンスを作成させ、
前記作成された関係インスタンスのそれぞれに対して、ソースおよびターゲットインスタンスを前記関係インスタンスに関連付けさせることを特徴とする請求項１５に記載の１つまたは複数のコンピュータ可読媒体。
【請求項１７】
前記選択された定義が、あるインスタンスが別のインスタンスに含まれることができることを記述する包含関係を定義することを特徴とする請求項１６に記載の１つまたは複数のコンピュータ可読媒体。
【請求項１８】
前記選択された定義が、別個に配置されたソフトウェア要素間の相互作用を記述する通信関係を定義することを特徴とする請求項１６に記載の１つまたは複数のコンピュータ可読媒体。
【請求項１９】
前記選択された定義が、インスタンス間の従属関係を取り込むために使用されるレファレンス関係を定義することを特徴とする請求項１６に記載の１つまたは複数のコンピュータ可読媒体。
【請求項２０】
前記選択された定義が、ホストをそのゲストメンバインスタンスの１つまたは複数と関連付けるホスティング関係を定義することを特徴とする請求項１６に記載の１つまたは複数のコンピュータ可読媒体。
【請求項２１】
前記選択された定義が、２つのシステムの通信エンドポイントを関連付ける委任関係を定義することを特徴とする請求項１６に記載の１つまたは複数のコンピュータ可読媒体。
【請求項２２】
前記選択された定義がオブジェクトを定義する場合に、前記追加の定義によって記述されているように、前記１つまたは複数のプロセッサに適切なインスタンスを生成させる前記命令が、前記１つまたは複数のプロセッサにさらに、
前記選択された定義で特定される前記オブジェクトの最小発生数を特定させ、
前記特定された最小発生数と、前記選択された定義でいくつのインスタンスが既に生成されているかに基づいて、生成すべき前記選択された定義のインスタンス数を特定させ、
前記選択された定義のインスタンスを前記特定された数だけ生成させることを特徴とする請求項１５に記載の１つまたは複数のコンピュータ可読媒体。
【請求項２３】
前記選択された定義がオブジェクトを定義する場合に、前記追加の定義によって記述されているように、前記１つまたは複数のプロセッサに適切なインスタンスを生成させる前記命令が、前記１つまたは複数のプロセッサにさらに、
リスナが前記追加情報によって記述されているように前記適切なインスタンスを作成することを可能にするイベントをトリガさせることを特徴とする請求項１５に記載の１つまたは複数のコンピュータ可読媒体。
【請求項２４】
前記選択された定義が関係を定義する場合に、前記追加の定義によって記述されているように、前記１つまたは複数のプロセッサに適切なインスタンスを生成させる前記命令が、前記１つまたは複数のプロセッサにさらに、
リスナが前記追加情報によって記述されているように前記適切なインスタンスを作成することを可能にするイベントをトリガさせることを特徴とする請求項１５に記載の１つまたは複数のコンピュータ可読媒体。
【請求項２５】
前記データセンタの前記システムの配置を開始する前に前記命令が実行されることを特

徴とする請求項１５に記載の１つまたは複数のコンピュータ可読媒体。
【請求項２６】
１つまたは複数のプロセッサによって実行されると、前記１つまたは複数のプロセッサに、
データセンタの環境で使用されるように設計中のシステムを記述するインスタンス空間の１つまたは複数のフローを特定させ、
前記１つまたは複数のフローの少なくとも１つのそれぞれに対して、
前記フローに対して前記インスタンス空間の他のインスタンスから取得した１つまたは複数の入力値を特定させ、
少なくとも一部には前記入力値に基づいて前記フローに対する出力値を生成させる複数の命令を記憶していることを特徴とする１つまたは複数のコンピュータ可読媒体。
【請求項２７】
前記１つまたは複数のプロセッサに、前記フローに対して１つまたは複数の入力値を特定させる前記命令が、前記１つまたは複数のプロセッサにさらに、
前記入力値が割り当てられているか否かを特定させ、
前記入力値が割り当てられている場合、前記他のインスタンスから前記入力値を取得させ、
前記入力値の少なくとも１つがまだ割り当てられていない場合、まだ割り当てられていない前記入力値のそれぞれに対して、
前記入力値を設定する１つの他のフローを特定させ、
前記他のフローに対して、前記インスタンス空間の他のインスタンスから取得した１つまたは複数の入力値を特定させ、
少なくとも一部には前記入力値に基づいて前記他のフローに対する出力値を生成させることを特徴とする請求項２６に記載の１つまたは複数のコンピュータ可読媒体。
【請求項２８】
前記１つまたは複数のプロセッサに、少なくとも一部には前記入力値に基づいて前記フローに対する前記出力値を生成させる前記命令が、前記１つまたは複数のプロセッサにさらに、
結果を生成するために実行することのできる前記フローに関連付けられた一組の命令を特定させ、
前記特定された一組の命令を実行させ、
前記生成された結果を前記フローに対する前記出力値として使用させることを特徴とする請求項２６に記載の１つまたは複数のコンピュータ可読媒体。
【請求項２９】
前記システムが前記環境に配置されるべきアプリケーションを含むことを特徴とする請求項２６に記載の１つまたは複数のコンピュータ可読媒体。
【請求項３０】
前記環境がデータセンタのハードウェア記述を含むことを特徴とする請求項２６に記載の１つまたは複数のコンピュータ可読媒体。
【請求項３１】
前記環境で前記システムの配置を開始する前に前記命令が実行されることを特徴とする請求項２６に記載の１つまたは複数のコンピュータ可読媒体。
【請求項３２】
１つまたは複数のプロセッサによって実行されると、前記１つまたは複数のプロセッサに、
データセンタの環境で使用されるよう設計中のシステムを記述するインスタンス空間の１つまたは複数の制約条件を特定させ、
前記１つまたは複数の制約条件が満たされているか否かをチェックさせ、
前記１つまたは複数の制約条件のそれぞれに対して、前記制約条件が満たされているか否かを特定する値を戻させる複数の命令を記憶していることを特徴とする１つまたは複数

のコンピュータ可読媒体。
【請求項３３】
前記１つまたは複数の制約条件が、設定制約条件、関係制約条件、およびオブジェクト制約条件を含むことを特徴とする請求項３２に記載の１つまたは複数のコンピュータ可読媒体。
【請求項３４】
前記１つまたは複数のプロセッサに前記１つまたは複数の制約条件が満たされているか否かをチェックさせる前記命令が、前記制約条件の１つに対して、前記１つまたは複数のプロセッサにさらに、
結果を生成するために実行することのできる前記制約条件に関連付けられる一組の命令を特定させ、
前記特定された一組の命令を実行させ、
前記生成された結果を前記制約条件が満たされているか否かを示す前記戻された値を使用させることを特徴とする請求項３２に記載の１つまたは複数のコンピュータ可読媒体。
【請求項３５】
前記１つまたは複数のプロセッサに前記１つまたは複数の制約条件が満たされているか否かをチェックさせる前記命令が、前記制約条件の１つに対して、前記１つまたは複数のプロセッサにさらに、
前記制約条件のターゲットインスタンスに対するロールおよびオブジェクト定義を特定させ、
前記制約条件の前記ロールおよび前記オブジェクト定義が前記ターゲットインスタンスの前記ロールおよび前記オブジェクト定義と一致するか否かをチェックさせ、
前記制約条件の前記ロールおよび前記オブジェクト定義が前記ターゲットインスタンスの前記ロールおよび前記オブジェクト定義と一致するか否かに基づいて、前記制約条件が満たされているか否かを示す前記戻された値を生成させることを特徴とする請求項３２に記載の１つまたは複数のコンピュータ可読媒体。
【請求項３６】
前記１つまたは複数のプロセッサに前記１つまたは複数の制約条件が満たされているか否かをチェックさせる前記命令が、前記制約条件の前記１つに対して、前記１つまたは複数のプロセッサにさらに、
前記制約条件の前記ターゲットインスタンスに対する副次的なロールおよび副次的なオブジェクト定義を特定させ、
前記制約条件の前記副次的なロールおよび前記副次的なオブジェクト定義が前記ターゲットインスタンスの前記ロールおよび前記オブジェクト定義と一致するか否かをチェックさせ、
前記制約条件の前記ロールおよび前記オブジェクト定義が、前記ターゲットインスタンスの前記ロールおよび前記オブジェクト定義と前記ターゲットインスタンスの前記副次的なロールおよび前記副次的なオブジェクト定義の両方と一致するか否かに基づいて、前記制約条件が満たされているか否かを示す前記戻される値を生成させることを特徴とする請求項３５に記載の１つまたは複数のコンピュータ可読媒体。
【請求項３７】
前記１つまたは複数のプロセッサに前記１つまたは複数の制約条件が満たされているか否かをチェックさせる前記命令が、前記制約条件の前記１つに対して、前記１つまたは複数のプロセッサにさらに、
前記ターゲットインスタンスに対して１つまたは複数のネストされた制約条件を評価させ、
前記ネストされた制約条件に対して前記１つまたは複数のネストされた制約条件が満たされているか否かを示す１つまたは複数の戻り値を受け取らせ、
前記ネストされた制約条件に対する前記１つまたは複数の戻り値に基づいて、前記制約条件が満たされているか否かを示す前記戻される値を生成させることを特徴とする請求項

３５に記載の１つまたは複数のコンピュータ可読媒体。
【請求項３８】
前記１つまたは複数のプロセッサに、前記１つまたは複数の制約条件のそれぞれに対して前記制約条件が満たされているか否かを示す値を戻させる前記命令が、前記制約条件の１つに対して、前記１つまたは複数のプロセッサにさらに、
前記制約条件が満たされていないことを示す値が戻されるべき場合、前記制約条件に対してエラーメッセージが生成されるべきか否かをチェックさせ、
前記エラーメッセージが生成されるべき場合、前記制約条件を示す情報を含む前記エラーメッセージを生成させることを特徴とする請求項３２に記載の１つまたは複数のコンピュータ可読媒体。
【請求項３９】
前記１つまたは複数のプロセッサに前記１つまたは複数の制約条件が満たされているか否かをチェックさせる前記命令が、前記制約条件の１つに対して、前記１つまたは複数のプロセッサにさらに、
一致数変数を初期設定させ、
前記制約条件の前記ターゲットインスタンスが参加する１つまたは複数の関係インスタンスを特定させ、
前記１つまたは複数の関係インスタンスのそれぞれに対して、前記関係インスタンスが前記制約条件を満たしているか否かを評価させ、
前記制約条件を満たしている前記１つまたは複数の関係インスタンスのそれぞれに対して前記一致数変数を増分させ、
前記１つまたは複数の関係インスタンスが評価された後で前記一致数変数の前記値に基づいて、前記制約条件が満たされているか否かを示す前記戻される値を生成させることを特徴とする請求項３２に記載の１つまたは複数のコンピュータ可読媒体。
【請求項４０】
前記１つまたは複数のプロセッサに、前記１つまたは複数の関係インスタンスのそれぞれに対して前記関係インスタンスが前記制約条件を満たしているか否かを評価させる前記命令が、前記制約条件の前記１つに対して、前記１つまたは複数のプロセッサにさらに、
前記制約条件の関係定義が前記関係インスタンスの関係定義と一致するか否かをチェックさせ、
前記制約条件の指示が前記関係インスタンスの指示と一致するか否かをチェックさせ、
前記関係インスタンスに対するすべてのネストされた制約条件が満たされているか否かをチェックさせ、
前記制約条件の前記関係定義が前記関係インスタンスの前記関係定義と一致し、前記制約条件の前記指示が前記関係インスタンスの前記指示と一致し、前記関係インスタンスに対するすべてのネストされた制約条件が満たされている場合にのみ、前記制約条件が満たされていることを示す値を戻させることを特徴とする請求項３９に記載の１つまたは複数のコンピュータ可読媒体。
【請求項４１】
前記１つまたは複数のプロセッサに、前記１つまたは複数の関係インスタンスのそれぞれに対して前記関係インスタンスが前記制約条件を満たしているか否かを評価させる前記命令が、前記制約条件の前記１つに対して、前記１つまたは複数のプロセッサにさらに、
前記制約条件のターゲットオブジェクトが前記関係インスタンスの前記他端のインスタンスと一致するか否かをチェックさせ、
前記制約条件の前記関係定義が前記関係インスタンスの前記関係定義と一致し、前記制約条件の前記指示が前記関係インスタンスの前記指示と一致し、前記関係インスタンスに対するすべてのネストされた制約条件が満たされており、前記制約条件の前記ターゲットオブジェクトが前記関係インスタンスの前記他端の前記インスタンスと一致する場合にのみ、前記制約条件が満たされていることを示す値を戻させることを特徴とする請求項３９に記載の１つまたは複数のコンピュータ可読媒体。

【請求項４２】
前記１つまたは複数のプロセッサに、前記１つまたは複数の関係インスタンスが評価された後で前記一致数変数の前記値に基づいて、前記制約条件が満たされているか否かを示す前記戻される値を生成させる前記命令が、前記制約条件の１つに対して、前記１つまたは複数のプロセッサにさらに、
前記一致数変数が前記制約条件の少なくとも１つの最小値であるが、前記制約条件の最大値よりも大きくないか否かをチェックさせ、
前記一致数変数が前記制約条件の少なくとも前記最小値であるが、前記制約条件の前記最大値よりも大きくない場合には前記制約条件が満たされていることを示す前記戻される値を生成させ、そうでない場合は前記制約条件が満たされていないことを示す前記戻される値を生成させることを特徴とする請求項３９に記載の１つまたは複数のコンピュータ可読媒体。
【請求項４３】
前記命令が、前記環境で前記システムの配置を開始する前に実行されるべきことを特徴とする請求項３２に記載の１つまたは複数のコンピュータ可読媒体。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、システムの設計に関し、より詳細には、システムの設計時検証に関する。
【背景技術】
【０００２】
過去数年間にインターネットの使用量は爆発的に増加し、今も成長を続けている。人々は、電子メール、オンラインショッピング、ニュースおよび情報の収集、音楽鑑賞、ビデオクリップ鑑賞、求職活動などのようなワールドワイドウェブ（または単に「ウェブ」）で提供される多くのサービスに非常に満足するようになってきた。インターネットベースのサービスへの需要が増大するのに合わせて、ウェブサイトをホスティングし、それらのサイトに対するバックエンドサービスを提供し、それらのサイトに関連するデータを記憶すること専用のコンピュータシステムが驚異的な成長を見せている。
【０００３】
分散型コンピュータシステムの１つの種類は、ネットワークベースのサービスをホスティングするための多くのコンピュータを収容する特別に設計された複合システムであるデータセンタ（例えば、インターネットデータセンタ（ＩＤＣ）または企業データセンタ（ＥＤＣ））である。「ウェブファーム」または「サーバファーム」という名称でも通用しているデータセンタは、通常、空調された物理的に安全な建物内に数百から数千ものコンピュータを収容する。データセンタは、通常、信頼性のあるインターネットアクセス、信頼性のある電力供給、および安全な動作環境を提供する。
【発明の開示】
【発明が解決しようとする課題】
【０００４】
様々なデータセンタは、そのデータセンタで実行されるアプリケーションに課される様々な要件を有することができる。例えば様々なデータセンタは、様々な種類のセキュリティが施行されることを要求し、またはアプリケーションが様々な通信プロトコルをサポートすることを要求する場合がある。さらに、いくつかのデータセンタは、経時的にそれらの要件を変更する場合がある。データセンタ間でのこれらの違いはアプリケーションの設計プロセスを困難にする。つまり、所望の１つ以上のデータセンタで彼らのアプリケーションが実行されることができるように、設計者はそれらの異なる要件に気付き、彼らのアプリケーションの設計にそれらの要件を反映させなければならない。
【０００５】
さらに、現状、設計者は、通常、アプリケーションをデータセンタに配置する際に、彼らのアプリケーションが特定のデータセンタで実行されるか否かを評価することしかでき

ない。データセンタの要件の１つまたは複数がアプリケーションによって満たされないために配置が不成功になる場合、設計者はアプリケーションに関する１つ以上のその問題を修正し、データセンタにおけるそのアプリケーションの配置を再試行するよう試みることが要求される。このプロセスは、多くの回数反復されることがあり、設計プロセスを効率の悪くいらいらさせるようなものにする可能性がある。

本明細書に記載のシステムの設計時検証はこれらおよび他の問題を解決する。

【課題を解決するための手段】

【０００６】

システムの設計時検証を本明細書に記載する。

システムの設計時検証のある態様によれば、設計中のシステムの記述および環境の記述が受け取られる。受け取られた両方の記述は、システムを設計中、およびシステムの配置を試みる前に、環境に対してシステムを検証するために使用される。

【発明を実施するための最良の形態】

【０００７】

同様の特徴を参照するために図面全体を通して同じ番号が使用される。

以下の開示は、システムの設計時検証のためのアーキテクチャに関するいくつかの態様を説明している。この開示は、サービス定義モデル（ＳＤＭ）とも称することのできるシステム定義モデル（ＳＤＭ）に関する検討も含む。ＳＤＭは、アプリケーション立案者が抽象的な方法で分散型コンピュータアプリケーションおよびデータセンタを設計するためのツールおよびコンテキストを提供する。モデルは、最終的に物理的コンピュータ資源およびソフトウェアによって実施されるアプリケーションの機能単位を表す一組の要素を定義する。そのモデル要素に関連付けられているのは、構成要素によって表される機能動作をどのように指定するかを示すスキーマである。

【０００８】

本明細書で使用されるように、「有線」という用語は、「接続」、「通信」、または「通信関係」をも意味する場合がある。また「システム」という用語は「モジュール」を意味する場合があり、「資源空間」という用語は「資源」を意味する場合がある。さらに「アプリケーション空間」という用語は「アプリケーション」をも意味する場合があり、「インスタンス空間」という用語は「インスタンス」をも意味する場合がある。また、「クラス」という用語は「抽象定義」をも意味する場合があり、「ポート」という用語は「エンドポイント」をも意味する場合があり、「型」という用語は「定義」をも意味する場合がある。

【０００９】

図１は、ネットワーク設定例１００を示している。設定１００では、複数（ｘ）のコンピュータデバイス１０２（１）、１０２（２）、．．．、１０２（ｘ）はネットワーク１０６に結合されている。ネットワーク１０６は、（公衆および／または独自のプロトコルを含めて）様々な従来型ネットワークプロトコルを利用する（有線および／または無線ネットワークを含めて）様々な従来型ネットワークのトポロジおよび種類のどれかを表すことを意図している。ネットワーク１０６は、例えばローカルエリアネットワーク（ＬＡＮ）、ワイドエリアネットワーク（ＷＡＮ）、インターネットの一部などを含んでよい。設定１００は、例えばデータセンタ（例えば、インターネットデータセンタ（ＩＤＣ））、オフィスまたは企業設定、家庭用設定、教育または研究施設、小売りまたは販売設定、データ記憶設定などを含めて幅広い設定のどれでも表す。

【００１０】

コンピュータデバイス１０２は、デスクトップＰＣ、ワークステーション、メーンフレームコンピュータ、サーバコンピュータ、インターネット機器、ゲーミングコンソール、ハンドヘルドコンピュータ、セルラ電話、パーソナルデジタルアシスタンス（ＰＤＡ）などを含めて様々な従来型のコンピュータデバイスのどれであってもよい。デバイス１０２の１つまたは複数は、同じ種類のデバイスであっても、または異なる種類のデバイスであってもよい。あるいは、たとえ複数のデバイスが同じ種類のデバイスであっても、それら

複数の種類のデバイスは異なる構成を取ることができる（例えば、２つのデバイス１０２はサーバコンピュータであるが、異なるプロセッサ、異なる量のＲＡＭ、異なるサイズのハードディスクドライブなど異なるハードウェア構成を有することができる）。

【００１１】

１つまたは複数のコンピュータデバイス１０２は、設定１００に追加された後で再構成することもできる。例えば、特定のコンピュータデバイス１０２は、ある機能を実行してある期間だけ（例えば分、時間、日、月などにより）動作することができ、管理者は別の機能が望ましいと決定することができる（例えば、サーバコンピュータであることからワークステーションコンピュータへの変更、ウェブサーバからローカルファイルサーバへの変更など）。

【００１２】

図２は、ＳＤＭを使用するアーキテクチャ２００例を示すブロック図である。ＳＤＭは、システムの設計プロセス中に使用されるように設計されており、設計プロセスの完了後にシステムの配置および／または管理のために使用することもできる。システムは、協同して共通機能を達成することのできる一組の関連ソフトウェアおよび／またはハードウェア資源である。このようなシステムの一例は、様々な機能を実行するコンピュータデバイスによって使用され、または実行されることのできる一組の命令を意味するアプリケーションである。アプリケーション例には、ゲームのような娯楽アプリケーション、ワードプロセッサのような生産アプリケーション、電子百科事典のようなレファレンスアプリケーション、ウェブサービスまたは経済分析のために使用することのできるような分散アプリケーションなどが含まれる。そのようなシステムの別の例は、アプリケーション（または別の環境）を配置することができる一環境である。環境は、アプリケーション（または別の環境）が配置されるソフトウェアおよび／またはハードウェア資源を意味する。そのような環境は相互の上に階層化することができる。

【００１３】

一般に、システムに対する設計プロセスまたは設計段階中（開発プロセスまたは開発段階とも称される）に、通信ソフトウェアおよび／またはハードウェア構成要素からなるシステムを定義するためにＳＤＭを活用する開発ツールが使用される。システム定義は、要求された資源、構成、動作特性、ポリシーなどを含めて、分散型システムを配置し、動作させるために必須のすべての情報を含んでいる。設計段階の完了後、システムを自動的に配置し、要求されるソフトウェアおよびハードウェア（例えば、サーバ、ストレージ、およびネットワーク）資源を動的に割り当て、構成するために配置段階でもシステム定義を使用することができる。異なるホスト環境および異なるスケールに配置するために同じシステム定義を使用することができる。

【００１４】

アーキテクチャ２００は、ＳＤＭ定義モデルならびにＳＤＭ定義モデル内で機能動作を定義するスキーマを利用する。定義モデルは、「定義」と総称される様々な異なる種類のデータ構造を含む。ＳＤＭの機能は、アプリケーションプログラムインターフェース（ＡＰＩ）のような１つまたは複数のプラットフォームサービスによって公開される。

【００１５】

システムの設計段階中、開発コンポーネント２０２はＳＤＭ文書２０４のようなシステム定義を含んでいる文書を生成する。開発コンポーネント２０２は、ワシントン州レッドモンドのＭｉｃｒｏｓｏｆｔ（登録商標）社製のＶｉｓｕａｌ　Ｓｔｕｄｉｏ（登録商標）開発システムのような様々な開発コンポーネントのどれであってもよい。ＳＤＭ文書２０４は、システムの配置（および、任意選択で管理）に関するすべての情報（本明細書では知識とも称する）を定義する。システムを配置し、またはシステムを管理するために必須の、またはその際に使用されるいかなる知識でもＳＤＭ文書２０４に含まれる。本明細書では単一文書として記載しているが、知識が複数の文書に展開され、保存され得ることを理解されたい。

【００１６】

システム定義は、システムを１つまたは複数の資源、エンドポイント、関係（relationship）、およびサブシステムに関して定義する。システム定義はＳＤＭ文書（例えば、ＸＭＬ文書）で宣言される。資源はハードウェア資源であってもソフトウェア資源であってもよい。エンドポイントは複数のシステムを横断する通信（communication）を表す。関係は、システム、資源およびエンドポイント間の関連を定義する。サブシステムは完全なシステムとして取り扱うことができ、典型的には、より大きなシステムの一部である。

【００１７】

システム定義は、動的システムの基本構造を取り込む。システム定義は、すべての他の情報が追加されるスケルトンと見なすことができる。この構造は、通常、開発プロセス中に設計者および開発者によって指定され、通常、頻繁には変更されない。この構造に加え、ＳＤＭは、配置情報、インストールプロセス、構成用スキーマ、イベントおよび計測、自動タスク、ヘルスモデル、動作ポリシーなどを含むことができる。分散型システムの存続期間全体を通じて、他の情報を、オペレーションスタッフ、供給業者、および／または管理システムによって追加することができる。

【００１８】

ＳＤＭ文書２０４は、システムが配置され、かつ／または実行される環境が充足しなければならないシステムの１つまたは複数の制約条件（要件とも称される）を含む。環境自体は、「論理インフラストラクチャ」（ＬＩＭ）文書２０６、またはデータセンタ記述またはデータセンタモデルと称されるＳＤＭ文書を使用しても記述される。ＳＤＭ文書２０４と類似して、ＬＩＭ文書２０６は単一文書であっても、複数の文書によって構成されてもよい。ＬＩＭ文書２０６は、開発コンポーネント２０２と類似のコンポーネントのような様々な開発コンポーネントのどれかを使用して生成することができる。ＬＩＭ文書２０６は、開発コンポーネント２０２を使用して開発することもできる（すなわち、ＳＤＭ文書２０４とＬＩＭ文書２０６の両方を生成するために同じ開発コンポーネントを使用することができる）。

【００１９】

ＬＩＭ文書２０６によって記述された環境は、単一のコンピュータデバイスであっても、またはコンピュータデバイス（例えば、データセンタ）、アプリケーションホストなどの集合であってもよい。異なるシステムまたは同じシステムを異なる環境にインストールすることができる。例えば、データセンタは、５０個のコンピュータデバイスを含むことができ、１つのシステムをそれらコンピュータデバイスのうちの５個に配置し、他のシステムをそれらコンピュータデバイスの３５個に配置することができる。これらの要件は、システムが配置されるべき１つ以上のコンピュータデバイスに関するハードウェア要件（例えば、最低プロセッサ速度、最低メモリ量、ハードドライブ空き領域の最低量、使用可能なネットワーク帯域幅の最低量、使用可能な特定のセキュリティ機構など）、システムが配置されるべき１つ以上のコンピュータデバイスに関するソフトウェア要件（例えば、特定オペレーティングシステム、同様にインストールされなければならない１つまたは複数の他のアプリケーション、特定システムおよび／またはオペレーティングシステムを構成すべき方法に関する規定、使用されている特定の種類のセキュリティまたは暗号化など）、システムが配置されるべき１つ以上のコンピュータデバイスに関する他の要件（例えば、使用可能な特定のセキュリティキー、施行されるべきデータセンタポリシー、使用されている認証、環境トポロジなど）のような様々な形式を取ることができる。ＳＤＭ文書２０４に記載されている単一システムを、別のＬＩＭ２０６に記載されているような複数の異なる環境に対して検証することができる。

【００２０】

要件は他の方向に向かうこともできる。すなわち、環境はインストールされるべきシステムの構成に関する（例えば、環境の標準またはポリシーを実施するための）制約条件または要件を有することができる。これらは、システムが有する必要のある特定の設定または構成、システムが提供またはサポートする必要のある特定機能、システムがサポートする必要のある特定のセキュリティ機構などのように、環境のオペレータによって作成され

る「明示的な」要件であってよい。これらは、環境の特定の構成によって生じる「潜在的な」要件であってもよい。例えば、環境内のホストコンピュータデバイスが特定の種類のファイルシステムを使用している場合、（別のファイルシステムを使用してそれらの同じアクションを実行することができたとしても）そのファイルシステムを使用していくつかのアクションを実行することができない場合がある。

【００２１】

システムの設計段階中、検証コンポーネント２０８によって１つまたは複数の特定の環境に対してシステムを検証するためにＳＤＭ文書２０４を使用することができる。これは、システムが環境に対して検証され、環境がシステムに対して検証されるという双方向の検証である。検証コンポーネント２０８は、ＳＤＭ文書２０４で特定された要件をＬＩＭ文書２０６に記載されている環境と比較し、要件のすべてがその環境によって満たされているか否かを判定することによって、システムに対して環境を検証することができる。検証コンポーネント２０８は、ＬＩＭ文書２０６で特定された要件をＳＤＭ文書２０４に記載されているシステムと比較し、要件のすべてがそのシステムによって満たされているか否かを判定することによって環境に対してシステムを検証することができる。要件のすべてが環境およびシステムによって満たされている場合、設計者または開発者は、そのシステムをその環境に配置することができ、そこで作動すると分かる。しかし、要件のすべてが環境および／またはシステムによって満たされていない場合、検証コンポーネント２０８は満たされていない要件を設計者または開発者に通知することができ、したがってシステムがその環境に配置され、そこで作動するために、ＳＤＭ文書２０４（同様にシステム）および／または環境にどのような変更を行う必要があるかが設計者または開発者に通知される。

【００２２】

ＬＩＭ文書２０６は、現実の環境またはシミュレートされた環境に関するものであってよい。現実の環境の場合、ＬＩＭは、現存しているあるいは既に記述されているがまだ施工されていない環境を記述する。例えば、現実の環境は現存するデータセンタであっても、既に設計済みであるがまだ実際には実施されていないデータセンタであってもよい。シミュレーションされた環境の場合、ＬＩＭは、必ずしも存在するとはいえないある予想された環境を記述する。現実の環境に対してはそのような仮定を行う必要はないが、予想された環境に関しては様々な仮定を行うことができる。通常、システムに対する設計プロセス中、ＬＩＭ文書２０６はシミュレーションされた環境を記述する。しかし、現実の環境が特定の時点で存在する場合には、その環境を記述しているＬＩＭ文書２０６を設計段階中に使用することができる。

【００２３】

検証コンポーネント２０８は、開発中のシステムを配置することができるデータセンタから分離されている１つまたは複数のコンピュータデバイスであってよい。例えば、検証コンポーネント２０８は単一コンピュータデバイスであってよいが、あるいは検証の負担を複数のデバイス全体に分散してもよい。あるいはまた、検証コンポーネント２０８は、開発中のシステムが配置されるべきデータセンタのコンピュータデバイス１０２の１つまたは複数であってよい。

【００２４】

ＳＤＭは、横軸および縦軸にわたってシステムの機能の合成を可能にする。横軸に沿った合成はシステムおよびサブシステムによって行われる。縦軸に沿った合成は「層」によって行われる。アプリケーション、サービス、ネットワークトポロジ、およびハードウェアは、分散されたシステムで１つのロール（roll）を満たすが、典型的には独立して定義され、異なるチームまたは組織によって所有される。階層化は、ホスト上の一組の制約条件を定義する構成要素によって達成され、またその逆も行われる。

【００２５】

図３は階層化設定例を示す。図３には、層３０２、層３０４、層３０６、および層３０８の４つの層が示されている。図３には４つの層が示されているが、実際の層の数は多様

であってよく、４つより多くても少なくてもよい。さらに、異なる層の内容は異なる実施形態で異なっていてよい。図３を見ると分かるように、様々な層が他の層の上および／または下に位置する（例えば、層３０６は層３０４の上、層３０８の下にある）。

【００２６】

１つの層内の異なるシステムおよびサブシステムはそれぞれ相互作用することができ、また異なる層のシステムおよびサブシステムとも相互作用することができる。例えば、層３０８のサブシステム３１０は層３０８のサブシステム３１２ならびに層３０６のサブシステム３１４と相互作用することができる。さらに、各層は１つ上の層に対する環境と見なすことができる。例えば、層３０６は層３０８のシステムおよびサブシステムに対する環境であり、層３０４は層３０６のシステムおよびサブシステムに対する環境である。各層３０２、３０４、３０６、および３０８はそれ独自の関連付けられたＳＤＭ文書を有する。

【００２７】

異なる層３０２、３０４、３０６、および３０８は異なる内容を表すことができる。ある実施形態では、層３０２はハードウェア層であり、層３０４はネットワークトポロジおよびオペレーティングシステム層であり、層３０６はアプリケーションホスト層であり、層３０８はアプリケーション層である。ハードウェア層は、階層化されたシステム（例えば、図１のデバイス１０２）が構築される物理デバイス（例えば、コンピュータデバイス）を表す。ネットワークトポロジおよびオペレーティングシステム層は、コンピュータデバイスのネットワークトポロジ（例えば、図１のネットワーク設定１００）ならびにそれらのコンピュータデバイスにインストールされたオペレーティングシステムを表す。アプリケーションホスト層は、他のアプリケーションをホストすることのできるコンピュータデバイス（例えば、ＳＱＬサーバ、ＩＩＳなど）にインストールされたアプリケーションを表す。アプリケーション層は、他のアプリケーションをホストしないコンピュータデバイスにインストールされたアプリケーション（例えば、ゲームのような娯楽アプリケーション、ワードプロセッサのような生産アプリケーション、電子百科事典のようなレファレンスアプリケーション、ウェブサービスまたは経済分析のために使用することのできるような分散型アプリケーションなど）を表す。

【００２８】

図２に戻ると、アプリケーション開発者は開発コンポーネント２０２を使用してアプリケーションを設計し、開発することができる。開発者が自分のシステムの様々な部分およびそれらの部分の相互関係を定義すると、それらの部分および関係を取り込んだＳＤＭファイル２０４が生成される。開発者は、ＬＩＭファイル２０６に記述されている環境に対してシステム記述を検証するために検証コンポーネント２０８を使用することができる。設計段階中のこの検証は「設計時検証」とも称される。

【００２９】

以下の節「ＳＤＭ実施態様例」で詳述するＳＤＭは、分散システム（モデルシステム）の構成要素の構成、相互作用、および変更の記述をサポートするように設計されている。ＳＤＭはオブジェクト関係モデルに基づいている。「オブジェクト」はシステムに存在するエンティティを記述し、「関係」は様々なエンティティ間のリンクを示す。オブジェクトと関係は、ＳＤＭに関する意味情報を取り込むようさらに定義される。具体的には、オブジェクトは構成要素、エンドポイント、および資源に分割される。関係は、接続（通信とも称される）、包含、ホスティング、委託、およびレファレンスに分割される。オブジェクトおよび関係に関するさらなる詳細を後述する。

【００３０】

ＳＤＭは、システム部分の共通カテゴリを提供し、広範囲のシステムにツールのサポートを提供し、設計時における定義チェックの基礎を提供する「抽象定義」を含む。一組の抽象定義は、サービス設計に対する包括的な基礎を提供する。「具体的定義」は実際のシステムまたはデータセンタ設計の部分を表す。具体的定義は、抽象定義を選択し、具体的定義のメンバおよびそのプロパティに対する設定値を定義するインプリメンテーションを

定義することによって生成される。アプリケーションは、それらの具体的定義の集合を使用して生成される。

【００３１】

ＳＤＭは、関係のインスタンスが参加できる許可された関係の組に基づいてモデルが制限する「制約条件」も含んでいる。関係に関与するオブジェクトの構成によって異なる要件を記述する際に制約条件は有益である。例えば、通信プロトコルの各終端での参加者が互換性のあるセキュリティ設定を使用しているか否かを判定するために制約条件を使用することができる。

【００３２】

ＬＩＭファイル２０６とＳＤＭファイル２０４は、検証コンポーネント２０８のインターフェース／処理層２１０に入力される。インターフェース／処理層２１０はＳＤＭファイル２０４の検証を制御する。検証コンポーネント２０８はコンパイラとも称することができる。

【００３３】

インターフェース／処理層２１０は、ローダ２１２、検証部２１４、およびシミュレータ２１８を含む。ローダ２１２はＳＤＭファイル２０４を構文解析し、参照されるファイルまたはメモリ内型空間（in-memory type space）内の型をどれでもロードする。ＳＤＭファイル２０４はＳＤＭクラス（例えば、ＸＭＬで）を定義し、ローダ２１２がそれをメモリ内型空間内のＳＤＭクラスに変換する。検証部２１４は、ローダ２１２がＳＤＭファイル２０４をロードするためにＳＤＭファイル２０４が適切に書かれているか否かを検証して、ＳＤＭファイル２０４にエラーがあるか否かをチェックする。検証部２１４またはローダ２１２がエラーに遭遇した場合、エラーは、そのエラーの原因となった問題の表示を任意選択で含めて開発コンポーネント２０２に戻される。検証部２１４はさらに、設定値、型、およびＳＤＭファイルレファレンスをチェックして型空間を「ウォーク（walk）」する。通常、検証部２１４またはローダ２１２がエラーに遭遇すると、設計時検証は中止される。

【００３４】

次いでシミュレータ２１８は、後述するように、ＬＩＭ文書２０６によって定義された環境をシミュレーションし、拡張エンジン２２０、フローエンジン２２２、および制約条件エンジン２２４を呼び出す。シミュレータ２１８が遭遇したいかなるエラーも、任意選択でそのエラーの原因となった問題の表示を含めてコンポーネント２０２に（例えば、結果２２６として）戻される。通常、シミュレータ２１８がエラーに遭遇した場合、そのエラーの表示は開発コンポーネント２０２に戻されるが、設計時検証は先に進む。

【００３５】

インターフェース／処理層２１０は、ＳＤＭファイル２０４の検証を実行するために拡張エンジン２２０、フローエンジン２２２、および制約条件エンジン２２４を呼び出す。一般に、拡張エンジン２２０は、要求されるすべての追加インスタンスおよびアクションを、バイバリューの子（by-value children）および関係を含めて追加するために、または適切なインスタンスおよび関係を除去するために、ＳＤＭファイル２０４に基づいてインスタンス空間を作成するためにそのアクションの組を拡張する。フローエンジン２２２はインスタンス「グラフ（graph）」上を「ウォーク」し、そのインスタンスに対して定義されたフローを見つけ、フローの値を適切な値に設定する。制約条件エンジン２２４はインスタンス空間上を「ウォーク」し、各制約条件メンバを見つけ、評価する。

【００３６】

ある実施形態では、制約条件のすべてが真であると評価され（すなわち、制約条件のすべてが満たされ）、ＳＤＭ文書２０４を処理する際に検証コンポーネント２０８がエラーに遭遇しない場合、検証コンポーネント２０８はＳＤＭ文書２０４にデジタル署名する。このデジタル署名は、ＳＤＭ文書２０４が正常にコンパイルされたことを示している。様々な異なる方法のどの方法によってもＳＤＭ文書２０４にデジタル署名することができるが、これは、通常、ＳＤＭ文書２０４に基づくデジタル署名と、検証コンポーネント２０

８に関連付けられた公開／秘密鍵対とを生成を伴う。したがって、デジタル署名されたＳＤＭ文書は、（例えば、それらが配置される際に）既に検証されたものとして引き続き取り扱うことができる。デジタル署名されたＳＤＭ文書は、コンパイルされたＳＤＭ文書と称することもできる。

【００３７】

様々な制約条件を評価することに加え、検証コンポーネント２０８は、それら参照された文書のフルネーム（および任意選択でバージョン）によって他のＳＤＭ文書へのレファレンスを更新することもできる。したがって、コンパイルされたＳＤＭ文書はそれらフルネームを含んでいる。例えば、ＳＤＭ文書は別のＳＤＭ文書をインポートし、バージョンではなく名前によって当該他のＳＤＭ文書を参照することができ、検証コンポーネント２０８がその参照された文書をインポートする際、インポートされる文書のバージョンが特定され、コンパイルされたＳＤＭ文書に追加される。

【００３８】

本明細書での説明は、ＳＤＭ文書２０４に含まれる得る様々なメンバまたは定義を参照している。定義は、特定のインスタンスがインスタンス空間でどのように作成されるかを定義する。換言すれば、定義は何が実現可能かを定義する。メンバは定義に基づいて特定の用法またはシナリオを記述する。例えば、定義は特定のインスタンスには何が実現可能かを記述し、メンバはそれらの可能性のどれが特定のインスタンスに含まれるかを記述する。次の表に、メンバまたは定義の例ならびに各メンバまたは定義の要約を示す。

【００３９】

【表１】

メンバ／定義	説明
abstract （抽象）	モデル化されたシステムオブジェクトに対して作用するために必要となる設定を定義するために使用される型。
communication （通信）	エンドポイントメンバ間の通信関係を記述する。独立して配置されたソフトウェア要素間の相互作用を記述するために使用される。
constrain （制約条件）	制限を記述する。定義のメンバの値設定または関係の参加者に対する制限を特定するために使用することができる。
containment （包含）	包含関係定義は、1つの型が別の型を含むことができることを宣言する。包含関係メンバは、1つのオブジェクトメンバが別のオブジェクトメンバを含むことを宣言する。
delegation （委任）	エンドポイントメンバ間の委任関係を記述する。1つのシステムが別のシステムに動作を転送することを可能とするために使用される。
endpoint （エンドポイント）	通信エンドポイントを記述する。
flow （フロー）	一組の設定値に加えられる特定の変形を記述する。
hosting （ホスティング）	オブジェクトメンバ間のホスティング関係を記述する。ゲストが構築されるためにホストを要求していることを示すために使用される（例えば、ウェブサービスオブジェクト定義はIISオブジェクトホストを要求する場合がある）。
manager （マネージャ）	型および関係がカスタム動作を挿入することができる機構を記述する。複合制約条件およびフローのための実施態様を提供するために使用することができる。
object （オブジェクト）	独立して配置可能なアプリケーションの部分またはシステムが公開できる通信エンドポイントを記述するために使用されるシステム定義のような、（シミュレーションされた環境であっても）モデル化されているシステムに存在するエンティティを記述する。
reference （参照）	オブジェクトメンバ間の参照関係を設定する。オブジェクトメンバ間の依存関係を取り込むために使用される。
relationship （関係）	作成された際のオブジェクトメンバ間の関係を記述する。
resource （資源）	システム定義の一部として特定できる特有の要素を記述する。
subsystem （サブシステム）	完全なシステムとして取り扱うことができるが、一般的にはより大きなシステムの一部であるデータ、ソフトウェア、および／またはハードウェア要素の集合を記述する。
system （システム）	データ、ソフトウェア、および/またはハードウェア要素の集合を記述する。

【００４０】

本明細書で検討するエンジンは、バイバリューメンバ（by value member）、バイレファレンスメンバ（by reference member）、拡張、リスナ（listener）、およびトップレベル定義を参照する。バイバリューメンバは、拡張中にインスタンスが自動的に作成されるメンバを参照する。ある実施態様では、バイバリューメンバは抽象であることはできな

い。バイレファレンスメンバは、拡張中に拡張エンジンによってインスタンスを自動的に作成することができないメンバである。逆に、拡張エンジンでイベントを待ち受けている外部ソースはインスタンスを作成する（例えば、インターフェース／処理層２１０はシミュレーション中にインスタンスを作成することができ、もしくはシミュレーション中または配置中にインスタンスを作成することを設計者または開発者に（例えば、ユーザインターフェースによって）照会することができる。）。

【００４１】

拡張は、ルートインスタンスが与えられた場合、そのルートインスタンスに関してインスタンス空間にデータが取り込まれるプロセスを参照する。ルートインスタンスのメンバのすべては反復的に作成される。バイレファレンスメンバは拡張エンジンによっては作成されず、外部ソースによって作成される。

【００４２】

イベントがトリガされた場合、リスナ（例えば、シミュレータ２１８）に通知される。そのようなイベントの一例は、バイレファレンスメンバを作成することの要求である。トップレベル定義は、ＳＤＭ文書のルートレベルで定義された定義を参照する。ＳＤＭ文書は、通常、トップレベル定義を１つだけ有しているが、別の実施形態では、ＳＤＭ文書は２つ以上のトップレベル定義を有することができる。

【００４３】

図４は、設計時検証のプロセス例４００を示す流れ図である。プロセス４００は、図２の検証コンポーネント２０８によって実施され、ソフトウェア、ファームウェア、ハードウェア、またはこれらの組合せで実行することができる。

【００４４】

システムの作成、削除、または修正要求に応じて、ＳＤＭ文書およびＳＤＭ文書に関連付けられた任意のレファレンスを（動作４０２）、任意選択でそのＳＤＭ文書をロードすることができることを検証する。ある実施形態では、１つ以上のＬＩＭ文書は既にロード済みとなっており、１つ以上のＬＩＭ文書によって記述された１つ以上のインスタンス空間が既に作成されている。しかし、これがまだ行われていない場合、動作４０２で１つ以上のＬＩＭ文書がロードされ、１つ以上のＬＩＭ文書によって記述された１つ以上のインスタンス空間が作成される。ある実施形態では、動作４０２でＳＤＭ文書をロードする前に、本明細書に記載のように１つ以上のＬＩＭ文書によって記述された１つ以上のインスタンス空間が拡張、フロー、および制約条件エンジンを使用して作成される。

【００４５】

次いで、ＳＤＭ文書にエラーがあるか（動作４０４）またはＳＤＭ文書がロードされた際に型空間が作成されたか否かに関してチェックが行われる。動作４０４でのチェックとは、設計中のシステムではなく、ＳＤＭ文書自体の構造またはフォーマットを対象としている。例えば、ＳＤＭ文書のフォーマットが適切であること（例えば、文書が適切なＸＭＬフォーマットであること）、またはＳＤＭ文書に未知の要素または型がないこと、またはＳＤＭ文書内の型レファレンスが有効であることなどを検証するために、動作４０４でチェックを行うことができる。エラーがある場合、それらエラーに関する表示が戻され（動作４０６）、処理４００は終了する。これらのエラーは、例えば結果２２６として図２の開発コンポーネント２０２に戻すことができる。

【００４６】

しかし、エラーがない場合、動作４０２でロードされたＳＤＭ文書のトップレベル定義が選択される（動作４０８）。ある実施形態では、ＳＤＭ文書にはトップレベル定義が１つしかない。この場合、選択された定義に対するインスタンスが作成され（動作４１０）、インスタンス空間が拡張される（動作４１２）。トップレベル定義は、図４の拡張エンジン４１０によって実行される拡張の開始点として使用される。トップレベル定義を分析すること、ルートインスタンスの各メンバのインスタンスを作成すること、次いでそれら作成されたインスタンスのそれぞれを分析すること、それらのメンバのそれぞれに対する１つ以上のインスタンスを作成すること、等々によってインスタンス空間が拡張される。

このプロセスは「拡張エンジン」という見出しで以下に詳述する。

【００４７】

トップレベル定義に基づく拡張が完了すると、拡張されたインスタンス空間のインスタンスに対して定義されたフローが特定され、フローの値が適切な値に設定される（動作４１４）。このプロセスは「フローエンジン」という見出しで以下に詳述する。

【００４８】

フローの評価による値が設定された後、制約条件がチェックされる（動作４１６）。インスタンス空間の各制約条件は、「制約条件エンジン」という見出しで以下に詳述するように制約条件チェックの一部として評価される。これらの制約条件は、通常、（ＬＩＭ２０６で記述されるように）システムが検証される対象であるシミュレーションされた環境の設定に対してチェックされる。

【００４９】

制約条件チェックが完了すると、未選択のＳＤＭ文書に追加のトップレベル定義があるか否かに関してチェックされる（動作４１８）。ＳＤＭ文書に１つまたは複数の追加のトップレベル定義がある場合、それら追加のトップレベル定義のうちの１つが選択され（動作４２０）、処理は動作４１０に戻され、選択されたトップレベル定義のインスタンスがそこで作成される。

【００５０】

しかし、未選択のトップレベル定義がない場合、エラーが検出されたか否かに関してチェックされる（動作４２２）。動作４２２のエラーとは、動作４１２の拡張、動作４１４のフロー、動作４１６の制約条件チェック中に生成されたエラーおよび／または警告のことである。エラーまたは警告がある場合、それらのエラーまたは警告に関する表示が戻され（動作４０６）、処理４００が終了する。これらのエラーは、例えば結果２２６として図２の開発コンポーネント２０２に戻すことができる。

【００５１】

しかしエラーがない場合、図４の設計時検証プロセスは成功であり、成功の表示が戻される（動作４２４）。成功の表示は、例えば結果２２６として図２の開発コンポーネント２０２に戻すことができる。

【００５２】

動作４０６で開発コンポーネント２０２にエラーおよび／または警告を戻すことによって、または動作４２４で成功の表示を戻すことによって、ＳＤＭ文書２０４に記述されたシステムの設計時検証の結果が開発コンポーネント２０２に戻される。これらのエラーおよび／または警告もしくは成功の表示は、開発コンポーネント２０２を使用する設計者にも提示することができる。これにより、設計者に、設計中の（および、ＳＤＭ文書２０４に記述された）システムが検証済みであるか否か、または潜在的な問題があるか否かを通知することができる。設計者に特定のエラーおよび／または警告の表示を戻すことによって、潜在的な問題が何であるか、その問題を解決する方法（または、それらを解決する必要さえあるか否か）を設計者により詳細に知らせることができる。このフィードバックは設計プロセス中に設計者に提供することができるが、これは設計者に、設計したシステムに問題があることが判明する前にデータセンタでシステムを配置するよう試みることを要求しない。

【００５３】

シミュレーション

上述のように、図２のインターフェース／処理層２１０は、ＬＩＭ文書２０６に記述されている環境をシミュレートする。これによって、ＳＤＭ文書２０４によって記載されたシステムをシステム設計中に予測された環境に対して検証することができ、環境に対してシステムを配置する前にエラーを特定することが可能になる。

【００５４】

インスタンス空間内の多くのインスタンスは、以下で詳述するように拡張エンジンによって作成することができる。しかしそれらインスタンスのいくつかは、拡張エンジンでな

くシミュレータ２１８によって作成される。ある実施形態では、シミュレータ２１８によって作成されたインスタンスはトップレベル定義およびバイレファレンスメンバである。ある実施態様では、各トップレベル定義はそれ独自のインスタンス空間でシミュレーションされる。

【００５５】

シミュレータ２１８は、コンパイル中の文書のルートで見つかった定義の２つのカテゴリである（１）システム定義と、（２）ホストおよびゲストがシステム定義であるホスティング定義とをシミュレーションする。これら定義の２つのカテゴリは、大半がシミュレータ２１８によってインスタンスを作成することができるスタンドアロン定義である。他の定義は、一般に他の定義のコンテキストで使用されるだけであり、したがって拡張エンジン２２０によって作成される。ある実施形態では、シミュレータ２１８は、通常はあったとしてもほとんど意味のない検証を生じるような抽象定義はシミュレーションしない。

【００５６】

システム定義は、定義に対するインスタンスを作成し、そのインスタンスをインスタンス空間のルートインスタンスとして追加することによってシミュレーションされる。次いでそのインスタンスのメンバおよびネストされたメンバが、拡張エンジン２２０を呼び出すことによってインスタンス生成される。

【００５７】

ホスティング定義のシミュレーションは、ホストとゲストの間の検証を提供する。ある実施形態では、ホスティング関係のインスタンスはルートインスタンスとしては使用されず、したがって拡張エンジンが呼び出される前に特別なルートインスタンスが作成される。この特別なルートインスタンスは、３つのメンバである、（１）ホスティング関係のホストに対するシステム定義と、（２）ホスティング関係のゲストに対するシステム定義と、（３）最初の２つのメンバのホストおよびゲストシステム定義のインスタンスに関連付けられたホスティング定義とを有するシステム定義である。次いで拡張エンジン２２２は、特別なルートインスタンスのメンバおよびネストされたメンバをインスタンス生成するために呼び出される。

【００５８】

作成すべきインスタンス数は未知であり、配置環境によって異なるので、拡張エンジン２２０はバイレファレンスメンバのインスタンスは作成しない。さらに、バイレファレンスメンバは抽象である場合があり、したがってメンバをインスタンス生成するために使用される具体的定義は未知である。配置中にこれらの質問に回答するために人的介入（例えば、システム管理者による）が通常利用される。このような介入は任意選択で設計時検証中にも利用することができる。しかし、より一般的には、インターフェース／処理層２１０（例えば、シミュレータ２１８）は、次の通りに人間のオペレータが何を選ぶかをシミュレーションし、これはシミュレーション中の人間のオペレータの介入へのいかなる要求をも緩和する。シミュレータ２１８は、１つまたは最小発生数よりも大きなメンバのインスタンス数を作成する（例えば、バイレファレンスメンバのためにパラメータＭｉｎＯｃｃｕｒｓで指定されたように）。バイレファレンスメンバが抽象である場合、シミュレータ２１８はそのメンバのインスタンスは生成せず、情報を伴う（例えば、抽象バイレファレンスメンバをシミュレーションできないことを示す）警告をユーザに戻す。

【００５９】

拡張エンジン

拡張エンジン２２０は幅広いコマンドを検証コンポーネント２０８に入力することを可能とし、それらのコマンドの影響を受ける適切なオブジェクトおよび関係を特定するためにそれらのコマンドを拡張する。通常、これら多様なコマンドは、特定のオブジェクトまたは関係を作成または削除するためのコマンドである（例えば、ウェブアプリケーションを作成するためのコマンド、またはウェブアプリケーションを削除するためのコマンド）。拡張エンジン２２０は、バイバリューの子および関係を含めて要求されるすべての追加インスタンスおよび関係を追加し、または適切なインスタンスおよび関係を除去するため

に、ＳＤＭファイル２０４に基づくインスタンス空間を作成するためにそのアクションの組を拡張する。

【００６０】

最初、シミュレーション中にトップレベル定義のインスタンス（トップレベルインスタンスとも称される）が作成された後、インスタンス空間にデータを取り込むために拡張エンジン２２０が呼び出される。ルートインスタンスが拡張の開始位置として拡張エンジン２２０に渡される。拡張によって作成されたインスタンスはルートインスタンスと関係している。拡張エンジン２２０が特定のインスタンス空間に対して呼び出された最初の時点で、拡張エンジン２２０に渡されたルートインスタンスはトップレベルのインスタンス（トップレベル定義のインスタンス）である。しかし拡張エンジン２２０は、通常はそうであるが、繰り返し呼び出される場合がある。したがって、トップレベルインスタンスのメンバおよびネストされたメンバはそのような反復呼び出しのルートインスタンスとなる。

【００６１】

拡張エンジン２２０は、ルートインスタンスの各メンバの１つ以上のインスタンスを作成することから開始する。各メンバがインスタンス生成されると、それらはネストされたメンバに対して検査される。別のメンバ内で定義されているメンバは、当該他のメンバ内にネストされているものと称される。メンバのネスティングは複数レベルを通して行うことができる（例えば、メンバはその中にネストされた複数のメンバを定義することができ、それら複数のメンバのそれぞれはその中にネストされた複数のメンバを定義することができ、さらにそのそれぞれがその中にネストされた複数のメンバを定義することができ、．．．という具合に）。拡張プロセスは、インスタンス空間がルートインスタンスに関して完全に作成されるまでメンバおよびネストされたメンバのそれぞれに対して反復的に実行される。

【００６２】

ある実施形態では、拡張中に様々な型のメンバの１つの部分集合だけがインスタンス生成される。拡張中に作成されるインスタンスのこの部分集合は、次のオブジェクトメンバ、すなわちサブシステムと、資源と、エンドポイントとを含む。拡張中に作成されたインスタンスのこの部分集合は、次の関係メンバ、すなわち包含と、通信と、ホスティングと、委任と、レファレンスも含む。この部分集合に含まれない他のメンバは拡張によってはインスタンス生成されず、上記のインスタンスの１つが拡張によって作成された際に作成プロセスがメンバまたは定義を検査し、当該他のメンバに対する子メンバのインスタンスを自働的に作成する。当該他のメンバの例としては、フローメンバと制約条件メンバとが含まれる。

【００６３】

ある実施形態では、拡張エンジン２２０が呼び出された際にメンバのインスタンスが既に存在する場合、そのインスタンスは再度作成されない。しかしインスタンス空間が完全に作成されたこと（また、まだ作成されていないネストされたメンバをこの反復的な呼び出しから作成すること）を検証するために拡張エンジン２２０は各インスタンスに対して反復的に進む。

【００６４】

図５Ａおよび５Ｂは、インスタンス空間を作成する反復拡張プロセス例５００を説明する流れ図である。インスタンス空間は、図２のＳＤＭ文書２０４のようなＳＤＭ文書によって定義される。プロセス５００は、図２の拡張エンジン２２０によって実施されるが、ソフトウェア、ファームウェア、ハードウェア、またはこれらの組合せによって実行することができる。

【００６５】

最初にインスタンス定義がアクセスされる（動作５０２）。呼び出されると、拡張エンジン２２０には拡張されるべきインスタンス定義が通知され（本節で既に説明したルートインスタンス）、動作５０２でこのインスタンス定義がアクセスされる。拡張エンジン２２０にはインスタンス定義を渡すことができ、インスタンス定義のある場所の識別子（例

えば、ポインタ）を渡すことができ、またはインスタンス定義にアクセスすることができる場所および／またはその方法を通知することができる。プロセス５００は、トップレベル定義であるインスタンス定義を使用してシミュレータ２１８によって呼び出すか、またはプロセス５００自体によって反復的に呼び出すことができる。

【００６６】

次いでインスタンス定義のメンバが選択される（動作５０４）。メンバはインスタンス定義からいかなる順番でも選択することができるが、通常はインスタンス空間を定義するＳＤＭファイルにある順番で選択される。ある実施形態では、関係メンバが選択される前にすべてのオブジェクトメンバが選択される。

【００６７】

選択されたメンバがオブジェクトメンバか関係メンバかに関して次いでチェックされ（動作５０６）、このチェックの結果に基づいてプロセス５００は進む。選択されたメンバがオブジェクトメンバの場合、作成中のインスタンス空間には選択されたメンバの正確な数のインスタンスが既にあるか否かに関してチェックされる（動作５０８）。各メンバは、メンバのインスタンスをいくつ作成すべきかに関する表示を含む。ある実施形態では、メンバがより多くのインスタンスを作成するよう指定しない限り（メンバに明示的に指定されているか否かにかかわらず）各メンバの１つのインスタンスが作成されるように、１つのインスタンスというディフォルト値が使用される。

【００６８】

正確な数のインスタンスがまだ存在していない場合、オブジェクトメンバがバイバリューメンバであるか否かに関してチェックされる（動作５１０）。オブジェクトメンバがバイバリューメンバである場合、メンバ定義で指定されているようにオブジェクトの最小発生数が作成される（動作５１２）。しかしオブジェクトメンバがバイバリューメンバでない場合、それはバイレファレンスメンバであり、したがってリスナ（例えば、図２のシミュレータ２１８）がバイレファレンスメンバのインスタンスを作成することを可能にするイベントがトリガされる（動作５１４）。

【００６９】

オブジェクトメンバの１つ以上のインスタンスが動作５１２と５１４のいずれにかかわらず作成された後、または動作５０８でメンバの正確な数のインスタンスが既に存在すると判定された場合、プロセス５００は動作５１６に進む。動作５１６で、動作５１２または動作５１４で作成された１つ以上のインスタンス（または、動作５０８で既に作成されたと判定された１つ以上のインスタンス）の各メンバインスタンスのためにプロセス５００が呼び出される。プロセス５００を反復的に呼び出す場合、プロセス５００が呼び出された各メンバインスタンスは呼び出されたプロセス５００に対するインスタンス定義となる。したがって、インスタンス空間ですべてのインスタンスが作成されるまでプロセス５００はオブジェクトメンバで反復的に呼び出される。

【００７０】

さらに、動作５０４で選択されていなかったインスタンス定義に追加メンバがあるか否かに関してチェックがなされる（動作５１８）。インスタンス定義に１つまたは複数のそのような追加メンバがある場合、プロセス５００は動作５０４に戻り、それらメンバのうちの１つがそこで選択される。しかしインスタンス定義にそのような追加メンバがない場合、プロセス５００はこのインスタンス定義については完了している。

【００７１】

動作５０６に戻って、選択されたメンバが関係メンバである場合、プロセス５００は関係に関与するソースインスタンス数とターゲットインスタンス数とに基づいて作成すべき関係インスタンス数を特定する（図５Ｂの動作５２２）。作成すべき関係インスタンス数は、ソースインスタンス数にターゲットインスタンス数を乗じ、その結果を作成すべき関係インスタンス数とすることによって決定される。次いで、関係の特定された数のインスタンスが作成される（動作５２４）。

【００７２】

あるいは、ソースインスタンス数にターゲットインスタンス数を乗じることによって関係インスタンスの最大数を得ることができ、動作５２４でその最大数の部分集合を作成することができる。例えば複数の実現可能なホストが存在する場合、動作５２４でそのうちの１つだけを選択することができる。上記のバイレファレンスメンバと同様に、関係インスタンスの部分集合を選択するためにリスナ（例えば、人間のオペレータまたはシミュレータ２１８）を呼び出すことができる。シミュレータ２１８は、例えば関係インスタンスのどの部分集合を作成するかを決定するために使用されるべき様々な基準または従うべき規則によって構成またはプログラムすることができる。

【００７３】

関係メンバの型に応じて関係インスタンスは様々な方法で作成される。ある実施形態では、包含定義メンバ、通信定義メンバ、レファレンス定義メンバ、ホスティング定義メンバ、および委任定義メンバという型の関係メンバを使用することができる。包含定義メンバは、親を子メンバの各インスタンスに関連付ける１対多の関係である。包含定義メンバは、１つのオブジェクトメンバを他のオブジェクトメンバに含めることができることを記述している。通信定義メンバは、サーバとクライアントメンバインスタンスのすべての組合せを関連付ける多対多の関係である。通信定義メンバは、別個に配置されたソフトウェア要素間の相互作用を記述する。レファレンス定義メンバは、ソースメンバインスタンスに依存するすべての組合せを関連付ける多対多の関係である。レファレンス定義メンバは、（ホスティング定義メンバに加えて）インスタンス間の従属関係を取り込むために使用される。これらの従属関係は、例えば配置中の構築順番とインストールおよび更新中のシステム間のフローパラメータとを制御するために使用することができる。ホスティング定義メンバは、ホストをそのゲストのメンバインスタンスのそれぞれに関連付ける１対多の関係である。ホスティング定義メンバは、構築されるためにゲストがホストを必要とすることを特定するために使用される。委任定義メンバは、２つのシステムの通信エンドポイントを関連付ける１対１の関係である。委任定義メンバは、１つのシステムから別のシステムに動作を転送するために使用される（例えば、既に指示されているすべての相互作用を１つのシステムから別のシステムのエンドポイントに転送する）。

【００７４】

関係の各インスタンスに関して、ソースおよびターゲットインスタンスが関係のインスタンスに関連付けられる（動作５２６）。次いでプロセス５００は図５Ａの動作５１８に戻り、インスタンス定義に選択されるべき追加メンバがあるか否かに関してそこでチェックがなされる。

【００７５】

したがって、プロセス５００は反復的に呼び出され、プロセスが反復的に呼び出されるたびに様々なオブジェクトおよび関係インスタンスが作成される。反復拡張プロセスは、すべてのメンバインスタンスが作成された際にインスタンス空間の作成が何時完了したかを認識している。

【００７６】

拡張プロセスの反復的な性質により、結果として生じたインスタンスは、図２のフローエンジン２２２および制約条件エンジン２２４へのエントリポイントであるトップレベルのインスタンスによって相互にリンクされる。このようにして作成されたインスタンス空間は、ツリーの各ノードがメンバインスタンスを表すインスタンスツリーと見なすことができる。さらに、ツリーの各ノード（トップレベルのノードを除いて）は、ツリーのノードを作成するために拡張プロセスを呼び出したメンバインスタンスを表す親ノードにリンクされる。さらにツリーの各ノードは、ツリーのそのノードのメンバインスタンスのメンバである０またはそれより多い子ノードにリンクされる。リーフメンバがネストされたメンバを有しない場合（例えば、すべてのリーフメンバが０個の子ノードを有する場合）、反復拡張プロセスはインスタンスグラフの作成が完了したことを認識している。さらに、その型であるメンバを有する定義により拡張プロセスが無限になる可能性がある状況を避けるために、ある実施形態では、定義が祖先インスタンスとしてまだインスタンス生成さ

れていない場合、拡張プロセスはインスタンスを作成することしか選ばない。例えばディレクトリの定義があり、それが中にディレクトリを有する場合、単一ディレクトリがあるならば拡張エンジンはこれの拡張を停止する。

【００７７】

フローエンジン

拡張エンジン２２０が拡張プロセスを完了した後、シミュレータ２１８は拡張プロセスから生じたインスタンス空間に対してフローエンジン２２２を呼び出す。フローエンジン２２２は、インスタンス空間のインスタンスグラフ上を「ウォーク」し、インスタンスグラフで定義されたフローインスタンスを見つける。次いでフローエンジン２２２はフローの値を適切な値に設定する。

【００７８】

設定はＳＤＭファイル２０４の定義に対して宣言することができ、対応する設定値はそれらの定義から作成されたインスタンス上に存在することができる。設定値（setting values）は非計算設定値（non-calculated values）であっても、計算設定値（calculated setting values）であってもよい。

【００７９】

非計算設定値は、（例えば、ＳＤＭファイル２０４の）定義からのディフォルト値または初期値によって設定された設定値であるか、または値を明示的に設定するユーザによって設定された設定値である。非計算設定値は、それらが属するＳＤＭインスタンスの作成に対してだけ設定されるよう制限されても、ＳＤＭインスタンスの存続期間を通して変更されてもよい。非計算設定値は、フローの出力ではなく、フローへの入力として動作することができる。

【００８０】

計算設定値は、フローの出力である設定値である。その結果、フローは計算設定値の値を設定することができるが、一般に他のエンティティは設定できない。ある実施態様では、特定の計算設定値に値を供給する１つのフローしかない場合がある。そのような実施態様では、複数のフローが同じ設定値インスタンスに出力する場合、エラーが報告され、フローの出力がエラー状態に設定される。計算設定値は１つまたは複数の入力に基づいている。これらの入力は計算設定値および／または非計算設定値であってよい。

【００８１】

ある実施形態では、計算設定値は、構成値と非構成値という２つのカテゴリに分離することができる。構成値はフローによって計算される値であり、その結果、それらを含むＳＤＭインスタンスの構成が得られる。例えば、ファイルまたはデータベース名へのフルパスは構成値である。

【００８２】

非構成値はフローによって計算された値だが、これから構成は得られない。むしろ、これらは制約条件に交換された値としてのみ存在する。例えば、この結果得られたすべての継承したｗｅｂ．ｃｏｎｆｉｇ値を考慮するｗｅｂ．ｃｏｎｆｉｇ設定は非構成値である。非構成値の別の例は、この結果として得られた、ファイルの構成されたアクセス制御リストならびにファイルが含んでいるディレクトリおよびボリュームを考慮したファイルのアクセス制御リストである。

【００８３】

フローエンジン２２２は、任意選択で構成値と非構成値とを別個に取り扱うことができる。例えば、構成値に高い優先順位を与えることができ、したがって構成値を設定するフローを、非構成値を設定するフローの前に評価することができる。

【００８４】

一般に、インスタンス空間内のフローインスタンスを特定し、フローインスタンスへの入力の値を特定する（例えば、それらの入力のソースである他のインスタンスの設定値を特定する）ことによってフローは評価される。命令のモジュールまたは組はフローインスタンスによって参照されるが、その命令のモジュールまたは組はフローインスタンスに対

応するフロー機能を処理することができる。フロー機能は入力の値に基づいて処理され、そのフロー機能の結果が得られる。フロー機能の結果はフローインスタンスの出力として設定される。１つのフローは０またはそれを超える入力値と１つまたは複数の出力とを有することができる（例えば、１つまたは複数のインスタンスに対する設定値をフローの出力に基づいて設定することができる）。

【００８５】

幅広い種類のフロー機能のどれでも使用することができる。フロー機能はシステムの開発者および／または他の関係者によって定義することができる。例えばＳＤＮファイル２０４を作成しているのと同じ設計者が１つまたは複数のフロー機能を定義することができる。別の例として、開発コンポーネント２０２の製造業者またはある種の他のサードパーティ開発者が１つまたは複数のフロー機能を定義することができる。フロー機能自体は、代数、幾何、微積分などを利用する計算、日付および／または時刻に基づく計算、乱数計算などのような様々な計算および／または動作のどれでも含むことができる。

【００８６】

ある実施形態では、ＳＤＭファイル２０４で定義されたフローメンバがマネージャ定義に関連付けられている。マネージャ定義は、フロー機能を処理するために実行することができる一組の命令またはコードへのポインタを含む。したがって、フローインスタンスに対するフローを評価するために、関連付けられたマネージャ定義が特定され、関連付けられたマネージャが（まだインスタンス生成されていない場合は）インスタンス生成される。当該一組の命令またはコードへのポインタがマネージャから得られるが、これはフロー出力を決定するために当該一組の命令またはコードを得て、実行することを可能にする。

【００８７】

さらにある実施形態では、マネージャは型変換をサポートすることもできる。このような実施形態では、マネージャは、フローを異なる型の２つ以上の設定値に基づかせることができる。例えば、入力値は出力値と異なる型であってよい。マネージャは、そのような型の変換を行うことを可能にする１つまたは複数の方法を公開するか、または変換を実行するために取得し実行されるべき一組の命令またはコードへのポインタを含む。異なる変換を異なるマネージャによってサポートすることができる（例えば、マネージャ設計者は、所望の型の変換ならどれでもサポートするようマネージャを設計することができる）。

【００８８】

図６Ａおよび６Ｂは、評価フローのためのプロセス例６００を示す流れ図である。プロセス６００は、図２のフローエンジン２２２によって実施され、ソフトウェア、ファームウェア、ハードウェア、またはこれらの組合せで実行することができる。

【００８９】

最初に、インスタンス空間のインスタンスのすべての設定が見つけられる（動作６０２）。これらの設定は、インスタンスグラフの各ノードを設定値に関してチェックすることによるような様々な方法で見つけることができる。次いでインスタンス空間のすべてのフローインスタンスが特定される（動作６０４）。インスタンス空間の各フローインスタンスは「ダーティ」および「訪問なし」とマーク付けされる（動作６０６）。「ダーティ」というマーク付けは、フローがまだ実行されていないか、または評価されていないことを示している。「訪問なし」というマーク付けは、フローの実行または評価がまだ開始されていないということを示している。フローインスタンスのマーク付けは、幅広い様々な方法で実行することができる。例えば、各フローインスタンスは、フローインスタンスを「ダーティ」とマーク付けするよう設定ができる「ダーティ」フラグと、フローインスタンスを「訪問なし」とマーク付けするよう設定できる「訪問なし」フラグとを有することができる。他の例として、フローエンジン２２２は、リスト、表、またはエンジン２２２がフローインスタンスの記録およびそれらの関連付けられたマーク付けを保持することができる１つ以上の他のデータ構造を維持することができる。

【００９０】

フローインスタンスの結果として値を受け取る設定が次いで特定される（動作６０８）

。動作６０８で、計算された設定値のすべてが特定される。これらの設置自体は、それらが計算設定値であるかまたは非計算設定値であるかを示している（例えば、それらが非計算設定値であることを示すよう設定することができる「固定」属性のようなフラグまたは属性を有することができる）。動作６０８では、値の従属関係を示す従属関係グラフを任意選択で生成することもできる。従属関係グラフは、各設定値に関して、他のどの設定値にそれが依存しているか（すなわち、その入力である他の設定値）と他のどの設定値がそれに依存しているか（すなわち、それを入力として使用する他の設定値）とを示す。

【００９１】

動作６０４で「ダーティ」および「訪問なし」とマーク付けされていると特定されたフローインスタンスのうちの１つが次いで選択される（動作６１０）。フローインスタンスは、動作６０４でフローインスタンスが特定された順番に基づく順番、無作為な順番、入力値の番号に基づく順番などのようなどのような順番でも選択することができる。選択されたフローインスタンスは、次いで「訪問あり」とマーク付けされるが、依然として「ダーティ」とマーク付けされたままである（動作６１２）。これは、フローインスタンスに関連付けられたフローの評価または実行は開始されたが、そのフローはまだ実行も評価もされていないということを示している（例えば、フローに対する出力値はまだ決定されていない）。

【００９２】

次いでフローインスタンスに対する入力が特定される（動作６１４）。各フローインスタンスは、その入力のそれぞれに対するソース（例えば、別のインスタンスの特定の設定値）を特定するパラメータを有する。特定された入力にまだ値が割り当てられていないか否かに関するチェックが次いで行われる（図６Ｂの動作６１６）。非計算設定値である入力は、既に評価済みのフローインスタンスによって計算された計算設定値である入力がそうであるように、割り当てられた値となっている。しかし未評価のフローインスタンスによって計算された計算設定値にはまだ値は割り当てられていない（例えば、それらはヌル値を有する場合がある）。

【００９３】

特定された入力の１つまたは複数に割り当てられた値がない場合、（動作６１０で選択された）このフローインスタンスの評価は中断される（動作６１８）。入力値の計算を可能とするために評価が中断される。まだ値を有さない各入力に対して、入力値を設定するフローインスタンスが特定される（動作６２０）。次いでプロセス６００は、そのフローインスタンスの評価を開始するために図６ａの動作６１２に戻る（動作６２２）。次いで入力のすべてが値を有すると、動作６１８で中断されたフローインスタンスの評価が再開される（動作６２４）。例えば後述する動作６１６または動作６２６で、評価を再開することができる。動作６２０で特定されたフローインスタンスの１つまたは複数の評価自体は、まだ値を有していない特定された入力のために中断する場合があることに留意されたい。

【００９４】

動作６１６に戻ると、特定された１つ以上の入力が値を有している場合、その１つ以上の入力のその１つ以上の値はそれの１つ以上のソースから取得される（動作６２６）。その１つ以上の値を、ＳＤＭファイル２０４、ＳＤＭファイル２０４に基づいてインスタンス生成されたオブジェクト、フローインスタンス（フローの出力）などのような様々なソースから取得することができる。フローメンバに対してフロー機能を評価するために実行されるべき当該一組の命令またはコードが次いで特定される（動作６２８）。あるいは、実行される一組の命令またはコードではなく、フロー機能をハードウェアで実施することができる。当該一組の命令またはコードが次いで実行され、次にその実行された一組の命令またはコードの結果がフローの出力とも称されるフローインスタンスの出力として使用される（動作６３０）。

【００９５】

次いでそのフローが既に実行済みであるかまたは評価済みであることを示すために、フ

ローインスタンスが「ダーティでない」とマーク付けされる（動作６３２）。フローインスタンスは、動作６１２で既に「訪問あり」とマーク付けされているべきであるが、フローインスタンスがまだ「訪問あり」とマーク付けされていない場合は動作６３２でフローインスタンスに「訪問あり」とマーク付けすることができる。

【００９６】

「ダーティ」または「訪問なし」とマーク付けされたインスタンス空間で特定された追加のフローインスタンスがあるか否かに関して次いでチェックがなされる（動作６３４）。そのような追加のフローインスタンスがある場合、プロセス６００は図６Ａの動作６１０に戻り、そこでそのような追加のフローインスタンスのうちの１つが選択される。すべてのフローインスタンスが評価されると（例えば、動作６０４で特定されたフローインスタンスがどれも「ダーティ」とも「訪問なし」ともマーク付けされていない場合）、フロー評価プロセス６００は完了する（動作６３６）。フローが完了すると、シミュレータ２１８は設定に関する値を伴って完全なインスタンス空間を有する。

【００９７】

プロセス６００は、図６Ａおよび６Ｂで、まだ値を有さない入力を有しており、それらの値を設定する１つ以上のフローインスタンスを評価するフローインスタンスの中断中の評価として説明されている。しかし、プロセス６００は別の方法で実施することもできる。例えばそれらの値を設定した１つ以上のフローインスタンスを評価するのではなく、プロセス６００は、図６Ａの動作６１０に戻ることができ、そこでフローインスタンスのうちの１つが選択される（中断したフローに対して入力値を設定するか否かにかかわらず）。この代替方法でのフローインスタンスの評価は、いくつかのフローインスタンスの評価は完了しているが、すべての他のフローインスタンスの評価は中断されているという状況をもたらす場合がある。このような状況が発生した場合、特定された入力が値を有しているか否かを判定するために、動作６１６を各中断されたフローインスタンスに対して反復することができる。中断しているフローインスタンスの少なくとも１つに対する入力は値を有するべきであり、反復された動作６１６のこのプロセスはすべてのフローインスタンスが評価されるまで続けることができる。

【００９８】

フローエンジン２２２は、図６Ａおよび６Ｂのフロー評価プロセス中に循環レファレンスをも検出する。循環レファレンスとは、１つまたは複数のフローインスタンスが、それらの入力の少なくとも１つに関してそれぞれに他の出力に（直接的または間接的に）依存している状況のことである。例えば、フローインスタンスＡは、フローインスタンスＡへの入力値でもある出力値を有する。別の実施例によれば、フローインスタンスＡはフローインスタンスＢからの出力である入力値を有し、フローインスタンスＢはフローインスタンスＡからの出力である入力値を有する。さらに別の実施例では、フローインスタンスＡはフローインスタンスＢからの出力である入力値を有し、フローインスタンスＢはフローインスタンスＣからの出力である入力値を有し、フローインスタンスＣはフローインスタンスＡからの出力である入力値を有する。このような循環レファレンスは評価されないが、フローエンジン２２２によって検出され、循環レファレンスの設定値はエラー状態に設定される。

【００９９】

循環レファレンスは「ダーティ」および「訪問あり」のマーク付けを使用して検出することができる。上記の動作６１８でフローの評価が中断した場合、そのフローは「訪問あり」とマーク付けされるが「ダーティ」ともマーク付けされ、別のフローの評価が開始される。さらに当該他のフローの評価も中断することができる。フロー評価のこの中断中、フローが以前に「ダーティ」とマーク付けされているが依然として「ダーティ」であると評価された場合、循環レファレンスが発生している。換言すれば、１つまたは複数のフローの評価が中断し、その中断中に既に「訪問あり」および「ダーティ」とマーク付けされている評価のために別のフローが選択される。この場合、評価のために選択された当該他のフローの評価が以前に中断したことにより循環レファレンスが発生している。

【０１００】
あるいは、様々な他の方法のどれでも循環レファレンスを検出することができる。例えば、値の従属関係を示す従属関係グラフを上述のように動作６０８で作成することができ、循環レファレンスを特定するためにこの依存関係グラフを使用することができる。
【０１０１】
循環レファレンスが検出されると、その循環レファレンスの一部であるフローのすべてはエラーとマーク付けされる（例えば、周期性フローエラー）。循環レファレンスが検出された際に評価中のフローは循環レファレンスの一部であり、循環レファレンスが検出された際に評価中のフローから入力を受け取るすべての他のフローであり、それらのフローの１つから入力を受け取るすべての他のフローであり、以下同断である。次いで循環レファレンスの一部であるフローのすべての評価が停止する。残りのフローの評価を続行することができるように、これらのフローを任意選択で「訪問あり」および「ダーティでない」とマーク付けすることができる。
【０１０２】
フローは決定論的であっても非決定論的であってもよいということに留意されたい。決定論的フローは、入力値の特有の組によって呼び出された時は何時でも同じ値を出力する。例えば、フローは２つの入力値の和を出力することができる。入力値の特有の組が与えられるとフローは常に同じ値を出力するので、このようなフローは決定論的である。しかし非決定論的フローは、入力値の特有の組によって呼び出されると何時でも異なる値を出力することができる。例えば、フローは１つの入力値に基づいて乱数を発生することができる。同じ入力値によって呼び出されるたびに異なる乱数を発生することができるので、そのようなフローは非決定論的である。入力値を有しないフローは決定論的（常に同じ値を出力する）であっても非決定論的（呼び出されるたびに異なる値を出力する）であってもよい。
【０１０３】
ある実施形態では、設定値はエラー状態を有することができる。エラー状態では、設定値はエラー情報を保持することができる（例えば、エラーコードまたはエラーの説明）。設定値がエラー情報を保持しない場合、エラー情報は何らかの他の方法で（例えば、設定をエラー状態に置いたイベントにより）報告することができる。設定値は様々な方法でエラー状態に設定することができる。例えば、フローエンジン２２２は、フローを実行している際にエラーが発生した場合（例えば、フローがエラーを戻したから、フローエンジンが循環レファレンスを検出した場合または入力の問題によりフローが実行できなかった場合などにフローエンジンがフローを実行することができなかったからなどの理由から）、フローの出力値をエラー状態に設定する。
【０１０４】
エラー状態はフローによって伝播され、したがってエラー状態の入力値を有するいかなるフローによってもフローエンジンはそのエラー状態をその出力に伝播させることになる。このようなエラー状態は最終的にフローエンジン２２２によって開発コンポーネント２０２に（例えば、図２の結果２２６として）戻すことができる。あるいは、元のエラーの原因となったイベントは、フローによってエラー状態の伝播を戻すのではなく、フローエンジン２２２によって開発コンポーネント２０２に（例えば、図２の結果２２６として）戻すことができる。
【０１０５】
さらにある実施形態では、フロー入力および／または出力は、現在存在していないインスタンスを意味することができる（例えば、入力または出力は作成されていないメンバを意味することができる）。フロー入力が現在存在しないインスタンスを表している場合、その入力は未設定と見なされ、そのフローでディフォルトが定義されている場合はディフォルトに戻る（例えば、割り当てられた値に戻る）。フローにそのような割り当てられた値がない場合、入力はメンバのデータ型に関してディフォルト値に戻る。
【０１０６】

フロー出力が現在存在していないインスタンスを表している場合、その出力値は使用されない。フローの出力が存在していない場合、フローは値を設定しないので実行される必要はない。しかしすべての入力が現在存在しているわけではなくても、少なくとも１つの出力が存在していればそのフローは実行される。

【０１０７】

設計時検証が継承した設定パターンを考慮することが望ましい場合もある。継承した設定とは、インスタンス空間がオブジェクトの階層を有しており、設定値をオブジェクトに適用することができ、その結果、階層でそのオブジェクトの下にあるすべてのオブジェクトに対する設定用の値が得られるという状態のことである。事実上、当該オブジェクトの下のオブジェクトがディフォルト値を変更するために独自の明示的に適用される値を有しない限り、これによって当該オブジェクトの下にあるオブジェクトに対するディフォルト値が作成される。これらを、オブジェクトに対して適用した設定（特定オブジェクトに適用した設定）およびオブジェクトに対して結果として得られた設定（親オブジェクトから継承した設定と組み合わされた適用した設定の結果）として参照することができる。

【０１０８】

そのような継承された設定には異なる方法で対応することができる。ある実施形態では、そのような継承した設定はＳＤＭファイル２０４の作成中に対応することができる。ＳＤＭファイル２０４の設計者は、そのような継承した設定が存在する状態に気付いており、ＳＤＭファイル２０４の設計でそれらを考慮する。例えば、ＳＤＭファイル２０４の設計者は、適用した設定と結果として得られた設定の両方を宣言することができるが、結果として得られた設定だけに対する制約条件しか宣言することはできない。

【０１０９】

別の実施形態では、各設定が適用した設定値と結果として得られた設定値の両方を有する。したがって１つのインスタンス内の各設定は、実際には１つではなく２つの値を有することになる。適用した設定値は、初期設定または値の明示的な設定によって設定された値である。結果として得られた設定値は、フローによって設定される。ディフォルトにより、設定へのレファレンスは、フローが結果の設定値を設定していない限り当該結果として得られた設定値を取る。この場合、レファレンスは適用した設定値を取ることになる。一方、フロー出力へのレファレンスはディフォルトによる結果値を表す。さらにすべてのレファレンスは、２つの設定値のうちの１つを明示的に表すために、任意選択でバリュー型修飾子（value type qualifier）を追加することができる。

【０１１０】

通常、フローの出力はフローが評価されるたびに置き換えられる。すなわち、フローを実行すると、フローのそれ以前のいかなる出力も上書きする。しかし各設定が適用した設定値と結果設定値の両方を有する実施形態では、適用した設定値をその入力のうちの１つとして取るためにフローを書くことができる。適切に構築されたフロー機能が与えられると、フローの出力をフローの他の１つ以上の入力に添付された適用した設定とすることができる。したがって、これらの実施形態では、フロー出力に新しい入力を効果的に添付することが可能である。

【０１１１】

制約条件エンジン

シミュレータ２１８は、ＳＤＭ文書２０４によって定義されたシステムに対して制約条件を評価するために制約条件エンジン２２４を呼び出す。制約条件エンジン２２４によって実行される評価は、（環境がＳＤＭ文書２０４によって述べられた制約条件を満たすことを検証するために）環境に対してＳＤＭ文書２０４で定義された制約条件を評価し、かつ／または（ＳＤＭ文書２０４で定義されたシステムが環境によって述べられた制約条件を満たすことを検証するために）ＳＤＭ文書２０４（例えば、ＬＩＭ文書２０６で）で定義されたシステムに対して環境によって定義された制約条件を評価する。

【０１１２】

制約条件エンジン２２４はインスタンス空間上を「ウォーク」し、各制約条件メンバを

見つけてこれを評価する。制約条件エンジン２２４は、ＳＤＭ文書２０４によって定義された制約条件メンバならびにＬＩＭ文書２０６によって定義された制約条件メンバを評価することができる。通常、シミュレータ２１８はフローエンジン２２２がフローの評価を終了した後で制約条件エンジン２２４を呼び出す。したがって、拡張およびフロー評価プロセスから結果として得られたインスタンス空間に基づいて制約条件チェックが実行される。しかし代替形態では、フローエンジン２２２がフローの評価を終了する前に制約条件エンジン２２４が呼び出される場合がある。

【０１１３】

制約条件は様々な形態を取ることができ、ある実施形態では、制約条件は設定制約条件、関係制約条件、およびオブジェクト制約条件を含む。設定制約条件は、オブジェクトに対する設定を制約し、マネージャまたはコードが評価を実行することによってサポートされる。関係制約条件は、オブジェクトが参加することのできる１つ以上の関係を制約する。オブジェクト制約条件は、関係の１つ以上のオブジェクトを制約する。

【０１１４】

制約条件定義は、通常、制約条件が適用されるターゲット定義を特定し、その制約条件が何であるかをさらに定義する。設定制約条件の場合、一組の命令またはコード（またはハードウェアモジュール）は、ターゲットが制約条件を満たしているか否かを評価するために実行することができる制約条件（例えば、制約条件定義によって示された）に関連付けられている。ある実施形態では、制約条件定義は、当該一組の命令またはコードを指し示し、評価フローに関して上記のマネージャ定義と類似しているマネージャ定義に関連付けられている。

【０１１５】

図７Ａ～７Ｅは、評価制約条件のプロセス例７００を示す流れ図である。プロセス７００は、図２の制約条件エンジン２２４によって実施され、ソフトウェア、ファームウェア、ハードウェア、またはこれらの組合せで実行することができる。

【０１１６】

最初、インスタンス空間のすべての制約条件が特定される（動作７０２）。動作７０２でのこの特定は、設定制約条件、関係制約条件、およびオブジェクト制約条件の特定を含む。動作７０２の特定は、（例えば、ＳＤＭ文書２０４で）システムによって定義された制約条件ならびに（例えば、ＬＩＭ文書２０６で）環境によって定義された制約条件を特定することを含むことができる。

【０１１７】

次いで特定された制約条件のうちの１つが選択される（動作７０４）。その制約条件は様々な異なる方法のどれでも選択することができる。例えば、制約条件は動作７０２で特定された順番で選択することができ、異なる型の制約条件を他の型の前に選択することができ（例えば、設定制約条件が関係制約条件の前に選択される）、制約条件は無作為に選択することができる、などである。選択された制約条件は、次いでその制約条件が関係制約条件か、設定制約条件か、オブジェクト制約条件かに基づいて評価される。

【０１１８】

制約条件が設定制約条件である場合、制約条件のターゲットインスタンスからの１つ以上の適切な値が取得される（動作７０６）。制約条件は、評価されるべきターゲットインスタンスを特定する（例えば、制約条件のターゲットであるオブジェクトを名前により特定する）。制約条件は、ターゲットインスタンスのどの１つ以上の値が評価されるべきかをも特定し、それら特定された１つ以上の設定値がターゲットインスタンスから取得される。制約条件のために制約条件機能を評価するために実行されるべき当該一組の命令またはコードが次いで特定される（動作７０８）。あるいは、実行される一組の命令またはコードではなく、制約条件機能をハードウェアで実施することができる。動作７０６で取得された１つ以上の値が制約条件を満たしているか否かを判定するために、次いで当該一組の命令またはコードが実行される（動作７１０）。例えば、当該一組の命令は、１つ以上の取得された値を制約条件の値と比較し、その取得された１つ以上の値の１つまたは複数

が制約条件の１つまたは複数の設定値と同じ（一致する）場合はその制約条件が満たされると判定することができる。

【０１１９】

次いで制約条件機能の結果が戻される（動作７１２）。その結果は、結果２２６の一部として図２の開発コンポーネント２０２に戻すことができる。制約条件は、名前または動作７１２で結果の一部として戻すことができる制約条件を記述する他の情報を任意選択で含むことができ、これによってエラーを記述する追加情報を開発コンポーネント２０２で設計者に提示することができる。

【０１２０】

未選択のインスタンス空間に追加の制約条件があるか否かに関してチェックがなされる（動作７１４）。制約条件のすべてが選択され、評価された場合、制約条件のチェックプロセスは完了する（動作７１６）。しかし１つまたは複数の制約条件が未選択の場合、プロセス７００は特定された制約条件の別の１つを選択するために動作７０４に戻る。

【０１２１】

動作７０４に戻ると、選択された制約条件がオブジェクト制約条件である場合、その制約条件のターゲットインスタンスに対する主要なロールと主要なオブジェクト定義が特定される（図７Ｂの動作７１８）。オブジェクト制約条件の場合、制約条件のターゲットインスタンスは関係インスタンスである。ターゲットインスタンスの主要なオブジェクト定義は、選択された制約条件が目標とする１つ以上のオブジェクトの名前を表している。ターゲットインスタンスの主要なロールは、主要なオブジェクト定義によって特定される定義に一致しなければならない関係のロールを特定する。次いで制約条件が目標とする副次的なロールと副次的なオブジェクト定義があるか否かに関してチェックがなされる（動作７２０）。ある実施形態では、制約条件は複数のオブジェクトおよび／またはロールを目標とすることができ、制約条件が目標とする第２のロールおよび／またはオブジェクト定義を特定するために副次的なロールと副次的なオブジェクト定義を使用することができる。

【０１２２】

制約条件が目標とする副次的なロールと副次的なオブジェクト定義がある場合、制約条件の主要なロールと副次的なロールの両方および主要なオブジェクト定義と副次的なオブジェクト定義の両方がターゲットインスタンスと一致するか否かに関してチェックがなされる（動作７２２）。ロールの１つまたは複数またはオブジェクト定義の１つまたは複数がターゲットインスタンスと一致しない場合、以下で詳述するように、一致数変数は０に設定され（動作７２４）、プロセス７００は図７Ｃの動作７２６に進む。

【０１２３】

しかし制約条件の主要なロールと副次的なロールの両方および主要なオブジェクト定義と副次的なオブジェクト定義の両方がターゲットインスタンスと一致する場合、主要なオブジェクトインスタンスに対するすべてのネストされた制約条件が評価される（動作７２８）。ネストされた制約条件は、制約条件定義のいかなる子インスタンスならびにそれらの子のいずれをも表す。ネストされた制約条件は、オブジェクト制約条件、関係制約条件、および／または設定制約条件であってよい。ネストされた制約条件のどれかが以前に評価されている場合、それらの結果は記録されており、それらの結果を動作７２８で使用することができる。しかし、まだ評価されていないネストされた制約条件のどれであっても、プロセス７００を呼び出し、ネストされた制約条件のそれぞれを図７Ａの動作７０４で選択された制約条件として使用することにより制約条件は評価される。

【０１２４】

すべてのネストされた制約条件が一度評価されると、すべてのネストされた制約条件が真であると評価するか否かに関してチェックがなされる（動作７３０）。ネストされた制約条件の１つまたは複数が真と評価しない場合、一致数変数は０に設定される（動作７２４）。しかしネストされた制約条件のすべてが真であると評価する場合、以下で詳述するように一致数変数は１に設定され（動作７３２）、プロセス７００は図７Ｃの動作７２６

に進む。

【０１２５】

あるいは、動作７３０はネストされた制約条件のすべてが真と評価するか否かに関するテストなので、ある条件下ではネストされた制約条件のすべてを評価する必要はない。ネストされた制約条件が真でないと評価されるとすぐに、ネストされた制約条件の評価を中止することができ、プロセス７００は動作７３０に進むことができる。例えば、３つのネストされた制約条件があり、評価される第２のネストされた制約条件が偽、すなわち真でないと評価する場合、第３のネストされた制約条件が真であると評価するか否かにかかわらず、動作７３０での結果は同じ（動作７２４に進む）になる。

【０１２６】

動作７２０に戻り、制約条件が目標とする副次的なロールと副次的なオブジェクト定義がない場合、制約条件の主要なロールと主要なオブジェクト定義がターゲットインスタンスと一致するか否かに関してチェックがなされる（動作７３４）。動作７３４は動作７２２と類似しているが、これは主要なロールと主要なオブジェクト定義に基づいてのみ実行される。つまり、動作７３４には副次的なロールと副次的なオブジェクト定義はなく、主要なロールと主要なオブジェクト定義だけが考慮される。

【０１２７】

主要なロールまたは主要なオブジェクト定義がターゲットインスタンスと一致しない場合、以下で詳述するように一致数変数は０に設定され（動作７３６）、プロセス７００は図７Ｃの動作７２６に進む。しかし制約条件の主要なロールと主要なオブジェクト定義がオブジェクトインスタンスと一致しない場合、プロセス７００は動作７２８に進み、そこで主要オブジェクトインスタンスに対するすべてのネストされた制約条件が評価される。

【０１２８】

次に図７Ｃの動作７２６に進み、一致数の値が少なくとも最小数であり、最大数よりも大きくないか否かに関してチェックがなされる（動作７２６）。これらの最小数および最大数は制約条件によって特定され、いかなる値であってもよい。一致数が少なくとも最小数であるが最大数よりは大きくない場合、真の値がこの制約条件に戻される（動作７３８）。真の戻り値は他のインスタンスを評価しているプロセスにも戻すことができる。例えば、ネストされた制約条件を図７Ｂの動作７２８で評価中の場合に、ネストされた制約条件が真であると評価したと動作７３８が判定したならば、動作７３０で判定を行うために親ノードを評価するプロセスにその真値を戻すことができる。

【０１２９】

真の戻り値は、結果２２６の一部として図２の開発コンポーネント２０２に任意選択で戻すこともできる。さらに、満たされた（真であると評価された）名前または制約条件の他の特定情報を任意選択で開発コンポーネント２０２に戻すことができる。

【０１３０】

しかし一致数が最小数でさえないかまたは最大数よりも大きい場合、この制約条件に対してエラーメッセージを生成すべきか否かに関してチェックがなされる（動作７４０）。制約条件は、制約条件に対してエラーメッセージを戻すべきか否かを示すパラメータまたは設定を有する。このパラメータまたは設定は、例えば制約条件の設計者または開発者によって設定することができる。動作７４０のチェックは、制約条件に対するこのパラメータまたは設定がエラーメッセージを生成すべきであると示しているか否かをチェックすることによって行うことができる。

【０１３１】

制約条件に対してエラーメッセージが生成されるべきである場合、エラーメッセージが生成される（動作７４２）。エラーメッセージは、ユーザ（例えば、設計者または開発者）がエラーの原因となった制約条件の性質を特定するのに役立つ名前または他の特定情報を任意選択で含むことができる。次いでこの制約条件に対して偽の値が戻される（動作７４４）。偽の値および／または動作７４２で生成されたエラーメッセージを、結果２２６の一部として図２の開発コンポーネント２０２に戻すことができる。動作７３８での真の

戻り値と同様に、偽の戻り値および／またはエラーメッセージを他のインスタンスを評価しているプロセスにも戻すことができる。

【０１３２】

動作７３８で真の値が戻された後、または動作７４４で偽の値が戻された後、未選択の追加の制約条件があるか否かをチェックするためにプロセス７００は図７Ａの動作７１４に戻る。

【０１３３】

図７Ａの動作７０４に戻ると、選択された制約条件が関係制約条件である場合、一致数変数は初期設定される（図７Ｄの動作７４６）。一致数変数は、例えば一致数変数の値を０に設定することによって初期設定することができる。制約条件のターゲットオブジェクトインスタンスが参加する１つ以上の関係インスタンスが次いで特定され（動作７４８）、特定された関係インスタンスのうちの１つが特定される（動作７５０）。

【０１３４】

次いで、関係定義および制約条件の指示が選択された関係インスタンスと一致するか否かに関してチェックがなされる（動作７５２）。関係定義は関係の型を表し（例えば、ホスティング、委任、通信など）、当該指示はロールまたは関係の指示（例えば、制約条件がホスティング関係のホストを目標としているかゲストを目標としているか）を表す。関係定義および制約条件の指示が関係定義および選択された関係インスタンスの指示と同じ場合、関係定義および制約条件の指示は特定された関係インスタンスと一致する。

【０１３５】

関係定義および制約条件の指示が選択された関係インスタンスと一致しない場合、動作７４８で特定された未選択の追加の関係があるか否かに関してチェックがなされる（動作７５４）。１つまたは複数のそのような追加の関係がある場合、プロセス７００は動作７５０に戻り、未選択の追加の関係のうちの１つがそこで選択される。しかしそのような追加の特定された関係がない場合、プロセス７００は以下で詳述する図７Ｅの動作７５６に進む。

【０１３６】

動作７５２に戻ると、関係定義および制約条件の指示が選択された関係インスタンスと一致する場合、制約条件にターゲットオブジェクト定義があるか否かに関してチェックがなされる（動作７５８）。ターゲットオブジェクト定義は制約条件内のオプションであり、評価中の関係インスタンスの他の側にあるオブジェクトインスタンスを表している。例えば、関係インスタンスがホスティング関係であり、ターゲットオブジェクトインスタンスがホストである場合、ホスティング関係の他の側のオブジェクトインスタンスはゲストインスタンスである。制約条件にターゲットオブジェクト定義がある場合、制約条件のターゲットオブジェクト定義が関係インスタンスの他端のインスタンスと一致するか否かに関してチェックがなされる（動作７６０）。制約条件のターゲットオブジェクト定義と関係インスタンスの他端のインスタンスが同じ場合、制約条件のターゲットオブジェクト定義は関係インスタンスの他端のインスタンスと一致する。

【０１３７】

制約条件内のターゲットオブジェクト定義が関係インスタンスの他端のインスタンスと一致しない場合、未選択の追加の特定された関係があるか否かに関してチェックするためにプロセス７００は動作７５４に進む。

【０１３８】

しかし制約条件のターゲットオブジェクト定義が関係インスタンスの他端のインスタンスと一致しない場合、または制約条件にターゲットオブジェクト定義がない場合、関係インスタンスに関してすべてのネストされた制約条件を評価するためにプロセス７００は先に進む（図７Ｅの動作７６２）。次いでプロセス７００は、ネストされた制約条件のすべてが真と評価するか否かに基づいて先に進む（動作７６４）。動作７６２および７６４の評価およびチェックは、図７Ｂの動作７２８および７３０の評価およびチェックと同様である。

【０１３９】
ネストされた制約条件のすべてが真と評価したわけではない場合、未選択の追加の特定された関係があるか否かをチェックするためにプロセス７００は図７Ｄの動作７５４に戻る。しかしネストされた制約条件のすべてが真と評価した場合、一致数変数は増分される（動作７６６）。一致数変数は、値１だけのように異なる量だけ増分することができる。未選択の追加の特定された関係があるか否かをチェックするためにプロセス７００は次いで図７Ｄの動作７５４に戻る。
【０１４０】
次いで図７Ｃの動作７２６と同様に、一致数の値が少なくとも最小数であり最大数より大きくないか否かに関してチェックがなされる（動作７５６）。一致数の値が少なくとも最小数であり最大数より大きくない場合、図７Ｃの動作７３８と同様に真の値が戻される（動作７６８）。しかし一致数の値が最小数でさえなく最大数よりも大きくない場合、図７Ｃの動作７４０と同様にこの制約条件に対してエラーが発生するか否かに関してチェックがなされる（動作７７０）。エラーが発生することになる場合、図７Ｃの動作７４２と同様にエラーメッセージが生成される（動作７７２）。エラーメッセージが生成される際、または生成されるべきエラーメッセージがない場合、図７Ｃの動作７４４と同様に偽の値が戻される（動作７７４）。
【０１４１】
さらに、制約条件チェックプロセスが数群の制約条件に対しても同様に行われることに留意されたい。一群の制約条件を評価するためにプロセス７００を使用して各制約条件を別個に評価するが、当該群の結果は個々の制約条件評価の結果に依存する。ある実施形態では、当該群の少なくとも１つの制約条件が真と評価するならば、当該群は真と評価する。
【０１４２】
プロセス７００で成功の制約条件検証（例えば、真の値）または失敗の制約条件検証（例えば、偽の値）を示す値をエラーメッセージと同様に開発コンポーネント２０２に戻すことにより、ＳＤＭ文書２０４に記述されたシステムの設計時検証の結果が開発コンポーネント２０２に戻される。これらの表示および／またはエラーメッセージは、開発コンポーネント２０２を使用する設計者にも提示することができる。これにより、その設計者に、設計中の（そしてＳＤＭ文書２０４に記述されている）システムが検証済みであるか否か、または潜在的に問題があるか否かを通知することができる。エラーメッセージを戻すことにより、設計者に、潜在的な問題が何であるか、またそれらをどのようにして解消するかに関してより詳しく通知することができる。設計されたシステムに問題があることに気付く前にデータセンタでシステムを配置しようと設計者が試みる必要なしに、このフィードバックを設計プロセス中に設計者に提供することができる。
【０１４３】
エラー実施例
本節では、図２の評価構成要素２０８によってエラーおよび警告を報告することのできる方法の一実施態様を説明する。エラーおよび警告は、本節で検討するＸＭＬ形式例のような様々なフォーマットのどれであってもよい。検証コンポーネント２０８がＣＬＲ（Ｃｏｍｍｏｎ　Ｌａｎｇｕａｇｅ　Ｒｕｎｔｉｍｅ（共通言語ランタイム））を通して使用される場合、本節で検討するＸＭＬフォーマットと等価の情報を有するクラスを戻すことができる。
【０１４４】
本節は、エラーのクラスに共通の値を示す。ここでは追加の値は除外しない。
【０１４５】
型空間の検証中に発生したエラー（構文解析および解決エラー）は、コンパイルされたＳＤＭ文書が書かれることを阻止する。しかしインスタンス空間検証中に発生したエラー（フローおよび制約条件エラー）は、コンパイルされたＳＤＭ文書が書かれることを阻止しない。

【０１４６】
基礎エラーフォーマット
基礎エラーフォーマットは、すべてのエラーに共通のエラー情報を含んでいる。基礎エラーフォーマットに対するＸＭＬフォーマット例を以下に示す。
【０１４７】
【表２】

```
<xs:complexType name="DocumentError">
        <xs:attribute name="category" type="Severity" use="required"/>
        <xs:attribute name="code" type="xs:string" use="required"/>
        <xs:attribute name="description" type="xs:string" use="required"/>
        <xs:attribute name="sdmFile" type="xs:string" use="required" />
```

【０１４８】
【表３】

```
</xs:complexType>
```

【０１４９】
基礎エラーフォーマットの属性または要素を以下の表に示す。
【０１５０】
【表４】

属性/要素	説明
category	エラーまたは警告（場所未特定）
code	エラーコード（場所未特定）
description	エラーの説明（場所特定済み）
sdmFile	エラーが発生したsdmファイル名（ネーム空間名でなくファイル名）

【０１５１】
構文解析エラー
構文解析エラーは、ＳＤＭ文書のロードを試みる際の障害に起因する。このエラーはラインおよびコラム番号を含む。構文エラーのＸＭＬフォーマット例を以下に示す。
【０１５２】

【表５】

```
<xs:complexType name="DocumentParseError">
    <xs:complexContent>
        <xs:extension base="DocumentError">
            <xs:attribute name="lineNumber" type="xs:int"/>
            <xs:attribute name="columnNumber" type="xs:int"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０１５３】

構文解析エラーフォーマットの属性または要素を以下の表に示す。

【０１５４】

【表６】

属性/要素	説明
lineNumber	sdmファイル属性によって特定されるファイルのライン番号。
columnNumber	エラーの開始を特定するコラム番号。

【０１５５】

解決エラー

解決エラーは、ファイルの型をロードし、解決する際の障害に起因する。このクラスのエラーは、インポートを解決する障害と、型、メンバ、および経路を解決する障害を含んでいる。この段階で完全に構文解析されたオブジェクトモデルがあるので、ファイルからのラインおよびコラム番号でなくオブジェクトモデルに関してエラーを発生させる部分に対して参照を行うことができる。解決エラーフォーマットに対するＸＭＬフォーマットの一例を以下に示す。

【０１５６】

【表７】

```
<xs:complexType name="DocumentResolutionError">
    <xs:complexContent>
        <xs:extension base="DocumentError">
            <xs:attribute name="statement" type="documentPath"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０１５７】

解決エラーフォーマットの属性または要素を以下の表に示す。

【０１５８】
【表８】

属性/要素	説明
statement	エラーの原因となったステートメントへの経路。

【０１５９】
フローエラー
フローが実行され、その入力または出力の１つまたは複数に対してエラーを戻す際にフローエラーが発生する。フローメンバは、現行の入力および出力値の組を含めてその型と、障害が発生したコンテキストのままで、宛先設定メンバと共に特定される。フローエラーフォーマットに対するＸＭＬフォーマット例を以下に示す。

【０１６０】
【表９】

```
<xs:complexType name="DocumentFlowError">
    <xs:complexContent>
        <xs:extension base="DocumentError">
            <xs:sequence>
                <xs:element name="inputValue" type="flowInput"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="outputValue" type="flowOutput"
                    minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="flowType" type="documentPath"/>
            <xs:attribute name="flowMember" type="documentPath"/>
            <xs:attribute name="memberContext"
```

【０１６１】
【表１０】

```
                type="containmentPath"/>
            <xs:attribute name="flowErrorID" type="xs:int"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０１６２】
フローエラーフォーマットの属性または要素を以下の表に示す。

【０１６３】
【表１１】

属性/要素	説明
InputValue	ソース設定、フロー設定、およびフローに対する値の記述
OuputValue	宛先設定、フロー設定、およびフローに対する値の記述
flowType	フローメンバが解決される型
flowMember	フローメンバの宣言
memberContext	このフローが評価された包含階層。この階層のルートは、本発明者らがシミュレーションしているターゲット型となり、中間ノードは、ルートとフローメンバの間のメンバとなる。
flowErrorID	設計データセクションからのエラー記述を検索するために使用されるフローから戻されたカスタムエラーid。

【０１６４】
フロー入力フォーマットの記述に対するＸＭＬフォーマット例を以下に示す。
【０１６５】
【表１２】

```
<xs:complexType name="flowInput">
    <xs:sequence>
```

【０１６６】
【表１３】

```
        <xs:element name="value" type="xs:anyType" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="setting" type="sdm:simpleName"/>
    <xs:attribute name="source" type="documentPath"/>
</xs:complexType>
```

【０１６７】
フロー入力フォーマットの記述の属性または要素を以下の表に示す。
【０１６８】
【表１４】

属性/要素	説明
Value	入力の値
Setting	これが入力であるフロー型に対するフロー設定
Source	本発明者らが値の入手源とする設定メンバ

【０１６９】
フロー出力フォーマットの記述に対するＸＭＬフォーマット例を以下に示す。
【０１７０】

【表１５】

```
<xs:complexType name="flowOutput">
    <xs:sequence>
        <xs:element name="value" type="xs:anyType" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="setting" type="sdm:simpleName"/>
    <xs:attribute name="destination" type="documentPath"/>
</xs:complexType>
```

【０１７１】

フロー出力フォーマットの記述の属性または要素を以下の表に示す。

【０１７２】

【表１６】

属性/要素	説明
Value	出力の値
Setting	出力値を提供するフロー設定
Destination	本発明者らが値をフローする先の設定メンバ

【０１７３】

制約条件エラー

設定制約条件が評価されてエラーを戻す場合、またはガードがそのカーディナリティ制約条件に失敗した場合、または１つの制約条件グループの少なくとも１つのメンバが真と評価することに失敗した場合、制約条件エラーが発生する。

【０１７４】

設定制約条件が失敗した場合、制約条件メンバ宣言が、制約条件型、制約条件が実行されるコンテキスト、および制約条件に対する当該一組の入力値と共に戻される。制約条件は、追加エラー情報を検索するために使用することのできるカスタムエラーｉｄを戻すこともできる。そのような制約条件エラーフォーマットに対するＸＭＬフォーマット例を以下に示す。

【０１７５】

【表１７】

```
<xs:complexType name="DocumentConstraintError">
    <xs:complexContent>
        <xs:extension base="DocumentError">
            <xs:sequence>
                <xs:element name="inputValue"
```

【０１７６】

【表１８】

```
                    type="constraintInput"/>
            </xs:sequence>
            <xs:attribute name="constraintMember"
                type=" documentPath"/>
            <xs:attribute name="constraintType"
                type=" documentPath"/>
            <xs:attribute name="memberContext"
                type="containmentPath"/>
            <xs:attribute name="constraintErrorID" type="xs:int"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０１７７】
そのような制約条件エラーフォーマットの属性または要素を以下の表に示す。
【０１７８】
【表１９】

属性/要素	説明
inputValue	制約条件に対する入力値のリスト
constaintMember	制約条件メンバの宣言への経路
constraintType	制約条件メンバが解決される制約条件型
Member context	この制約条件が評価された包含階層。この階層のルートは、本発明者らがシミュレーションしているターゲット型となり、中間ノードは、ルートと制約条件メンバの間のメンバとなる。

【０１７９】
制約条件入力フォーマットの記述に対するＸＭＬフォーマットを以下に示す。
【０１８０】

【表２０】

```
<xs:complexType name="constraintInput">
    <xs:sequence>
        <xs:element name="value" type="xs:anyType"/>
    </xs:sequence>
    <xs:attribute name="setting" type="sdm:simpleName"/>
    <xs:attribute name="source" type=" documentPath"/>
</xs:complexType>
```

【０１８１】

フロー入力フォーマットの記述の属性または要素を以下の表に示す。

【０１８２】

【表２１】

属性/要素	説明
Value	入力の値
Setting	値の対象となる制約条件設定
Source	入力値に対するソースメンバ

【０１８３】

ガードまたはグループが失敗した場合、制約条件メンバがメンバコンテキストおよび評価コンテキストと共に特定される。ネストされた制約条件の評価対象である型または関係を評価コンテキストが特定する場合、評価コンテキストはネストされた制約条件に対するメンバコンテキストとは異なる。このような制約条件エラーフォーマットに対するＸＭＬフォーマット例を以下に示す。

【０１８４】

【表２２】

```
<xs:complexType name="constraintGuardOrGroupError">
    <xs:attribute name="constraintMember" type=" documentPath"/>
    <xs:attribute name="evaluationContext" type="containmentPath"/>
    <xs:attribute name="memberContext" type="containmentPath"/>
</xs:complexType>
```

【０１８５】

このような制約条件エラーフォーマットの属性または要素を以下の表に示す。

【０１８６】

【表２３】

属性/要素	説明
ConstraintMember	ガード（型制約条件または関係制約条件）またはグループ（型制約条件グループまたは関係制約条件グループ）の宣言。
EvaluationContext	制約条件の評価対象であるメンバに対するルートからのメンバ経路
memberContext	制約条件が宣言されるメンバに対するルートからのメンバ経路

【０１８７】
文書経路
これは、ＳＤＭ文書内の特定のＳＤＭ要素への経路である。文書経路フォーマットに対するＸＭＬフォーマット例を以下に示す。
【０１８８】
【表２４】

```
<xs:simpleType name="documentPath">
    <xs:restriction base="xs:string">
    </xs:restriction>
</xs:simpleType>
```

【０１８９】
この経路は、一般に以下の形式を有する。
【０１９０】
namespace/root_type(/nested_type)*/member(/nested_member)*
関係および制約条件に対するよく知られた設定がメンバとして取り扱われる。
【０１９１】
包含経路
包含経路は、ルートから特定メンバへの一連のメンバを特定する。この経路は、メンバをルートシミュレーション構成要素からそれらのメンバなどを介して、エラーを起こしたメンバに達するまでナビゲートする。文書経路フォーマットに対するＸＭＬフォーマット例を以下に示す。
【０１９２】
【表２５】

```
<xs:simpleType name="containmentPath">
    <xs:restriction base="xs:string">
    </xs:restriction>
</xs:simpleType>
```

【０１９３】
この経路は一般に次の形式を有する。
【０１９４】

【表２６】

Namespace / rootType (/ nested_type) * member (/ nested_member)*

【０１９５】
エラー例
コンパイラは、ユーザに表示されるコンパイルからエラーおよび警告を戻す。ＳＤＭ文書のコンパイル中に捉えたエラーは、様々なレベルの重大さを有し得る。大部分のエラーは致命的だが、他のエラーの原因となる可能性があるかまたは無視しても安全であるものが一部にはある。レベルのそれぞれに対する重大さに関する詳細な例を以下に示す。

【０１９６】
エラー：このカテゴリのエラーは致命的であり、以下のような含みを有する。
・出力ｓｄｍファイル（．ｓｄｍＤｏｃｕｍｅｎｔ）はコンパイルから作成することはできない。
・エラーは、コンパイラにコンパイルの現在の段階を完了させ、中止させる。例えば、ＸＭＬ構文解析エラーが発生した場合、文書はロードされない。ロードエラーが発生した場合、さらに多くのロードエラーが発見され得るが、フローおよび制約条件チェックを続けない。

【０１９７】
警告１（Ｗ１）：警告はＳＤＭ文書がコンパイルされることを防がない。警告１に対するガイドラインは以下のものを含む。
・コンパイルの将来の段階でエラーを生じる可能性がある。例えば、レファレンスファイルが見つからない場合、そのレファレンスからいかなる型が使用されていたとしてもターゲットＳＤＭファイルのロード中にエラーは発生しない。
・インスタンス空間シミュレーション中のエラー。

【０１９８】
警告２（Ｗ２）：警告２は重大さの最も低いエラーである。例えば、ＸＭＬファイルで余分な属性が見つかる。警告２に対するガイドラインを以下に示す。
・警告２は無視しても安全である。
・ＳＤＭをクリーンアップするために固定することが推奨される。

【０１９９】
警告３（Ｗ３）：警告３は情報メッセージである。
・警告３は情報を提供する。
・ディフォルトにより、これらの警告は報告されない。

【０２００】
以下の表に、図２の検証コンポーネント２０８によって報告することのできるエラーの例を列挙する。一般的なコンパイラエラー、文書ロードエラー、定義空間検証エラー、フローエラー、および制約条件エラーの例を示すために別個の表を使用する。エラーそれらのレベルの記述が以下の表に含まれる。エラーのそれぞれに対するフォーマットも示される（例えば、エラー（基礎エラー）フォーマット、構文解析フォーマット、解決フォーマット、フローフォーマット、および制約条件フォーマット）。

【０２０１】

【表２７】
一般的なコンパイラエラー

フォーマット	レベル	説明	規則
Error	エラー	予想外の例外	コンパイル中に例外が投入された
Error	エラー	未発見のSDMファイル	
Error	W1	未発見のレファレンスファイル	ターゲットファイルのコンパイル前にレファレンスがロードされた場合、そのレファレンスは発見されない。
Error	エラー	SDMファイルのロードに失敗	ロード中に発生し、この文書のロード失敗の原因となった構文解析エラー。
Error	エラー	中途終了したコンパイル。検出されたエラーが多すぎる。	コンパイル中に100を超えるエラーが検出された。
Error	W1	無効なレファレンスファイル	レファレンスファイルはコンパイルされたsdm文書ではない。
Error	エラー	ソースファイルがレファレンスとしてロードされた。	ソースファイルはレファレンスファイルとは同じではあることはできない。
Error	エラー	ファイルのハッシュ取得に失敗した。	Sdmに列挙されているファイルは存在するはずであり、そのハッシュを取得するための読み出しアクセスを有する必要がある。
Error	W3	sdmファイルは遅延の符号が付いている。	Sdmファイルには遅延の符号だけでなく、完全な符号を付けるべきである。
Resolution	エラー	ソース設定値がファイル資源から消失している。	構築中にファイルを探し出せるように、System.OperatingSystem.FileSystem: Fileの型の資源メンバは、設定値「Source」を含んでいる必要がある。
Resolution	エラー	資源内にファイルは見つからなかった。	System.OberatingSystem.FileSystem:Fileの型の資源メンバに対する「Source」設定値によって指定された経路でこのファイルを見つけることはできなかった。
Error	エラー	キーファイルが存在しないか、破損しているか、または秘密鍵が見つからず、ターゲットには完全に符号が付いている。	keyFile変数は無効である。ファイルが存在しないか、キーファイルが破損しているか、または秘密鍵が見つからない場合、delaySign = 偽である。
Error	エラー	キーコンテナが存在しない。	keyContainer変数は無効である。
Error	エラー	公開および/または秘密鍵を抽出するためにキーファイルにアクセスすることができない。	keyFileが（コンテナでなく）ファイルである場合、それはアクセス可能でなければならない。

【０２０２】

【表２８】
文書ロードエラー

フォーマット	レベル	説明	規則
Parse	エラー	無効なXML文書	このxml文書は有効なxml文書ではなかったか、またはxsdスキーマに準拠しなかった。例えば、消失属性/要素、未知の属性/要素、属性/要素に対する無効値
Parse	エラー	無効な設定値スキーマ	失敗した設定値に対する検証
Resolution	エラー	見つかった未知のオブジェクト	型を構文解析する際に見つかった未知のオブジェクト
Resolution	エラー	未知のメンバ	メンバを構文解析する際に見つかった未知のメンバ
Resolution	エラー	未知のロールリスト値	見つかった未知のロールリスト値
Resolution	エラー	既に定義済みの型名	単一sdm文書内に設定、構成要素、ポートおよび資源型、関係、制約条件およびフロー型、およびマネージャに対する単一ネーム空間がある。
Resolution	エラー	現在の型内で既に定義済みのメンバ名	型、関係、制約条件、フローメンバ、および設定宣言は、型内の単一ネーム空間のすべての部分である。
Resolution	エラー	制約条件は既に存在する	所与の名前に対して制約条件は1つしか存在することができない。
Resolution	エラー	設定値は既に存在する	所与の名前に対して型またはメンバに設定値は1つしか存在することができない。

【０２０３】

【表２９－１】
定義空間検証エラー

フォーマット	レベル	説明	規則
Resolution	エラー	このオブジェクトの型の解決に失敗した	メンバおよび型は有効な型を参照しなければならない
Resolution	エラー	ホスティング関係のホストの解決に失敗した	メンバは有効な型を参照しなければならない。これは、設定宣言、型、関係、フローおよび制約条件メンバを含む。
Resolution	エラー	ホスティング関係のゲストの解決に失敗した	
Resolution	エラー	レファレンス関係の従属メンバの解決に失敗した	
Resolution	エラー	レファレンス関係のソースメンバの解決に失敗した	
Resolution	エラー	包含関係の子メンバの解決に失敗した	
Resolution	エラー	包含関係の親メンバの解決に失敗した	
Resolution	エラー	通信関係のサーバメンバの解決に失敗した	
Resolution	エラー	通信関係のクライアントメンバの解決に失敗した	
Resolution	エラー	委任関係のプロキシメンバの解決に失敗した	
Resolution	エラー	委任関係の代表メンバの解決に失敗した	
Resolution	エラー	包含関係の子型の解決に失敗した	関係型は有効な型を参照しなければならない。
Resolution	エラー	包含関係の親型の解決に失敗した	
Resolution	エラー	ホスティング関係のホスト型の解決に失敗した	
Resolution	エラー	ホスティング関係のゲスト型の解決に失敗した	
Resolution	エラー	レファレンス関係の従属型の解決に失敗した	
Resolution	エラー	レファレンス関係のソース型の解決に失敗した	
Resolution	エラー	通信関係のサーバ型の解決に失敗した	
Resolution	エラー	通信関係のクライアント型の解決に失敗した	

【０２０４】

【表２９－２】

Resolution	エラー	委任関係のプロキシ型の解決に失敗した	
Resolution	エラー	委任関係の代表型の解決に失敗した	
Resolution	エラー	関係の基礎の解決に失敗した	関係は実施または拡張された有効な型を有する必要がある
Resolution	エラー	この設定値に関して設定宣言型が見つからなかった	設定宣言の型は無効である
Resolution	エラー	無効設定宣言型	設定宣言の型は無効である。
Resolution	エラー	マネージャの解決に失敗した	オブジェクトがマネージャを有する場合、それを解決しなければならない。
Resolution	エラー	メンバ宣言は、メンバ宣言型と一致しない型を参照した	メンバは有効な型を参照しなければならない。これは、設定宣言、型、関係、フローおよび包含メンバを含む。
Resolution	エラー	メンバ名は既に基準型で定義されている	基準型のメンバは導出された型によって変更することができない
Resolution	エラー	インポートステートメントの解決に失敗した	すべてのインポートステートメントはsdm文書に解決しなければならない
Resolution	エラー	基準型の解決に失敗した	構成要素のための基準型は解決可能でなければならず、構成要素と同じ型でなければならない
Resolution	エラー	無効な基準型	
Resolution	エラー	設定型宣言はサポートされていない基準型を使用した	設定型はSDMスキーマで定義された一組の限定されたxsd型から導出されなければならない
Resolution	エラー	設定ネーム空間包含はsdm包含と一致しない	設定スキーマに含まれるネーム空間は、設定スキーマを含むsdm文書にインポートされたsdmネーム空間と一致しなければならない
Resolution	W2	複合設定型はUIが表示するには複雑すぎる	表示は、属性を有さず、簡素な型から導出した一連の単集合の元だけを含む複合型しかサポートすることはできない
Resolution	エラー	経路の解決に失敗した	経路上のすべての要素は、経路上の以前の要素または経路のルートコンテキストに関連付けられた型のメンバに解決しなければならない。最後の要素に関連付けられた型は、経路の宛先と整合しなければならない。
Resolution	エラー	経路型は一致しない	
Resolution	エラー	入力経路は書き込み専用設定を目標とする	
Resolution	エラー	出力経路は固定設定を目標とする	

【０２０５】

【表２９－３】

Resolution	エラー	フロー入力型とソース設定型の間に使用可能な変換はない	本発明者らは、基礎データ型間の少数のロスレス変換しかサポートしない。
Resolution	エラー	フロー出力型と宛先設定型の間に使用可能な変換はない	
Resolution	エラー	制約条件入力型とソース設定型の間に使用可能な変換はない	
Resolution	エラー	設定値は、配置中にしか指定することはできない。	この設定値は配置時用にマーク付けされている。
Resolution	エラー	設定値は設定型に準拠しなかった	すべての設定値は設定型に準拠しなければならない
Resolution	エラー	設定値はnillableでない	設定値はブランクであり、フロー値のターゲットではない
Resolution	エラー	マネージャの解決に失敗した	型によって特定されたマネージャは解決可能でなければならず、コンパイラおよびランタイムによってロード可能でなければならない。
Resolution	エラー	マネージャのロードに失敗した	
Resolution	エラー	単一フロー入力のために提供された複数の入力宣言	フローまたは制約条件入力には経路が1つだけ許可される
Resolution	エラー	単一制約条件入力のために提供された複数の入力宣言	

【０２０６】

【表３０】

フローエラー

フォーマット	レベル	説明	規則
Flow	エラー	要求された入力は定義されていない	フローに対するすべての要求された入力は、そのフローを評価することができるようになる前に定義されなければならない。
Flow	エラー	フロー出力はターゲット設定型に準拠しなかった	フローによって生成された値は、フローによって宣言された出力型に準拠しなかった。
Flow	エラー	フローが失敗した。詳細はフロー型を参照されたい。	実行された場合、フローはエラーを戻す。

【０２０７】

【表３１】
制約条件エラー

フォーマット	レベル	説明	規則
Constraint	エラー	要求された制約条件は真と評価しなかった	制約条件宣言はその最低カーディナリティを満たさなかった。
Constraint	エラー	限界のある制約条件が上限を超えた	制約条件がその最大カーディナリティを超えた。
Constraint	エラー	不法制約条件が真と評価した	最大カーディナリティ0の制約条件が真と評価した。
Constraint	エラー	制約条件グループの制約条件はどれも真と評価しなかった	制約条件グループの少なくとも1つの制約条件は真と評価しなければならない。
SettingConstraint	エラー	要求された入力が定義されていない	制約条件に対するすべての要求された入力は、その制約条件を評価することができるようになる前に提供されなければならない。
SettingConstraint	エラー	制約条件が失敗した。詳細は制約条件を参照されたい。	制約条件が評価され、エラーが戻された。（本発明者らは、実行中の障害を偽が戻されたことと区別すべきか）
Constraint	W3	制約条件をチェックすることができなかった	制約条件をチェックするために十分な情報が使用可能でない。

【０２０８】
ＳＤＭ実施例
ＳＤＭの実施例を以下に示す。このＳＤＭ実施例は、「定義」の節（節１）、「アーキテクチャ概要」の節（節２）、および「実施態様詳細」の節（節３）を含む。この実施例では様々な具体的な値および要件が記載されており、すべての実施態様がそれらすべての具体的な値および要件に限定されるわけではないということが理解されよう。
【０２０９】
１　定義
【０２１０】
【表３２】

用語	定義
要求変更	モデル化されたシステムに対する一組の変更を記述する平叙文書
オブジェクト定義	オブジェクト定義は、モデル化されたシステムの要素の構成を取り込む。
関係定義	モデル化されたシステム要素間の相互作用を記述するために定義が使用される
システム定義モデル(SDM)文書	オブジェクト、関係、設定、フロー、および制約条件に関する定義を含んでいるxml文書。
ソフトウェア分散ユニット（SDU）	型をSDM管理システムに配置するために要求される一組のSDM文書および関連付けられたバイナリ情報（ファイル）の組合せ
SDMインスタンス空間	モデル化されたシステムを表す一組の具体的な型および関係

【０２１１】
２　アーキテクチャ概要
システム定義モデル（ＳＤＭ）は、本発明者らが「モデル化されたシステム」と称する接続されたシステムの構成およびこのシステムの要素間の相互作用の記述をサポートする。
【０２１２】
ＳＤＭは、オブジェクト関係モデルに基づいている。本発明者らは、モデル化されたシステムに存在する要素を記述するために「オブジェクト」を使用し、それらの間のリンクを示すために「関係（relationships）」を使用する。ＳＤＭは、ＳＤＭにとって重要な意味論を取り込むためにオブジェクトと関係をさらに改良する。具体的には、本発明者らは、オブジェクトを「システム（systems）」、「エンドポイント（endpoints）」、および「資源（resources）」に分け、本発明者らは、関係を「通信（communication）」、「包含（containment）」、「ホスティング（hosting）」、「委任（delegation）」、および「レファレンス（reference）」に分割する。
【０２１３】
本発明者らは、広範なアプリケーションに対するツールサポートを可能にし、設計時の型チェックに対する基礎を提供するシステム部分の共通の範疇を提供するために「抽象オブジェクト定義」を使用する。本発明者らは、当該一組の抽象定義がシステム設計に包括的基礎を提供するものと推測し、本発明者らはそれらがゆっくりと徐々に変化するものと推測する。
【０２１４】
この場合、本発明者らは、実際のアプリケーションまたはデータセンタ設計の部分を表す「具体的定義」を定義するための基礎として抽象定義を使用する。本発明者らは、抽象定義を取り、具体的な型のメンバと設定値とを定義する実施態様をそのプロパティに提供する。次いで、本発明者らは、これら定義の集合からシステムを構築する。
【０２１５】
「制約条件」は、インスタンスが参加することのできる許可された一組の関係に対して制約をモデル化するために使用される。本発明者らは、関係に関与するオブジェクトの構成に依存するきめの細かい要件を取り込むために制約条件を使用する。例えば、通信プロトコルの各端部の参加者が整合性のあるセキュリティ設定を使用していることを検証するために制約を使用することができる。
【０２１６】
ターゲットシステムに対する変更を有効にするために、「変更要求」と呼ばれる要求された変更を記述する宣言を使用することができる。ＳＤＭは、「ＳＤＭ実行モデル」の一部として変更要求を拡張、検証、および実行するために使用されるプロセスを定義する。
【０２１７】
「インスタンス空間」は、管理されたアプリケーションの所望の状態と現在の状態の両方を取り込む。本発明者らは、インスタンス空間の変更を追跡することができ、変更を開始した変更要求にそれらを関連付ける。
【０２１８】
カスタマイズされた動作をランタイムに提供し、かつランタイムとモデル化されたシステムの間の相互作用をサポートするために「マネージャ」が使用される。
【０２１９】
２．１　オブジェクト
モデル化されたシステムの論理的態様と物理的態様の両方を表すために、オブジェクトが使用される。例えば、これらはＩＩＳ内のファイル、ディレクトリ、および構成を表すために使用することができる。これらは、アプリケーションまたは分散型システム境界を表すためにも使用することができる。
【０２２０】
本発明者らは、オブジェクトを３つのカテゴリに分ける。各カテゴリは、属性または特

定モデルのインスタンス全体に強制された継承構造によってではなく、ＳＤＭモデル自体の一部として公開され、理解されるべきであると本発明者らが考えるモデル化されたシステムの態様を表す。

【０２２１】

２．１．１　資源

資源は、システムモデルを作成するために組み合わせることができる動作の基本的単位を表す。システムモデルに構造を追加するために他の資源を１つにグループ化するよう資源を使用することができる。各資源は、それのモデル化された動作を実行するために要求することのできる他の資源に対する従属関係を表すことができる。資源のインスタンスを作成するために、その資源に対してホスト環境が特定されなければならない。

【０２２２】

資源の例には、オペレーティングシステムの一部としてファイル、ディレクトリ、およびレジストリキーおよび値、ＩＩＳおよび表の一部としてウェブディレクトリおよびウェブファイル、およびデータベースの一部として行および記憶された手順が含まれる。

【０２２３】

２．１．２　システム

システムは、洗練されたタスクを実行する資源の集合を表すため、または洗練されたタスクを実行するために相互作用するシステムの集合を表すために使用される。システム境界内の資源は、その境界外の資源に対する従属関係を表すことはできない。同時に、同じシステム境界内の他の資源と通信するためにそれらが使用する機構を文書化するために本発明者らは資源を要求しない。

【０２２４】

これは、システム間の相互作用は、通信関係を使用して明示的にモデル化されなければならないということを意味している。これはまた、一般にシステムは配置の最小単位であるということをも意味している。資源は他の資源との文書化されていない相互作用を有する場合があるので、本発明者らは、それらが独立して配置された場合にそれらが機能することは保証できない。

【０２２５】

システムの例には、オペレーティングシステム、ＩＩＳでホストされるウェブアプリケーション、およびＳＱＬによってホストされるデータベースが含まれる。

【０２２６】

２．１．３　エンドポイント

システムが他のシステムとの通信をサポートするために公開するインターフェースを定義するためにエンドポイントが使用される。本発明者らは資源がシステム境界を横断する従属関係を有することを許可しないので、システムが正確に動作できるようにするために必要となる相互作用をモデル化するためにエンドポイントを使用する必要がある。

【０２２７】

エンドポイントの例には、Ｈｔｔｐ、Ｔｃｐｉｐ、およびＳｏａｐエンドポイントが含まれる。

【０２２８】

２．２　関係

本発明者らは、オブジェクト間に発生する相互作用の態様を取り込むために関係を使用する。すべての関係はバイナリであり有向である。オブジェクト間の相互作用を取り込むのと同様に、関係は、関係に参加するオブジェクトに制約条件を課すことができ、また参加者間に構成情報をフローすることができる。

【０２２９】

ここでもう一度、本発明者らは、それら関係に特定の意味論を付加するためにＳＤＭモデルによって理解される一組の関係を特定する。これにより本発明者らはモデルシステムのランタイム動作に関して判断することができる。

【０２３０】

具体的には、本発明者らが特定のオブジェクトインスタンスに関して記述する必要のある態様は以下の通りである。
ａ）その存続期間
ｂ）それと相互作用できる人物
ｃ）システムの構造に表示される場所
ｄ）それが通信する人物
ｅ）その作業を行うために依存する人物
ｆ）それが実行されるべき環境
ｇ）それが正確に動作しているか否か、また使用可能か否か

【０２３１】

本発明者らは、アプリケーションの許容される構造を定義するためにも関係を使用する。これらの態様には以下のものが含まれる。
ａ）他のオブジェクトが含むことのできるのはどのようなオブジェクトか
ｂ）１つに接続できるのはどのエンドポイントか
ｃ）特定のオブジェクトをホストすることができるのはどのような環境か
以下に示す関係のそれぞれは、上記態様の１つまたは複数の一部を取り込む。

【０２３２】

２．２．１　包含

ＳＤＭモデルの包含構造を定義するために包含関係が使用される。あるオブジェクトが別のオブジェクトを含むことができることを示すために包含関係の存在が使用される。設計者または立案者がシステム、資源、およびエンドポイントを関連付けることのできる方法について設計者および立案者に選択肢を公開する際に、この情報は設計環境をガイドするために設計時に使用される。

【０２３３】

ＳＤＭモデルの構造を限定するために包含関係が定義される。例えば、立案者は、ウェブディレクトリ、ファイルシステムディレクトリ、およびファイルを含むことのできるウェブアプリケーションモデルを設計する場合がある。別の立案者は、ウェブサイトを含むことができるようにモデルを拡張する場合がある。これらモデルの基準構造を理解することによって、オペレータおよびツールビルダは、モデルのどの環境インスタンスが要求することができるか、またそれが配置されたシステム上に特定のモデルがどのような効果をもたらすことができるかに関して判断することができる。例えば、上記で定義した第１のモデルで、オペレータはアプリケーションを配置することができるようにするためにウェブサイトを供給する。

【０２３４】

包含関係は、オブジェクトの存続期間、オブジェクトの所有者、およびオブジェクトがモデル化されたシステムの構造のどこに表示されるかを定義する。オブジェクトに対して単一の親コンテナがある場合、親の存続期間は含まれているオブジェクトの存続期間に境界を設ける。親は、含まれているオブジェクトを所有すると表現することもできる。これは、親がオブジェクトの可視性を制御しており、オブジェクトまたはそのオブジェクトの一部をその親に公開するか否かを判定することができることを意味している。最終的に、親はオブジェクトにコンテキスト情報を提供する。このコンテキスト情報は、設計者がそれらアプリケーションの構造を他の設計者または当該アプリケーションのユーザに伝達するのに役立つよう使用することができる。例えば、実行中アプリケーションのエラーの場所を特定する際、包含チェーンは、アプリケーションのどの部分に障害が生じたかに関する詳細を示す豊かなコンテキスト情報を提供することができる。

【０２３５】

包含関係の用途には、ウェブアプリケーションの仮想ディレクトリと、ウェブアプリケーションによって配置されたファイルおよびディレクトリの構造とを表すことが含まれる。包含は、ウェブアプリケーションと、それを含む接続されたシステムモデルとの間の関係を記述するためにも使用される。

【０２３６】
２．２．２　ホスティング
ホスティング関係は、モデル化中の環境の包含構造をモデル化するために使用される。これらの関係は、環境に対する機能上の制約を取り込む。例えば、複数のファイルが１つのディレクトリに含まれることを必要とするか、または複数のウェブディレクトリが１つのウェブサイトに含まれることを必要とする制約である。
【０２３７】
ホスティング関係は、オブジェクトの存続期間とそれが実行される環境の両方を定義する。存在するために、オブジェクトは少なくとも１つのホストを有するべきである。これは、そのホストの存続期間がオブジェクトの存続期間に限界をつけており、仮にそのホストがオフラインとマーク付けされているならば、それもまたオフラインとマーク付けされるべきであるということを意味している。
【０２３８】
ホスティング関係は、そのホストによって定義された環境内でオブジェクトのインスタンスを作成し、維持することを担当する。この属性分配は、ゲストまたはホストによってではなく、関係によってゲストのインスタンスを作成することの責任をホストに帰する。ホスティング関係の範囲を提供することによって、複数のホスト環境によってゲストをサポートすることができる。
【０２３９】
ホスティング関係の存在は、ホストオブジェクトにゲストオブジェクトを置くことが可能であるということを示している。別のシステムでホストされるべき多くの資源を含んでいるシステムの場合、そのゲストシステムのすべての資源はホストシステムの少なくとも１つの資源に対するホスティング関係を有するべきである。
【０２４０】
２．２．３　通信
通信関係は、接続されたシステムのサブシステム間で行われる通信をモデル化するために使用される。それらはエンドポイントオブジェクト間で表現されるので、それらはエンドポイントを公開するシステム間でしか確立することはできない。システムがそのシステムと同様にエンドポイントを公開するネストされたシステムを含んでいる場合、それらエンドポイントは外部システム上のプロキシエンドポイントを介して公開されるが、当該外部システム上のプロキシエンドポイントは内部システムのエンドポイントに対して委任されている。
【０２４１】
通信関係が一組のエンドポイント間に存在しない場合、それらエンドポイント間で接続を確立することはできない。
【０２４２】
システムがオフラインとマーク付けされている場合、そのシステムが使用不可能であることを反映するようそのシステムのすべてのクライアントが更新される。
【０２４３】
２．２．４　レファレンス
レファレンス関係は、正確な動作を達成するために要求されるが、従属オブジェクトの実行環境の一部とは見なされない資源またはシステム間の従属関係を取り込むために使用される。
【０２４４】
例えば、ウェブディレクトリは、ウェブコンテンツをウェブサービスの消費者に公開するためにローカルディレクトリまたは遠隔共有資源へのレファレンスを必要とする。これは、ウェブディレクトリとローカルディレクトリまたは遠隔共有資源との間の従属関係として表現される。
【０２４５】
２．２．５　委任

委任関係は、プロキシとの相互作用をプロキシによって公開された動作を実施するオブジェクトインスタンスに転送するために使用される。一般に使用される委任の例は、親システムのエンドポイントから含まれているシステムのエンドポイントに通信関係を転送している。

【０２４６】

２．３　定義

オブジェクトおよび関係定義は、その構成のインスタンスを作成するためにインスタンス生成することができる再利用可能な構成を作成するために使用される。この場合、これらインスタンスは定義によって特定された共通特性を共有する。

【０２４７】

不完全な抽象定義を直接インスタンス生成することはできない。抽象定義を完全にするためには、欠けている要素を追加するよう別の定義がこの抽象定義を拡張する。

【０２４８】

実際のシステムのモデルを構築する際に本発明者らが使用することのできる構築ブロックを作成するために、本発明者らは抽象定義を使用する。抽象定義間の関係は、抽象定義を拡張する定義の許容可能な構造を定義する。これによって、表面のユーザが実行することのできるアクションを定義するために設計表面が抽象定義を使用することが可能となる。例えば、抽象ウェブアプリケーションが参加する包含関係は、抽象定義を拡張するウェブアプリケーションの構造をガイドする。

【０２４９】

基本的に、抽象オブジェクト定義と関係定義の組合せは、ターゲットシステムをモデル化するためのスキーマを定義する。具体的なオブジェクト定義のロールは、１つまたは複数の抽象定義に基づいて再利用可能な構成を作成するために抽象定義空間の部分集合を使用することである。単なる類似として、抽象定義空間をデータベースに対するスキーマと比較することができ、その場合、具体的なオブジェクト定義はデータベースの一組の行に対して再利用可能なテンプレートを表す。具体的なオブジェクトのインスタンスが作成された場合、行はデータベースでしか作成されない。設計時検証を実行するために、本発明者らは、データベースの行をスキーマの包含（例えば外部キーなど）に対して検証するのと同様の方法で、具体的なオブジェクト定義を抽象定義空間に対して検証することができる。

【０２５０】

単純な値の要素を作成するために設定定義が使用される。次いでこれら値の要素は構成情報を記憶するために使用することができる。

【０２５１】

２．４　メンバ

定義は、他の定義を参照するメンバを含むことができる。メンバは、ある定義がその特定のアプリケーションに対してカスタマイズされた方法で別の定義を再利用することを可能にする。

【０２５２】

定義に関連付けられた構成情報を特定するために設定メンバが使用される。設定メンバは設定定義に基づいている。

【０２５３】

特定のオブジェクト定義のインスタンスを作成するためにオブジェクトメンバが使用される。オブジェクトに値を提供するために設定フローを使用することができる。オブジェクトメンバを宣言する際、ユーザは、外部システムが作成されるのと同時にオブジェクトメンバが作成されるか否か（値の意味論）、または後で行われる明白な新しい動作によって作成されるか否か（レファレンス意味論）を決めることができる。

【０２５４】

関係メンバは、オブジェクトメンバが作成された際にそれらオブジェクトメンバが参加する関係を定義する。オブジェクトメンバがその親に「含まれている」場合、包含関係メ

ンバはメンバと含んでいる定義との間で宣言されることになる。オブジェクトメンバが「委任されている」場合、委任関係メンバはオブジェクトメンバとソースオブジェクトメンバとの間で定義される。通信しているエンドポイント間で通信関係メンバを宣言することができる。従属関係メンバ（レファレンスおよびホスティング）を従属およびソースオブジェクトメンバの間で宣言することができる。

【０２５５】

制約メンバは、特定のオブジェクトが参加を意図する当該一組の関係または特定の関係に参加することのできる当該一組のオブジェクトを狭めるために使用される。オブジェクトまたは関係の設定を目標とすることができるか、またはオブジェクトまたは関係に関連付けられた相互作用を制約することができるオブジェクトまたは関係に対する制約条件をこれらは特定する。

【０２５６】

メンバ間で構成のフローを定義するためにフローメンバが使用される。これらは、メンバに対する設定からの入力値を収集し、その情報に対して何らかの処理を実行し、次いでその結果をメンバに対する設定に分配する。

【０２５７】

２．５　インスタンス

インスタンス空間は、モデル化されたシステムの現在の状態を反映する。作成されたインスタンスとこれらインスタンス間の関係との完全な記録を維持することもできる。各バージョンが変更要求にリンクされている場合、各インスタンスは関連付けられたバージョンヒストリを有する。

【０２５８】

新しいインスタンスを作成する処理は「変更要求」によって開始される。変更要求は、既存のインスタンスの特定メンバに関連付けられた型および関係に対する一組の作成、更新、および削除要求を定義し、ルートは特別な場合である。

【０２５９】

変更要求はランタイムによって拡張され、すべての制約条件に対して検証され、次いで構築される。拡張プロセスは、含んでいるオブジェクトの構築要求の一部として暗黙的に構築されるオブジェクトおよび関係インスタンスを特定し、次いですべての関係を通して設定フローが評価される。検証ステップは、すべての要求された関係が存在すること、その関係がすべての制約条件を満たすことをチェックする。最後に、構築プロセスは、適切なアクションを実行するために、各インスタンスの配置、更新、または除去の適切な順番付けを決定し、次いで正確な順番で各インスタンスをインスタンスマネージャに渡す。

【０２６０】

２．６　オブジェクトモデル

以下に示すＵＭＬ図は、ＳＤＭモデル内のオブジェクト間の幅広い相互作用を取り込んでいる。分かりやすくするために、導出された型間に実際の相互作用が存在し、したがってこれらがさらに特化される場合、これら相互作用の一部は基準型間に定義されている。

【０２６１】

ＳＤＭ文書は、その文書を説明する情報、文書内の定義に対するマネージャ、他の文書を参照するインポートステートメント、および文書によって記述されたオブジェクトおよび関係に関する一組の定義を含んでいる。多くのＳＤＭ要素は、設計表面内で当該要素の表示および操作に特有の情報によって設計表面をＳＤＭ要素の属性とすることを可能とする設計データ含むこともできる。

【０２６２】

図８はＳＤＭ文書の一例を示している。

【０２６３】

すべてのＳＤＭ定義は、図９に示すように共通の基礎定義に由来している。定義は、説明、一組のメンバ、一組の設定値、および設計データを含むことができる。本発明者らは、定義をオブジェクト、関係、制約条件、フロー、および設定定義に特化する。次いでオ

ブジェクト定義は、システム、資源、およびエンドポイント定義にさらに特化される。関係定義は、ホスティング、通信、委任、レファレンス、および包含定義にさらに特化される。

【０２６４】

名前を図１０に示すように特定のインスタンスまたはインスタンスのアレイに関連付けるためにメンバが使用される。すべてのメンバは定義を参照する。各メンバは、一組の設定値、説明、および設計データを含むことができる。次いでメンバは、それらが参照する定義の種類に基づいて特化される。フローおよび制約条件メンバは、含んでいる定義内の設定メンバから設定値を入手する入力を定義する能力を追加する。フローメンバは、処理されたフロー値が含んでいる定義内の設定メンバに転送されることを可能にする出力も追加する。

【０２６５】

設定メンバの一例を図１１に示す。設定メンバは、複雑な設定定義または簡素な設定定義を表現することができる設定定義を参照する。複雑な設定定義は、ネストされた設定メンバを含むことができる。簡素な設定定義は、単一値フィールドを定義する。設定メンバ間での設定値の転送を定義するために入力および出力オブジェクトが使用される。

【０２６６】

制約条件定義の一例を図１２に示す。設定値とＳＤＭモデルの構造とに対して制約を課すために制約条件が使用される。制約条件定義は特定のオブジェクトまたは関係定義を目標とする。制約条件定義は、オブジェクトが参加する関係または関係に参加しているオブジェクトに構造上の制約条件を課すことができる。これらの構造定義をオブジェクトまたは関係定義に直接追加することもできる。

【０２６７】

図１３に示すようにＳＤＭ要素の人間によって可読の説明を提供するために説明オブジェクトが使用される。これら要素は場所特定が可能であるべきだが、これはマネージャの責任である。設計データ要素は、ＳＤＭ文書を編集する設計表面によって定義される構造化されたデータを含んでいる。

【０２６８】

２．７　階層化

ＳＤＭモデルの目標は、アプリケーションの開発者とソフトウェアインフラストラクチャの設計者とデータセンタの立案者との間の関係の分離を可能にすることである。これらグループのそれぞれは、特定のサービスに焦点を合わせており、異なる一組の従属関係を有している。

【０２６９】

例えば、設計者は、主としてＳＱＬ、ＩＩＳ、およびＣＬＲのようなそれらが従属するホスト間の構成および接続性に配慮する。ホスト構成の設計者はネットワークトポロジとＯＳ構成に配慮し、一方、ネットワークトポロジ、ＯＳ構成、および記憶のマッピングを開発中の立案者はデータセンタに存在するハードウェアに関して知る必要がある。

【０２７０】

関係のこのような分離をサポートするために、ＳＤＭは階層化の概念を公開する。階層化は、ホスティング関係を使用することによって可能となる。オペレーティングシステムとｓｑｌのようなホストとｓｑｌが提供するサービス、例えばデータベースとの間の関係のホスティングを定義することによって、本発明者らはアプリケーション開発者がデータベースを含んでいるアプリケーションを定義し、立案者がそのｓｑｌホストを含んでいるシステムを定義することを可能にする。ゲストシステムとホストシステムとの間のホスティング関係を接続することによって本発明者らはこれらの分離されたシステムを層結合する。

【０２７１】

ホスティング関係が存在する時は何時でも階層境界が発生し得る。階層境界をどこに置くかの選択を簡素化するために、本発明者らはＳＤＭモデルの一部として４つの基礎階層

を定義する。次いで本発明者らはシステムをそれらの階層の１つに関連付ける。４つの階層を以下に示す。

【０２７２】

アプリケーション層

・アプリケーション層は、制約されたコンテキストでアプリケーションの構築をサポートする。コンテキストはホスト層で特定されたホストの構成によって定義される。

・アプリケーション層のシステム定義の例としては、ウェブサービス、データベース、およびｂｉｚｔａｌｋスケジュールが含まれる。

【０２７３】

ホスト層

・ソフトウェア構成要素からデータセンタを構築する。構成要素間の接続を構成する。これら構成要素の一部はアプリケーション層に対してホストとして働く。

・この層のシステム定義の例は、ＩＩＳ、ＳＱＬ、ＡＤ、ＥＸＣＨＡＮＧＥ、ＤＮＳ、およびＢｉｚｔａｌｋである。

【０２７４】

ネットワーク／ＯＳ／記憶層

・データセンタネットワークおよびプラットフォームを構築する。ネットワークセキュリティモデルおよびオペレーティングシステムプラットフォーム構成を構成する。ストレージをオペレーティングシステム構成に追加する。

・この層のシステム定義の例は、ＶＬＡＮ、Ｗｉｎｄｏｗｓ（登録商標）、Ｆｉｌｔｅｒ、Ｓｔｏｒａｇｅである。

【０２７５】

ハードウェア層

ハードウェア層は、データセンタに存在するマシンの種類とそれらマシン間に存在する物理接続とを特定する。

【０２７６】

図１４は、層４のウェブアプリケーションを層３のウェブサーバホストにマッピングする一例を示す。各層の外部ボックスはシステムを表し、境界上のボックスはエンドポイントを表し、内部のボックスは資源を表す。本発明者らは、それら要素のそれぞれをホスティング関係によって下の層のホストにマッピングする。

【０２７７】

３　実施態様の詳細

３．１　名前付け

ＳＤＭには、本発明者らがオブジェクトを特定するために強力な名前付けシステムを必要とするいくつかの場所がある。以下に示す名前付けシステムは、文書の定義のユーザが、開発者が元々発行した文書の定義とそれらが同じであることを確証できる方法で、ＳＤＭ文書の作成者が文書に署名することを可能にする。

【０２７８】

次に示すヘッダは、ＳＤＭ文書に対する識別子の一例である。

【０２７９】

【表３３】

```
<SystemDefinitionModel Name="System.OperatingSystem.FileSystem"
    Version="0.1.0.0"
    DocumentLanguage="en"
    xmlns="http://schemas.microsoft.com/SystemDefinitionModel/2003/10"
      Version="0.1.0.0"
      PublicKeyToken="AAAABBBBCCCCDDDD"
      Culture="neutral"
      Platform="neutral"
    >
 </ SystemDefinitionModel >
```

【０２８０】

別のネーム空間の型を参照するために、そのネーム空間をインポートする。

【０２８１】

【表３４】

```
<Import Alias="FileSystem" Name="System.OperatingSystem.FileSystem" Version="0.1.0.0"/>
```

【０２８２】

次に、別名を使用して、そのネーム空間内の定義を参照することができる。

FileSystem:File

【０２８３】

３．１．１　識別

ＳＤＭ名は、それらが定義されているネーム空間によってスコープとされる。ネーム空間は名前、バージョン、言語、および公開鍵トークンによって特定され、単一ファイル内に含まれる。

【０２８４】

識別の基礎形式は名前、バージョン、カルチャ、プラットフォーム、および公開鍵トークンを含む。

【０２８５】

【表３５】

```
<xs:attributeGroup name="Identity">
    <xs:attribute name="Name" type="Path" use="required"/>
    <xs:attribute name="Version" type="FourPartVersionType" use="required"/>
    <xs:attribute name="PublicKeyToken" type="PublicKeyTokenType" use="optional"/>
    <xs:attribute name="Culture" type="CultureNeutral" use="optional"/>
    <xs:attribute name="Platform" type="xs:string" use="optional"/>
</xs:attributeGroup>
```

【０２８６】

【表３６】

属性/要素	説明	ディフォルト
Name	SDMファイルの名前は、開発者がそのファイルの内容を参照するために使用することのできる使いやすい名前である。NameはPublicKeyTokenと組み合わせて、ファイルに対する強力な名前を提供する。	NA
Version	ファイルの内容のバージョンを特定するためにVersionが使用される。ファイルのすべての要素は同じバージョン番号を採用する。	NA
PublicKeyToken	公開鍵トークンは、ファイルに関連付けられた公開鍵に対する短い名前である。	Nil
Culture	バイナリのカルチャ	ニュートラル
Platform	バイナリに対するサポートされたプラットフォーム	x86

【０２８７】

ネーム空間識別と呼ばれる新しい強力な識別を作成するために、基礎識別を署名および公開鍵と共に使用することができる。ＳＤＭ文書を特定するためにこの識別が使用される。識別を作成するために文書は秘密鍵を使用して署名されるが、これによって文書のユーザは公開鍵を使用してその内容を検証することができる。

【０２８８】

公開鍵トークンは、公開／秘密鍵対の公開部分を特定する１６文字の１６進法ストリングである。これは公開鍵ではなく、単なる公開鍵の６４ビットハッシュである。

【０２８９】

【表３７】

```
<xs:simpleType name="PublicKeyTokenType">
    <xs:restriction base="xs:string">
        <xs:pattern value="([0-9]|[a-f]|[A-F]){16}"/>
    </xs:restriction>
</xs:simpleType>
```

【０２９０】

文書の言語は、言語を定義する２つの小文字と国または領域を定義する２つまたは３つの大文字とからなるカルチャ識別子を使用して指定される。

【０２９１】

【表３８】

```
<xs:simpleType name="Culture">
    <xs:restriction base="xs:string">
        <xs:pattern value="[a-z]{2}((-[A-Z]{2})?|(-[A-Z]{3})?)"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="CultureNeutral">
    <xs:union memberTypes="Culture">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="neutral"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:union>
</xs:simpleType>
```

【０２９２】

３．１．２　バージョン

ファイルバージョンは、０＜＝Ｎ＜６５５３５として、形式Ｎ．Ｎ．Ｎ．Ｎ．の４つの部分番号によって定義される。慣例により、この番号はＭａｊｏｒ．Ｍｉｎｏｒ．Ｂｕｉｌｄ．Ｒｅｖｉｓｉｏｎを意味する。バージョン番号は、文書に含まれるすべての定義のバージョンを特定する。これは、文書内のすべての定義が共に更新される際に文書がバージョニングの単位となることを意味している。

【０２９３】

【表３９】

```
<xs:simpleType name="FourPartVersionType">
    <xs:restriction base="xs:string">
        <xs:pattern value="([0-9]{1,4}|[0-5][0-9]{4}|64[0-9]{3}|655[0-2][0-9]|6553[0-5])(\.([0-9]{1,4}|[0-5][0-
9]{4}|64[0-9]{3}|655[0-2][0-9]|6553[0-5])){3}"/>
    </xs:restriction>
</xs:simpleType>
```

【０２９４】

３．１．３　単純名

単純名は英数字から構成され、句読法が限定されている。この名前は非数字文字から始まる。

【０２９５】

【表４０】

```
<xs:simpleType name="SimpleName">
    <xs:restriction base="xs:string">
        <xs:pattern value="[a-z,A-Z,_]{1}([0-9,a-z,A-Z,_])*"/>
    </xs:restriction>
</xs:simpleType>
```

【０２９６】

本発明者らは、識別子に関してはＣ＃定義に準拠することを予定しており、その適切なセクションが以下に示す表に挿入されている。本発明者らはＳＤＭモデルでは「@」という接頭辞を付けた名前をサポートしないことに留意されたい。

【０２９７】

【表４１－１】

本節に記載する識別子のための規則は、識別子としてキーワードを使用することを可能にするように、（Cプログラミング言語で従来そうであったように）最初の文字として下線が許可されており、ユニコードエスケープシーケンスが識別子で許可されており、「@」という文字が接頭辞として許可されていることを除いて、ユニコード標準付属文書15（Unicode Standard Annex 15）が推奨する規則に厳密に対応している。

識別子：
　使用可能な識別子
　@　識別子またはキーワード
使用可能な識別子：
　キーワードではない識別子またはキーワード
識別子またはキーワード：
　識別子開始文字　識別子部分文字$_{opt}$
識別子開始文字：
　英字文字
　_（下線文字U+005F）
識別子部分文字：
　識別子部分文字
　識別子部分文字　識別子部分文字

【０２９８】

【表４２－２】

識別子部分文字：
英字文字
10進数字文字
接続文字
結合文字
フォーマット文字
英字文字：
クラスLu、Ll、Lt、Lm、Lo、またはNlのユニコード文字
クラスLu、Ll、Lt、Lm、Lo、またはNlの文字を表すユニコードエスケープシーケンス
結合文字：
クラスMnまたはMcのユニコード文字
クラスMnまたはMcの文字を表すユニコードエスケープシーケンス
10進数字文字：
クラスNdのユニコード文字
クラスNdの文字を表すユニコードエスケープシーケンス
接続文字：
クラスPcの文字のユニコード文字
クラスPcの文字を表すユニコードエスケープシーケンス
フォーマット文字：
クラスCfのユニコード文字
クラスCfの文字を表すユニコードエスケープシーケンス
上記ユニコード文字のクラスに関する情報については、ユニコード標準、バージョン3.0、セクション4.5（The Unicode Standard、Version 3.0、section 4.5.）を参照されたい。

有効な識別子の例としては、「identifier1」、「_identifier2」、および「@if」が含まれる。適合するプログラムの識別子は、ユニコード標準付属文書15によって定義されるようにユニコード規格化形式Cによって定義された標準フォーマットであるべきである。規格化形式Cでない識別子に遭遇した場合の動作は、実施態様が定義されているが、診断は要求されない。

接頭辞「@」は、他のプログラミング言語とインターフェースする場合に有益な識別子としてキーワードを使用することを可能にする。文字@は、実際には識別子の一部ではなく、したがって識別子は他の言語ではその接頭辞なしで通常の識別子と見なすことができる。接頭辞@付きの識別子は「逐語識別子」と呼ばれる。キーワードでない識別子に対して接頭辞@を使用することは許可されているが、スタイル上は止めることが強く推奨される。

【０２９９】

【表４３－３】

例：

```
class @class
{
  public static void @static(bool @bool) {
        if (@bool)
              System.Console.WriteLine("true");
        else
              System.Console.WriteLine("false");
  }
}
  class Class1
{
  static void M() {
        cl＼u0061ss.st＼u0061tic(true);
  }
}
```

これは、「bool」と命名されたパラメータを取る「static」と命名された静的な方法で「class」と命名されたクラスを定義する。キーワードではユニコードエスケープが許可されていないので、トークン「cl＼u0061ss」は識別子であり、「@class」と同じ識別子であることに留意されたい。

２つの識別子に次に示す順で変形が加えられた後でそれらが同じである場合、その２つの識別子は同一と見なされる。

■使用されている場合は接頭辞「@」が除去される。

■各ユニコードエスケープシーケンスが対応するユニコード文字に変形される。

■フォーマット文字が除去される。２つの連続する下線文字を含んでいる識別子（U+00SF）が実施態様によって使用されるよう確保される。例えば、一実施態様は、2つの下線から始まる拡張されたキーワードを提供する。

【０３００】

３．１．４　予約名

ＳＤＭ定義のメンバに対する名前を作成する際にユーザが使用することを本発明者らが防止する予約名のリストを以下に示す。あるコンテキスト内である名前が予約される。コンパイラは、それらの名前のユーザを検出し、エラーをそのユーザに戻す。

【０３０１】

【表４４】

コンテキスト	名前
すべての定義	this
ホスティング関係定義	Guest, Host
包含関係定義	Parent, Member
通信関係定義	Client, Server
レファレンス関係定義	Source, Dependent
委任関係定義	Proxy, Delegate

【０３０２】

３．１．５　他のネーム空間へのレファレンス

本発明者らは、他のネーム空間を現在のネーム空間にインポートし、次いでそのネーム空間に別名を関連付けることによって、ネーム空間が他のネーム空間を参照することを可能にする。インポートされたネーム空間は名前、バージョン、および公開鍵トークンによって参照される。

【０３０３】

【表４５】

```
<xs:complexType name="Import">
    <xs:attribute name="Alias" type="SimpleName" use="required"/>
    <xs:attributeGroup ref="Identity"/>
</xs:complexType>
```

【０３０４】

【表４６】

属性/要素	説明
Alias	現在のSDMファイルの範囲内の外部SDMファイルを参照することに使用される別名。
Identity	節3.1.1で定義された当該一群の属性

【０３０５】

３．１．６　修飾名

修飾名は、現在のネーム空間または別名の付いたネーム空間で定義された定義またはマネージャを意味する名前である。名前が別名または「ドット分離記号」を含まない場合、その名前は修飾されておらず、スコープ規則に基づいて解決されるべきである。修飾されていない経路は、それらが評価されるコンテキストに基づいて異なる定義に解決することができる。これは、ネストされた定義がさらに広いスコープに定義を隠すことができるからである。節３．１．７を参照されたい。

【０３０６】

「ドット分離記号」を含むいかなる経路でも完全に修飾されるべきであり、文書のルートから解決されることになる。これは、ユーザが、複数の型の名前を含んでいるが宣言コンテキストからしか解決できない部分的に修飾された名前を作成することができないということである。

【０３０７】

【表４７】

```
[<alias> :] <simpleName> (. <simpleName>)*
```

【０３０８】

別名はインポートステートメントで定義される。

【０３０９】

【表４８】

```
<xs:simpleType name="QualifiedName">
    <xs:restriction base="xs:string">
        <xs:pattern value="[a-z,A-Z,_]{1}([0-9,a-z,A-Z,_])*(:[a-z,A-Z,_]{1}([0-9,a-z,A-Z,_])*)?(\.[a-z,A-Z,_]{1}([0-
9,a-z,A-Z,_])*)*"/>
    </xs:restriction>
</xs:simpleType>
```

【０３１０】

３．１．７　名前スコープ

定義の名前は、それらが定義されているコンテキストによってスコープとされる。最も

広いスコープは当該文書である。定義がネストされている場合、その名前はその定義によってスコープとされる。これは、それが名前を親スコープに隠すことができるということを意味している。これはまた、それが定義されているスコープ外からの定義に対するレファレンスがその親スコープの名前によって修飾されるべきであるということをも意味している。完全に修飾された名前は、文書のルートからネストされた定義までのすべての名前を含む。

【０３１１】

３．１．８　メンバ経路

メンバ経路は、横断される定義のメンバに対応する一連の名前である。経路は、経路が宣言されるコンテキストで定義されるメンバ名から始められるべきである。

【０３１２】

いくつかのメンバ名が定義に自動的に追加される。本発明者らはこれらを周知の名前と呼ぶ。周知の名前の例としては、定義またはレファレンス宣言の「ｔｈｉｓ」、ホスティング関係宣言の「Ｈｏｓｔ」および「Ｇｕｅｓｔ」、または制約条件宣言内で定義された「Ｔａｒｇｅｔ」が含まれる。設定ターゲットは、経路によって特定される設定のソース値または宛先設定として使用されることになる、関連付けられたフロー定義に対する設定も特定する。

【０３１３】

メンバ名はドットで分離される。

【０３１４】

【表４９】

<simpleName>(.<simpleName>)*

```
<xs:simpleType name="Path">
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z,A-Z]{1}([0-9,a-z,A-Z,_])*(\.[a-z,A-Z]{1}([0-9,a-z,A-Z,_])*)*"/>
    </xs:restriction>
</xs:simpleType>
```

【０３１５】

３．１．９　インスタンス経路

インスタンス空間内の経路は、ｘｐａｔｈの要素名がメンバ名に対応しており、ｘｐａｔｈの属性が設定に対応しているｘｐａｔｈに基づいている。

【０３１６】

３．１．１０　署名

文書の内容を検証し、その文書に強力な名前を関連付けるための機構を提供するためにコンパイルされたＳＭＤ文書はキーを使用して署名される。署名機構は、文書に署名するためにＤＳＩＧ標準（http://www.w3.org/2000/09/xmldsig#）を使用する。

【０３１７】

次のような署名ノードが署名された文書に埋め込まれる。

【０３１８】

【表５０】

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <Reference URI="">
            <Transforms>
                <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
                <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue>Dn9BN6rKnnzzgGxaWl3j0aW8/1g=</DigestValue>
        </Reference>
    </SignedInfo>

    <SignatureValue>JaNLAktv2+n4Xz3p0jM8zqwmUc8T91AOtwasKOVBwHgvYAGijRFl0U1kcVfHbDJoi8ln33prn
S5lTNj46rGYRn/UK11vjvpprfgz2a+Ql7G4o7hBwO0SZzLLPO7/y7+GKw0Cr6zaoDmOFSdP8lAATuVJC40pZhbDm
cmHJduaXm0=</SignatureValue>
        <KeyInfo>
```

【０３１９】

【表５１】

```
            <KeyValue xmlns="http://www.w3.org/2000/09/xmldsig#">
                <RSAKeyValue>

    <Modulus>wxxlClOui6J77CZcnzLOcBBxl/3uuFrUBEyFivVklRtTTzQpgirdaNouQ9ziCAK6aipeM6RjgYAoDPs/i
WFnrHmtDoxgP2H4/W9lttXO5EjQDb76XAZRPSvthJYtreKEGE0d6L6lTRdg25hYGFkyY/yS0s0ONh3+5VO0Wxq+
Fx0=</Modulus>
                    <Exponent>AQAB</Exponent>
                </RSAKeyValue>
            </KeyValue>
        </KeyInfo>
</Signature>
```

【０３２０】

このノードは、文書の他の内容よりも前に発生すべき文書内のルートノードである。本発明者らは、この要素を「ａｎｙ」というタグを使用して宣言するが、本発明者らは、この位置では「Ｓｉｇｎａｔｕｒｅ」ノードしか認識し、処理しない。

【０３２１】

【表５２】

```
<xs:any namespace="http://www.w3.org/2000/09/xmldsig#" processContents="skip" minOccurs="0"/>
```

【０３２２】

３．１．１１　ネーム空間の名前の定義、参照、および解決

本節では、本発明者らは、本発明者らがネーム空間レファレンスを特定のネーム空間と照合するために使用することのできるプロセスを説明する。節３．３．１で定義したように、ネーム空間の識別は次に示す５つの属性から構成される。

・単純名
・公開鍵トークン
・バージョン
・言語
・プラットフォーム

これらの属性は、要求されたネーム空間（ネーム空間レファレンス）を特定するためにネーム空間の消費者によって、またネーム空間に周知の識別（ネーム空間定義）を提供するためにネーム空間の開発者によって使用される。それら属性のそれぞれの解釈は、それらがネーム空間レファレンスの一部であるかネーム空間定義の一部であるかによって異なる場合がある。

【０３２３】
３．１．１１．１　ネーム空間定義
開発者が新しいネーム空間を作成し、発行することを希望する場合、その開発者は、
・ネーム空間に単純名を提供し、
・ネーム空間にバージョンを提供する
開発者は任意選択で、
・暗合化キーでネーム空間に署名し、
・ネーム空間に対して特有カルチャ（ディフォルトではニュートラルなカルチャ）を特定し、
・ネーム空間に対して特有プラットフォーム（ディフォルトではニュートラルなプラットフォーム）を特定する
ネーム空間が署名されていない場合、公開鍵トークンはなく、ネーム空間は弱い名前を有することになる。ネーム空間に遅延署名することができる。これによって、開発者はコンパイルプロセスから署名プロセスを分離することができる。コンパイル済みではあるが署名されていない遅延署名されたネーム空間は、コンパイル中に参照することはできるが、ランタイムで使用することはできない。
【０３２４】
３．１．１１．２　ネーム空間レファレンス
消費者がＳＤＭファイルのネーム空間を参照（インポート）する場合、消費者は、
・参照されるネーム空間の名前を提供する。
消費者は、任意選択で次のような追加情報を提供する。
・参照されるネーム空間の公開鍵トークン
・参照されるネーム空間の最低バージョン
・参照されるネーム空間のカルチャまたはワイルドカード（＊）
・参照されるネーム空間のプラットフォームまたはワイルドカード（＊）
提供されていないすべての属性は、ディフォルトではその属性に対するいかなる値とも一致することになるワイルドカードである。
【０３２５】
３．１．１１．３　定義に対するレファレンスの照合
ネーム空間レファレンスをネーム空間定義と照合するために以下に示すアルゴリズムが使用される。このアルゴリズムは、ネーム空間をコンパイルする際と、コンパイルされたネーム空間の間でレファレンスを解決する際のどちらでも使用される。
【０３２６】
コンパイルとコンパイル後のシナリオの間には１つの僅かな違いしかない。それは、コンパイル時にネーム空間レファレンスのバージョンフィールドが無視されるが、ランタイムにそれは存在するということである。コンパイル中にレファレンスが解決されたネーム空間から追加情報を追加するために、コンパイルプロセスはネーム空間レファレンスを更新する。これはネーム空間からのバージョンを常に追加する。公開鍵トークンが提供されておらず、解決されたネーム空間にそれが存在する場合、それはネーム空間レファレンスに追加される。
【０３２７】
論理上、解決アルゴリズムは以下のように機能する。
１．レファレンス内の名前と一致する名前を有するすべてのネーム空間を見つける。
２．レファレンスが公開鍵トークンを提供する場合、提供された公開鍵トークンとも一致するネーム空間だけを選択する。
３．特有の言語がレファレンスによって選択されている場合、その言語と一致しないすべてのネーム空間を除去する。その言語属性がワイルドカードに設定されているかまたは提供されていない場合、すべてのネーム空間を維持する。
４．特有のプラットフォームが提供されている場合、そのプラットフォームと一致しないすべてのネーム空間を除去する。プラットフォームの属性がワイルドカードに設定され

ているかまたは提供されていない場合、すべてのネーム空間を維持する。
５．次いで、当該一組の実現可能なネーム空間から特定のネーム空間を選択するためにバージョニングポリシーを使用する。
次節以降では、これらのステップをさらに詳細に定義する。
【０３２８】
３．１．１１．３．１　ネーム空間の名前の照合
名前の一致は正確なストリングの一致である。名前は大文字小文字を区別する。
【０３２９】
３．１．１１．３．２　公開鍵トークンの照合
トークンは正確な一致である。
【０３３０】
３．１．１１．３．３　言語の照合
言語の照合には次に示す３つのシナリオがある。
【０３３１】
１．レファレンス＝＊、定義＝特定言語
２．レファレンス＝特定言語、定義＝＊
３．レファレンス＝特定言語、定義＝特定言語
第１と第２の場合に関して、定義はレファレンスに対する一致である。第３の場合、言語が正確な一致ならば、定義はレファレンスに対する一致にすぎない。
【０３３２】
３．１．１１．３．４　プラットフォームの照合
プラットフォームの照合には次に示す３つのシナリオがある。
１．レファレンス＝＊、定義＝特定プラットフォーム
２．レファレンス＝特定プラットフォーム、定義＝＊
３．レファレンス＝特定プラットフォーム、定義＝特定プラットフォーム
第１と第２の場合に関して、定義はレファレンスに対する一致である。第３の場合、プラットフォームが正確な一致ならば、定義はレファレンスに対する一致にすぎない。
【０３３３】
３．１．１１．３．５　バージョンの照合
バージョンの一致は、コンパイラまたはランタイムが使用中のバージョニングポリシーに依存する。
【０３３４】
３．２設定
すべての定義は、ＸＭＬスキーマ内で設定宣言と呼ばれる設定メンバを公開することができる。定義に関連付けられた構成値を記述するためにこれらのメンバが使用される。ＳｅｔｔｉｎｇＶａｌｕｅステートメントを使用して値が提供される。これらは次のものによって使用することができる。
ａ）設定に対するディフォルト値を定義するために設定メンバを宣言する定義。
ｂ）基礎定義に対するディフォルト値または固定値を提供するために別の定義を拡張する定義によって、
ｃ）定義を参照するメンバによって、定義を参照するメンバで中止するメンバ経路を介して、
ｄ）関係に沿ったフローを介して、
設定を定義するには、まずｘｓｄを使用して設定の定義を定義する必要がある。
【０３３５】

【表５３】

```
<SettingDefinitions>
    <xs:schema targetNamespace="System.OperatingSystem.Registry"
               xmlns="System.OperatingSystem.Registry"
               xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:simpleType name="PathType">
            <xs:restriction base="xs:string"/>
        </xs:simpleType>
        <xs:simpleType name="RegistryValueType">
            <xs:restriction base="xs:NMTOKEN">
                <xs:enumeration value="binary"/>
                <xs:enumeration value="integer"/>
                <xs:enumeration value="long"/>
                <xs:enumeration value="expandString"/>
                <xs:enumeration value="multiString"/>
                <xs:enumeration value="string"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:schema>
</SettingDefinitions>
```

【０３３６】

次いで、定義を使用し、設定の動作を定義するために一組の属性を含む設定を宣言することができる。

【０３３７】

【表５４】

```
<SettingDeclaration Name="Definition" Definition="RegistryValueType" Access="Readwrite"
Required="true"/>
```

【０３３８】

一度設定宣言を有すると、その設定用の値を提供することができる。

【０３３９】

【表５５】

```
<SettingValue Name=" Definition" Fixed="true">long</ SettingValue >
```

【０３４０】

３．２．１　設定定義

本発明者らは、設定メンバによって使用される設定定義を定義するためにＸＳＤスキーマを使用する。本発明者らはスキーマからの単純型と複合型の使用しかサポートしない。ただし、これらの型の定義をサポートするために他のスキーマ要素が存在する場合もある。

【０３４１】

設定定義の節は、ネーム空間宣言とネーム空間インポートを含めて完全なｘｍｌスキーマを含むべきである。本発明者らは、ｘｓｄスキーマのネーム空間を除いてｘｓｄスキーマへのインポートがＳＤＭファイルへのインポートと一致することをチェックする。これは、すべての参照された型が別のＳＤＭファイルで定義されるべきであり、スキーマは任意のｘｓｄファイルで定義された型を参照することができないということを意味している。

【０３４２】

スキーマのターゲットネーム空間はＳＤＭ文書の名前と一致する必要がある。ネーム空間の名前が一意であることを保証するために本発明者らが文書に署名する際に本発明者らはネーム空間の名前に文書に対する公開鍵を追加する。したがってコンパイラは、ターゲットのＮａｍｅｓｐａｃｅ＝“Ｓｙｓｔｅｍ”をターゲットのＮａｍｅｓｐａｃｅ＝“Ｓ

ｙｓｔｅｍ／ＡＡＡＡＢＢＢＢＣＣＣＣＤＤＤＤ”に変更する。

【０３４３】

設定定義スキーマが別の文書からのネーム空間を参照する場合、それらは完全なネーム空間の名前ｘｍｌｎｓ：ｓｙｓｔｅｍ＝“Ｓｙｓｔｅｍ／ＡＡＡＡＢＢＢＢＣＣＣＣＤＤＤＤ”を使用する。

【０３４４】

【表５６】

```
<xs:complexType name="SettingDefinitions">
    <xs:sequence>
        <xs:any namespace="http://www.w3.org/2001/XMLSchema"/>
    </xs:sequence>
    <xs:attribute name="Manager" type="QualifiedName" use="optional"/>
    <xs:attribute name="ClrNamespace" type="xs:string" use="optional"/>
</xs:complexType>
```

【０３４５】

【表５７】

属性/要素	説明
xs:any	http://www.w3.org/2001/XMLSchmaネーム空間のスキーマ
Manager	このスキーマ用のヘルパクラスを含んでいるclrアセンブリ
ClrNamespace	これらのクラスが定義されるclrネーム空間。すべての設定型は、CLRシリアル化によるそれらのマッピングに基づいてclrに解決する。

【０３４６】

設定は次に示す３つの別個のネーム空間から解決可能である。

ａ）ＳＤＭネーム空間－本発明者らがシステム内の設定型、資源、エンドポイント、関係、制約条件、またはフロー型に言及する場合。

ｂ）ｃｌｒネーム空間－本発明者らがｃｌｒ内の強力な型のクラスを使用した設定に言及し、かつ設定型が他の設定型上に構築されている場合。

ｃ）ＸＳＤネーム空間－設定型が他の設定型を使用して構築される場合。

【０３４７】

これが機能するために、本発明者らが設定を宣言する方法に本発明者らはいくつかの制約を課す。

ａ）すべての設定は、ｃｌｒ、ＳＤＭ、およびｘｓｄネーム空間のそれぞれの中の同じグループ化でなければならない。すなわち、２つの設定が１つのネーム空間になる場合、これらをすべての３つのネーム空間に統合しなければならない。

ｂ）ｘｓｄスキーマ定義内のインポートされたネーム空間は、ＳＤＭファイル内のインポートされたネーム空間および関連付けられたヘルパアセンブリ内のインポートされたネーム空間と一致しなければならない。

ｃ）ｘｓｄネーム空間を例外として、ｘｓｄスキーマ内のすべてのインポートされたネーム空間をＳＤＭファイル内で定義しなければならない。

【０３４８】

インポートされたＳＤＭ文書からのＸＳＤ型にはＱＮａｍｅを使用してアクセス可能である。

【０３４９】

【表５８】

```
<alias>:<type-name>
```

【０３５０】

次の例は、上記のレジストリ例で定義された型を使用する。文書は、設定定義を含むレジストリ文書をインポートする。

【０３５１】

【表５９】

```
<Import Alias="Registry" Name="System.OperatingSystem.Registry" Version="0.1.0.0"
PublicKeyToken="AAAABBBBCCCCDDDD"/>
```

【０３５２】

次いで設定定義スキーマはＳｙｓｔｅｍ．Ｏｐｅｒａｔｉｎｇｓｙｓｔｅｍ．Ｒｅｇｉｓｔｒｙネーム空間も参照する。

【０３５３】

【表６０】

```
<SettingDefinitions>
    <xs:schema targetNamespace=" http://DocumentExamples"
        xmlns="http://DocumentExamples"
        xmlns:reg="System.OperatingSystem.Registry/AAAABBBBCCCCDDDD"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:simpleType name="UsesReg">
            <xs:restriction base="reg:RegistryValueType">
                <xs:enumeration value="guid"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:schema>
</SettingDefinitions>
```

【０３５４】

３．２．２　組み込み単純データ型

ＳＤＭは、ＸＳＤおよびＣ＃ネーム空間の共通部分である一組の限定された組み込みデータ型をサポートする。これらの型はＳＤＭランタイムによって本質的にサポートされており、次に示す表に定義される。これらの型に加えて、ユーザはｘｓｄとｃｌｓ型の間に当該ユーザ独自のマッピングを自由に構築し、使用する。

【０３５５】

【表６１】

型	説明	XSD型	C#型
String	Stringは一連のユニコード文字である	string	String
Int	32ビットの署名された整数型	int	System.Int32
Unsigned Int	32ビットの未署名整数	unsignedInt	System.UInt32
Long	64ビットの署名された整数型	long	System.Int64
Unsigned Long	64ビットの未署名整数	unsignedLong	System.UInt64
Float	単一の精度浮動小数点型	float	float
Double	ダブルの精度浮動小数点型	double	double
Boolean	Boolean値は真または偽である（受理された値は「真」、「偽」、0、および1である）	boolean	boot
Any	すべての他の型の基準型	anyType	object
DateTime	単純な日付	date	dateTime
Guid	大域的に一意の識別子	String（パターン付き）	System.Guid
Duration	時間間隔	Duration	System.Timesnan
Version	バージョン番号	ストリングパターン： =″([0-9]{1,4}|[0-5][0-9]{4}|64[0-9]{3}|655[0-2][0-9]|6553[0-5])(＼.([0-9]{1,4}|[0-5][0-9]{4}|64[0-9]{3}|655[0-2][0-9]|6553[0-5])){3}″/>	System.Version
Culture	カルチャ識別子	ストリングパターン：=	System.Culture Info

【０３５６】

これらの型をｃ＃およびｘｓｄ型空間のそれらの型の整合する導出にフローすることができる。例えば、ストリングに対する値は、ストリングに対する制約を定義したｘｓｄ型にフローすることができ、いかなる値でもｔｙｐｅ＝“ａｎｙ”を受理する設定にフローすることができる。

【０３５７】

３．２．２．１　ＸＳＤ組み込み型

図１５は、組み込みデータ型階層の一例を示している。図１５で、すべての複合型は拡張または制約によって導出される。ＮＭＴＯＫＥＮＳ、ＩＤＲＥＦＳ、およびＥＮＴＩＴＩＥＳの型はリストによって導出される。図１５に示す、すべての他の型は制約によって導出される。

【０３５８】

３．２．２．２　Ｃ＃データ型

【０３５９】

【表６２】

型	説明	例
object	すべての他の型の究極的な基準型	object o = null;
string	ストリング型；ストリングは一連のユニコード文字である	string s = "hello";
sbyte	8ビットの署名された整数型	sbyte val = 12;
short	16ビットの署名された整数型	short val = 12;
int	32ビットの署名された整数型	int val = 12;
long	64ビットの署名された整数型	long val1 = 12; long val2 = 34L;
byte	8ビットの未署名の整数型	byte val1 = 12;
ushort	16ビットの未署名の整数型	ushort val1 = 12;
uint	32ビットの未署名の整数型	uint val1 = 12; uint val2 = 34U;

ulong	64ビットの未署名の整数型	ulong val1 = 12;
		ulong val2 = 34U;
		ulong val3 = 56L;
		ulong val4 = 78UL;

float	単一の精度浮動小数点型	float val = 1.23F;
double	ダブルの精度浮動小数点型	double val1 = 1.23; double val2 = 4.56D;
bool	ブーリアン型；boolの値は真または偽である	bool val1 = true; bool val2 = false;
char	文字型；charの値はユニコード文字である	char val = 'h';
decimal	28の有効数字を有する精度10進数型	decimal val = 1.23M;

【０３６０】

３．２．２．３　サポートされる変換

ｘｓｄ型とｃｌｓ型の間に存在する変換である。

【０３６１】

【表６３－１】

XMLスキーマ（XSD）型	.NETフレームワーク型
anyURI	System.Uri
base64Binary	System.Byte[]
Boolean	System.Boolean
Byte	System.SByte
Date	System.DateTime
dateTime	System.DateTime
Decimal	System.Decimal
Double	System.Double
Duration	System.TimeSpan
ENTITIES	System.String[]
ENTITY	System.String
Float	System.Single
gDay	System.DateTime
gMonthDay	System.DateTime
gYear	System.DateTime
gYearMonth	System.DateTime
hexBinary	System.Byte[]
ID	System.String
IDREF	System.String
IDREFS	System.String[]
Int	System.Int32
Integer	System.Decimal

【０３６２】

【表６３－２】

Language	System.String
Long	System.Int64
Month	System.DateTime
Name	System.String
NCName	System.String
negativeInteger	System.Decimal
NMTOKEN	System.String
NMTOKENS	System.String[]
nonNegativeInteger	System.Decimal
nonPositiveInteger	System.Decimal
normalizedString	System.String
NOTATION	System.String
positiveInteger	System.Decimal
QName	System.Xml.XmlQualifiedName
Short	System.Int16
String	System.String
Time	System.DateTime
timePeriod	System.DateTime
Token	System.String
unsignedByte	System.Byte
unsignedInt	System.UInt32
unsignedLong	System.UInt64
unsignedShort	System.UInt16

【０３６３】

３．２．３　設定メンバ

命名された設定を作成するために設定宣言のセクションは前セクションからの設定定義を使用する。ｘｓｄスキーマ内の設定定義は、設定を含む文書に対する別名（設定定義が現在の文書にない場合は要求されない）と、次いでその設定の名前とを含む修飾された名前を使用して参照される。設定定義を参照する際、本発明者らはｘｓｄネーム空間を使用しない。

【０３６４】

各設定メンバに対するさらなる情報を提供するために属性が使用される。

【０３６５】

【表６４】

```
<xs:complexType name="SettingMember">
    <xs:complexContent>
        <xs:extension base="Member">
            <xs:attribute name="List" type="xs:boolean"/>
            <xs:attributeGroup ref="SettingsAttributes"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０３６６】

【表６５】

属性/要素	説明
SettingsAttributes	個々の設定宣言に適用することのできる一組の属性。節3.2.5を参照されたい。
List	この設定が単一の値ではなく値のリストであることを示すためにこの属性が使用される。

【０３６７】

３．２．４　リストのサポート

複数値設定の操作をサポートするために、本発明者らは設定値の単純なリストをサポートする。リストは、設定宣言と同じ定義の一連の値である。リストは、それらを置き換えられることができるかまたはそれらに添付することができる他のリストにフローすることができる。値をリストに添付することは設定フローを使用してより柔軟に行うことができ、また本発明者らはいかなる形式の順番も保証しないので、値をリストに添付する場合、本発明者らは複製検出をサポートしない。

【０３６８】

リスト宣言は、真に設定された属性リストを含む。

【０３６９】

【表６６】

```
<SettingDeclaration Name="roles" Definition="String" List="true"/>
```

【０３７０】

次いで設定ＶａｌｕｅＬｉｓｔを使用して値が提供される。リストを提供する場合、ユーザは以前の値を置き換えるかまたはそれに添付するかを指定することができる。

【０３７１】

【表６７】

```
<SettingValueList Path="roles" Fixed="true" Replace="true">
    <Value>staticContent</Value>
    <Value>aspPages</Value>
</SettingValueList>
```

【０３７２】

ＳＤＭは値のリストの単純な操作をサポートする。フローメンバからの経路が設置宣言を目標とする場合、その結果得られた動作は経路の一方の定義に依存する。

【０３７３】

【表６８】

ソース	宛先	結果
要素	リスト	置き換え＝偽－要素がリストに追加される 置き換え＝真－単一要素を有するリスト
リスト	リスト	置き換え＝偽－ソースおよび宛先リストがマージされる 置き換え＝真－ソースリスト
リスト	要素	SDMはこのリストからどの要素を選択すべきかを決定できないので、この組合せはサポートされない。これを機能させるにはフローを使用せよ。

【０３７４】

３．２．５　設定属性

特定設定の動作を記述するためにランタイムによって設定属性が使用される。

【０３７５】

【表６９】

```
<xs:attributeGroup name="SettingsAttributes">
    <xs:attribute name="Access">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="Readwrite"/>
                <xs:enumeration value="Readonly"/>
                <xs:enumeration value="Writeonly"/>
            </xs:restriction>
```

【０３７６】

【表７０】

```
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="Secure" type="xs:boolean"/>
    <xs:attribute name="DeploymentTime" type="xs:boolean"/>
    <xs:attribute name="Dynamic" type="xs:boolean"/>
    <xs:attribute name="Required" type="xs:boolean"/>
    <xs:attribute name="KeyValue" type="xs:boolean"/>
    <xs:attribute name="Nillable" type="xs:boolean"/>
</xs:attributeGroup>
```

【０３７７】

【表７１】

<table>
<tr><th>属性名</th><th>説明</th><th>ディフォルト</th></tr>
<tr><td>Access</td><td>設定のAccess属性は、設定値の読み取りおよび書き込みが許可されているか否かを指定し、設計者にSDMランタイムアクセス制御および表示/編集規則を提供する。
<table>
<tr><th>属性値</th><th>SDMランタイムでの意味</th><th>表示/編集規則</th></tr>
<tr><td>readwrite</td><td>設定値を読み取り、書き込みすることができることを示す。</td><td>値が表示されており、編集することができる。</td></tr>
<tr><td rowspan="3">readonly</td><td>設定値は読み取ることができるが、書き込むことはできないということを示す。この値はフローのターゲットになることはできない。</td><td rowspan="3">値は表示されているが、編集することはできない。</td></tr>
<tr><td>一般に、readonly設定は計算されるものであるか、または現実のインスタンスによって提供されるものである。</td></tr>
<tr><td>例：サーバ上の接続数を表す値</td></tr>
<tr><td rowspan="2">writeonly</td><td>設定値は書き込むことができるが、読み取られないということを示す。この値はフローのソースになることができない。</td><td rowspan="2">値はマスクされているが、編集することができる。</td></tr>
<tr><td>例：サービスアカウントのパスワード</td></tr>
</table></td><td>readWrite</td></tr>
<tr><td>Deployment Time</td><td>インスタンス用の配置プロセスの一部としてしか提供できない値。設計時制約条件をこの値に対して評価することはできない。定義またはメンバはこの設定に固定値を供給することはできない。</td><td>偽</td></tr>
<tr><td>Required</td><td>Requiredが真の場合、インスタンスが配置される前に値はこの設定に提供されなければならない。この設定はreadonlyであることはできない。</td><td>偽</td></tr>
<tr><td>Dynamic</td><td>Dynamicが真である場合、インスタンスが配置された後でインスタンスマネージャはこの値への変更をサポートする。</td><td>真</td></tr>
<tr><td>KeyValue</td><td>設定がホスティングスコープ内で一意であることを示すためにKeyvalueが使用される。インスタンスマネージャは、オブジェクトインスタンスを特定し、既存のインスタンスとの衝突を検出するために経路のこの設定を使用する。</td><td>偽</td></tr>
<tr><td>Secure</td><td>設定のSecure属性は、SDM文書に記憶される場合に設定の値が暗号化されるべきか否か（真か偽か）を指定する。インストールのような操作中にツールがこの値をログするか否かも示す。</td><td>偽</td></tr>
<tr><td>Nillable</td><td>設定のNillable属性は、値を含んでいない場合に設定値が有効か否かを示す。</td><td>偽</td></tr>
</table>

【０３７８】

３．２．６　設定値

設定が単一値として宣言されているかリストとして宣言されているかによって、設定に対する値を設定値要素または設定値リスト要素を使用して提供することができる。

【０３７９】

３．２．６．１　設定値

特定の設定宣言に対する値を提供するために設定値ステートメントが使用される。この値は、設定宣言に関連付けられた定義にキャスト可能であるかまたは変換されなければならない宣言に関連付けられた定義と一致しなければならない。この値が固定と宣言された場合、その値が固定されているポイントに応じて、提供された値がすべての導出された定義または参照するメンバで使用される。値は一度固定されると、上書きすることはできない。

【０３８０】

【表７２】

```
<xs:complexType name="SettingValue" mixed="true">
    <xs:sequence>
        <xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Path" type="Path" use="required"/>
    <xs:attribute name="Fixed" type="xs:boolean" use="optional"/>
    <xs:attribute name="Nil" type="xs:boolean"/>
    <xs:attribute name="Definition" type="QualifiedName" use="optional"/>
</xs:complexType>
```

【０３８１】

【表７３】

属性/要素	説明
Path	この値が適用されるべき設定宣言への経路。
Fixed	Fixed属性は、提供された値は後で新しい値によって変更することができるか否かを制御する。真のFixed値は、設定に提供された値を変更することはできないことを示す。Fixedの値が具体的な定義メンバの設定値に対して真である場合、その値はそのメンバのすべての配置に固定される。そうではなく、Fixedの値が偽である場合、その値はそのメンバのそれぞれの配置で変更可能である。Fixedの値が具体的な定義の設定値に対して真である場合、その値はその具体的な定義のすべてのメンバに対して固定されている（すなわち、使用する）。そうではなく、Fixedの値が偽である場合、その値はその具体的な定義の各メンバによって異なる（すなわち、使用する）場合がある。 Fixedの値が抽象定義の設定値に対して真である場合、その値は実施する具体的定義またはその抽象定義を拡張する抽象オブジェクトに対して固定されている。そうではなく、Fixedの値が偽である場合、導出した抽象定義またはメンバ宣言でその値を具体的な定義によって変更することができる。
Nil	最後に、設定値が真の値の属性Nilを有する場合、その設定値は内容なしに有効であると見なされる。このようにラベリングされた要素は、空でなければならないが、設定定義によって許可される場合は属性を保持することができる。
Definition	Definition属性は、設定定義に対するレファレンスを含むことができる。これが提供されている場合、設定値はこの定義で型される。これが提供されていない場合、この設定値を型するために経路末端の設定宣言の設定定義が使用される。

【０３８２】

３．２．６．２　設定値リスト

リストとして宣言された設定に対する１つまたは複数の値を提供するために設定値リストが使用される。値を宣言する際、ユーザは以前の値とマージするかまたはすべての以前の値を上書きするかを決定することができる。

【０３８３】

【表７４】

```
<xs:complexType name="SettingValueList">
    <xs:sequence>
        <xs:element name="Value" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType mixed="true">
                <xs:sequence>
                    <xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
    <xs:attribute name="Path" type="SimpleName" use="required"/>
    <xs:attribute name="Fixed" type="xs:boolean" use="optional"/>
    <xs:attribute name="Replace" type="xs:boolean" use="optional"/>
    <xs:attribute name="Definition" type="QualifiedName" use="optional"/>
</xs:complexType>
```

【０３８４】

【表７５】

属性/要素	説明
Path	この値が適用されるべき設定宣言の名前。
Fixed	Fixed属性は、提供された値は後で新しい値によって変更することができるか否かを制御する。真のFixed値は、設定に提供された値を変更することはできないことを示す。Fixedの値が具体的な定義メンバの設定値に対して真である場合、その値はそのメンバのすべての配置に固定される。そうではなく、Fixedの値が偽である場合、その値はそのメンバのそれぞれの配置で変更可能である。Fixedの値が具体的な定義の設定値に対して真である場合、その値はその具体的な定義のすべてのメンバに対して固定されている（すなわち、使用する）。そうではなく、Fixedの値が偽である場合、その値はその具体的な定義の各メンバによって異なる（すなわち、使用する）場合がある。 Fixedの値が抽象定義の設定値に対して真である場合、その値は実施する具体的定義またはその抽象定義を拡張する抽象オブジェクトに対して固定されている。そうではなく、Fixedの値が偽である場合、導出した抽象定義またはメンバ宣言でその値を具体的な定義によって変更することができる。
Replace	設定に対する新しい値が設定に対する任意の以前の非固定値に置き換わるかまたはこれとマージするかを示すためにReplace属性が使用される。 Replaceが真である場合、すべての以前の値は無視される。Replaceが偽である場合、新しい値と以前の値が単一のリストに結合される。マージプロセス中、複製は検出されない。
Definition	Definition属性は、設定定義に対するレファレンスを含むことができる。これが提供されている場合、設定値のそれぞれはこの定義で型される。これが提供されていない場合、この経路末端の設定宣言の設定定義が各設定値に対する型として使用される。

【０３８５】

３．２．７　設定の継承

設定の継承は、導出された定義が基礎定義からのすべての設定宣言を暗黙的に含んでいることを意味している。設定の継承のいくつかの重要な態様は次の通りである。

【０３８６】

設定の継承は推移的である。ＣがＢから導出され、ＢがＡから導出される場合、ＣはＡで宣言された設定と同様、Ｂで宣言された設定をも継承する。

【０３８７】

導出された定義はそれが拡張する基礎定義から継承するものに新しい設定宣言を追加することができるが、継承した設定の定義を除去または変更することはできない。

【０３８８】

３．２．８　設定インスタンス

インスタンス空間の設定インスタンスは設定メンバに対する実際の値を保持する。まず、設定インスタンスに対する値は定義空間の設定値ステートメントによって割り当てられる。次いで設定フローが評価され、フローのターゲットである設定インスタンスの値が更新される。

【０３８９】

シナリオによっては、本発明者らは設定値ステートメントから導出した設定値とフロー

によって後で提供される任意の値との間を区別しなければならない。本発明者らは、前者を「割り当てられた」値、後者を「結果として得られた」値と呼ぶ。

【０３９０】

３．２．９　値の転送

設定値をフローに渡し、または制約条件から得るために、本発明者らは値の転送を使用する。メンバに関連付けられた定義が評価される前に入力転送が評価され、メンバに関連付けられた定義が評価された後で出力転送が評価される。

【０３９１】

各値の転送は、入力の宛先または出力のソースとして使用されるべき定義に対する設定をＮａｍｅ属性によって示す。Ｐａｔｈ属性は、入力のソースまたは出力の宛先である設定宣言を示す。

【０３９２】

【表７６】

```
<xs:complexType name="ValueTransfer">
    <xs:attribute name="Name" type="SimpleName"/>
    <xs:attribute name="Path" type="Path"/>
</xs:complexType>
```

【０３９３】

【表７７】

属性/要素	説明
Name	経路が示した設定のソースまたは宛先であるフローまたは制約条件定義に対する設定の名前。
Path	メンバが定義されるコンテキストに対するソースまたは宛先設定への経路である。Pathは単一の設定を示す。これは、その経路上のすべてのメンバの最大カーディナリティが1または0であるべきことを示している。

【０３９４】

３．２．９．１　入力

入力ステートメントは、フローまたは制約条件が実行される前に転送されることになる入力値を示す。入力は、「割り当てられた」設定値または「結果として得られた」設定値が入力として使用されるか否かを選択するためにＶａｌｕｅＳｏｕｒｃｅ要素を使用することができる（「割り当てられた／結果として得られた」の定義については節ＸＸを参照のこと）。

【０３９５】

【表７８】

```
<xs:complexType name="Input">
    <xs:complexContent>
        <xs:extension base="ValueTransfer">
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０３９６】

３．２．９．２　出力

出力は、ターゲット値を固定し、置き換えるための意味論をサポートする値の転送の一種である。

【０３９７】

【表７９】

```
<xs:complexType name="Output">
    <xs:complexContent>
        <xs:extension base="ValueTransfer">
            <xs:attribute name="Fix" type="xs:boolean" use="optional"/>
            <xs:attribute name="Replace" type="xs:boolean" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０３９８】

【表８０】

属性/要素	説明
Fix	これは、固定されるべきターゲット値を宣言する。これは、後でこの値を修正しようとするいかなる試みによってもエラーが生じることを意味する。
Replace	リストが経路のターゲットである場合、本発明者らはそのリストの現在の内容とマージするかまたはこれと置き換えるかを選ぶことができる。ディフォルトの動作は現在の内容とマージすることである。

【０３９９】

３．２．１０　キャスティング

本発明者らは、設定定義継承ツリーのキャスティングアップおよびキャスティングダウンをサポートする。ソース定義が宛先定義から導出される場合、ソースから宛先定義へのキャストをアップキャストと呼ぶ。宛先定義がソース定義から導出される場合、ソースから宛先へのキャストをダウンキャストと呼ぶ。ツリーのルートは常にＡｎｙ定義なので、すべての定義をＡｎｙ定義にアップキャストすることができる。

【０４００】

図１６は、アップキャスティングとダウンキャスティングの一例を示している。図１６では、Ｕｓｅｒ定義は基礎定義Ａｎｙを拡張する。ＥｘｔｅｎｄｅｄＵｓｅｒ定義は基礎定義Ｕｓｅｒを拡張する。この定義の階層を構築することによって、本発明者らは、Ｕｓｅｒ定義が要求されるところならどこでもＥｘｔｅｎｄｅｄＵｓｅｒ定義を提供することができ、Ａｎｙ定義が要求されるところならどこでもＥｘｔｅｎｄｅｄＵｓｅｒとＵｓｅｒ定義の両方を提供することができることを示している。

【０４０１】

次に示す設定宣言ステートメントは、図１６からの定義を使用しており、次節の例で使用される。

【０４０２】

【表８１】

```
<SettingDeclaration Name="First" Definition="ExtendedUser"/>
<SettingDeclaration Name="Second" Definition="Any"/>
<SettingDeclaration Name="Third" Definition="User"/>
```

【０４０３】

３．２．１０．１　アップキャスト

図１６で、ＥｘｔｅｎｄｅｄＵｓｅｒ定義のインスタンスがＵｓｅｒまたはＡｎｙの定義を要求する設定宣言に割り当てられ際にアップキャストが発生する。アップキャストの一例は、設定宣言Ｆｉｒｓｔから設定宣言Ｓｅｃｏｎｄへの値の転送である。これはフロー入力宣言の一部として行うことができる。

【０４０４】

アップキャストは自動的にサポートされる。いかなる値でも、開発者側に特別な対話を必要とせずにその基礎定義の１つにキャストすることができる。設定インスタンスがアップキャストされる際、インスタンスの実際の定義は設定値と共に保持される。

【０４０５】

３．２．１０．２　ダウンキャスト

図１６で、Ａｎｙとマーク付けされた設定宣言からの設定値をＵｓｅｒとマーク付けされた設定宣言にユーザが転送する際にダウンキャストが発生する。この場合、インスタンスに関連付けられた定義がＵｓｅｒかまたはＥｘｔｅｎｄｅｄＵｓｅｒの場合にのみ割り当てを行うことができる。これは、設定インスタンスが、現在関連付けられている設定宣言と異なる定義を有することができることを示している。本発明者らは、それ自体のインスタンスを中心としたインスタンスの実際の定義にレファレンスを渡すことによってインスタンス空間内でこれを追跡する。

【０４０６】

ダウンキャストの一例は、インスタンスがそもそもＦｉｒｓｔから来たものであると仮定すると、本発明者らが設定インスタンスをＳｅｃｏｎｄからＴｈｉｒｄに渡す際に発生する場合がある。この場合、設定インスタンスはＥｘｔｅｎｄｅｄＵｓｅｒ定義のインスタンスである。この例は、設定インスタンスがフロー定義から転送される際に発生する場合がある。

【０４０７】

ダウンキャストは、開発者によって明示的に要求されており、インスタンスのランタイム定義がターゲット定義に等しいかまたはターゲット定義から導出された場合には失敗する。ダウンキャストを要求するために、開発者は関連付けられた転送動作（入力または出力）のＣａｓｔ属性を真に設定する。キャストが要求される場合、宛先定義がソース定義から導出されたものでないならばユーザはコンパイル時エラーを受け取り、設定インスタンスの定義が宛先定義と同じでないかまたは宛先定義から導出されたものでないならばユーザはランタイムエラーを受け取る。

【０４０８】

３．２．１１　変換

ユーザが設定インスタンスを、インスタンスと異なる定義を有する設定宣言に転送することを希望し、設定宣言定義がインスタンスの定義の基礎定義でない場合、型変換が要求される。

【０４０９】

図１７は、型変換の一例を示している。図１７に示すように、型変換は、継承ツリーの異なる分岐内の定義間で、宛先定義が継承ツリーの１つの分岐内のソース定義の下にある時に発生する場合がある。

【０４１０】

型変換は設定マネージャによって取り扱われる。ソースマネージャとターゲットマネージャはどちらも変換サポートを公開することができる。これは、図１７の定義と、Ｕｓｅｒ定義からＳｔｒｉｎｇ定義への割り当てがあると仮定すると、ＵｓｅｒのためのマネージャがＳｔｒｉｎｇへの変換を公開することができるか、またはＳｔｒｉｎｇのためのマネージャがＵｓｅｒからの変換を公開することができることを意味している。

【０４１１】

ランタイムはまずソースマネージャにインスタンスを変換するよう要求し、次いでターゲットマネージャに要求し、最後に使用可能な変換がないならばエラーを戻す。

【０４１２】

型変換を要求するために、開発者は設定転送（入力または出力）で属性Ｃｏｎｖｅｒｔを真にマーク付けする。Ｃｏｎｖｅｒｔが指定されておらず、ソース定義がターゲット定義と同じでないかまたはターゲット定義から導出されない場合、コンパイラはエラーを戻す。Ｃｏｎｖｅｒｔが指定されている場合、コンパイラは、両方のマネージャにそれらが変換をサポートするか否かを問うことによって、２つの型間に使用可能な変換があるか否

かをチェックする。

【０４１３】

インスタンスが設定宣言に関連付けられた定義から導出されたものである定義のものである場合、本発明者らは、アップキャスティングにフォールバックし次いで値を変換する前に、導出された定義を宛先定義にまず変換するよう試みる。本発明者らがアップキャストする場合、本発明者らは変換プロセスの一部として情報を失う可能性があることに留意されたい。例えば、本発明者らがＵｓｅｒとＳｔｒｉｎｇの間の変換を指定しておらず、実際のインスタンスがＥｘｔｅｎｄｅｄＵｓｅｒであった場合、本発明者らはＵｓｅｒからＳｔｒｉｎｇにフォールバックする前にＥｘｔｅｎｄｅｄＵｓｅｒをＳｔｒｉｎｇにまず変換するよう試みる。

【０４１４】

３．２．１２　ＣＬＲからＳＤＭへの型マッピング

ＳＤＭモデルでは、すべての設定定義は対応するＣＬＲ型を有する。これらの型は、ＳＤＭモデルのＳｅｔｔｉｎｇＤｅｆｉｎｉｔｉｏｎｓ要素に関連付けられたマネージャによって定義される。

【０４１５】

フロー、制約条件、または配置マネージャがＳＤＭ定義の代わりに機能する場合、それは、ＳＤＭのＳｅｔｔｉｎｇ定義のインスタンスに対してではなくＣＬＲ型のインスタンスに対して機能することによってこれを行う。これは、より自然なプログラミングモデルを開発者に提供する。これはまた、ＳＤＭランタイムからマネージャに情報を渡すためにＳＤＭランタイムがＳＤＭ定義からＣＬＲ型にマップすることができるべきであり、マネージャから戻された情報を受理するためにＣＬＲ型からＳＤＭ定義にマップすることができるべきであることも意味している。

【０４１６】

例えば、ＳＤＭモデルでは、開発者は次に示す設定定義を定義することができる。

【０４１７】

【表８２】

```
<xs:simpleType name="AppPoolIdentityType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="LocalSystem"/>
        <xs:enumeration value="LocalService"/>
        <xs:enumeration value="NetworkService"/>
        <xs:enumeration value="UserDefined"/>
    </xs:restriction>
</xs:simpleType>
```

【０４１８】

次いで、開発者は、次に示すＣＬＲ型のインスタンスを取る設定定義に関連付けられた値を処理するマネージャを書くことができる。

【０４１９】

【表８３】

```
public enum AppPoolIdentityType
{
  LocalSystem,
  LocalService,
  NetworkService,
```

【０４２０】

【表８４】

```
   UserDefined,
 }
```

【０４２１】
この問題には２つの部分がある。第１は定義から型にマップし、また戻すことができ、第２はＳＤＭインスタンスからｃｌｒインスタンスにマップし、また戻すことができるということである。

【０４２２】
３．２．１２．１　型へのマッピング定義
設定定義からｃｌｒ型にマップするために、本発明者らはマネージャ宣言、設定定義ネーム空間、および設定定義名を使用する。ＳｅｔｔｉｎｇＤｅｆｉｎｉｔｉｏｎｓ要素のＭａｎａｇｅｒ属性によって示されたマネージャ宣言から、本発明者らは強力な名前と型を含んでいるアセンブリに対する位置を得ることができ、ＳｅｔｔｉｎｇＤｅｆｉｎｉｔｉｏｎｓ要素のＣｌｒＮａｍｅｓｐａｃｅ属性から、本発明者らは型を含んでいるネーム空間の名前を得ることができる。最後に、本発明者らは、アセンブリからｃｌｒ型を取り出すためにネーム空間の名前に設定定義の名前を添付することができる。

【０４２３】
ｃｌｒ型からＳＤＭ定義にマップするために、逆の処理は同じ情報を同じ情報を使用する。まず、本発明者らはランタイムによって提供された型情報からアセンブリ名を取得する。次いで本発明者らは、１つまたは複数のモデルファイルを本発明者らに提供するアセンブリを参照するマネージャ宣言を見つける。次いでモデルファイルの設定定義セクションから、本発明者らはネーム空間の名前と型の名前の対応する対を見つける。

【０４２４】
３．２．１２．２　変換インスタンス値
設定インスタンスをＸＭＬのＳＤＭ表現からＣｌｒオブジェクトインスタンスに変換し、また戻すために、本発明者らはここでもまたＳｅｔｔｉｎｇＤｅｆｉｎｉｔｉｏｎｓ要素に関連付けられたマネージャを使用する。この場合、翻訳を実施するＣｌｒクラスは、ＣｌｒＣｌａｓｓＮａｍｅ属性とマネージャ属性によって示されたマネージャ宣言からのアセンブリ情報との組合せによって特性される。

【０４２５】
Ｃｌｒクラスは、マネージャインターフェース基準（参照）に記載のＩＳｅｔｔｉｎｇＳｅｒｉａｌｉｚａｔｉｏｎインターフェースを実施する。このインターフェースは、ＳＤＭ　Ｘｍｌ表現からＣｌｒインスタンスへの変換と、Ｃｌｒオブジェクトインスタンスから ＳＤＭ　Ｘｍｌ表現への変換の両方をサポートする。Ｍｏｄｅｌファイル内で設定定義を作成するのと同じ開発者は、変換を実行するコードを書くことを担当する。

【０４２６】
Ｃｌｒクラスは、ＳｅｔｔｉｎｇＤｅｆｉｎｉｔｉｏｎｓ要素内のすべての設定定義に対する変換をサポートすべきである。

【０４２７】
３．３　属性
ＳＤＭ内のオブジェクトの多くは、オブジェクトの中心的な動作に直交する捕獲動作をその属性とすることができる。本発明者らは以下に定義する一般的な属性モデルを使用する。

【０４２８】
３．４　定義
資源およびシステムの再利用可能な構成を作成するために定義が使用される。これらはオブジェクト指向言語のクラスに類似している。

【０４２９】
３．４．１　定義

定義は、そこからオブジェクト定義、関係定義、制約条件定義、およびフロー定義が導出される基礎である。すべての定義は、説明、設定メンバ、設定値、および設計表面データを含むことができる。各定義は単純名によって特定され、マネージャへを参照する。マネージャは、定義によって特定されたクラスによってこの特定の定義に対するＳＤＭランタイムにサポートを提供することを担当している。

【０４３０】

定義の名前は、定義を含んでいるスコープ内のすべての定義に対して一意であるべきである。スコープはＳＤＭ文書であっても別の定義であってもよい。

【０４３１】

【表８５】

```
<xs:complexType name="Definition">
    <xs:sequence>
        <xs:element name="Description" type="Description" minOccurs="0"/>
        <xs:element name="DesignData" type="DesignData" minOccurs="0"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="SettingDeclaration" type="SettingMember"/>
            <xs:element name="SettingValue" type="SettingValue"/>
            <xs:element name="SettingValueList" type="SettingValueList"/>
        </xs:choice>
    </xs:sequence>
    <xs:attribute name="Name" type="SimpleName" use="required"/>
    <xs:attribute name="Manager" type="QualifiedName" use="optional"/>
    <xs:attribute name="ClrClassName" type="xs:string" use="optional"/>
</xs:complexType>
```

【０４３２】

【表８６】

属性/要素	説明
SettineDeclaration	設定の宣言
SettingValue	定義またはその基礎定義に対する設定値。値は、定義内の設定宣言に対して一度しか提供できない。
SettingValueList	定義またはその基礎定義に対する書き込み可能リスト設定値のリスト。
DesignData	設計表面特有データ
Name	含んでいるSDMファイルのスコープ内で一意のこの定義に対する名前。
Manager	この定義に対するマネージャ宣言へのレファレンス。マネージャの定義に関しては節3.5を参照されたい。
ClrClassName	ランタイムでこの定義をサポートするclrクラスの名前。クラスは、マネージャによって示されたアセンブリ内に存在すべきである。この属性がある場合はマネージャの属性もなければならない。
Description	定義のテキスト記述。

【０４３３】

３．４．２　オブジェクト定義

抽象オブジェクト定義は、一組の設定宣言を公開する。抽象オブジェクト定義は、それが参加し、ランタイム内の関連するマネージャを有する関係に対する制約条件を含むことができる。

【０４３４】

ウェブサーバに対する抽象システム定義を次に示す。ウェブサーバは、２つの設定を有しており、少なくとも１つのｖｓｉｔｅ型を含むことをそれに要求する関係制約条件を有する。

【０４３５】
【表８７】

```
<SystemDefinition Abstract="true"
                  Name="WebServer"
                  ClrClassName="micorosft.sdm.IISSupportClass"
                  Manager="IISSupportCode">

    <SettingDeclaration Name="serverName" Definition="String"/>
    <SettingDeclaration Name="category" Definition="String"/>
    <SettingDeclaration Name="roles" Definition="String" List="true"/>

        <RelationshipConstraint  Name="containsVsites"
                                 RelationshipDefinition="containmentRelationship"
                                   TargetRole="Member"
                                   TargetObjectDefinition="vsite"
                                   MinOccurs="1"
                                   MaxOccurs="unbounded"/>

</SystemDefinition>
```

【０４３６】
ｖｓｉｔｅはサーバ結合情報を含んでいる抽象エンドポイント定義である。
【０４３７】
【表８８】

```
<EndpointDefinition Abstract="true" Name="vsite">
    <SettingDeclaration Name="ipAddress" Definition="ipaddress" required="true"/
    <SettingDeclaration Name="port" Definition="Int"/>
    <SettingDeclaration Name="domainName" Definition="Int"/>
    <SettingDeclaration Name="securityModel" Definition="securityModelEnum"/>
</EndpointDefinition>
```

【０４３８】
フロントエンドウェブサーバに対する具体的なシステム定義は、静的内容としてウェブサーバのカテゴリを特定し、１と１００の間のエンドポイントインスタンスを表すことのできる単一のｂｙＲｅｆｅｒｅｎｃｅエンドポイントメンバを含んでいる。エンドポイントに対する具体的なエンドポイント定義はシステム定義内にネストされており、ｖｓｉｔｅがエンドポイント８０になるｉｐエンドポイントを定義する。
【０４３９】
【表８９】

```
    <SystemDefinition Name="FrontendWebServer" Extends="WebServer">
        <SettingValue Path="serverName" Fixed="true">myFEServer</SettingValue>
        <SettingValueList Path="roles" Fixed="true" Replace="true">
            <Value>staticContent</Value>
            <Value>aspPages</Value>
        </SettingValueList>
        <EndpointDefinition Name="port80Vsite" Extends="vsite">
            <SettingValue Path="port" Fixed="true">80</SettingValue>
        </EndpointDefinition>
        <Endpoint  Name="contentOnlyVsite" Definition="port80Vsite" Reference="true" MinOccurs="1"
MaxOccurs="100"/>
    </SystemDefinition>
```

【０４４０】
３．４．２．１　オブジェクト定義
抽象オブジェクトおよび具体的オブジェクトは、次に示す基礎オブジェクト定義を拡張する。基準型のＤｅｆｉｎｉｔｉｏｎの要素に加えて、これらはオブジェクトが参加する関係を制約する機能を共有する。

【０４４１】
【表９０】

```
<xs:complexType name="ObjectDefinition">
    <xs:complexContent>
        <xs:extension base="Definition">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="Flow" type="FlowMember" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="RelationshipConstraintGroup" type="RelationshipConstraintGroup"/>
                <xs:element name="RelationshipConstraint" type="RelationshipConstraint"/>
                <xs:element name="Constraint" type="ConstraintMember"/>
            </xs:choice>
            <xs:attribute name="Visibility" type="VisibilityEnum" use="optional"/>
            <xs:attribute name="Layer" type="xs:string" use="optional"/>
            <xs:attribute name="Extends" type="QualifiedName" use="optional"/>
            <xs:attribute name="Abstract" type="xs:boolean" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０４４２】
【表９１】

属性/要素	説明
Flow	フローメンバ宣言
RelationshipConstraint	それらの型が参加することのできる関係に対する制約条件。
RelationshipConstraintGroup	それらの型が参加することのできる関係に対する一群の制約条件。当該グループの少なくとも1つのメンバは真と評価すべきである。
Layer	Layer属性は、この抽象オブジェクト定義を使用することのできる層を示す。これが提供されていない場合、抽象オブジェクト定義はいかなる層でも使用することができる。
Extends	このオブジェクト定義がそこから導出される基礎オブジェクト定義を示す。
Abstract	このオブジェクトをインスタンス生成することができるか否かを示す。Abstractとマーク付けされている場合、欠けている要素を完成させるには定義を拡張しなければならない。ディフォルトは偽である。
Visibility	このクラスのメンバがディフォルトで公開になるか秘密になるかを示す。ディフォルトは公開である。

【０４４３】
設計表面を見えるようにし、そこからすべての具体的オブジェクトが導出される構築ブロックを定義するために抽象オブジェクト定義が使用される。具体的なオブジェクト定義は、抽象オブジェクト定義を実施する。
【０４４４】
抽象オブジェクト定義は単純な継承を追加することによってＳＤＭオブジェクトを拡張する。すなわち、抽象オブジェクト定義に対する基礎オブジェクト定義を示すために「ｅｘｔｅｎｄｓ」属性が使用される。次いで抽象オブジェクト定義は、その基礎オブジェクト定義から設定および関係制約条件を継承する。継承により、オブジェクト定義は、新しい設定および制約条件を追加することにより抽象オブジェクト定義の設定および制約条件を拡張することができる。

【０４４５】
抽象オブジェクト定義は、参加を希望する関係に制約条件を追加することもできる。例えば、抽象オブジェクト定義は、ある関係の存在を要求することができ、または当該関係の他端に置かれる場合のあるオブジェクト定義を制約することができ、または所与の関係に参加しているインスタンスに対する設定を制約することができる。
【０４４６】
オブジェクト定義は、ネストされた定義の宣言も含むことができる。これらの定義を、含んでいる定義のスコープ内のメンバに対して使用し、また定義のスコープ外の制約条件で参照することができる。
【０４４７】
具体的なオブジェクト定義は、抽象オブジェクト定義に一実施態様を提供する。この実施態様は、オブジェクトおよび関係メンバ、実施された抽象定義の設定に対する値、新しい設定宣言、メンバとメンバに対する制約条件との間のフローから構築される。
【０４４８】
３．４．２．２　暗黙の基礎定義
他のオブジェクト定義を拡張しないすべてのオブジェクト定義は、エンドポイント、システム、または資源の基礎定義のうちの１つを暗黙的に拡張する。これらの基礎定義は、関係および制約条件宣言内で使用することのできるツリーのそれぞれに対するルートを形成する。これによって、関係または制約条件は、ルートから導出された型のどれでも特定されたルート定義の代わりに使用可能なことを示すことができる。これらルート型は、常に抽象であり、これらを直接的にインスタンス生成することはできない。
【０４４９】
これらの型の定義は、モデル内のそれらのインスタンス生成を制御する基礎制約条件を含む。
【０４５０】
３．４．２．３　エンドポイント定義
エンドポイント定義は、ネストされた資源型、資源メンバおよびホスト、包含およびレファレンス関係メンバを宣言する機能を追加することによって基礎オブジェクト定義を拡張する。
【０４５１】
【表９２】

```
<xs:complexType name="EndpointDefinition">
    <xs:complexContent>
        <xs:extension base="ObjectDefinition">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="ResourceDefinition" type="ResourceDefinition"/>
                <xs:element name="Resource" type="ResourceMember"/>
                <xs:element name="Hosting" type="HostingMember"/>
                <xs:element name="Containment" type="ContainmentMember"/>
                <xs:element name="Reference" type="ReferenceMember"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０４５２】

【表９３】

属性/要素	説明
ResourceDefinition	外部型定義のスコープ内の型メンバによって使用することのできるネストされた資源定義。
Resource	資源型を参照する資源メンバ宣言。
Hostin	ホスティング関係メンバ宣言。
Containment	包含関係メンバ宣言。
Reference	レファレンス関係メンバ宣言。

【０４５３】

３．４．２．４　システム定義

システム型は、ネストされたエンドポイント、システムおよび資源型、エンドポイント、システム、および資源メンバ、およびホスト、包含、接続、委任およびレファレンス関係に関してサポートを追加することにより基準型を拡張する。

【０４５４】

【表９４】

```
<xs:complexType name="SystemDefinition">
    <xs:complexContent>
        <xs:extension base="ObjectDefinition">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="EndpointDefinition" type="EndpointDefinition"/>
                <xs:element name="SystemDefinition" type="SystemDefinition"/>
                <xs:element name="ResourceDefinition" type="ResourceDefinition"/>
                <xs:element name="Endpoint" type="EndpointMember"/>
                <xs:element name="Subsystem" type="SystemMember"/>
                <xs:element name="Resource" type="ResourceMember"/>
                <xs:element name="Hosting" type="HostingMember"/>
                <xs:element name="Containment" type="ContainmentMember"/>
                <xs:element name="Connection" type="CommunicationMember"/>
                <xs:element name="Delegation" type="DelegationMember"/>
                <xs:element name="Reference" type="ReferenceMember"/>
            </xs:choice>
            <xs:attribute name="SimulationRoot" type="xs:boolean" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０４５５】

【表９５】

属性/要素	説明
EndpointDefinition	外部システム定義のスコープ内でメンバが使用することのできるネストされたエンドポイント定義
ResourceDefinition	外部システム定義のスコープ内でメンバが使用することのできるネストされた資源定義。
SystemDefinition	外部システム定義のスコープ内でメンバが使用することのできるネストされたシステム定義
Endpoint	エンドポイント定義を参照するエンドポイントメンバ宣言。
Subsystem	システム定義を参照するサブシステムメンバ宣言。
Resource	資源定義を参照する資源メンバ宣言。
Containment	包含関係メンバ宣言。
Hosting	ホスティング関係メンバ宣言。
Connection	接続関係メンバ宣言。
Delegation	委任関係メンバ宣言。
Reference	レファレンス関係メンバ宣言。
SimulationRoot	設計時チェックを行うために定義がSDM文書のコンパイル中にシミュレーションされたそのルートインスタンスを有するべきか否かを指定する。この属性が指定されていない場合、シミュレーション用のディフォルトコンパイラ規則が適用される。シミュレーションは首尾よくパスしないが定義の実際の処理は正常に機能する場合、この属性が必要とされる。

【０４５６】

３．４．２．５　資源定義

資源型は、ネストされた資源型定義、資源メンバ、およびホスト、包含、およびレファレンス関係メンバを含むことができる。

【０４５７】

【表９６】

```
<xs:complexType name="ResourceDefinition">
    <xs:complexContent>
        <xs:extension base="ObjectDefinition">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="ResourceDefinition" type="ResourceDefinition"/>
                <xs:element name="Resource" type="ResourceMember"/>
                <xs:element name="Hosting" type="HostingMember"/>
                <xs:element name="Containment" type="ContainmentMember"/>
                <xs:element name="Reference" type="ReferenceMember"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０４５８】

【表９７】

属性/要素	説明
ResourceDefinition	外部資源定義のスコープ内でメンバが使用することのできるネストされた資源定義。
Resource	資源定義を参照する資源メンバ宣言。
Hosting	ホスティング関係メンバ宣言。
Containment	包含関係メンバ宣言。
Reference	レファレンス関係メンバ宣言。

【０４５９】

３．４．２．６　関係規則

オブジェクト定義の特定のインスタンスに関して、インスタンスが果たすことのできるロールのそれぞれに関連付けられたカーディナリティを次の表に示す。

【０４６０】

３．４．２．６．１　システム規則

【０４６１】

【表９８】

定義	ロール	システム	エンドポイント	資源
System	親 (0...*)	含む	含む	含む
	メンバ (1...1)	含まれる	禁止	禁止
	プロキシ (0...*)	禁止	禁止	禁止
	代表 (0...*)	禁止	禁止	禁止
	クライアント (0...*)	禁止	禁止	禁止
	サーバ (0..*)	禁止	禁止	禁止
	ゲスト (1..1)	ホストされる	ホストされる (??)	ホストされる
	ホスト (0..*)	ホストする	ホストする	ホストする
	ソース (0..*)	提供する	禁止	禁止
	従属 (0..*)	消費する	禁止	禁止

【０４６２】

３．４．２．６．２　エンドポイント規則

【０４６３】

【表９９】

定義	ロール	システム	エンドポイント	資源
Endpoint	親（0...*）	禁止	禁止	含む
	メンバ（1...1）	含まれる	禁止	禁止
	プロキシ（0...*）	禁止	委任する	禁止
	代表（0...*）	禁止	実施する	禁止
	クライアント（0...*）	禁止	接続する	禁止
	サーバ（0..*）	禁止	サービスを提供する	禁止
	ゲスト（1..1）	ホストされる	ホストされる	ホストされる
	ホスト（0..*）	ホストする	ホストする	ホストする
	ソース（0..*）	禁止	提供する	提供する
	従属（0..*）	禁止	消費する	消費する

【０４６４】

３．４．２．６．３　資源規則

【０４６５】

【表１００】

定義	ロール	システム	エンドポイント	資源
Resource	親（0...*）	禁止	禁止	含む
	メンバ（1...1）	含まれる	含まれる	含まれる
	プロキシ（0...*）	禁止	禁止	禁止
	代表（0...*）	禁止	禁止	禁止
	クライアント（0...*）	禁止	禁止	禁止
	サーバ（0..*）	禁止	禁止	禁止
	ゲスト（1..1）	ホストされる	ホストされる	ホストされる
	ホスト（0..*）	ホストする	ホストする	ホストする
	ソース（0..*）	禁止	提供する	提供する
	従属（0..*）	禁止	消費する	消費する

【０４６６】

３．４．２．６．４注記

各インスタンスは１つの包含関係および少なくとも１つのホスティング関係に参加すべきである。

【０４６７】

これは、次のことを意味する。

Ａ）非レファレンス（バイバリュー）メンバは同じ定義内の包含関係を示すべきである。

Ｂ）構築されるには、レファレンスメンバは包含関係を特定すべきである。

【０４６８】

３．４．３　関係定義

型間の実現可能な相互作用を示すために関係が使用される。これらはバイナリであり有向である。これらは、それぞれが関係に参加することのできるインスタンスの型を示している。関係は、関係に参加するインスタンスの設定も制約することができ、関係を横断して設定値をフローすることができる。

【０４６９】
型の節で説明したウェブサーバのｗｅｂＡｐｐｌｉｃａｔｉｏｎに対する実現可能なホスティング関係を次に示す。関係は、２つのシステムのセキュリティモデルが整合性を有することを検証する制約条件を含み、またｖｓｉｔｅからｖｄｉｒにサーバ名をコピーする設定フローメンバを含む。
【０４７０】
【表１０１】

```
<HostingDefinition Name="vsiteHostsVdir" GuestDefinition="vdir" HostDefinition="vsite">
    <ObjectConstraint Name="checkCompatibility" PrimaryRole="Guest" PrimaryObjectDefinition="vdir">
        <Constraint Name="constrainSecurityModel" Definition="SimpleOperatorComparison">
            <SettingValue Path="operator">=</SettingValue>
            <Input Name="LHS" Path="host.securityModel"/>
            <Input Name="RHS" Path="guest.securityModel"/>
        </Constraint>
    </ObjectConstraint>
    <Flow Definition="copy" Name="copyServerToVdir">
        <Input Name="source" Path="host.server"/>
        <Output Name="desination" Path="guest.server"/>
    </Flow>
</HostingDefinition>
```

【０４７１】
関係に参加することになる型メンバを示す関係メンバを宣言することによって関係が使用される。
【０４７２】
【表１０２】

```
<SystemDefinition Name="TestSystemt" Extends="DistributedApplication">
    <Resource Name="myVdir" Definition="vdir" Reference="false"/>
    <Resource Name="myVsite" Definition="vsite" Reference="false"/>
```

【０４７３】
【表１０３】

```
        <Hosting Name="HostMyVsite" Definition="vsiteHostsVdir" GuestMember="myVdir"
HostMember="myVsite"/>
    </SystemDefinition>
```

【０４７４】
３．４．３．１　関係定義
基礎関係定義は、オブジェクト制約条件およびフローを定義に追加する。オブジェクト制約条件は、この関係のインスタンスに参加するオブジェクトインスタンスに対する設定値に関するステートメントである。例えば、ＤＣＯＭ接続を表す通信関係は、クライアントとサーバに対するセキュリティ設定に整合性があることをチェックすることができる。この場合、設計プロセスの一部として容易に取り込むことのできる設定間に厳密な関係があり、当該関係には４つの階乗設定の組合せがあるが、有効な組合せはさらに少数である。
【０４７５】
フローは、あるインスタンスから別のインスタンスに値を転送する機能を関係設計者に提供する。これによりオブジェクト定義は、それらの実現可能な相互作用とは別個に開発可能となり、また、インスタンスは、特定インスタンスを完全に記述するために関係グラフの部分集合を要求するのではなく、情報に対するレファレンスポイントとして単独で存在することが可能となる。
【０４７６】
具体的な関係は、２つの具体的なオブジェクト定義間の関係である。各具体的な関係は抽象関係を実施すべきである。抽象関係は、具体的なオブジェクト定義によって直接的ま

たは（継承により）間接的に実施される対応する一対の抽象オブジェクト定義の間にあるべきである。

【０４７７】

関係の名前は、関係を含んでいるネーム空間内で一意であるべきである。

【０４７８】

【表１０４】

```
<xs:complexType name="RelationshipDefinition">
    <xs:complexContent>
        <xs:extension base="Definition">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
```

【０４７９】

【表１０５】

```
                <xs:element name="ObjectConstraintGroup" type="ObjectConstraintGroup"/>
                <xs:element name="ObjectConstraint" type="ObjectConstraint"/>
                <xs:element name="Flow" type="FlowMember"/>
            </xs:choice>
            <xs:attribute name="Extends" type="QualifiedName" use="optional"/>
            <xs:attribute name="Abstract" type="xs:boolean" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０４８０】

【表１０６】

属性/要素	説明
ObjectConstraint（グループ）	この関係に参加するインスタンスに対する制約条件。節3.7を参照されたい。
Flow	この関係に参加するインスタンスの間のフロー。
Extends	この定義をそこから導出する基礎定義を示す。
Abstract	この関係をインスタンス生成することができるか否かを示す。abstractとマーク付けされている場合、欠けている要素を完成させるために定義を拡張すべきである。ディフォルトは偽である。

【０４８１】

３．４．３．２　通信関係

エンドポイント定義間の実現可能な通信リンクを取り込むために通信関係が使用される。独立して配置されたソフトウェア要素間の相互作用を記述するためにこれらが使用される。通信関係スキーマは、クライアントおよびサーバのエンドポイントレファレンスを追加することによって基礎関係スキーマを拡張する。

【０４８２】

【表１０７】

```
<xs:complexType name="CommunicationDefinition">
    <xs:complexContent>
        <xs:extension base="RelationshipDefinition">
            <xs:sequence>
                <xs:element name="Connection" type="HostingMember" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="ClientDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="ServerDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０４８３】

【表１０８】

属性/要素	説明
ClientDefinition	通信関係に関与するクライアントインスタンスの定義
ServerDefinition	関係に関与するサーバインスタンスの型

【０４８４】

通信関係には次に示す抽象型対の組合せが有効である。

【０４８５】

【表１０９】

クライアント型	サーバ型
Endpoint	Endpoint

【０４８６】

３．４．３．３　ホスティング関係

構築されるためにゲストがホストを要求しているという事実を取り込むためにホスティング関係が使用される。ゲストに対して複数の実現可能なホストがあり得る場合があるので、ホスティング関係はホストでのゲストの構築を担当することも示している。したがってオブジェクトのインスタンスを作成するために、ホスティング関係はゲストから整合性のあるホストに対して存在すべきである。

【０４８７】

例えば、ホスティング関係はＷｅｂｓｅｒｖｉｃｅオブジェクト定義とＩＩＳオブジェクト定義の間に存在することができる。この場合、ＭｙＷｅｂｓｅｒｖｉｃｅとＭｙＩＩＳがウェブサービスとＩＩＳを別々に実施すると仮定すると、関係は、ホスティング関係のマネージャを使用してシステムＭｙＷｅｂｓｅｒｖｉｃｅのインスタンスをシステムＭｙＩＩＳのインスタンス上に作成することが可能な場合があることを示している。システムと関係の両方に存在する制約条件を本発明者らが評価するまでは、関係を作成することが可能か否かは本発明者らには分からない。

【０４８８】

本発明者らがデータセンタにアプリケーションを配置する際、本発明者らはアプリケーション内のシステムに対するすべての顕著なホスティング関係を解決する必要がある。これを行うために、オペレータは、要求されたホスティング関係のそれぞれに対してホスティングメンバを作成することが必要である。オペレータのタスクを簡素化し、開発者が配置プロセスをガイドすることができるようにするために、開発者は代わりに具体的なホスティング関係を作成することができる。アプリケーションを配置する際にオペレータが単一のホスティングメンバを宣言するだけでよいような方法で、一組のホスティング関係メンバをグループ化するために具体的なホスティング関係が使用される。

【０４８９】

【表１１０】

```
A guest can be bound to a host iff
For each guestMember in Guest
There exists one or more hostMember in Host where
guestMember.Type has a hosting relation with hostMember.type
and guestMember.hostConstraints validate against hostMember.settings
and hostMember.guestConstraints validate against guestMember.settings
and for each member of guestMember there exists a binding to a member of hostMember
```

【０４９０】

例えば、次に示す具体的な関係は層３システム（Ｂｉｋｅ）を層２ホスト（ｏｐｅｒａｔｉｎｇＳｙｓｔｅｍ）と結合する。この場合、本発明者らは、ディフォルト値「ｓｙｓｔｅｍ　ｆｏｌｄｅｒ」でホスティング関係に対する設定を定義する。本発明者らは、この設定を、層３アプリケーションのシステムと層２ホストのシステムの間でホスティング関係を定義する３つのホスティングメンバのうちの１つにフローする。

【０４９１】

【表１１１】

```
<HostingDefinition Name="DefaultBikePlacement" GuestDefinition="Bike"
                   HostDefinition="OperatingSystem:OperatingSystem">
    <SettingDeclaration Name="fileLocationRelativeToRoot" Definition="xs:sting" Access="Readwrite" />
    <SettingValue Path="dirPath">systemFolder</SettingValue>
    <Flow Name="copyPath" Definition="copy">
        <Input Name="source" Path="dirPath"/>
        <Output Name="destination" Path="bikeExecutableHost.hostRelativePath"/>
    </Flow>

    <Hosting  Name="bikeExecutableHost" Definition="fileDirectoryHost" GuestMember="guest.bikeFile"
              HostMember="host.FileSystem"/>
    <Hosting Name="bikeEventKeyHost" Definition="registryKeyRegistryKeyHost"
GuestMember="guest.bikeEventKey"
              HostMember="host.Registry.ApplicationEventKey"/>
    <Hosting Name="bikeSoftwareKeyHost" Definition="registryKeyRegistryKeyHost"
              GuestMember="guest.bikeSoftwareKey" HostMember="host.Registry.HKLM"/>
</HostingDefinition>
<xs:complexType name="HostingDefinition">
    <xs:complexContent>
        <xs:extension base="RelationshipDefinition">
            <xs:sequence>
                <xs:element name="Hosting" type="HostingMember" minOccurs="0"
maxOccurs="unbounded"/>
```

【０４９２】

【表１１２】

```
            </xs:sequence>
            <xs:attribute name="GuestDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="HostDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="SimulationRoot" type="xs:boolean" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０４９３】

【表１１３】

属性/要素	説明
GuestDefinition	ゲストインスタンスの定義を示す。
HostDefinition	ホストインスタンスの定義を示す。
SimulationRoot	設計時チェックを行うために定義がSDM文書のコンパイル中にシミュレーションされたそのルートインスタンスを有するべきか否かを指定する。この属性が指定されていない場合、シミュレーション用のディフォルトコンパイラ規則が適用される。シミュレーションは首尾よくパスしないが定義の実際の処理は正常に機能する場合、この属性が必要とされる。

【０４９４】

ホスティング関係には次に示す抽象定義対の組合せが有効である。

【０４９５】

【表１１４】

ゲスト型	ホスト型
Endpoint	Endpoint
Resource	Resource
Resource	System
System	Resource
System	System

【０４９６】

３．４．３．４　包含関係

２つの抽象オブジェクトの間の包含関係は、ｐａｒｅｎｔＴｙｐｅに基づく具体的な型はｍｅｍｂｅｒＴｙｐｅに基づくメンバを含むことができるという事実を取り込む。包含は、親インスタンスがメンバインスタンスの存続期間を制御することができ、メンバインスタンスに動作を委任できることを示している。

【０４９７】

【表１１５】

```
<xs:complexType name="ContainmentDefinition">
    <xs:complexContent>
        <xs:extension base="RelationshipDefinition">
            <xs:sequence>
                <xs:element name="Containment" type="HostingMember" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="ParentDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="MemberDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０４９８】

【表１１６】

属性/要素	説明
ParentDefinition	メンバを含むことになるインスタンスの定義を示す。
MemberDefinition	含まれるメンバになるインスタンスの定義を示す。

【０４９９】
包含関係には次に示す抽象定義対の組合せが有効である。
【０５００】
【表１１７】

親型	メンバ型
System	Endpoint
System	Resource
System	System
Endpoint	Resource
Resource	Resource

【０５０１】
３．４．３．５　委任関係
外部システムから含まれるシステムに動作を転送するために委任が使用される。本発明者らがこれを行う方法は、外部システムのエンドポイントを内部システムのエンドポイントに委任する方法である。これは外部システムに方向付けられていたすべての相互作用を内部システムのエンドポイントに効果的に転送する。委任は繋げることができ、これによって内部システムはその動作を別のシステムにさらに委任することができる。
【０５０２】
委任関係は、委任に参加することのできる抽象エンドポイント定義の対を定義する。各関係は、プロキシおよびそれが動作を委任することのできる抽象エンドポイント定義として機能することのできる抽象エンドポイント定義を示す。
【０５０３】
【表１１８】

```
<xs:complexType name="DelegationDefinition">
    <xs:complexContent>
        <xs:extension base="RelationshipDefinition">
            <xs:sequence>
                <xs:element name="Delegation" type="HostingMember" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="ProxyDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="DelegateDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０５０４】
【表１１９】

属性/要素	説明
ProxyDefinition	その動作を内部エンドポイントに委任する外部エンドポイントの定義を示す。
DelegateDefinition	要求された動作を提供する内部エンドポイントの定義を示す。

【０５０５】
委任関係には次に示す抽象型対の組合せが有効である。
【０５０６】

【表１２０】

プロキシ型	代表型
Endpoint	Endpoint

【０５０７】

本発明者らは、資源およびシステム委任が層間の結合をサポートすることを可能にする。例えば、ＩＩＳがファイルシステムの一部を配置することを必要とせずにそのファイルシステムの一部を公開することを可能にすることを目的としている。

【０５０８】

３．４．３．６　レファレンス関係

本発明者らは、ホスティング関係従属関係に加えてインスタンスの間の強力な従属関係を取り込むためにレファレンス関係を使用する。配置中に構築の順番を制御するために、またインストールおよび更新中にシステム間でパラメータをフローするためにこれらの従属関係が使用される。レファレンス関係は強力な従属関係を示すので、本発明者らはレファレンス関係がシステムの境界を越えられるようにすることはできない。これは、あるシステム内の資源が別のシステム内の資源に対して従属関係を有することができないということを意味している。これでシステムは最早、配置の独立した単位ではなくなる。システム間に従属関係が存在する場合、本発明者らは通信関係を使用する。通信関係は、システムの再インストールを要求せずに経時的に変更することができる。

【０５０９】

【表１２１】

```
<xs:complexType name="ReferenceDefinition">
    <xs:complexContent>
        <xs:extension base="RelationshipDefinition">
            <xs:sequence>
                <xs:element name="Reference" type="HostingMember" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="DependentDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="SourceDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０５１０】

【表１２２】

属性/要素	説明
DependentDefinition	ソースインスタンスに依存するインスタンスの定義
SourceDefinition	ソースインスタンスの定義。従属関係に気付くためにこのインスタンスは要求されない。

【０５１１】

レファレンス関係には次に示す抽象型対の組合せが有効である。

【０５１２】

【表１２３】

従属型	ソース型
System	System
Resource	Resource
Resource	Endpoint
Endpoint	Resource
Endpoint	Endpoint

【０５１３】

３．４．３．７　暗黙の基礎関係

すべての抽象関係は、基礎関係定義のうちの１つを暗黙的に拡張する。これらの定義は、関係ツリーのそれぞれに対するルートを形成する。図１８は、関係ツリーの一例を示している。これを行うことによって、本発明者らは、制約定義内からのルート定義を参照することができ、また本発明者らはそのルート型から共通型制約条件を継承することができる。

【０５１４】

３．５　メンバ

３．５．１　メンバ

ランタイムに存在することのできる特定の定義のインスタンスを示すためにメンバが使用される。すべてのメンバは、当該メンバを含んでいる定義のスコープ内の当該一組のメンバ内で一意の名前によって示される。メンバは、それが参照する定義に対する設定を提供することができる。これは、設計表面特有データを含むこともできる。

【０５１５】

【表１２４】

```
<xs:complexType name="Member">
    <xs:sequence>
        <xs:element name="Description" type="Description" minOccurs="0"/>
        <xs:element name="DesignData" type="DesignData" minOccurs="0"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="SettingValue" type="SettingValue"/>
            <xs:element name="SettingValueList" type="SettingValueList"/>
        </xs:choice>
    </xs:sequence>
    <xs:attribute name="Name" type="SimpleName" use="required"/>
    <xs:attribute name="Definition" type="QualifiedName" use="required"/>
    <xs:attribute name="Visibility" type="VisibilityEnum" use="optional"/>
</xs:complexType>
```

【０５１６】

【表１２５】

属性/要素	説明
Description	メンバの記述
DesignData	メンバに関する設計表面特有情報
SettingValue	参照される型に関して書くことのできる設定に対応する設定に対する値。これらの値が固定とマーク付けされている場合、メンバに対してインスタンスが作成される際にこれらが使用されるべきである。これらが固定されていない場合、これらの値は配置またはフローされたパラメータによって上書きすることができる。
SettingValueList	参照される型に対する書くことのできるリスト設定に対する値のリスト。
Name	含んでいる型のスコープ内のメンバに対する一意の名前。
Definition	このメンバが参照する定義の名前。
Visibility	メンバが公開か秘密かを示すためにVisibility属性が使用される。秘密である場合、それは当該メンバを含んでいる定義内からしか参照することはできない。公開である場合、それは定義外から参照することができる。ディフォルトの動作は、定義のvisibility属性によって定義される。

【０５１７】

３．５．２　オブジェクトメンバ

オブジェクトメンバは、抽象または具体的なオブジェクト定義を参照すべきである。これらはインスタンスのアレイを表すことができる。この場合、これらはそのアレイに対する上限と下限を定義することができる。これらがレファレンスメンバである場合、オブジェクトをインスタンス生成しているユーザは、そのメンバに対するインスタンスを明示的に構築すべきである。これらがレファレンスメンバでない場合、ランタイムは外部オブジェクトが作成されるのと同時にインスタンスを作成する。

【０５１８】

【表１２６】

```
<xs:complexType name="ObjectMember">
    <xs:complexContent>
        <xs:extension base="Member">
            <xs:attribute name="MinOccurs" type="MinOccurs" use="optional"/>
            <xs:attribute name="MaxOccurs" type="MaxOccurs" use="optional"/>
            <xs:attribute name="Reference" type="xs:boolean" use="required"/>
        </xs:extension>
    </xs:complexContent>
```

【０５１９】

【表１２７】

```
</xs:complexType>
```

【０５２０】

【表１２８】

属性/要素	説明
MinOccurs	このメンバに関連付けられたいくつかのインスタンスの下限。これが0の場合、メンバは任意選択である。ディフォルトは1である。
MaxOccurs	このメンバに関連付けられたいくつかのインスタンスの上限。1またはそれ以上であるべきである。ディフォルトは1である。
Reference	この値が真の場合、当該メンバに関連付けられたインスタンスはオペレータによって明示的に作成されるか、または別の型で参照されるべきである。偽の場合、その型が作成される際にインスタンスが作成される。

【０５２１】

ＳＤＭモデルでは、本発明者らは、親が構築される際に作成され、親が破壊される際に破壊されるメンバを、その親とは別個に存続期間を有することのできるメンバと区別することが必要になる。本発明者らは、この目的でＩｓＲｅｆｅｒｅｎｃｅ属性を使用する。単なる類似は、インスタンスを作成するために「ｎｅｗ」が使用されるか否かに基づくスタックベースおよびヒープベースの構築を可能にするＣ＋＋宣言との類似である。メンバがＩｓＲｅｆｅｒｅｎｃｅとマーク付けされている場合、インスタンスを作成し、それをメンバに関連付けるには、オペレータ側で明示的な新しい動作が要求される。

【０５２２】

本発明者らがこれを行ういくつかの理由を以下に示す。

１．オペレータがシステムを構築する際、本発明者らはｉｓＲｅｆｅｒｅｎｃｅメンバを構築する機能しか公開しない。これはオペレータの作業を大幅に簡素化する。

２．ＳＤＭ文書を処理する際、本発明者らは、文書のインスタンス空間を具体的定義空間のそれと区別することのできる明白な境界を有する。

【０５２３】

３．５．３　関係メンバ

関係メンバは、オブジェクトメンバが作成される際にそのオブジェクトメンバ間に存在することになる関係を示す。関係インスタンスは、オペレータによって明示的に作成されるかまたはランタイムによって暗黙的に作成される。前者の例はインスタンス間のホスティング関係であり、後者はシステム間の通信関係である。

【０５２４】

【表１２９】

```
<xs:complexType name="RelationshipMember">
    <xs:complexContent>
        <xs:extension base="Member"/>
    </xs:complexContent>
</xs:complexType>
```

【０５２５】

３．５．４　エンドポイントメンバ

【０５２６】

【表１３０】

```
<xs:complexType name="EndpointMember">
    <xs:complexContent>
        <xs:extension base="ObjectMember"/>
    </xs:complexContent>
</xs:complexType>
```

【０５２７】

３．５．５　システムメンバ

【０５２８】

【表１３１】

```
<xs:complexType name="SystemMember">
    <xs:complexContent>
        <xs:extension base="ObjectMember"/>
    </xs:complexContent>
</xs:complexType>
```

【０５２９】

３．５．６　資源メンバ

【０５３０】

【表１３２】

```
<xs:complexType name="ResourceMember">
    <xs:complexContent>
        <xs:extension base="ObjectMember"/>
    </xs:complexContent>
</xs:complexType>
```

【０５３１】

３．５．７　ホスティングメンバ

２つのオブジェクトメンバ間のホスティング関係を宣言するためにホストメンバが使用される。オブジェクトメンバは、含んでいる定義の直接のメンバであっても、定義とメンバーシップ関係を有するネストされたメンバであってもよい。参照されるメンバと含んでいる定義の間のメンバーシップチェーンがあるべきである。

【０５３２】

【表１３３】

```
<xs:complexType name="HostingMember">
    <xs:complexContent>
        <xs:extension base="RelationshipMember">
            <xs:attribute name="GuestMember" type="Path" use="required"/>
            <xs:attribute name="HostMember" type="Path" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０５３３】

【表１３４】

属性/要素	説明
GuestMember	関係のゲストと整合性のある定義を有するメンバを示す。メンバをネストすることができる。
HostMember	関係のホストと整合性のある定義を有するメンバを示す。メンバをネストすることができる。

【０５３４】

３．５．８　通信メンバ

定義の隣接するシステムメンバのエンドポイントメンバ間の通信関係を宣言するために通信メンバが使用される。

【０５３５】

【表１３５】

```
<xs:complexType name="CommunicationMember">
    <xs:complexContent>
        <xs:extension base="RelationshipMember">
            <xs:attribute name="ClientMember" type="Path" use="required"/>
            <xs:attribute name="ServerMember" type="Path" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０５３６】

【表１３６】

属性/要素	説明
ClientMember	関係のクライアント定義と整合性のある定義を有するエンドポイントメンバを示す。エンドポイントメンバは、定義の隣接するシステムメンバのメンバであるべきである。
ServerMember	関係のサーバ定義と整合性のある定義を有するエンドポイントメンバを示す。エンドポイントメンバは、定義の隣接するシステムメンバのメンバであるべきである。

【０５３７】

３．５．９　包含メンバ

型メンバが型によって含まれることを宣言するために包含メンバが使用される。各型メンバは含まれていても、委任されていてもよい。包含メンバは、関係のこのポインタになるべき包含関係の親の値を自動的に設定する。

【０５３８】

【表１３７】

```
<xs:complexType name="ContainmentMember">
    <xs:complexContent>
        <xs:extension base="RelationshipMember">
            <xs:attribute name="ChildMember" type="Path" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０５３９】

【表１３８】

属性/要素	説明
ChildMember	親によって含まれることになる隣接するオブジェクトメンバを示す。

【０５４０】

３．５．１０　委任メンバ

外部型のエンドポイント定義メンバと外部型の隣接するシステムメンバのエンドポイント定義メンバとの間の委任関係を設定するために委任メンバが使用される。

【０５４１】

【表１３９】

```
<xs:complexType name="DelegationMember">
    <xs:complexContent>
        <xs:extension base="RelationshipMember">
            <xs:attribute name="ProxyMember" type="Path" use="required"/>
            <xs:attribute name="DelegateMember" type="Path" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０５４２】

【表１４０】

属性/要素	説明
ProxyMember	Proxyは、システムに対する包含関係を有しないシステムの隣接するエンドポイントメンバを示す。メンバの定義は、委任関係のプロキシの定義と一致すべきである。
DelegateMember	Delegateは、型の隣接するメンバのエンドポイントメンバを示す。エンドポイントメンバの型は委任関係の代表型と一致すべきである。

【０５４３】

３．５．１１　レファレンスメンバ

外部定義の２つの隣接するメンバまたはネストされたメンバの間のレファレンス関係を設定するためにレファレンスメンバが使用される。

【０５４４】

【表１４１】

```
<xs:complexType name="ReferenceMember">
    <xs:complexContent>
        <xs:extension base="RelationshipMember">
            <xs:attribute name="DependentMember" type="Path" use="required"/>
            <xs:attribute name="SourceMember" type="Path" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０５４５】

【表１４２】

属性/要素	説明
dependentMember	ソースメンバに依存するオブジェクトメンバ。レファレンス関係の従属オブジェクトの定義に対応すべきである。
sourceMember	ソースオブジェクトメンバ。レファレンス関係のソースオブジェクトの定義に対応すべきである。

【０５４６】

３．６　フロー

オブジェクト定義のメンバ間、および関係の参加者間でパラメータを渡すために設定フローが使用される。フローの一部として、ユーザは、設定値を結合または分離し、新しい設定値を計算するために変形を使用することができる。

【０５４７】

すべての設定フローメンバは、変形を実施するためにフロー定義を使用する。ｕｒｌを構文解析するフロー定義を次に示す。

【０５４８】
【表１４３】

```
<FlowDefinition Name="urlToComponents">
    <SettingDeclaration Name="url" Definition="url" Access="Writeonly"/>
    <SettingDeclaration Name="protocol" Definition="String" Access="Readonly"/>
    <SettingDeclaration Name="server" Definition="String" Access="Readonly"/>
    <SettingDeclaration Name="Path" Definition="String" Access="Readonly"/>
    <SettingDeclaration Name="file" Definition="String" Access="Readonly"/>
</FlowDefinition>
```

【０５４９】
次いでフローメンバは、オブジェクトまたは関係内で宣言される。フローメンバは、フロー定義に入力を提供し、次いで出力をフローからターゲット設定に出力を方向付ける。
【０５５０】
【表１４４】

```
<Flow Name="deconstructUrl" Definition="urlToComponents">
    <Input Name="url" Path="webservice.url"/>
    <Output Name="server" Path="webservice.server" Replace="false"/>
</Flow>
```

【０５５１】
３．６．１　フロー定義
本発明者らが一組の設定値に適用することを望む特定変形を定義するために本発明者らはフロー定義を使用する。フロー定義は、入力設定（書き込み専用設定）と出力設定（読み取り専用設定）、変形を定義するための入力インターフェースのような設計表面特有情報のためのＤｅｓｉｇｎＤａｔａセクション、およびＳＤＭファイルをブラウジングする際に使用するための説明を定義する設定スキーマを公開する。フロー定義は、それが定義されているネーム空間内の名前によって示される。この定義は、それがフローを評価する際にランタイムをサポートするマネージャも示す。
【０５５２】
本発明者らは、単純明快な変形が要求される場合にフロー要素の構築を簡素化するためにランタイムがいくつかの標準フロー定義を含むだろうと推測する。例は、コピー、マージ、およびストリングの置換を含むことができる。フロー定義をパラメータ化することができるので、本発明者らは、構成パラメータに基づいて異なるアクションを実行する１つまたは複数の簡素な変形があるだろうとも推測する。
【０５５３】
【表１４５】

```
<xs:complexType name="FlowDefinition">
    <xs:complexContent>
        <xs:extension base="Definition"/>
    </xs:complexContent>
</xs:complexType>
```

【０５５４】
３．６．２　フローメンバ
各フローメンバは、１つまたは複数の入力設定、１つまたは複数の宛先設定を示す。各フローメンバは固定された設定値を提供することができる。各フローメンバはフロー定義を示すべきである。フローが評価される際、ソースデータが入力から収集され、固定設定値と結合され、変形のためにフロー定義に渡される。定義からの出力は、フローメンバに列挙される出力に従って宛先設定に渡される。
【０５５５】

ソース値の１つが変更される時は何時でもフローの再評価がトリガされる。この理由から、本発明者らは値を変更させる回路の流れを防止する必要がある。値が一定に維持される場合、ループは中止する。ランタイムは、スタックの深さを追跡することによって無限ループを検出し中止する。

【０５５６】

別個のフローメンバからの出力がオブジェクトインスタンスの同じ設定メンバを目標とすることはエラーである。

【０５５７】

【表１４６】

```
<xs:complexType name="FlowMember">
    <xs:complexContent>
        <xs:extension base="Member">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="Input" type="SettingTarget"/>
                <xs:element name="Output" type="OutputPath"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０５５８】

【表１４７】

属性/要素	説明
Input	フローへの入力として使用される設定値への経路のリスト。各入力は、フロー定義に対する書き込み専用設定を示すべきである。
Output	フローの結果として設定される設定への経路のリスト。各出力は、フロー定義に対する読み取り専用設定を示すべきである。

【０５５９】

３．７　制約条件

定義のメンバの設定値または関係の参加者に対する制約を示すために制約条件が使用される。これらの制約は、設計時と開発時の両方でインスタンス空間内で評価される。

【０５６０】

すべての設定制約条件は、設定値を評価するために制約条件定義を使用する。制約条件定義は、それが含んでいる値を示すために設定宣言を使用する。次に示す制約条件定義は、２つの引数と１つの演算子を取り、制約条件を評価し、最後に成功またはエラーを戻す簡素な比較関数を実施する。

【０５６１】

【表１４８】

```
<ConstraintDefinition Name="SimpleOperatorComparison">
    <SettingDeclaration Name="LHS" Definition="Any" Access="Writeonly"/>
    <SettingDeclaration Name="operator" Definition="operator" Access="Writeonly"/>
    <SettingDeclaration Name="RHS" Definition="Any" Access="Writeonly"/>
</ConstraintDefinition>
```

【０５６２】

次いで制約条件メンバは、評価に制約条件型への値を提供するために使用される。

【０５６３】

【表１４９】

```
<Constraint Name="constraintSecurityMode" Definition="SimpleOperatorComparison">
        <SettingValue Path="operator">=</SettingValue>
        <SettingValue Path="RHS">basicAuth</SettingValue>
        <Input Name="LHS" Path="webservice.securityMode"/>
    </Constraint>
```

【０５６４】

３．７．１　制約条件定義

制約条件定義は、一組の入力値に対して機能する制約条件を定義する。制約条件は、カスタム動作を選択するか、またはその動作を定義するためにパラメータを使用する簡素な制約条件エンジンをサポートするために、パラメータ化することができる。本発明者らは、抽象オブジェクト間の周知の関係をサポートするために簡素なパラメータ値制約条件と一組の複雑な制約条件のために一組の標準制約条件定義が書かれるだろうと推測する。

【０５６５】

【表１５０】

```
<xs:complexType name="ConstraintDefinition">
    <xs:complexContent>
        <xs:extension base="Definition">
```

【０５６６】

【表１５１】

```
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="RelationshipConstraint" type="RelationshipConstraint"/>
                <xs:element name="RelationshipConstraintGroup" type="RelationshipConstraintGroup"/>
                <xs:element name="ObjectConstraint" type="ObjectConstraint"/>
                <xs:element name="ObjectConstraintGroup" type="ObjectConstraintGroup"/>
            </xs:choice>
            <xs:attribute name="TargetDefinition" type="QualifiedName" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０５６７】

【表１５２】

属性/要素	説明
TargetDefinition	この制約条件を適用することができるターゲット定義を示す。これが供給されていない場合、制約条件はいかなるオブジェクトまたは関係定義にも適用可能であると推定される。

【０５６８】

３．７．２　制約条件メンバ

制約条件メンバは、特定の制約条件定義に関して一組の入力値を示す。メンバは、設定に関して静的な値を示すことができ、また制約条件設定を経路に結合するために入力ステートメントを使用することができる。

【０５６９】

【表１５３】

```
<xs:complexType name="ConstraintMember">
    <xs:complexContent>
        <xs:extension base="Member">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="Input" type="Input"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０５７０】

【表１５４】

属性/要素	説明
input	制約条件への入力リスト。入力は、制約条件に渡されるソース設定値および結果として設定される制約条件設定への経路を示す。ソース設定定義と制約条件設定定義は整合すべきである。

【０５７１】

３．７．３　構造的制約条件

本発明者らは、特定の関係で使用される際に具体的な空間のトポロジを定義し、オブジェクトの設定を制約するためにオブジェクトおよび関係制約条件を使用する。

【０５７２】

例えば、抽象オブジェクト定義（Ａ）内で、本発明者らは、この抽象定義の実施態様が別の抽象オブジェクト定義（Ｂ）の１つのインスタンスを含むべきであると示したい場合がある。少なくとも１つの適切な包含関係が既に存在すると仮定すると、これを行うために、本発明者らは、次に示すようなＡ内の関係制約条件を使用する。

【０５７３】

【表１５５】

```
<RelationshipConstraint name="AContainsB"
                relationship="ContainmentDefinition"
                myRole="parent"
                targetType="B"
                minOccurs="1"
                maxOccurs="1"/>
```

【０５７４】

制約条件は、Ａの実施態様が親のロールを果たし、関係（メンバ）の他端の型が型Ｂである包含関係が存在すべきことを示す。本発明者らがＢの構成に関してさらに制御しようとする場合、本発明者らは次のように型Ｂの設定に対する制約条件を追加することができる。

【０５７５】

【表１５６】

```
<RelationshipConstraint Name="AContainsB" RelationshipDefinition="containementRelationship"
                        TargetRole="Member" TargetObjectDefinition="B" MinOccurs="1"
MaxOccurs="1">

    <Constraint Definition="simpleValueConstraint" Name="BValueConstraint">
        <SettingValue Path="operator">=</SettingValue>
        <SettingValue Path="RHS">myPort</SettingValue>
        <Input Path="member.Name" Name="LHS"/>
    </Constraint>
</RelationshipConstraint>
```

【０５７６】

この場合、本発明者らは、メンバの名前がストリング「ｍｙＰｏｒｔ」と等しくなることを要求した制約条件を追加した。

【０５７７】

本発明者らは、関係に制約条件を追加することもできる。本発明者らはこれらをオブジェクト制約条件と呼ぶ。関係内から、本発明者らは関係に参加するオブジェクトを制約する。関係のそれぞれのロールに関して、本発明者らはオブジェクト定義を示すことができ、次いで本発明者らはそれらのオブジェクト定義に設定制約条件を追加することができる。関係の観点から、カーディナリティは常にｍｉｎＯｃｃｕｒｓ＝１、およびｍａｘＯｃｃｕｒｓ＝１であり、したがってこれは制約条件宣言には表示されない。

【０５７８】

【表１５７】

```
<ObjectConstraint Name="allowedPair"
                  PrimaryRole="Host"
                  PrimaryObjectDefinition="IIS"
                  SecondaryRole="Guest"
                  SecondaryObjectDefinition="webApp"/>
```

【０５７９】

最後に、本発明者らは制約条件をネストすることができる。これは、本発明者らに制約条件をつなぐ機能、すなわち外部制約条件が内部制約条件に対するコンテキストを設定する機能を提供する。特定型のエンドポイントだけを含むｗｅｂＡｐｐを制約するｗｅｂａｐｐシステムをホストするＩＩＳシステムの一例を次に示す。

【０５８０】

この場合、本発明者らは、少なくとも１つが真であるべきである一組の可能性を指定するために一群のオブジェクト制約条件を使用する。

【０５８１】

【表１５８】

```
<SystemDefinition Name="test">
    <RelationshipConstraint Name="WebAppHostConstraint"
                            RelationshipDefinition="HostingDefinition"
                            TargetRole="Guest"
                            TargetObjectDefinition="webApp">
        <RelationshipConstraint Name="WebAppContainsPort1"
                                RelationshipDefinition="containmentRelationship"
                                TargetRole="Member"
                                TargetObjectDefinition="EndpointDefinition"
                                MinOccurs="1"
                                MaxOccurs="unbounded">
            <ObjectConstraintGroup>
                <ObjectConstraint Name="hasWebPort" PrimaryRole="Member"
PrimaryObjectDefinition="webPort"/>
                <ObjectConstraint Name="hasSqlPort" PrimaryRole="Member"
PrimaryObjectDefinition="sqlPort"/>
            </ObjectConstraintGroup>
        </RelationshipConstraint>
    </RelationshipConstraint>
```

【０５８２】

このネストされた制約条件は、本発明者らが外部から評価することができる経路を形成する。この経路に対する各制約条件は、現在のインスタンスと同様、当該経路に対する以前のインスタンスの設定にもアクセスすることができる。ネストされた制約条件の評価は、特定されたシステム内で制約条件が定義されているかのように実行される。

【０５８３】

ｆｏｏの観点から、次に示す２つのシナリオは等価であるべきである。第１のｆｏｏは、制約されたシステムｂａｒに対してネストされた制約条件を課し、第２のｆｏｏでは、型ｂａｒは既にその制約条件を含んでいる。

【０５８４】

シナリオ１：

【０５８５】

【表１５９】

```
< SystemDefinition name="foo">
    <RelationshipConstraint Name="containsBar"
                            RelationshipDefinition="containment"
                            TargetRole="Member"
                            TargetObjectDefinition="bar"
                            MinOccurs="1">
        <RelationshipConstraint Name="containsX"
                                RelationshipDefinition="containment"
                                TargetRole="Member"
                                TargetObjectDefinition="X"
                                MinOccurs="1"/>

    </ RelationshipConstraint >
</ SystemDefinition >

< SystemDefinition name="bar"/>
```

【０５８６】

シナリオ２：

【０５８７】

【表１６０】

```
<SystemDefinition Name="foo">
    <RelationshipConstraint Name="containsBar"
                            RelationshipDefinition="containment"
                            TargetRole="Member"
                            TargetObjectDefinition="bar"
                            MinOccurs="1"/>
</SystemDefinition>

<SystemDefinition Name="bar">
    <RelationshipConstraint Name="containsX"
                            RelationshipDefinition="containment"
                            TargetRole="Member"
                            TargetObjectDefinition="X"
                            MinOccurs="1"/>
</SystemDefinition>
```

【０５８８】

３．７．３．１　制約条件モデル

制約条件モデルには、ガードと述部という２つの部分がある。本発明者らは、述部を実行するコンテキストを定義するためにガードを使用する。例えば、関係内で、本発明者らは、述部を実行することを本発明者らが希望する型の特定の組合せを示すためにガードを使用する。オブジェクト内で、本発明者らは、他のオブジェクトに対して一組の関係を示すためにガードを使用する。

【０５８９】

次いで述部は、そのガードの要件が満たされている場合に実行される。本発明者らは、設定値を検証する設定制約条件と、一組の制約条件を検証するグループ制約条件という２つの形式の述部を有する。

【０５９０】

本発明者らは、ガード内にガードをネストすることができる。この場合、内部ガードは外部ガードが満たされた場合にのみチェックされる。これによって、本発明者らは関係構造の検証をサポートする経路を構築することができる。

【０５９１】

ガードとその述部の組合せは、ガードが一致し、述部が真と評価する回数を示すカーディナリティを有することができる。

【０５９２】

より正式には、

【０５９３】

【表１６１】

```
        Guard :== ObjectConstraint(ObjectDefintion,
  ObjectDefintion ,required)
                    { (Guard | predicate) * } |

                  RelationshipConstraint(RelationshipDefinition,
                                         TargetObject,lBound,uBound)
                    { (Guard | predicate) * }
```

【０５９４】

である。

【０５９５】

ガードは、ＯｂｊｅｃｔＣｏｎｓｔｒａｉｎｔまたはＲｅｌａｔｉｏｎｓｈｉｐＣｏｎｓｔｒａｉｎｔと定義される。オブジェクト制約条件は、関係のどちらか一方の端部に関連付けられた２つのオブジェクト定義を示す。関係制約条件は、関係定義とターゲットオブジェクト定義とを示す。オブジェクト制約条件は任意選択であっても、関係制約条件が

下限と上限を有する場合に要求されてもよい。カーディナリティのこのような違いは、型は複数の関係に参加することができるのに対し、関係は２つの型しか示せないという事実を反映している。

【０５９６】

【表１６２】

```
Predicate :== SettingsConstraint(rule)  | group{ (guard)* }
```

【０５９７】

述部は、規則を含んでいる設定制約条件かまたは一組のガードを含んでいるグループである。述部は、ガードのコンテキストで評価される。設定制約条件の場合、述部は、ルートガードの所有者とそれぞれのネストされたガードが示すコンテキストとから設定を示すことができる。少なくとも１つが一致し、真と評価する一組のガードを示すためにグループが使用される。

【０５９８】

例：

１．

【０５９９】

【表１６３】

```
RelationshipConstraint(containmentRelationship,webapp,0,1){
}
```

【０６００】

この例は、ｗｅｂａｐｐに対して制約条件関係がある時は何時も真と評価するガードを示している。このガードは、多くて１回しか真と評価することはできない。それ以上一致した場合はユーザにエラーが戻されることになる。

【０６０１】

２．

【０６０２】

【表１６４】

```
RelationshipConstraint(containmentRelationship,webapp,0,1)
{
   SettingsConstraint(webapp.name=2)
}
```

【０６０３】

この例は、ガードに述部を追加する。ガードは、関係とターゲット定義が一致し、設定制約条件が真と評価する場合にのみ真と評価する。関係とターゲット定義が一致し、設定制約条件が真でない場合、ユーザにエラーが戻される。関係とターゲット型が一致し、設定制約条件が複数回真と評価する場合、ユーザにエラーが戻される。

【０６０４】

３．

【０６０５】

【表１６５】

```
RelationshipConstraint(containmentRelationship,webapp,0,1)
{
   RelationshipConstraint(containmentRelationship,vdir,0,1)
}
```

【０６０６】
この例で、本発明者らはガード内にガードをネストする。外部ガードが真の場合（制約条件を含んでいる型がｗｅｂａｐｐも含んでいる）、本発明者らは外部ガードのコンテキスト内の内部ガードを評価する。これは、内部関係制約条件がｗｅｂａｐｐインスタンスのコンテキストで評価されることを意味している。ｗｅｂＡｐｐが０または１のｖｄｉｒを含んでいる場合、内部制約条件は真を戻し、複数のｖｄｉｒを含んでいる場合、制約条件はユーザにエラーを戻す。
【０６０７】
４．
【０６０８】
【表１６６】

```
ObjectConstraint(webapp,iis,0,1)
{

RelationshipConstraint(containmentRelationship,systemType,0,1)
   {
         ObjectConstraint(webapp,vdir,0,1)
   }
}
```

【０６０９】
オブジェクト制約条件のコンテキストは主要オブジェクト定義（第１のオブジェクト定義）である。これは、関係制約条件がｗｅｂａｐｐのコンテキストで評価されることを意味している。関係制約条件は、第１はオブジェクト制約条件に対するコンテキストとなる関係であり、第２は関係制約条件に対するコンテキストであるターゲットオブジェクト定義である、２つの実現可能なコンテキストを定義する。
【０６１０】
５．
【０６１１】
【表１６７】

```
RelationshipConstraint(containmentRelationship,webapp,0,1)
{
   group
   {
      RelationshipConstraint(containmentRelationship,vdir,0,1)

RelationshipConstraint(containmentRelationship,directory,0,1)
   }
}
```

【０６１２】
この例で、本発明者らは、Ｗｅｂａｐｐのコンテキストでどちらも評価されることになる２つの関係制約条件を含めるために１つのグループを使用する。これら関係の少なくとも１つが発火して真を戻さない限り、このグループはエラーを生じる。この場合、ＷｅｂａｐｐはＶｄｉｒまたはディレクトリを含めるべきである。
【０６１３】
３．７．３．２　基礎制約条件
すべての構造的制約条件は、ＳｔｒｕｃｔｕｒａｌＣｏｎｓｔｒａｉｎｔと呼ばれる共通の基礎制約条件から導出される。各構造的制約条件は、記述要素と設計データの両方を含むことができる。要求された制約条件が失敗した場合に記述要素はエラーメッセージと

して戻される。各制約条件は、名前と、その制約条件が要求されているかまたはより大きな制約条件の一部を構成する単なるテストであるかを示すフラグとを有する。最後に、制約条件が何時評価されるべきかを示すために制約条件の評価要素を使用することができる。制約条件評価に関する更なる情報については節３．７．３．７を参照されたい。

【０６１４】

【表１６８】

```
<xs:complexType name="StructuralConstraint">
    <xs:sequence>
        <xs:element name="Description" type="Description" minOccurs="0"/>
        <xs:element name="DesignData" type="DesignData" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="Name" type="SimpleName" use="required"/>
    <xs:attribute name="Evaluate" type="ConstraintEvaluation" use="optional"/>
    <xs:attribute name="Required" type="xs:boolean" use="optional"/>
</xs:complexType>
```

【０６１５】

【表１６９】

属性/要素	説明
Name	この制約条件の名前
DesignData	この制約条件に関する設計表面特有情報
Description	制約条件のテキスト記述
Evaluation	制約条件を何時評価するかに関する情報。ディフォルトは設計時である。
Required	この制約条件が失敗した場合にユーザにメッセージが戻されるか否かを示すフラグ。ディフォルトは真である。

【０６１６】

３．７．３．３　オブジェクト制約条件

オブジェクト制約条件は、関係のロールの一方または両方に対する制約条件を記述する。この制約条件は、制約条件が失敗した場合にその制約条件の特定を支援するために名前を有する。オブジェクトは、ネストされた関係制約条件、または関係に参加しているインスタンスとロールが一致する場合に評価されることになる制約条件メンバを含むことができる。

【０６１７】

制約条件は次のように評価される。

ａ）主要なロールと主要な定義をオブジェクトインスタンスと照合し、提供されているならば副次的なロールと副次的な定義をオブジェクトインスタンスと照合する。一致しない場合、一致＝０と設定してｃ）に進む。

ｂ）主要オブジェクトインスタンスのコンテキストのすべてのネストされた制約条件を評価する。これらすべてが真と評価する場合は、一致＝１と設定し、そうでない場合は、一致＝０と設定する。

ｃ）一致＞ＭｉｎＯｃｃｕｒｓ、かつ一致＜ＭａｘＯｃｃｕｒｓの場合は、結果＝真と設定し、そうでない場合は、結果＝偽と設定する。

ｄ）Ｒｅｑｕｉｒｅｄが真であり、結果が偽である場合、メッセージをユーザに戻す。

ｅ）結果を親コンテキストに戻す。

【０６１８】

【表１７０】

```
<xs:complexType name="ObjectConstraint">
    <xs:complexContent>
        <xs:extension base="StructuralConstraint">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="Constraint" type="ConstraintMember"/>
                <xs:element name="RelationshipConstraint" type="RelationshipConstraint"/>
                <xs:element name="RelationshipConstraintGroup" type="RelationshipConstraintGroup"/>
            </xs:choice>
            <xs:attribute name="PrimaryRole" type="RolesList" use="required"/>
```

【０６１９】

【表１７１】

```
            <xs:attribute name="PrimaryObjectDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="SecondaryRole" type="RolesList" use="optional"/>
            <xs:attribute name="SecondaryObjectDefinition" type="QualifiedName" use="optional"/>
            <xs:attribute name="MinOccurs" type="MinOccurs" use="optional"/>
            <xs:attribute name="MaxOccurs" type="MaxOccurs" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０６２０】

【表１７２】

属性/要素	説明
Constraint	この制約条件にネストされている制約条件メンバのリスト。
RelationshipConstraint	ネストされた関係制約条件。関係は、主要なロールに関連付けられた定義で宣言されているかのうように評価される。
RelationshipConstraintGroup	ネストされた関係グループ。このグループ内の関係は、主要なロールに関連付けられている定義で宣言されているかのように評価される。
PrimaryRole	この制約条件が目標とする関係でのロールの名前。
PrimaryObjectDefinition	主要なロールに関連付けられたオブジェクト定義の名前。
SecondaryRole	この制約条件が目標とする関係での他のロールの名前。（任意選択）
SecondaryObjectDefinition	関係での副次的なロールに関連付けられたオブジェクト定義の名前。副次的なロールが指定されている場合にこれが要求される。
MaxOccurs	関係でこの制約条件がオブジェクトと一致することのできる最大回数。
MinOccurs	関係でこの制約条件がオブジェクトと一致することのできる最小回数。
Required	Requiredが真ならば、偽と評価した場合に制約条件はエラーを生成する。ディフォルトは真である。

【０６２１】
３．７．３．４　オブジェクト制約条件グループ
オブジェクト制約条件グループは、少なくとも１つの意味論を使用して複数組のオブジェクト制約条件を評価することができるようにその複数組のオブジェクト制約条件を１つにグループ化することを可能にする。グループの制約条件の少なくとも１つが真と評価しない限り、このグループは偽を戻す。制約条件グループは次のように評価される。
ａ）それぞれのネストされた制約条件を順次評価する。少なくとも１つが真と評価した場合は、結果＝真と設定し、そうでない場合は、結果＝偽と設定する。
ｂ）結果＝偽であり、かつＲｅｑｕｉｒｅｄ＝真である場合、エラーを発生する。
ｃ）結果を親コンテキストに戻す。

【０６２２】
【表１７３】

```
<xs:complexType name="ObjectConstraintGroup">
    <xs:complexContent>
        <xs:extension base="Constraint">
            <xs:sequence>
                <xs:element name="ObjectConstraint" type="ObjectConstraint" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０６２３】
【表１７４】

属性/要素	説明
ObjectConstraint	グループ内にネストされたオブジェクト制約条件のリスト。

【０６２４】
３．７．３．５　関係制約条件
オブジェクトが参加することのできる関係を制約するために関係制約条件が使用される。関係制約条件は関係定義を示し、任意選択で関係の他端のインスタンスのオブジェクト定義と関係のカーディナリティとを示す。制約条件には、エラーメッセージで識別されることができるよう名前が与えられる。関係制約条件の本体は、この制約条件をさらに改良するネストされた制約条件を制約する。

【０６２５】
関係制約条件はいくつかの目的で使用することができる。すなわち、追加の述部なしに単にカーディナリティを使用して、インスタンスが正常に動作するよう提供されるべき関係を示すために関係制約条件を使用することができ、述部を使用して、このオブジェクトが相互作用を意図するインスタンスに関する当該一組の構成を狭めるために関係制約条件を使用することができる。

【０６２６】
関係制約条件は次のように評価される。
ａ）一致＝０と設定する。
ｂ）オブジェクトインスタンスが参加する各関係インスタンスに対して、
ａ．関係定義を関係インスタンスの定義と照合し、関係の方向をターゲットのロールが示す方向と照合し、提供されているならばターゲットオブジェクト定義を関係の他端のインスタンスの定義と照合する。これらが一致しない場合、次の関係に進む。
ｂ．関係インスタンスのコンテキストのすべてのネストされた制約条件を評価する。これらすべてが真と評価する場合、一致＝一致＋１と設定する。
ｃ）一致＞ＭｉｎＯｃｃｕｒｓ、かつ一致＜ＭａｘＯｃｃｕｒｓの場合は、結果＝真と設定し、そうでない場合は、結果＝偽と設定する。
ｄ）Ｒｅｑｕｉｒｅｄが真であり、結果が偽である場合、メッセージをユーザに戻す。

ｅ）結果を親コンテキストに戻す。

【０６２７】

【表１７５】

```
<xs:complexType name="RelationshipConstraint">
    <xs:complexContent>
        <xs:extension base="StructuralConstraint">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="Constraint" type="ConstraintMember"/>
                <xs:element name="RelationshipConstraint" type="RelationshipConstraint"/>
                <xs:element name="RelationshipConstraintGroup" type="RelationshipConstraintGroup"/>
                <xs:element name="ObjectConstraint" type="ObjectConstraint"/>
                <xs:element name="ObjectConstraintGroup" type="ObjectConstraintGroup"/>
            </xs:choice>
            <xs:attribute name="RelationshipDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="TargetRole" type="RolesList" use="required"/>
            <xs:attribute name="TargetObjectDefinition" type="QualifiedName" use="optional"/>
            <xs:attribute name="MinOccurs" type="MinOccurs" use="optional"/>
            <xs:attribute name="MaxOccurs" type="MaxOccurs" use="optional"/>
            <xs:attribute name="DelegationAware" type="xs:boolean" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０６２８】

【表１７６】

属性/要素	説明
Constraint	関係内の設定値または関係の他端のオブジェクトに対する制約条件。
RelationshipConstraint	この要素は、ターゲットオブジェクトインスタンスのコンテキストで評価される関係制約条件を定義する。
RelationshipConstraint Group	この要素は、ターゲットオブジェクトインスタンスのコンテキストで評価される関係グループを定義する。
ObjectConstraint	この要素は、照合した関係インスタンスのコンテキストで評価されるオブジェクト制約条件を定義する。
Obj ectConstraintGroup	この要素は、照合した関係インスタンスのコンテキストで評価されるオブジェクト制約条件グループを定義する。
Name	制約する定義のスコープ内の制約条件に対する一意の名前。
RelationshipDefinition	照合されることになる関係の定義。
TargetRole	関係の方向を示すためにターゲットロールが使用される。関係の他端のインスタンスが参加するロールの名前である。
TargetObjectDefinition	関係の反対側にあるオブジェクトインスタンスの定義の任意選択の名前
MaxOccurs	この制約条件を照合できる関係インスタンスの最大数。
MinOccurs	この制約条件を照合できる関係インスタンスの最小数。
Required	Requiredが真ならば、偽と評価した場合に制約条件はエラーを生成する。ディフォルトは真である。
DelegationAware	制約は委任を意識していることを示す。ディフォルトは偽である。さらなる情報については節3.8を参照されたい。

【０６２９】

３．７．３．６　関係制約条件グループ

関係制約条件グループは、少なくとも１つの意味論を使用して複数組の関係制約条件を述部として評価することができるようにその複数組の関係制約条件を１つにグループ化することを可能にする。制約条件グループは次のようにグループ化される。

ｄ）それぞれのネストされた制約条件を順次評価する。少なくとも１つが真と評価した場合は、結果＝真と設定し、そうでない場合は、結果＝偽と設定する。

【０６３０】

ｅ）結果＝偽、かつＲｅｑｕｉｒｅｄ＝真である場合、エラーを発生する。

【０６３１】

ｆ）結果を親コンテキストに戻す。

【０６３２】

【表１７７】

```
<xs:complexType name="RelationshipConstraintGroup">
    <xs:complexContent>
        <xs:extension base="StructuralConstraint">
            <xs:sequence>
                <xs:element name="RelationshipConstraint" type="RelationshipConstraint"
maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

【０６３３】

【表１７８】

属性/要素	説明
RelationshipConstraint	グループ内にネストされた関係制約条件

【０６３４】

３．７．３．７　制約条件評価

制約条件を評価することができる３つの別個の時点がある。すなわち、設計プロセス中、アプリケーション配置中、およびオペレータがアプリケーションの構成のチェックまたは更新を希望した任意の時点で一度アプリケーションが配置された時である。

【０６３５】

制約条件が評価される時点は、制約条件に使用可能な一組の情報を変更する。設計時に評価される制約条件は、配置中にのみ供給される値または配置後にのみ使用可能な値に依存することはできない。配置時に評価される制約条件は、配置後にのみ使用可能な値に依存することはできない。配置後に評価される制約条件は、アプリケーションのいかなる設定にもアクセスすることができる。

【０６３６】

本発明者らは、制約条件が実行されるべき時点に応じて制約条件開発者がその制約条件にマーク付けすることを可能にする。入力が使用可能でない時に本発明者らが制約条件を実行する場合、発生することになるエラーを回避するためにこの情報が使用される。次いでコンパイラは、配置または検証まで使用可能でないと認識している値に設計制約条件が依存しないことをチェックすることができ、同様に、配置後まで使用可能でないと認識している値に配置制約条件が依存しないことをチェックすることができる。設定が使用可能な時点に関する知識をコンパイラが完成した場合、制約条件を実行することができる時点を計算することができるが、残念ながらこれは実現する可能性が低い。値を設定に提供することができる時点は、例えば予め知ることができるか、または一意の識別子を生成したアルゴリズムを用いて配置中に作成することのできるディレクトリ名を変更することができる。

【０６３７】

制約条件は次のようにマーク付けすることができる。

ａ）設計－設計時、配置時、または検証中に実行することができることを示す。
ｂ）配置－配置時または検証中に実行することができることを示す。
ｃ）検証－検証中にのみ実行することができることを示す。または
ｄ）なし－実行されないことを示す。

【０６３８】

これらの選択肢を取り込むために次に示す一覧表が使用される。

【０６３９】

【表１７９】

```
<xs:simpleType name="ConstraintEvaluation">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Design"/>
        <xs:enumeration value="Deployment"/>
        <xs:enumeration value="Validate"/>
        <xs:enumeration value="Never"/>
    </xs:restriction>
</xs:simpleType>
```

【０６４０】

構造的制約条件と制約条件メンバの両方に対して制約条件の評価動作を設定することができる。どちらの場合も、すべてのネストされた制約条件に構造的制約条件または制約条件メンバを適用する。

【０６４１】

ディフォルトの制約条件は設計時制約条件とマーク付けされており、何時でも実行可能である。

【０６４２】

３．８　委任

委任の目的は、モデルの開発者が、ホストまたは含まれた構成要素からゲストまたは親に特有の動作を提示できるようにすることである。これによって開発者は、既存の実施態様を再利用し、そのクライアントにより包括的な実施態様を提供する構成要素を設計することができる。図１９は、ホストの実施態様をゲストに公開するために委任を使用する一例を示している。

【０６４３】

図１９で、アプリケーションプラットフォーム構成要素は、ファイルシステムをそのホストであるオペレーティングシステム構成要素からそのゲストであるアプリケーションに公開するために委任関係を使用する。これを行うことにより、アプリケーションプラットフォームは、そのゲストにとってオペレーティングシステムのどの部分が使用可能かを、それらを直接実施することを必要とせずに制御することができる。

【０６４４】

図２０は、メンバの実施態様を親に公開するために委任を使用する一例を示している。図２０で、クライアントアプリケーションは予約サービスのエンドポイントを発券アプリケーションに公開する。これは、発券アプリケーションが、その時点でエンドポイントをＳｅｒｖｅｒアプリケーションに接続することを可能とするが、Ｓｅｒｖｅｒアプリケーションはそれ自体のエンドポイントをＵｓｅｒＤａｔａＳアプリケーションに委任する。

【０６４５】

３．８．１　原理

次に示す原理は委任モデルの設計をガイドする。

・プロキシはターゲットシステムに存在しない。プロキシはモデルにとって便利なものであり、物理的表現を有しない。これが真でない場合、委任の使用はターゲットシステムに対して悪影響を有することになる。

・プロキシは、代表が変更をサポートしない限り代表の動作を変更することはできない。プロキシはターゲットシステムに存在しないので、プロキシはモデル化された現実でな

いシステム動作しか変更することはできない。代表がその動作に影響を与える設定を公開する場合、他の関係と同様に、委任関係もフローによってそれらの設定を変更することができる。

・外部ユーザの観点から、ユーザは、特定のインスタンスがプロキシかまたは現実の実施態様かを知る必要があるとされるべきではない。これが真でない場合、制約条件およびフローの著者は、複雑かつ脆弱なコードを生じるどちらの場合もカバーする必要がある。

・ユーザは、制約条件表現の一部としてプロキシを明示的に目標とすることができるべきである。これは、構成要素の境界を経て動作の公開を制御する制約条件を実施するために必要である。

【０６４６】

３．８．２　シナリオ

システムの代表の使用に関連する問題を示す２つのシナリオを次に示す。

【０６４７】

３．８．２．１　通信接続性の検証

クライアントポートがサーバポートに接続できることを検証するために、通信関係は、サーバまでウォークし、次いでそのサーバのホストまでウォークし、その後、層３で通信関係に沿って戻り、さらに層４のクライアントまでホスティング関係を上がり、最後に通信関係に戻る制約条件を含んでいる。これを図２１に示す。

【０６４８】

委任関係が、包含境界が存在することにより両方の層でクライアントとサーバの間に存在する可能性がある。この一例を図２２に示す。この経路に従う制約条件を書くために、ユーザはその時点で任意の数のそれらの関係を処理することができるが、これは制約条件の記述を複雑化し、制約条件言語の利便性を低下させる。

【０６４９】

これを防ぐために、本発明者らは、プロキシがあるか否かと、そのプロキシがいくつあるかを予め認識せずにユーザが制約条件を書けるようにする必要がある。

【０６５０】

３．８．２．２　ゾーン境界

データセンタで、システムを異なるセキュリティ要件または動作と区別するためにゾーンがしばしば使用される。通信関係がゾーン境界を横断する場合、ユーザはその関係の動作に対して制約条件を課そうとする場合がある。例えば、ユーザは、バックエンドサーバに対してあるポートを介してｈｔｔｐ通信を可能にすることだけを望む場合がある。これを行う場合、ユーザは、ゾーン境界を越える際に通信関係を示すことができるべきである。プロキシはユーザがこれを行うことのできる唯一の方法である。図２３は、ゾーン境界の一例を示している。

【０６５１】

図２３で、開発者がゾーン１からゾーン２への発信する通信を制約しようとする場合、開発者は、ゾーン境界に委任されたエンドポイントを制約することによってこれを行う。例えば、開発者は、着信ｈｔｔｐと発信ｓｑｌ通信フローだけを可能にする制約条件を書く場合がある。この制約条件はゾーンが公開する各プロキシエンドポイントに対して発火する。

【０６５２】

プロキシを制約するために、プロキシは制約条件に対して可視的であるべきである。これは、制約条件が、動作を実施する代表とは別にプロキシが参加する関係を目標とすることができるべきであることを意味している。いくつかのシナリオではプロキシはユーザに見えないので、これは以前のシナリオに予想される動作を複雑化する。

【０６５３】

３．８．３　モデルの他の部分との相互作用

代表に対する規則を図２４に示すインスタンス空間例を使用して示す。

【０６５４】

３．８．３．１　プロキシおよび代表

プロキシは、代表と同じかまたは代表の基礎クラスであるべきである。図２４の例で、プロキシとして機能するｂは、代表であるｃと同じかまたはその基礎定義であるべきである。プロキシｂはｃの動作しか公開することはできないので、これが要求される。これらの型が異なってよい場合は、ｂの公開された動作はｃの動作と異なってよいが、要求された動作を実施することができるｂの物理的表現はない。

【０６５５】

インスタンス空間で、プロキシインスタンスは１つしか代表を有することはできない。すなわち、ｂが関係のプロキシとして機能する場合、ｂからの委任関係は１つだけあってよい。この制約条件がないと代表から設定値またはメンバの集合を戻すことができず、ランタイムはその値に対するソースとしてどの代表を選ぶべきかを知らないので、この制約条件が要求される。

【０６５６】

３．８．３．２　関係のロール

プロキシは次に示す関係のロールに参加することができる。

【０６５７】

【表１８０】

ホスティング (Hosting)	ホスト	参加	ホスティング関係がプロキシを目標とする場合、ゲストは代表に経路変更される。ゲストが代表で直接ホストされたような結果が得られる。
	ゲスト	不参加	これは禁止である。モデル化されたシステムにプロキシが存在したか、またはプロキシの構成に基づいて代表が配置されたことを示す。
包含 (containment)	親	不参加	プロキシは包含関係の親として参加することはできない。参加すると、許可されていない代表のメンバが変更される。
	メンバ	参加	プロキシはアプリケーションの包含構造に存在し、したがって包含関係のメンバとして参加することができるべきである。
委任 (Delegation)	プロキシ	参加	要求を代表に転送可能とするために、プロキシは委任関係に参加すべきである。
	代表	不参加	プロキシは別の委任関係に対する代表として機能することができる。これは、委任をつなぐことを可能にし、包含の複数の層を介してインスタンスを公開する。
レファレンス (Reference)	ソース	参加	プロキシは、レファレンス関係に対するソースとして機能することができる。ソースに対する要求は代表に宛先変更される。
	従属	不参加	代表がレファレンスを要求し、したがって代表の動作を変更することを示す場合があるので、プロキシはレファレンス関係で従属として機能することはできない。
通信 (communication)	クライアント	参加	本発明者らはコンテナを介してエンドポイントを公開するために委任を使用するので、プロキシはクライアントとして機能可能であるべきである。
	サーバ	参加	本発明者らはコンテナを介してエンドポイントを公開するために委任を使用するので、プロキシはサーバとして機能可能であるべきである。

【０６５８】

３．８．３．３　メンバのフィルタリングおよび経路の解決

プロキシは、プロキシの定義の一部と定義された代表からメンバを公開するだけである。プロキシと代表の定義が同一の場合、代表のすべてのメンバはプロキシによって公開されることになる。プロキシ定義が代表の定義の基礎定義である場合、プロキシは両方の定義に共通のメンバを公開するだけである。

【０６５９】

図２４のＢとＣに対して次に示す定義を仮定する。

【０６６０】

【表１８１】

```
<Definition Name="B">
        <SettingDeclaration Name="height" Definition="Int"/>
</Definition>

<Definition Name="C" Extends="B">
        <SettingDeclaration Name="weight" Definition="Int"/>
</Definition>
```

【０６６１】
この場合、プロキシｂは高さ設定を公開するだけであり、重さ設定は公開しない。
【０６６２】
設定値がプロキシで設定される場合、委任関係を介してアクションが代表に転送される。新しい設定値が代表に直接記録され、その代表を目標とするいかなる他のプロキシにも可視的となる。
【０６６３】
設定値がプロキシから取り出される場合、要求が代表に転送され、設定値は代表から直接的に戻される。
【０６６４】
オブジェクトおよび関係メンバのフィルタリングは、設定のフィルタリングと同じ方法で動作する。プロキシの定義の一部として宣言された共通メンバだけがプロキシによって公開される。メンバへの変更は代表のメンバに直接影響を与え、その代表を目標とするすべてのプロキシに対して可視的となる。
【０６６５】
メンバ経路がプロキシのメンバを参照する場合、その経路は代表のメンバに自動的に解決される。これは、メンバがプロキシか代表かによって経路のシンタックスは変化しないということを意味している。
【０６６６】
３．８．３．４　関係のフィルタリング
各オブジェクトインスタンスは一組の関係に参加する。これらの関係は、制約条件とフローの両方を評価するために使用され、またインスタンス生成のプロセスを制御するためにも使用される。プロキシと代表の両方は、プロキシと代表の両方の一貫したピクチャをユーザに提示するために１つに結合されるべき複数組の関係に参加する。
【０６６７】
プロキシから、本発明者らは、プロキシの関係と組み合わせて代表の関係の部分集合を公開することを必要とする。本発明者らが実施態様を隠している場合、プロキシは独自の包含関係と、可能であれば代表の関係の小さな部分集合だけをユーザに戻す必要がある。プロキシが完全に透明であることを本発明者らが望む場合、本発明者らはすべての代表の関係を公開し、プロキシの関係を隠す。
【０６６８】
代表から、本発明者らは、そのプロキシが参加している関係の部分集合と組み合わせてその直接的な関係を公開することを必要とする。すなわち、プロキシが別のエンドポイントと通信関係を有する場合、代表がこの関係に直接参加しているように表示されるべきである。一方、本発明者らは、プロキシの包含関係を代表の一部として公開することを望まない。制約条件に対するロールアップ動作は節３．８．３．６で説明する。
【０６６９】
３．８．３．４．１　プロキシ識別および関係のフィルタリング
本発明者らは、透過的プロキシと不透明なプロキシという２つの型のプロキシを有する。開発者が代表の実施態様をプロキシのクライアントに公開することを望む場合は透過的プロキシが使用され、開発者が代表の実施態様をプロキシのクライアントから隠す場合は不透明なプロキシが使用される。

【０６７０】
３．８．３．４．１．１　透過的プロキシ
代表の実施態様をクライアントに公開するために透過的プロキシが使用される。ゾーンが純粋にモデル化を目的としたものであり、それらが含んでいるシステムを隠すべきでない場合、その典型例を図２３に示す。この場合、ｚｏｎｅ２でＳｑｌシステムと通信する開発者は、委任されたエンドポイントの実施態様がゾーンではなくＳｑｌサーバであることを知っているべきである。
【０６７１】
透過的プロキシは、クライアントからではなく代表からすべての識別情報を戻す。これは、インスタンスＩＤ、定義、バージョン、および包含情報はすべてプロキシからではなく代表から入手されるということを意味している。
【０６７２】
照会された場合、透過的プロキシは、プロキシの関係ではなく代表が参加している当該一組の関係も戻す。
【０６７３】
３．８．３．４．１．２　不透明なプロキシ
開発者がクライアントを中断せずにその実施態様を自由に変更するために、代表の実施態様を隠すよう不透明なプロキシが使用される。例えば図１９で、ＩＩＳシステムはファイルシステムの実施態様を隠すために不透明なプロキシを使用することができる。この結果、ＩＩＳのゲストはファイルシステムを実施するオペレーティングシステムには依存しない。
【０６７４】
不透明なプロキシはインスタンスＩＤ、定義、およびバージョンと、代表ではなくプロキシからの包含情報とを戻す。
【０６７５】
照会された場合、不透明なプロキシは、委任関係を含まず、それが直接参加している関係だけを戻す。
【０６７６】
３．８．３．５　フロー
プロキシを目標とするフローは、代表に自動的に宛先変更されるが、この場合、フローはターゲットが実際の実施態様ではなくプロキシであることに気付く必要はない。プロキシはメンバに対するすべての要求を代表に関連付けられたインスタンスに転送するので、そのフローの影響はプロキシによって代表に直接転送される。
【０６７７】
プロキシの一部として宣言されたフローは評価されない。これは、代表がプロキシの定義と同一かまたはプロキシの定義から導出されたものであるべきなので、フローが代表の一部としても評価されるからである。それが評価される場合、本発明者らは常に代表の設定値に複製フローエラーを有する。
【０６７８】
３．８．３．６　制約条件
制約条件は、委任を意識するようにも意識しないようにも書くことができる。制約条件が委任を意識する場合、プロキシが参加している関係のロールアップなしに各プロキシインスタンスに対してその制約条件が評価される。委任関係は制約条件に公開される。
【０６７９】
例えば、図２４のｘからｂを目標として書かれる委任を意識している通信制約条件は、プロキシｂとｘ、ｃ、ｄ、ｅ、およびｉと共に参加している関係とを参照する。この制約条件がｃへの委任関係に従っていると仮定すると、この制約条件はｃからｂ、ｆ、ｈ、およびｇへの関係を参照する。
【０６８０】
制約条件が委任を意識しているようにマーク付けされていない場合、制約条件エンジン

は、プロキシを目標とする関係を評価する場合、透過的プロキシに自動的に進む。制約条件のターゲットインスタンスがプロキシである場合、制約条件エンジンはそれを代表によって置き換える。制約条件は委任関係を参照しない。

【０６８１】

同じ制約条件が委任を意識せずに評価される場合、その制約条件はｂを全く参照しない。代わりに、制約条件はｃに対して評価されるだけである。ｃを評価する場合、制約条件はｃとｘ、ｄ、ｆ、ｈ、ｇ、およびｉとの間の関係を参照する。この場合、本発明者らはその包含関係を除いてｂからのすべての関係をロールアップしており、本発明者らは委任関係を隠しているということに留意されたい。

【０６８２】

制約条件は、ＤｅｌｅｇａｔｉｏｎＡｗａｒｅ属性を構造的制約条件で真に設定することによって委任を意識するようマーク付けされる（オブジェクトまたは関係制約条件－節３．７．３．を参照のこと）。

【０６８３】

プロキシの定義の一部として宣言された制約条件はプロキシに対して評価されない。これは、制約条件はプロキシと代表の定義の両方に共通であるべきなので、制約条件が代表に対しても評価されるからである。

【０６８４】

３．９　マネージャ

マネージャは、型と関係がランタイム環境にカスタム動作を挿入する機構である。マネージャが管理する各型のためにマネージャがサポートすることのできるいくつかのロールがある。マネージャは型のインストールに参加することができ、型のＣＬＲ表現を提供することができ、型間の結合がどのように解決されるかに関してポリシー決定に関与することができ、複雑な制約条件とフローに実施態様を提供することができる。

【０６８５】

すべてのオブジェクトマネージャのロールは、強力に命名されたクラスへのエントリポイントとしてＣＬＲによって公開される。オブジェクトマネージャは、ＳＤＭの他の型と同じ方法でパッケージされ、バージョン付けされる。これらは、システム分散ユニットに分配され、これらのバージョンと強力な名前は、これらが宣言されたＳＤＭファイルから導出される。

【０６８６】

【表１８２】

```
<xs:complexType name="ManagerDeclaration">
    <xs:sequence>
        <xs:element name="Description" type="Description" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="Name" type="SimpleName" use="required"/>
    <xs:attribute name="AssemblyName" type="xs:string" use="required"/>
    <xs:attribute name="Version" type="FourPartVersionType" use="optional"/>
    <xs:attribute name="PublicKeyToken" type="PublicKeyTokenType" use="optional"/>
    <xs:attribute name="Culture" type="CultureNeutral" use="optional"/>
    <xs:attribute name="Platform" type="xs:string" use="optional"/>
    <xs:attribute name="SourcePath" type="xs:string" use="optional"/>
</xs:complexType>
```

【０６８７】

【表１８３】

属性/要素	説明
Name	含んでいるSDMファイルのスコープのこのマネージャに対する一意の名前。
Description	マネージャのテキスト記述
AssemblyName	アセンブリ名
Version	アセンブリバージョン
PublicKeyToken	アセンブリに対する公開鍵トークン
Culture	アセンブリのカルチャ
Platform	アセンブリのプラットフォーム
SourcePath	SDU内のアセンブリへの経路

【０６８８】

３．９．１　規則

オブジェクトマネージャは、それがサポートする各型に対する１つまたは複数のロールをサポートすることができる。これらのロールには、次のものが含まれる。

ａ）型または関係に対する制約条件の評価

ｂ）型または関係に対するフローの評価

ｃ）型に対する構築／破壊／更新サポート

ｄ）型または関係に対する設定のオブジェクト表現の公開

ｅ）型または関係に対する発見の実行

ｆ）型または関係を中心とした設計表面特有ＵＩのサポート

【０６８９】

３．９．２　注記

マネージャは他のマネージャに対する従属関係を有する場合があるが、その従属関係は、そのマネージャを使用するＳＤＭファイルのインポートステートメントに反映されるべきである。この従属関係が記述されていない場合、特定のマネージャが依存する他のマネージャはランタイムによってロードされず、当該マネージャの実行は失敗する。

【０６９０】

３．１０　設計データおよび記述

記述は、関連するＳＤＭ要素を記述するテキストを含んでいる。このテキストは、ＤｏｃｕｍｅｎｔＬａｎｇｕａｇｅ属性が示す文書の言語で書かれている。テキストの場所の特定をサポートするために、資源識別子を提供することができる。ランタイムは、その記述で特定されたマネージャか、またはそれが供給されていない場合は当該記述に関連付けられた、場所が特定されたテキストのための文書に対するディフォルトマネージャを要求する。

【０６９１】

【表１８４】

```
<xs:complexType name="Description" mixed="true">
    <xs:attribute name="ResourceId" type="xs:string" use="optional"/>
    <xs:attribute name="Manager" type="QualifiedName" use="optional"/>
</xs:complexType>
    A design data type contains structured data that defined by a design surface. Each design
    surface should use their own schema to identify and to structure the data.
<xs:complexType name="DesignData">
    <xs:sequence>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
```

【０６９２】

３．１１　ＳＤＭ文書構造

ＳＤＭ文書は、一組の関係、オブジェクト、およびマネージャに対する強力な識別、バージョン付け、および場所特定情報を提供する。

【０６９３】

【表１８５】

```
<xs:element name="SystemDefinitionModel">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Information" type="Information" minOccurs="0"/>
            <xs:element name="Import" type="Import" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="DesignData" type="DesignData" minOccurs="0"/>
            <xs:element name="SettingDefinitions" type="SettingDefinitions" minOccurs="0"/>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="CommunicationDefinition" type="CommunicationDefinition"/>
                <xs:element name="ContainmentDefinition" type="ContainmentDefinition"/>
                <xs:element name="DelegationDefinition" type="DelegationDefinition"/>
                <xs:element name="ReferenceDefinition" type="ReferenceDefinition"/>
                <xs:element name="HostingDefinition" type="HostingDefinition"/>
                <xs:element name="EndpointDefinition" type="EndpointDefinition"/>
                <xs:element name="ResourceDefinition" type="ResourceDefinition"/>
                <xs:element name="SystemDefinition" type="SystemDefinition"/>
                <xs:element name="ConstraintDefinition" type="ConstraintDefinition"/>
                <xs:element name="FlowDefinition" type="FlowDefinition"/>
                <xs:element name="Manager" type="ManagerDeclaration"/>
            </xs:choice>
        </xs:sequence>
        <xs:attributeGroup ref="NamespaceIdentity"/>
        <xs:attribute name="DocumentLanguage" type="Culture"/>
    </xs:complexType>
</xs:element>
```

【０６９４】

３．１１．１　情報

ＳＤＭ文書の情報の節は、ＳＤＭ文書の識別と管理をサポートするために人間によって可読の情報を含んでいる。

【０６９５】

【表１８６】

```
<xs:complexType name="Information">
    <xs:annotation>
```

【０６９６】

【表１８７】

```
            <xs:documentation>Human readable information about the SDM Definition
library.</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="FriendlyName" type="xs:string" minOccurs="0"/>
            <xs:element name="CompanyName" type="xs:string" minOccurs="0"/>
            <xs:element name="Copyright" type="xs:string" minOccurs="0"/>
            <xs:element name="Trademark" type="xs:string" minOccurs="0"/>
            <xs:element name="Description" type="Description" minOccurs="0"/>
            <xs:element name="Comments" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
```

【０６９７】

コンピュータ環境例

図２５は、本明細書に記載の技術を実施するために使用することのできる汎用コンピュータ環境８００を示している。コンピュータ環境８００は、コンピューティング環境の一例に過ぎず、コンピュータおよびネットワークアーキテクチャの用途または機能の範囲に関していかなる限定をも企図するものではない。またコンピュータ環境８００は、コンピュータ環境例８００に示すいかなる１つの構成要素または複数の構成要素の組合せに関するいかなる従属関係または要件をも有するものと解釈されるべきではない。

【０６９８】

コンピュータ環境８００は、コンピュータ８０２の形態の汎用コンピューティングデバイスを含む。コンピュータ８０２は、例えば図１のコンピューティングデバイス１０２、または図２の実施態様開発システム２０２または検証コンポーネント２０８であってよい。コンピュータ８０２の構成要素は、限定はしないが、１つまたは複数のプロセッサまたは処理ユニット８０４、システムメモリ８０６、およびプロセッサ８０４を含めて様々なシステム構成要素をシステムメモリ８０６に結合するシステムバス８０８を含むことができる。

【０６９９】

システムバス８０８は、メモリバスまたはメモリコントローラ、周辺バス、アクセラレーテッドグラフィックスポート、様々なバスアーキテクチャのどれかを使用するプロセッサまたはローカルバスを含めて複数種類のバス構造の任意の１つまたは複数を表す。一例として、このようなアーキテクチャは、業界標準アーキテクチャ（ＩＳＡ）バス、マイクロチャネルアーキテクチャ（ＭＣＡ）バス、拡張ＩＳＡ（ＥＩＳＡ）バス、ビデオエレクトロニクス規格協会（ＶＥＳＡ）ローカルバス、およびメザニンバスとしても知られている周辺装置相互接続（ＰＣＩ）バスを含むことができる。

【０７００】

コンピュータ８０２は、通常、様々なコンピュータ可読媒体を含む。このような媒体は、コンピュータ８０２によってアクセス可能ないかなる使用可能な媒体であってもよく、揮発性媒体と不揮発性媒体、取り外し可能媒体と取り外し不可能媒体の両方を含む。

【０７０１】

システムメモリ８０６は、ランダムアクセスメモリ（ＲＡＭ）８１０のような揮発性メモリ、および／または読み取り専用メモリ（ＲＯＭ）８１２のような不揮発性メモリの形態のコンピュータ可読媒体を含む。起動時などにコンピュータ８０２内の素子間での情報の転送に役立つ基本ルーチンを含んでいる基本入出力システム（ＢＩＯＳ）８１４はＲＯＭ　８１２に記憶されている。ＲＡＭ　８１０は、通常、処理ユニット８０４によって直接アクセス可能であり、かつ／またはそれに対して現在動作が行われているデータおよび／またはプログラムモジュールを含んでいる。

【０７０２】

コンピュータ８０２は、他の取り外し可能／取り外し不可能な揮発性／不揮発性コンピュータ記憶媒体を含むこともできる。一例として、図２５は、取り外し不可能な不揮発性磁気媒体（図示せず）から読み取り／に書き込むためのハードディスクドライブ８１６、取り外し可能な不揮発性磁気ディスク８２０（例えば、「フロッピー（登録商標）ディスク」）から読み取り／に書き込むための磁気ディスクドライブ８１８、ＣＤ－ＲＯＭ、ＤＶＤ－ＲＯＭ、または他の光媒体のような取り外し可能な不揮発性光ディスク８２４から読み取り／に書き込むための光ディスクドライブ８２２を示している。ハードディスクドライブ８１６、磁気ディスクドライブ８１８、および光ディスクドライブ８２２は、１つまたは複数のデータ媒体インターフェース８２６によってシステムバス８０８にそれぞれ接続されている。あるいは、ハードディスクドライブ８１６、磁気ディスクドライブ８１８、および光ディスクドライブ８２２は、１つまたは複数のインターフェース（図示せず）によってシステムバス８０８に接続することができる。

【０７０３】

ディスクドライブおよびそれらに関連するコンピュータ可読媒体は、コンピュータ可読命令、データ構造、プログラムモジュール、および他のデータの不揮発性記憶をコンピュ

ータ８０２に提供する。この実施例は、ハードディスク８１６、取り外し可能磁気ディスク８２０、および取り外し可能光ディスク８２４を示しているが、コンピューティングシステムおよび環境例を実施するには、磁気カセットまたは他の磁気記憶装置、フラッシュメモリカード、ＣＤ－ＲＯＭ、デジタル多用途ディスク（ＤＶＤ）、または他の光ストレージ、ランダムアクセスメモリ（ＲＡＭ）、読み取り専用メモリ（ＲＯＭ）、電気的消去可能書き込み可能ＲＯＭ（ＥＥＰＲＯＭ）などのようなコンピュータによってアクセス可能なデータを記憶することができる他の種類のコンピュータ可読媒体を使用することもできるということを理解されたい。

【０７０４】

一例としてオペレーティングシステム８２６、１つまたは複数のアプリケーションプログラム８２８、他のプログラムモジュール８３０、およびプログラムデータ８３２を含めてプログラムモジュールを幾つでも、ハードディスク８１６、磁気ディスク８２０、光ディスク８２４、ＲＯＭ　８１２、および／またはＲＡＭ　８１０に記憶することができる。そのようなオペレーティングシステム８２６、１つまたは複数のアプリケーションプログラム８２８、他のプログラムモジュール８３０、およびプログラムデータ８３２（またはこれらのいくつかの組合せ）のそれぞれは、分散型ファイルシステムをサポートする常駐構成要素のすべてまたは一部を実施することができる。

【０７０５】

ユーザは、キーボード８３４およびポインティングデバイス８３６（例えば、「マウス」）のような入力装置を介してコマンドおよび情報をコンピュータ８０２に入力することができる。他の入力装置８３８（具体的には図示せず）は、マイクロフォン、ジョイスティック、ゲームパッド、衛星パラボラアンテナ、シリアルポート、スキャナなどを含むことができる。これらおよび他の入力装置は、システムバス８０８に結合されるが、パラレルポート、ゲームポート、またはユニバーサルシリアルバス（ＵＳＢ）のような他のインターフェースおよびバス構造によって接続することができる入出力インターフェース８４０を介して処理ユニット８０４に接続される。

【０７０６】

モニタ８４２または他の種類の表示装置は、ビデオアダプタ８４４のようなインターフェースを介してシステムバス８０８に接続することもできる。モニタ８４２に加え、他の出力周辺装置は、入出力インターフェース８４０を介してコンピュータ８０２に接続することができるスピーカ（図示せず）およびプリンタ８４６のような構成要素を含むことができる。

【０７０７】

コンピュータ８０２は、リモートコンピュータデバイス８４８のような１つまたは複数の遠隔コンピュータへの論理接続を使用してネットワーク接続された環境で動作することができる。一例として、リモートコンピュータデバイス８４８は、パーソナルコンピュータ、携帯型コンピュータ、サーバ、ルータ、ネットワークコンピュータ、ピアデバイスまたは他の共通ネットワークノードなどであってよい。リモートコンピュータデバイス８４８は、コンピュータ８０２に関して本明細書に記載の要素および特徴の多くまたはすべてを含むことのできる携帯型コンピュータとして示される。

【０７０８】

コンピュータ８０２と遠隔コンピュータ８４８の間の論理接続は、ローカルエリアネットワーク（ＬＡＮ）８５０および一般的なワイドエリアネットワーク（ＷＡＮ）８５２として示される。このようなネットワーク接続環境は、職場、企業規模コンピュータネットワーク、インターフェース、およびインターネットでは一般的である。

【０７０９】

ＬＡＮネットワーク接続環境で実施される場合、コンピュータ８０２はネットワークインターフェースまたはアダプタ８５４を介してローカルネットワーク８５０に接続される。ＷＡＮネットワーク接続環境で実施される場合、コンピュータ８０２は、通常、モデム８５６またはワイドネットワーク８５２を介して通信を確立するための他の手段を含む。

コンピュータ８０２に内蔵されていても外付けであってもよいモデム８５６は、入出力インターフェース８４０または他の適切な機構を介してシステムバス８０８に接続することができる。図示したネットワーク接続は一例であり、コンピュータ８０２と８４８の間の１つ以上の通信リンクを確立する他の手段を利用することができるということを理解されたい。

【０７１０】

コンピューティング環境８００と共に図示されたネットワーク接続された環境では、コンピュータ８０２に関して示されたプログラムモジュールまたはその一部は、遠隔メモリ記憶装置に記憶することができる。一例として、遠隔アプリケーションプログラム８５８は、遠隔コンピュータ８４８のメモリ装置に常駐する。説明のために、アプリケーションプログラムとオペレーティングシステムのような他の実行可能なプログラム構成要素とを本明細書では別個のブロックとして示す。ただし、このようなプログラムおよび構成要素は様々な時点でコンピューティングデバイス８０２の様々な記憶構成要素に常駐し、コンピュータの１つ以上のデータプロセッサによって実行されることを理解されたい。

【０７１１】

本明細書では様々なモジュールおよび技術は、１つまたは複数のコンピュータまたは他の装置によって実行されるプログラムモジュールのようなコンピュータ実行可能命令の一般的な状況で説明することができる。一般に、プログラムモジュールは、特定タスクを実行し、特定の抽象データ型を実施するルーチン、プログラム、オブジェクト、構成要素、データ構造などを含む。通常、プログラムモジュールの機能は、適宜、様々な実施形態で結合されても分散されてもよい。

【０７１２】

これらモジュールおよび技術の実施態様は、コンピュータ可読媒体のある種の形態に記憶し、またはこれを横断して伝送することができる。コンピュータ可読媒体は、コンピュータがアクセスすることのできるいかなる使用可能な媒体であってもよい。限定ではなく、一例として、コンピュータ可読媒体は、「コンピュータ記憶媒体」および「通信媒体」を含むことができる。

【０７１３】

「コンピュータ記憶媒体」は、コンピュータ可読命令、データ構造、プログラムモジュール、または他のデータのような情報の記憶のためのいかなる方法または技術でも実施される揮発性および不揮発性の取り外し可能および取り外し不可能な媒体を含む。コンピュータ記憶媒体は、限定はしないが、ＲＡＭ、ＲＯＭ、ＥＥＰＲＯＭ、フラッシュメモリまたは他のメモリ技術、ＣＤ－ＲＯＭ、デジタル多用途ディスク（ＤＶＤ）、または他の光記憶装置、磁気カセット、磁気テープ、磁気ディスク記憶または他の磁気記憶装置、または所望の情報を記憶するために使用することができ、コンピュータがアクセスすることのできる任意の他の媒体を含む。

【０７１４】

「通信媒体」は、通常、コンピュータ可読命令、データ構造、プログラムモジュール、もしくは搬送波または他の搬送機構のような変調されたデータ信号形式の他のデータを実施する。通信媒体は、いかなる情報伝達媒体をも含む。「変調されたデータ信号」という用語は、情報を信号に符号化することに関する方法で設定または変更されるその特徴の１つまたは複数を有する信号を意味している。限定ではなく、一例として、通信媒体は、有線ネットワークまたは直接有線接続のような有線媒体と、アコースティック、ＲＦ、赤外線、および他の無線媒体のような無線媒体とを含む。上記のどの組合せでもコンピュータ可読媒体の範囲に含まれる。

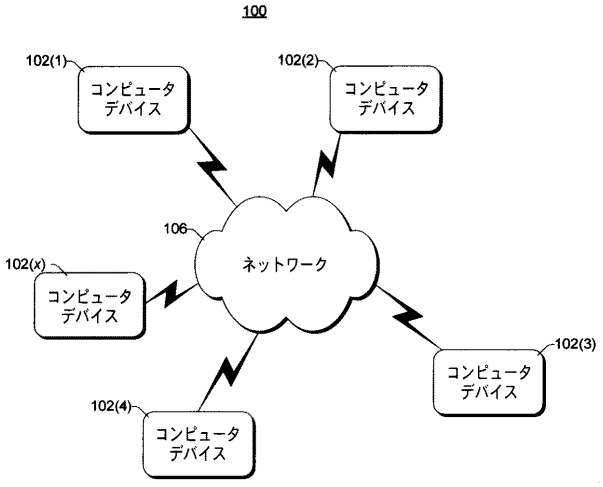
【０７１５】

あるいは、フレームワークの部分は、ハードウェア、またはハードウェア、ソフトウェア、および／またはファームウェアの組合せで実施することができる。例えば、１つまたは複数の特定用途向け集積回路（ＡＳＩＣ）またはプログラム可能論理装置（ＰＬＤ）は、フレームワークの１つまたは複数の部分を実施するために設計またはプログラムするこ

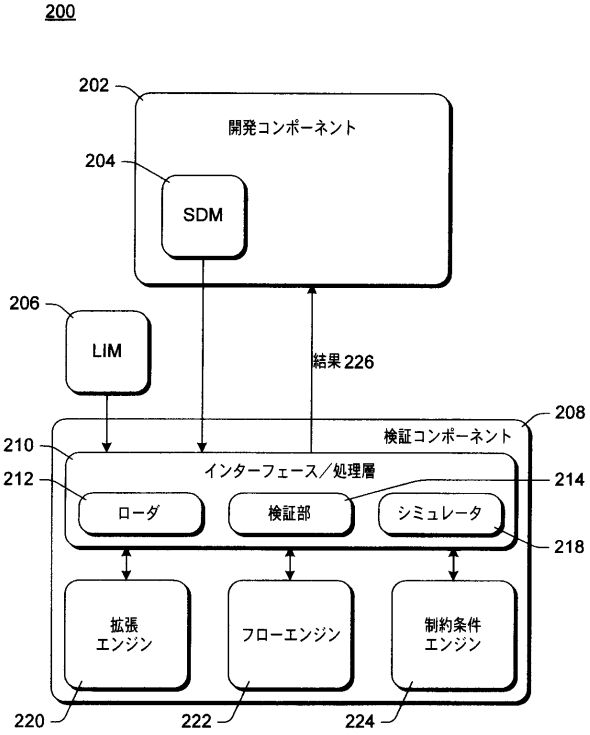
とができる。
【０７１６】
結論
以上、本発明は、構造上の特性および／または方法上の動作特有の言い回しで説明したが、首記の特許請求の範囲で定義した本発明を、記載の特有の機能または動作に限定することは必要ないということを理解されたい。そうではなく、これら特有の機能および動作は、請求された発明を実施する形式例として開示されている。
【図面の簡単な説明】
【０７１７】
【図１】ネットワーク設定例を示す図である。
【図２】システム定義モデルを使用するアーキテクチャ例を示すブロック図である。
【図３】階層化された設定例を示す図である。
【図４】設計時検証のためのプロセス例を示す流れ図である。
【図５Ａ】インスタンス空間を作成する反復拡張プロセス例を示す流れ図である。
【図５Ｂ】インスタンス空間を作成する反復拡張プロセス例を示す流れ図である。
【図６Ａ】評価の流れのプロセス例を示す流れ図である。
【図６Ｂ】評価の流れのプロセス例を示す流れ図である。
【図７Ａ】評価制約条件のプロセス例を示す流れ図である。
【図７Ｂ】評価制約条件のプロセス例を示す流れ図である。
【図７Ｃ】評価制約条件のプロセス例を示す流れ図である。
【図７Ｄ】評価制約条件のプロセス例を示す流れ図である。
【図７Ｅ】評価制約条件のプロセス例を示す流れ図である。
【図８】ＳＤＭ文書例を示す図である。
【図９】ＳＤＭ定義例を示す図である。
【図１０】ＳＤＭメンバ例を示す図である。
【図１１】設定メンバ例を示す図である。
【図１２】制約条件定義例を示す図である。
【図１３】記述オブジェクト例を示す図である。
【図１４】１つの層のウェブアプリケーションを別の層のウェブサーバホストにマッピングする例を示す図である。
【図１５】組み込みデータ型階層例を示す図である。
【図１６】アップキャスティングおよびダウンキャスティング例を示す図である。
【図１７】型変換例を示す図である。
【図１８】関係ツリー例を示す図である。
【図１９】ホストの実施態様をゲストに公開するために委任を使用する例を示す図である。
【図２０】メンバの実施態様を親に公開するために委任を使用する例を示す図である。
【図２１】通信接続検証例を示す図である。
【図２２】通信接続検証例を示す図である。
【図２３】ゾーン境界例を示す図である。
【図２４】インスタンス空間例を示す図である。
【図２５】本明細書に記載の技術を実施するために使用することのできる汎用コンピュータ環境例を示す図である。
【符号の説明】
【０７１８】
１０２　コンピュータデバイス
１０６　ネットワーク
２０２　開発構成要素
２０４　ＳＤＭ文書
２０６　ＬＩＭ文書

２０８　検証構成要素
８００　コンピュータ環境
８０２　コンピュータ

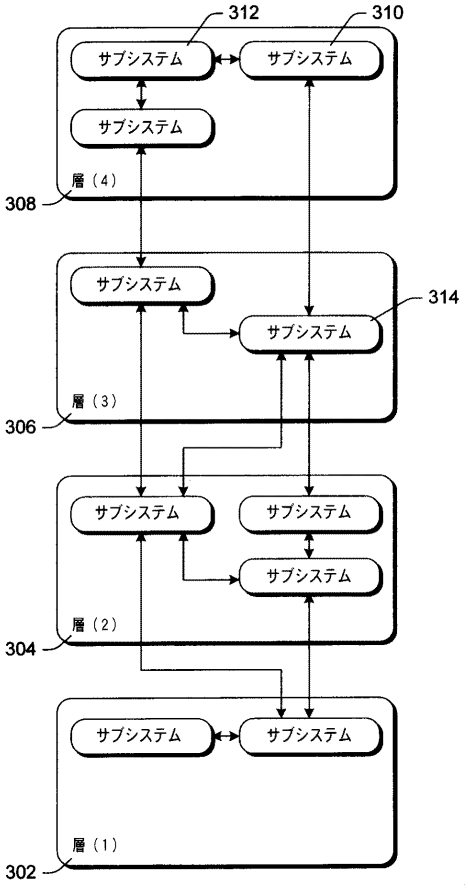
【図１】



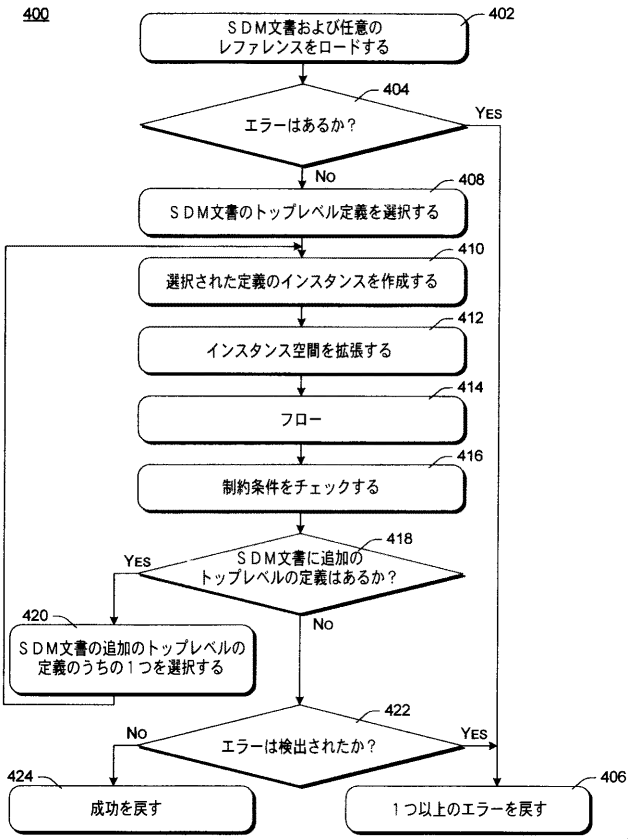
【図２】



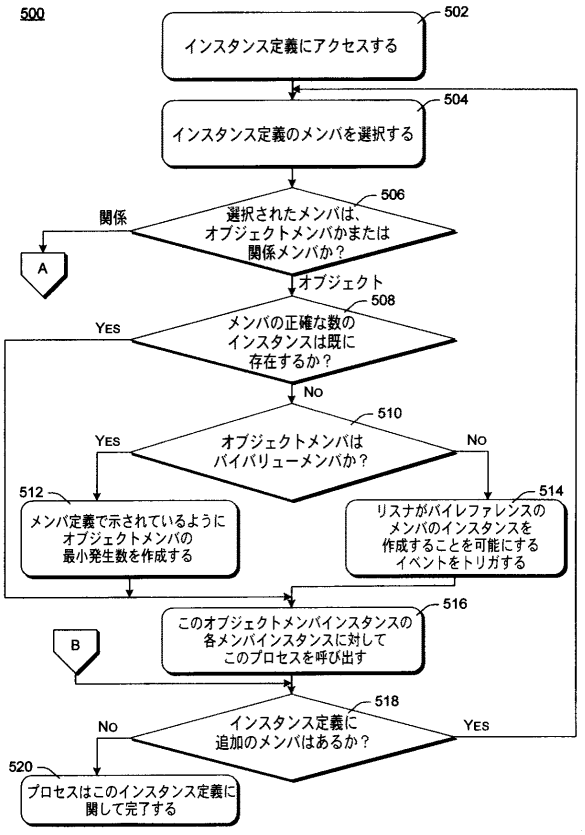
【図３】


312
310
サブシステム
サブシステム
サブシステム
層（4）
308
サブシステム
314
サブシステム
層（3）
306
サブシステム
サブシステム
サブシステム
層（2）
304
サブシステム
サブシステム
層（1）
302

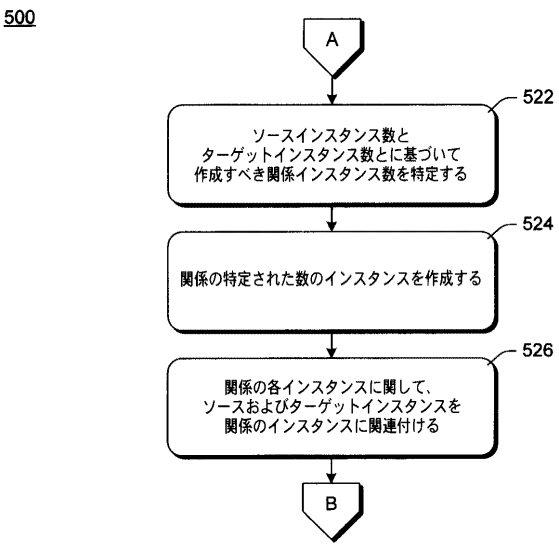
【図４】


400
SDM文書および任意のレファレンスをロードする
402
エラーはあるか？
404
YES
No
SDM文書のトップレベル定義を選択する
408
選択された定義のインスタンスを作成する
410
インスタンス空間を拡張する
412
フロー
414
制約条件をチェックする
416
SDM文書に追加のトップレベルの定義はあるか？
418
YES
No
SDM文書の追加のトップレベルの定義のうちの１つを選択する
420
エラーは検出されたか？
422
No
YES
成功を戻す
424
１つ以上のエラーを戻す
406

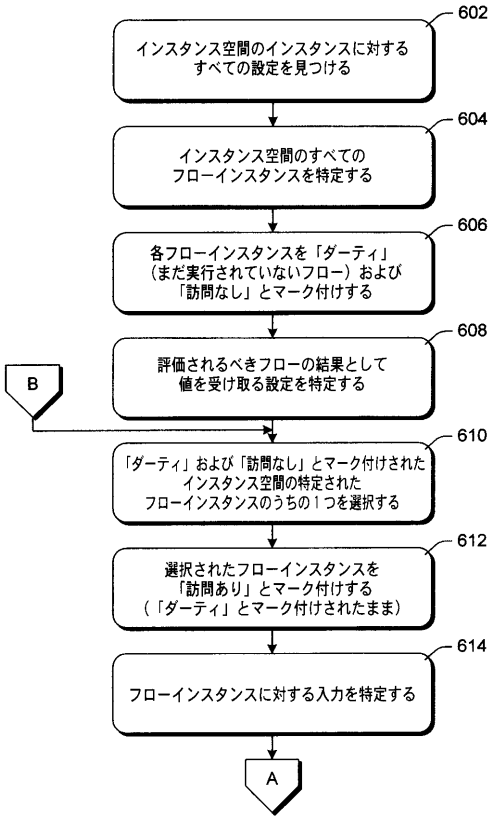
【図５Ａ】


500
インスタンス定義にアクセスする
502
インスタンス定義のメンバを選択する
504
選択されたメンバは、オブジェクトメンバかまたは関係メンバか？
506
関係
オブジェクト
A
メンバの正確な数のインスタンスは既に存在するか？
508
YES
No
オブジェクトメンバはバイバリューメンバか？
510
YES
No
メンバ定義で示されているようにオブジェクトメンバの最小発生数を作成する
512
リスナがバイレファレンスのメンバのインスタンスを作成することを可能にするイベントをトリガする
514
このオブジェクトメンバインスタンスの各メンバインスタンスに対してこのプロセスを呼び出す
516
B
インスタンス定義に追加のメンバはあるか？
518
No
YES
プロセスはこのインスタンス定義に関して完了する
520

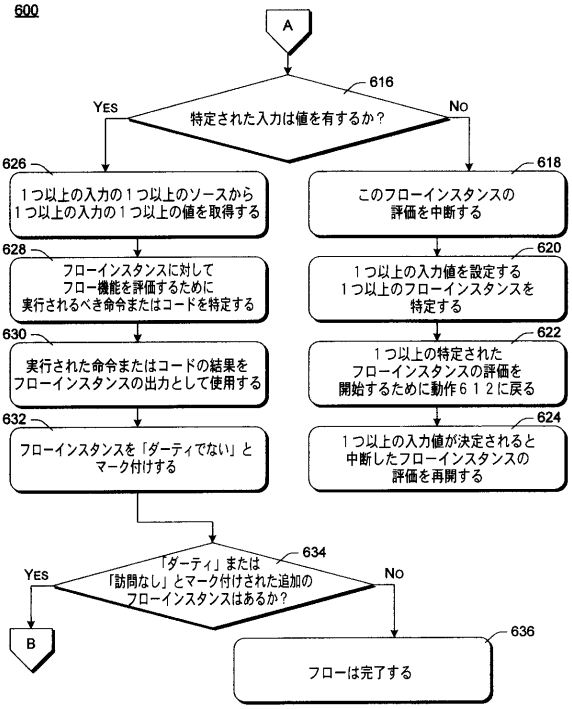
【図５Ｂ】


500
A
ソースインスタンス数とターゲットインスタンス数とに基づいて作成すべき関係インスタンス数を特定する
522
関係の特定された数のインスタンスを作成する
524
関係の各インスタンスに関して、ソースおよびターゲットインスタンスを関係のインスタンスに関連付ける
526
B

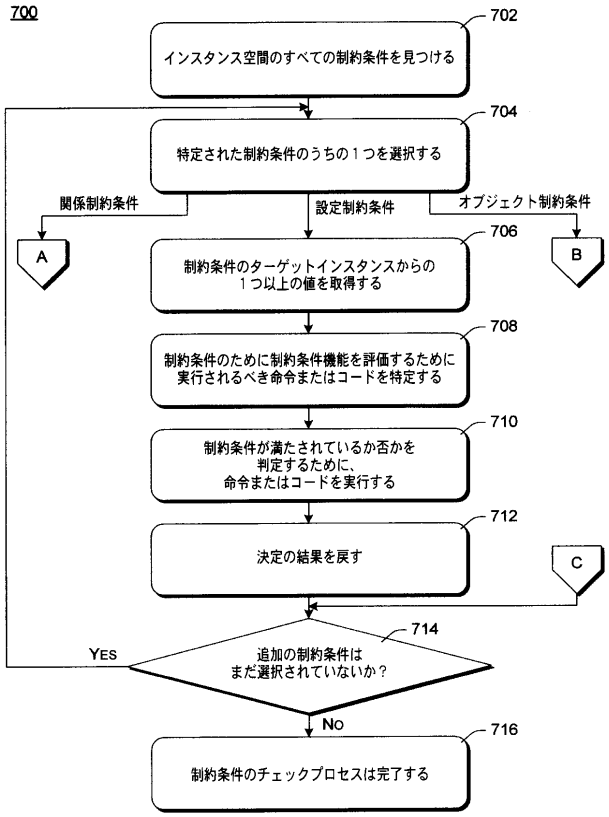
【図６Ａ】



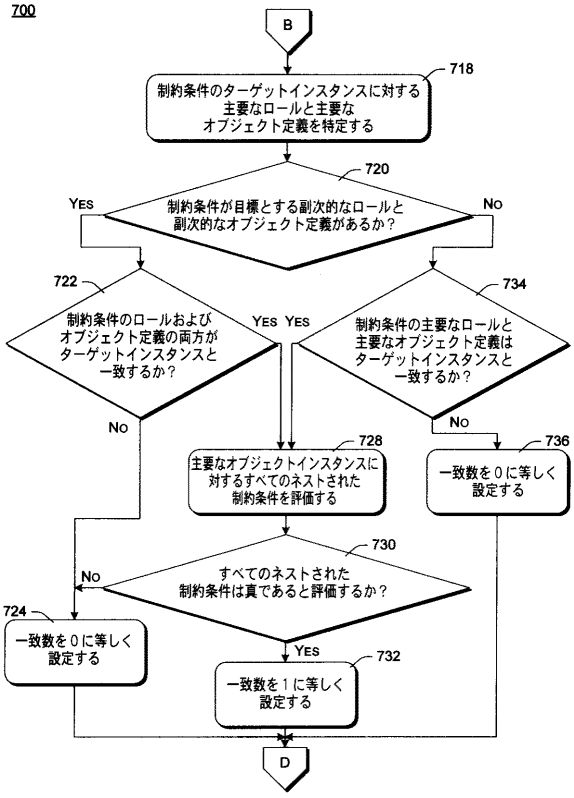
【図６Ｂ】



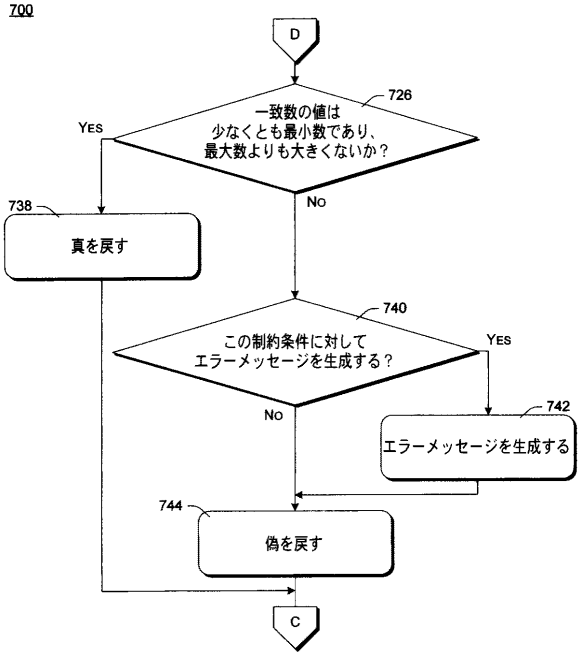
【図７Ａ】



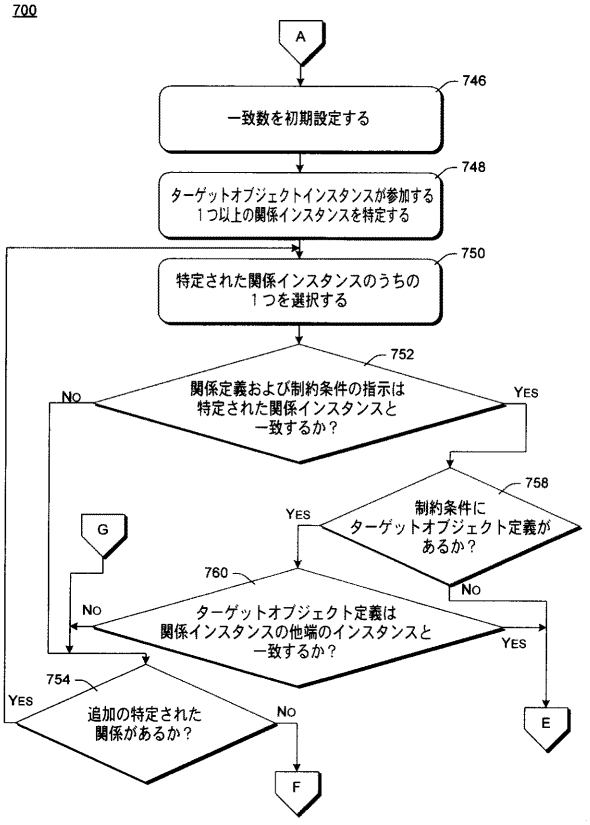
【図７Ｂ】



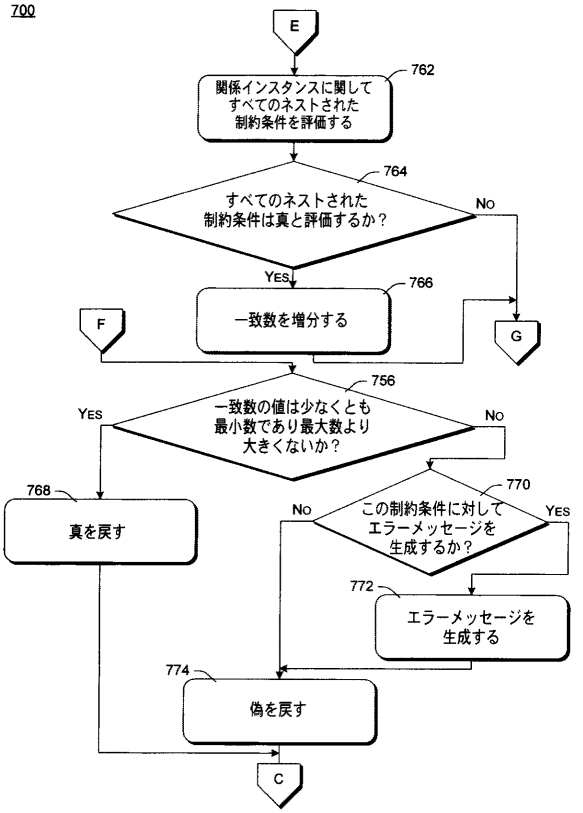
【図７Ｃ】



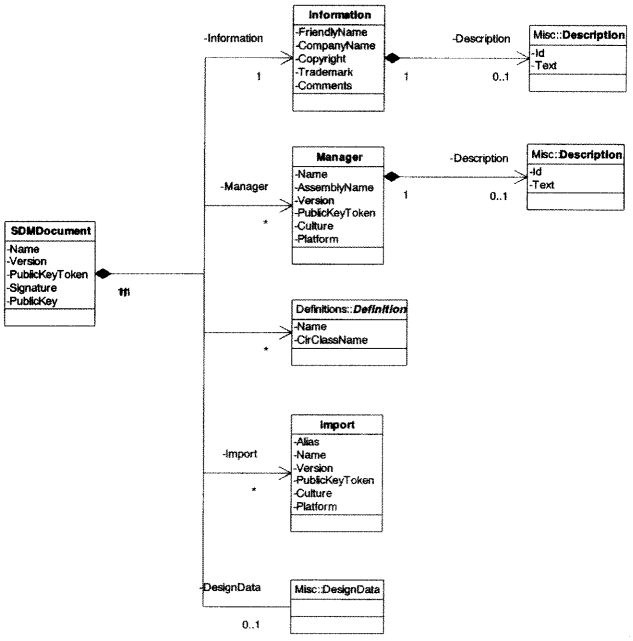
【図７Ｄ】



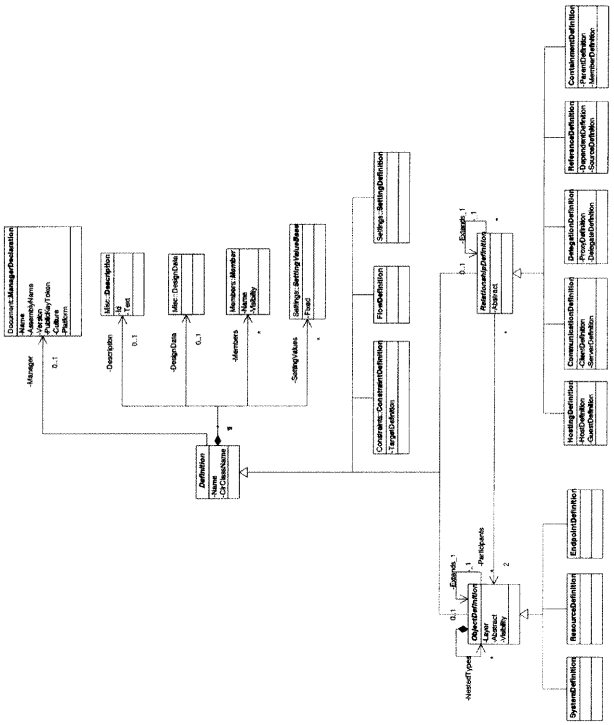
【図７Ｅ】



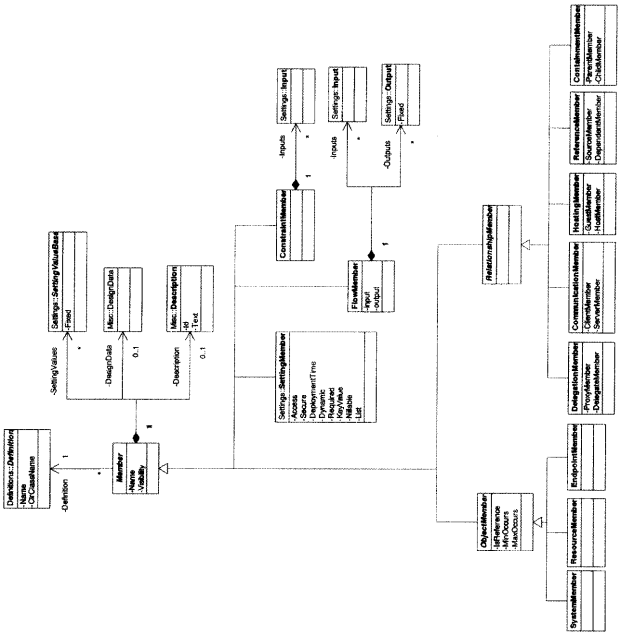
【図８】



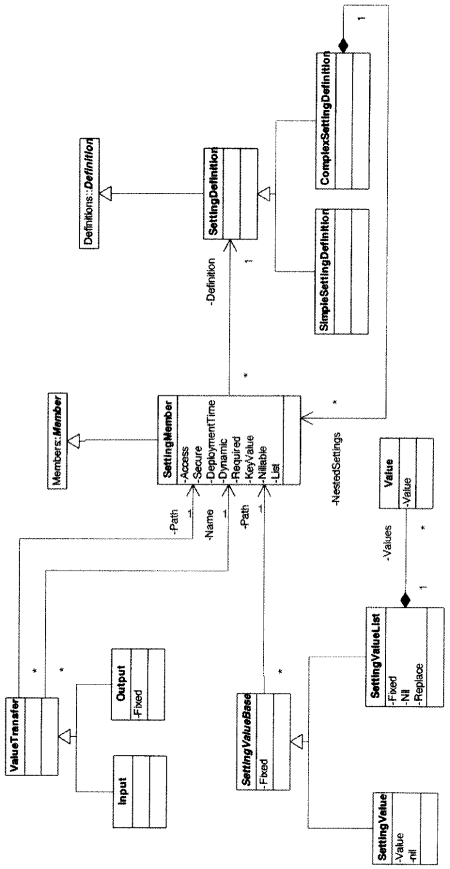
【図９】



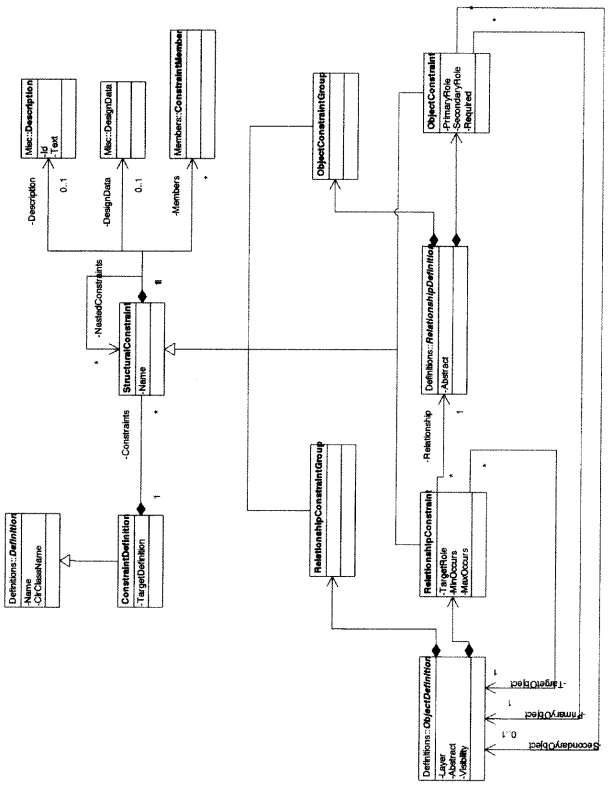
【図１０】



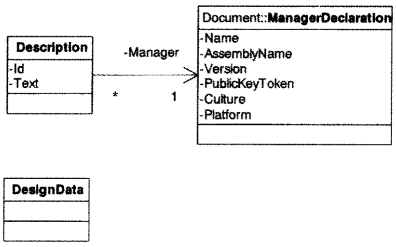
【図１１】



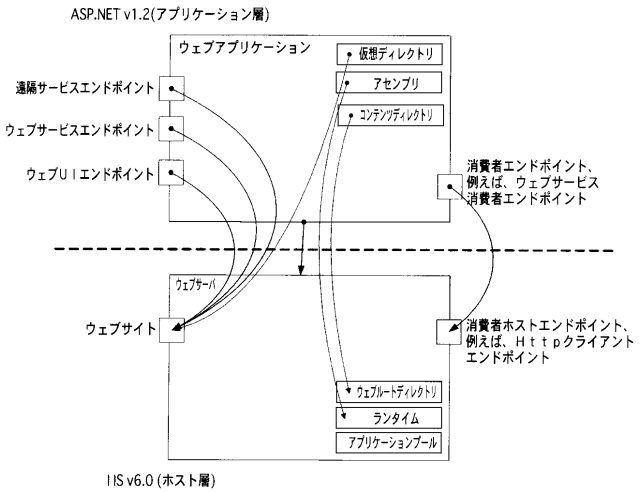
【図１２】



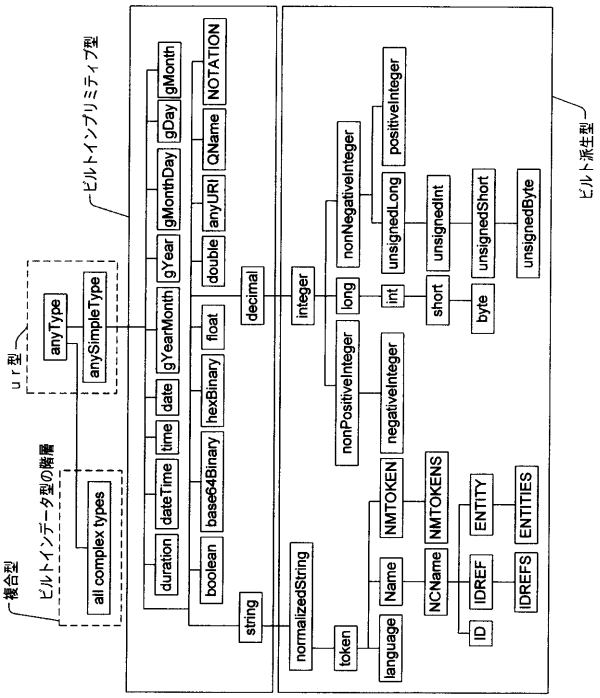
【図１３】



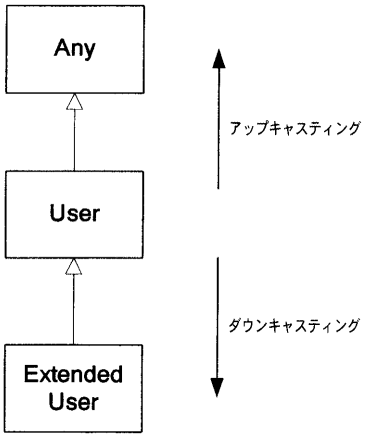
【図１４】



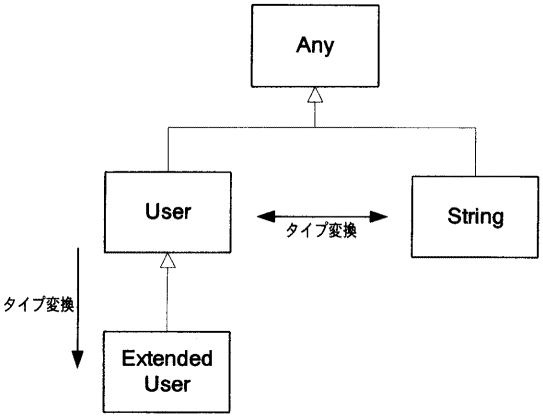
【図１５】



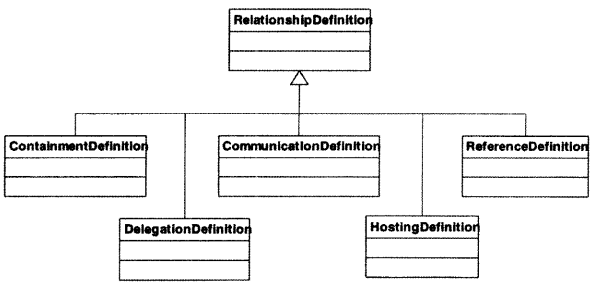
【図１６】



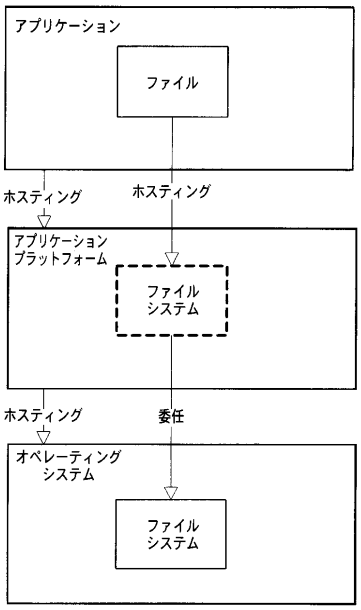
【図１７】



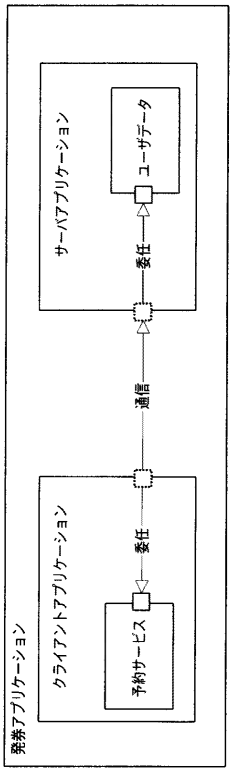
【図１８】



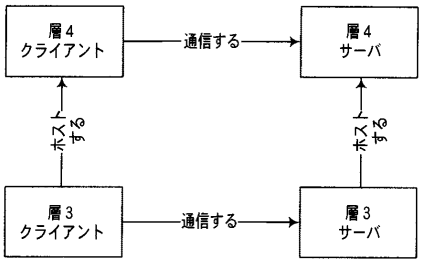
【図１９】



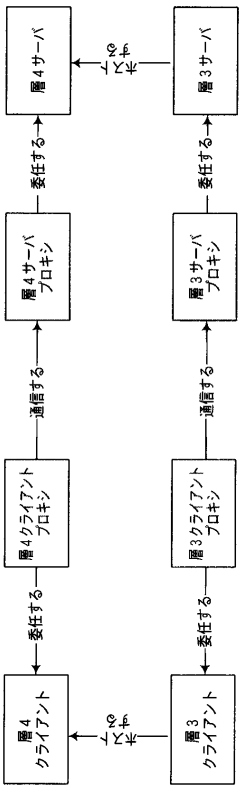
【図２０】



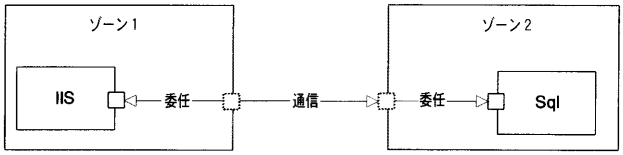
【図２１】



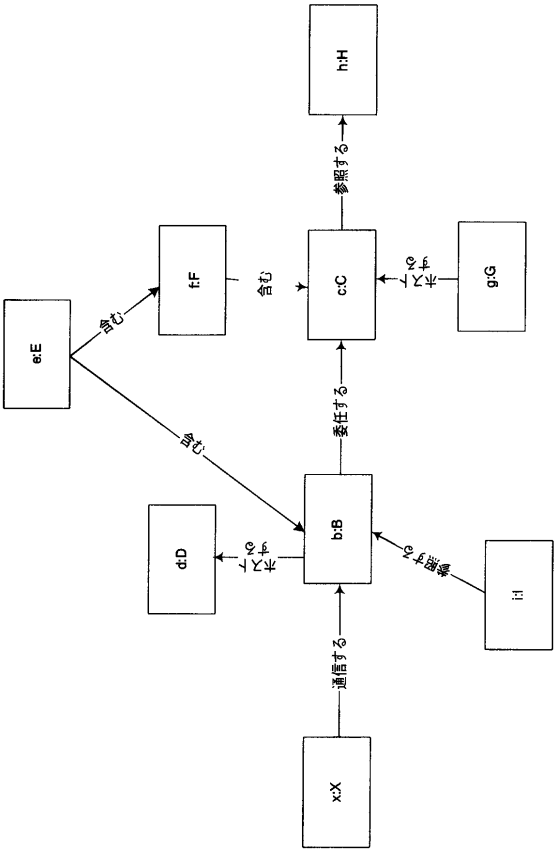
【図２２】



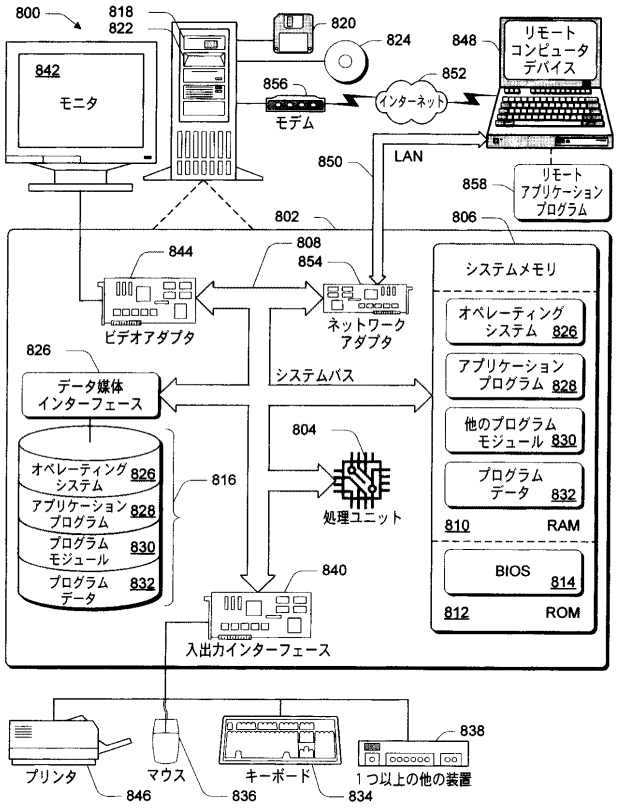
【図２３】


ゾーン1
IIS
委任
通信
委任
ゾーン2
Sql

【図２４】


h:H
参照する
e:E
含む
f:F
含む
c:C
オブジェクト
g:G
含む
委任する
b:B
d:D
オブジェクト
参照する
i:I
通信する
x:X

【図２５】


800
818
822
820
824
848
リモート
コンピュータ
デバイス
842
モニタ
856
モデム
852
インターネット
850
LAN
858
リモート
アプリケーション
プログラム
802
806
844
808
854
システムメモリ
ビデオアダプタ
ネットワーク
アダプタ
オペレーティング
システム 826
826
データ媒体
インターフェース
システムバス
アプリケーション
プログラム 828
他のプログラム
モジュール 830
804
816
オペレーティング
システム 826
アプリケーション
プログラム 828
プログラム
モジュール 830
プログラム
データ 832
処理ユニット
プログラム
データ 832
810 RAM
840
BIOS 814
812 ROM
入出力インターフェース
838
プリンタ
マウス
キーボード
1つ以上の他の装置
846
836
834

フロントページの続き

(72)発明者　ケビン　グレアリッシュ
アメリカ合衆国　９８１０３　ワシントン州　シアトル　デイトン　アベニュー　ノース　４０３５
(72)発明者　ロブ　メンシング
アメリカ合衆国　９８０５２　ワシントン州　レッドモンド　１６４　アベニュー　ノースイースト　１０５０７
(72)発明者　バッサム　タッバラ
アメリカ合衆国　９８１０７　ワシントン州　シアトル　ベーカー　アベニュー　ノースウエスト　４１１９

Ｆターム(参考) 5B076 EC02　EC10

【外国語明細書】
2004272909000001.pdf