(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4994580号
(P4994580)
(45)発行日　平成24年8月8日(2012.8.8)　(24)登録日　平成24年5月18日(2012.5.18)

(51)Int.Cl.　F I
G06F 9/44 (2006.01)　G06F 9/44 530D

請求項の数 11　外国語出願　(全 24 頁)

(21)出願番号	特願2004-148158 (P2004-148158)
(22)出願日	平成16年5月18日 (2004.5.18)
(65)公開番号	特開2005-4737 (P2005-4737A)
(43)公開日	平成17年1月6日 (2005.1.6)
審査請求日	平成19年5月17日 (2007.5.17)
審判番号	不服2011-107 (P2011-107/J1)
審判請求日	平成23年1月4日 (2011.1.4)
(31)優先権主張番号	10/458,460
(32)優先日	平成15年6月10日 (2003.6.10)
(33)優先権主張国	米国 (US)

(73)特許権者　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　110001243
特許業務法人　谷・阿部特許事務所
(74)復代理人　100115624
弁理士　濱中　淳宏
(72)発明者　エリック　メイジャー
アメリカ合衆国　９８０４０　ワシントン州　マーサー　アイランド　サウスイースト　７６　プレイス　８５５７

最終頁に続く

(54)【発明の名称】動的ランタイム環境でタグ付き型を用いるシステム及び方法

(57)【特許請求の範囲】
【請求項１】
実行時に動的に型付けされる動的プログラミング言語を実行するコンピュータシステムであって、
前記動的プログラミング言語によって宣言され、ポインター数値とその他の数値とを区別する１又は２以上のタグ付き数値の継承階層を宣言するクラスコンポーネントであって、前記継承階層が、分岐ツリーの一方に非タグ付き型のシステムオブジェクト階層を含み、もう一方に前記タグ付き数値を表すシールドされた型であるタグ付き型を含む、抽象ルート型を有し、前記タグ付き数値が、算術演算を実行するために変換されるクラスコンポーネントと、
ランタイム拡張を含む１つまたは複数の実行ルールを提供し、前記システムオブジェクト階層に含まれるユーザ定義の型が、前記抽象ルート型または前記タグ付き数値からプロパティを継承しないようにする少なくとも１つのメタデータ検証ルールを備えるルールコンポーネントと
を備え、
前記コンピュータシステムの記憶装置に記憶されたコンピュータ可読命令が、該コンピュータシステムの処理装置によって実行されると、
前記クラスコンポーネントおよび前記ルールコンポーネントを使用して、前記ユーザ定義の型が前記抽象ルート型または前記タグ付き数値からはプロパティを継承しないようにして、実行時に動的に型付けされる前記動的プログラミング言語を実行することを特徴と

するシステム。
【請求項２】
前記動的プログラミング言語は、仮想マシンによって実行される中間言語命令を生成するコンパイラーに供給されることを特徴とする請求項１記載のシステム。
【請求項３】
前記動的プログラミング言語は、パール（Ｐｅｒｌ）、スキーム（Ｓｃｈｅｍｅ）、ラビー（Ｒｕｂｙ）、ピーソン（Ｐｙｔｈｏｎ）及びスモールトーク（Ｓｍａｌｌｔａｌｋ）のうちの一つを少なくとも含むことを特徴とする請求項１記載のシステム。
【請求項４】
前記仮想マシンは、仮想実行システム（ＶＥＳ）、共通言語ランライム（ＣＬＲ）及びジャバ仮想マシンのうちの一つを少なくとも含むことを特徴とする請求項２記載のシステム。
【請求項５】
前記タグ付き数値は、共通言語インフラストラクチャ（ＣＬＩ）を定義する「ＥＣＭＡ標準」にしたがうことを特徴とする請求項１記載のシステム。
【請求項６】
前記ＥＣＭＡ標準は、少なくとも５つのパーティションを含み、前記パーティションは、パーティションＩ：アーキテクチャ、パーティションＩＩ：メタデータ定義及びセマンティクス、パーティションＩＩＩ：ＣＩＬ命令セット、パーティションＩＶ：プロファイル及びライブラリー及びパーティションＶ：アネックスを含むことを特徴とする請求項５記載のシステム。
【請求項７】
前記共通言語インフラストラクチャ（ＣＬＩ）は、共通型システム、メタデータ記述、共通言語仕様及び仮想実行システムのうちの少なくとも一つを含むことを特徴とする請求項５記載のシステム。
【請求項８】
前記抽象ルート型は、ＥＣＭＡパーティションＩＩＩで提供される型Ｏのオブジェクト参照に対応することを特徴とする請求項１記載のシステム。
【請求項９】
前記抽象ルート型と関連する数値を「．ｒｅｆ」接尾辞を備える命令バリアントのスタックにロードするコンポーネントを更に備えることを特徴とする請求項１記載のシステム。
【請求項１０】
前記タグ付き数値を操作する少なくとも１つの命令をさらに備え、前記命令は、キャストクラス、テストクラス、整数値をオブジェクト参照に変換するタグ命令、及びオブジェクト参照を整数値に変換するアンタグ命令を少なくとも一つ含むことを特徴とする請求項１記載のシステム。
【請求項１１】
請求項１記載の前記クラスコンポーネントと前記ルールコンポーネントとを使用して、コンピュータ上で、実行時に動的に型付けされる動的プログラミング言語を実行するための前記コンピュータ可読命令を格納したことを特徴とするコンピュータ記憶媒体。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、一般的にコンピュータシステムに関し、より詳細には、動的ランタイム環境における動的言語の操作及び実行を容易にするシステム及び方法に関する。
【背景技術】
【０００２】
コンピュータ科学が発展するにつれて、オブジェクト指向プログラミングは、設計者やプログラマーがコンピュータシステム内の機能を実行するために利用する数多くの良く知

られたモデルの中の一つとなっている。オブジェクトモデルは、一般的に、メソッドとクラスに属する関連付けられたデータ要素とを提供するクラスメンバーを含んだクラス構造によって定義される。クラスメンバーは、コンピュータプログラム内部に所望の機能を提供／定義し、そこで、オブジェクトは特定のクラスのインスタンスとして宣言される。よく知られたように、オブジェクトは、頻繁にデータ交換を行い、及び／又は、同じプラットフォーム上で動作する他のオブジェクトを呼び出し、及び／又は、遠隔プラットフォームに属するオブジェクトと通信をする。オブジェクト間で通信を行うために、インタフェースシステム及びインタフェース規格は、発展して、オブジェクトがどのように相互に通信をし、及び／又は、どのように相互に作用しあうかを定義してきた。

【０００３】

オブジェクト間で通信を行い、インタフェースをとる著名なシステムは、コンポーネント・オブジェクト・モデル（ＣＯＭ）として知られており、そこでは、別の類似システムがコモン・オブジェクト・リクエスト・ブローカー・アーキテクチャ（ＣＯＲＢＡ）として参照されている。その他の通信インタフェースは、例えば、Ｊａｖａ（登録商標）仮想マシンのオペレーティングフレームワーク内部のＪＡＶＡ（登録商標）等の言語の中で定義される。これらのシステム及び他のシステムが発展するにつれて、２つの共通オブジェクトアーキテクチャ又はモデルは一般に現れて、例えば、一般にマネージド及びアンマネージドのオブジェクトシステムの観点から定義される。

【０００４】

マネージドオブジェクトは、マネージドソフトウェア環境内部のヒープ（ｈｅａｐ）から割り当てられ、一般的にマネージング関連オブジェクトライフタイムに対して責任をもたない。マネージドオブジェクトは、データ型（例えば、メタデータ）という形で記述され、オブジェクトがもはやアクセスされない場合にはメモリからオブジェクトを除去するマネージド環境「不要データ収集器」（ガーベッジコレクター）によって自動的に収集（回収）される。これとは対照的に、アンマネージドオブジェクトは、標準的なオペレーティングシステムヒープから割り当てられ、そこで、オブジェクト自身は、オブジェクトへの参照がもはや存在しない場合に、オブジェクトが使用するメモリを開放すること（ｆｒｅｅｉｎｇ　ｍｅｍｏｒｙ）に対して責任をもつ。このことは、参照計数（ｒｅｆｅｒｅｎｃｅ　ｃｏｕｎｔｉｎｇ）等のよく知られた技法により達成され得る。

【０００５】

上述した通り、マネージドオブジェクトは、マネージドヒープから割り当てられ、不要データは自動的に収集される。これを達成するために、マネージドオブジェクトに対する参照が追跡（ｔｒａｃｅ）される。オブジェクトに対する最後の参照が除去される場合、ガーベッジコレクターは、オブジェクトによって占有されるメモリを回収し、マネージドオブジェクトを参照計数する必要性を軽減する。したがって、マネージド環境は、基本的に内部で参照計数を処理する。マネージド環境はオブジェクトに存在する未解決の参照の記録をとるため、追跡はマネージドコード内部で起こり得る。新しいオブジェクト参照がマネージドコード内部で各々宣言されると、マネージド環境は、ライブ参照のリストにその参照を加える。

【０００６】

所定の時間に、オブジェクト自身ではなくマネージド環境が、所定のオブジェクトに存在するライブ参照を認識する。参照がスコープからはずれるか又は値を変更すると、ライブ参照のリストは更新される。そして、参照がマネージドコード内にとどまる限り、マネージド環境はそれを追跡可能である。上述のオブジェクトライフタイムの課題のほかに、マネージドオブジェクトシステムとアンマネージドオブジェクトシステムは、一般的に、他の多くの重要な面で相違する。これらの相違には、例えば、オブジェクトシステムによる各オブジェクトシステム内部のオブジェクトインタフェースの提供の方法、データの構造化の方法及び／又は定義の方法、及び、エラーや例外の処理の方法が含まれる。

【０００７】

オブジェクト実行環境に関しては、動的プログラミング言語は、複数のアプリケーショ

ンを開発するための多種多様なコード型を提示する。Ｐｅｒｌ、Ｓｃｈｅｍｅ、Ｒｕｂｙ、Ｐｙｔｈｏｎ、Ｓｍａｌｌｔａｌｋ等の動的に型付けされた言語は、伝統的に様々なタグ付けスキーマを利用して、ヒープ上に小さな（通常は語長の）オブジェクトを割り当てるオーバーヘッドを克服するが、今もなおパラメータ操作に対する数値データの統一的表現の利益を持ち続けている。例えば、ポインターが４バイト境界で位置合わせされていると仮定すると、共通的手法は、３２ビット数値の最下位ビットを用いて、ポインターと即時値とを区別する。ウィンドウベースのシステムでは、例えば、ポインターが最高のメモリーセグメントの中でポイントすることは想定されないため、２つの最上位ビットを１に設定することが考えられる。

【発明の開示】

【発明が解決しようとする課題】

【０００８】

コード化（例えば、ポインターと整数値を区別するコード化）を使用するプログラムは、共通言語ランタイム（Ｃｏｍｍｏｎ　Ｌａｎｇｕａｇｅ　Ｒｕｎｔｉｍｅ）又はＪａｖａ（登録商標）仮想マシン等のマネージド実行環境で実行される場合は検証されないことが、多くのシステム設計者にとって明白である。現在のところ、統一的表現を実現する１つの検証手法は、ボクシング（ｂｏｘｉｎｇ）（例えば、ボクシングは、整数とオブジェクトとを関連付ける。）を利用することである。しかしながら、ボックス値（ｂｏｘｅｄ　ｖａｌｕｅｓ）上の動作は、内在する数値に直接取り組むことよりもプロセッサ実行性能の面で非常に遅い。

【課題を解決するための手段】

【０００９】

発明の幾つかの態様を基本的に理解させるため、以下に発明の要旨を簡略化して示す。この要旨は、発明の概要ではない。主要で重要な要素を識別すること又は発明の範囲を線引きすることを意図しない。唯一の目的は、後述する詳細な説明の前置きとして、単純化された形式で発明の概念を示すことにある。

【００１０】

本発明は、マネージドオブジェクト環境の動的プログラミング言語から生成されるタグ付き型の数値を処理するシステム及び方法に関する。タグ付き型の数値には、例えば、実行エンジンがポインターと即時データ数値とを区別できるようにする３２ビット数値の最下位ビット等の特別に符号化されたデータの一部が含まれる。本発明は、仮想マシンによって提供されるマネージドオブジェクト環境で実行可能な動的プログラミング言語に対する抽象ルート型クラスを定義する。この型のクラスは、クラスのトップポーションの下に定義されるデータ要素のオパーク（ｏｐａｑｕｅ）でナチュラルサイズの数値を表すクラス階層でトップポーションを有することとして定義される。

【００１１】

一例として、分岐ツリーは、クラスのトップポーション又はルートの下に作り出され、そこで、システムオブジェクト階層（例えば、非タグ付け要素）は、ツリーの一つの側に作り出され、タグ付け数値を表すシールド型は、ツリーの他の側に作り出される。次に、様々なルールが抽象ルート型クラスに適用されることにより、タグ付き型を使用する動的プログラミング言語の正常実行を容易化する（例えば、ユーザ定義型からタグ付き型へのデータアクセスをふさぐか止める）。より詳細には、実行ルールは、ユーザ定義の型と関連する数値又はデータを、抽象クラスのタグ付きメンバーからプロパティーを導くこと又は引き継ぐことから緩める。この手法によれば、タグ付き型数値は、型セーフ実行環境で実行できる。さらに、抽象クラスを定義し、クラスのメンバーへのアクセスを制限することによって、コード実行性能は、上述のボクシングプロトコルを介してタグ付き数値を使用する従来のアルゴリズム上で強化される。

【００１２】

本発明の一態様によると、タグ付けされた型の要素と型トップの要素は定義され、トップは、ツリーの１つの枝にタグ付けされた特殊型データを有し、ツリーの別の枝に非タグ

付けメンバーに関する特殊型データを有するツリーの最上位メンバーを定義する。メタデータルールを拡張することにより、非タグ付け枝のメンバーが、タグ付けされる特殊型又は型トップの要素からプロパティーを導かず、あるいは、引き出さないことを保証するのを助ける。ルールを拡張し、算術演算又はその他の型の演算をタグ付き数値に適用するような演算をサポートする。ルールは、スタック実行手続きの間に適用され、オブジェクトをトップエレメント型、及び／又は、タグ付き型にキャスティングするような演算を含む。これは、数値が個別クラスのメンバーか特殊型かを判定するテスト演算を含む。その他の実行ルールは、データの一つの型（例えば整数）をタグ付き型に変換すること及びその逆を含む。タグ付き数値と非タグ付き数値の両方を包含する抽象クラスを定義し、クラス特殊型間の継承を分離するルールを提供することによって、本発明は、動的プログラミング言語を実行する速くて安全な環境を提供する。

【発明を実施するための最良の形態】

【００１３】

以下の説明及び添付の図面を使って本発明の一態様を解説する。これらの態様は、発明が実施される様々な方法を表し、それらの全ての方法を本発明でカバーするつもりである。図面を考慮することにより、発明のその他の利点及び新規な特徴は、以下に述べる発明の詳細な説明から明白であろう。

【００１４】

本発明は、マネージドコード環境における動的プログラミング言語の実行を容易化するシステム及び方法に関する。動的プログラミング言語と関係する１又は複数のタグ付き数値の継承階層を宣言するクラスコンポーネントを提供する。タグ付き数値を実行する間に、ルールコンポーネントは、型セーフ・ランタイム・環境をサポートするためにタグ付き数値からプロパティーを引き継ぐ又は導くことからユーザ定義の型を緩和する。ツリーの一つの側に非タグ付き型の要素を定義し、ツリーの別の枝にタグ付きの型の要素数値を定義する分岐クラスツリーを提供する。ルールコンポーネントは、ツリーの別のコンポーネントからプロパティーを導く又は引き継ぐツリーの１コンポーネントからのデータを阻止するのを助けるランタイム拡張を分析する。ランタイム拡張は、キャストクラス拡張、テストクラス拡張、及び、１クラス特殊型から別の型へデータ型を変換する変換クラス拡張（例えば、タグ付き要素の非タグ付き要素への変換及びその逆）のような態様を含む。

【００１５】

本願では、「コンポーネント」、「クラス」、「階層」、「システム」等の用語は、コンピュータに関する構成要素、すなわち、ハードウェア、ハードウェアとソフトウェアの結合、ソフトウェア、実行中のソフトウェアを指す。例えば、コンポーネントは、限定はされず、プロセッサ上を走るプロセス、プロセッサ、オブジェクト、実行ファイル、実行スレッド、プログラム、及び／又はコンピュータである。実例として、サーバ上を走るアプリケーション及びそのサーバは、いずれもコンポーネントであり得る。１又は複数のコンポーネントは、実行プロセス又は／及び実行スレッドの内部に常駐し、コンポーネントは、１つのコンピュータ上に集中して置かれてもよいし、及び／又は、２又はそれより多いコンピュータ間で分散されてもよい。

【００１６】

最初に図１を参照すると、システム１００は、本発明の態様によるタグ付き型処理を示す。１又は複数の動的言語１１０は、仮想マシン１３０により実行される中間言語命令１２４を生成するコンパイラー１２０（例えば、タイムコンパイラー（Ｔｉｍｅ　ｃｏｍｐｉｌｅｒ）の中）に入力される。動的言語１１０には、例えばコンパイラー１２０によってコンパイルされ仮想マシン１３０によって実行されるＰｅｒｌ、Ｓｃｈｅｍｅ、Ｒｕｂｙ、Ｐｙｔｈｏｎ及びＳｍａｌｌｔａｌｋ等のあらゆる種類のコンピュータ言語が実質的には含まれる。そのようなマシンには、例えば、仮想実行システム（Ｖｉｒｔｕａｌ　Ｅｘｅｃｕｔｉｏｎ　Ｓｙｓｔｅｍ；　ＶＥＳ）、共通言語ランタイム（Ｃｏｍｍｏｎ　Ｌａｎｇｕａｇｅ　Ｒｕｎｔｉｍｅ；　ＣＬＲ）、又は、Ｊａｖａ（登録商標）仮想マシン等が含まれる。仮想マシン１３０は、中間言語コードを実行すると、（ローカル及び／又

はリモートネットワークシステムであり得る）単一コンピュータシステム又は複数のコンピュータシステム上で１又は複数のコンピュータアプリケーション１３４を動かす。本発明の１態様においては、動的言語１１０は、宣言し、そして、コンパイルされ次に仮想マシン１３０上で実行される１又は複数のタグ付き型１４０を生成する。タグ付き型１４０は、ポインター数値と中間データ数値等のその他の数値とを区別し、動的言語１１０によって頻繁に使用されるコード化を示す。

【００１７】

タグ付き型１４０は、クラス構造の片側又はルートにタグ付き数値を備え、もう片側又はルートに非タグ付き数値又はユーザ定義数値を備える分岐クラス構造の中で宣言され、この型クラス構造は、図２を参照して以下により詳細に説明する。中間命令１２４及びタグ付き型１４０を処理するために、仮想マシン１３０及び／又はコンパイラー１２０は、１組の検証ルールを使用して、ユーザ定義データ型がタグ付きデータ型から導かれず又は継承されないことを保証するのを助ける。（以下に説明する）各種スタック実行ルール１６０は、タグ付き数値及び非タグ付き数値を処理するために、スタック例外ハンドラーによって修正され、実行される。新しいクラス型フレームワーク、検証ルール１５０及び／又はスタック例外ハンドリング１６０を備えることによって、本発明は、仮想マシン１３０の実行性能を改善する。このことは、「ボックス」変数としてタグ付き数値を扱う手法を軽減する代わりにクラスフレームワーク及びルールにしたがってタグ付き数値を処理することによって達成される。

【００１８】

これらの変数は、共通言語インフラストラクチャ（Ｃｏｍｍｏｎ　Ｌａｎｇｕａｇｅ　Ｉｎｆｒａｓｔｒｕｃｔｕｒｅ　；ＣＬＩ）．を定義する「ＥＣＭＡ標準」のような標準の中に記述される。

【００１９】

一般に、システム１００は、ＥＣＭＡスタンダードと共通言語インフラストラクチャに準拠し、そこでは、複数高レベル言語で記述されるアプリケーションを、アプリケーションを書き換えることで異なるシステム環境の固有の特徴を考慮する必要性なく、これらの環境において実行することができる。動的言語及び／又は仮想実行環境を処理する他の標準を、同様に、本発明に従って適用できると理解できる。ＥＣＭＡ標準は、幾つかの節から構成され、別々の節の中にこれらのコンポーネントを記述することによって様々なコンポーネントの理解を容易にするために、これらの節はインターネット上で簡単に入手できる。これらの節は、

パーティション　Ｉ：アーキテクチャ
パーティション　II：メタデータ定義及びセマンティックス
パーティション　III：CIL命令セット
パーティション　IV：プロファイル及びライブラリー
パーティション　Ｖ：付録

である。

【００２０】

共通言語インフラストラクチャ（ＣＬＩ）は、実行コード及びそれが実行される実行環境（仮想実行システム又はＶＥＳ）を提供する。実行コードは、モジュールとしてＶＥＳに渡される。モジュールは、通常、実行コンテンツを指定の形式で含んだ単一のファイルである。一般的には、コンパイラー、ツール及びＣＬＩ自体によって共有されるシングル型システム、すなわち共通型システム（Ｃｏｍｍｏｎ　Ｔｙｐｅ　Ｓｙｓｔｅｍ　；ＣＴＳ）は、共通言語インフラストラクチャの中心に位置する。それは、型を宣言、使用及び管理するときにＣＬＩが従うルールを定義するモデルである。ＣＴＳは、クロスーラングエッジ（ｃｒｏｓｓ－ｌａｎｇｕａｇｅ）統合、型セーフティ及び高性能コード実行を可能にするフレームワークを確立する。ＣＬＩは、以下に示す基本コンポーネントを含む。

【００２１】

（共通型システム（Ｃｏｍｍｏｎ　Ｔｙｐｅ　Ｓｙｓｔｅｍ））

共通型システム（ＣＴＳ）は、多くのプログラミング言語の中に見出される型及びオペレーションを支援するリッチ型システムを提供する。共通型システムは、広範囲のプログラミング言語を完全に実行することをサポートすることを目的とする。

【００２２】

（メタデータ（Ｍｅｔａｄａｔａ））

ＣＬＩは、メタデータを用いて、共通型システムによって定義される型を記述し、参照する。メタデータは、特定のプログラミング言語とは独立して記憶（「持続」）される。メタデータは、プログラムを操るツール（コンパイラー、デバッカー等）間及びこれらツールと仮想実行システムとの間で使用される共通的な交換機構を提供する。

【００２３】

（共通言語仕様（Ｃｏｍｍｏｎ　Ｌａｎｇｕａｇｅ　Ｓｐｅｃｉｆｉｃａｔｉｏｎ））

共通言語仕様は、言語設計者とフレームワーク（クラスライブラリー）設計者との間の取り決めである。それは、ＣＴＳ型システムのサブセットとユーセージ取り決めのセットを指定する。言語は、ＣＬＳの一部であるＣＴＳのこれらの部分を少なくとも実行することによってフレームワークにアクセスする最大能力をこれらのユーザに提供する。同様に、フレームワークは、公表された態様（クラス、インタフェース、方法、フィールド等）がＣＬＳの一部である型を用いて、そして、ＣＬＳ取り決めに従う場合に、最も広く用いられるであろう。

【００２４】

（仮想実行システム（Ｖｉｒｔｕａｌ　Ｅｘｅｃｕｔｉｏｎ　Ｓｙｓｔｅｍ））

仮想実行システムは、ＣＴＳモデルを実装し、実行する。ＶＥＳは、ＣＬＩのために書かれたプログラムをロードし走せる役割を担う。それは、メタデータを使用して別々に生成されたモジュールをランタイム（遅い結合）に一緒に結合して、マネージドコード及びデータを実行するために必要とされるサービスを提供する。

【００２５】

ＣＬＩのこれらの態様は共に、分散型のコンポーネント及びアプリケーションの設計、開発、展開及び実行のための統一フレームワークを形成する。共通型システム(Common Type System)の適切なサブセットは、ＣＬＩをターゲットとする各プログラミング言語から利用可能である。言語ベースツールは、メタデータを使用してアプリケーションを構築するために用いられる型を定義し参照して、お互いに及び仮想実行システムと通信を行う。仮想実行システムは、メタデータを用いて、必要に応じて型のインスタンスを生成し、データ型情報をインフラストラクチャの他の部分（遠隔サービス、アセンブリダウンロード、セキュリティ等）に提供する。

【００２６】

図２を参照すると、ダイアグラム２００は、本発明の態様によるタグ付き型階層を示す。型セーフ方法でタグ付き数値をサポートするために、本発明は、トップ２１０として記される新しい（抽象）ルート型をオパキュー（ｏｐａｑｕｅ）でナチュラルサイズの数値を表す継承階層に提供する。この新しいルート２１０の下位において、現行のシステムオブジェクト階層２２０は、片側に宣言され、（密閉された）タグ付き型２３０は、もう一方の側に宣言される。型トップ２１０は、一般的に、ＥＣＭＡパーティションIIIで提供されるオブジェクト参照型Ｏの概念に対応する。本発明の主題は、Ｏを第一のクラス型に提供し、新しいタグ付き特殊型２３０を加える。２４０では、トップ２１０及びタグ付き型２３０として定義される要素への継承を許可しない１又は複数のメタデータ妥当性ルールが提供される。これらのルールを以下により詳細に説明する。トップ２１０の下位に２つの枝が描かれているが、他の枝を同じように備えることができることが理解されるであろう（例えば、システムオブジェクト及びタグ付き型の近傍又は下で定義される複数の枝）。

【００２７】

ＥＣＭＡパーティションIIIにおいて定義されるオブジェクト参照（型Ｏ）は、通常、完全にオパークであると考えられる。本発明は、以下に記述するように、タグ付き型の算

術演算を提供するが、オブジェクト参照をオペランドとして認める算術命令は通常存在しない。許可される比較操作は、テストクラス命令を提供する本発明を備えるオブジェクト参照間の等式（および不等式）である。ＥＣＭＡパーティションIIIのオブジェクト参照上で定義される変換操作はない。しかし、本発明は、タグ付き数値及び非タグ付き数値間の変換を提供する。オブジェクト参照は、通常、所定のＣＩＬオブジェクト命令（例えば、ｎｅｗｏｂｊ及びｎｅｗａｒｒ）によって作成される。これらの参照は、例えば、引数として渡され、ローカル変数として格納され、数値として返され、配列内に及びオブジェクトのフィールドとして格納される。本発明は、ＣＬＲの使用を実質的に容易にする共通ランタイム言語（ＣＬＲ）を真のマルチ言語ランライムとして明白にサポートすることを提供する。発明の一特殊態様は、拡張（例えば、算術操作、検証ルール）の追加を提供し、同様に動的言語をサポートする（例えば、オープンソースアプリケーション）。本発明は、オパークでナチュラルサイズの数値を表す継承階層を備える抽象ルート型（例えば、トップ）の使用を介して型セーフ手法におけるタグ付き数値をサポートする。本発明は、各種タグ化手法の使用の必要性と関連する従来の問題を軽減し、小さな（例えば、文字サイズ）のオブジェクトをヒープ上に割り当てるオーバーヘッドを克服するが、パラメータ操作に関する数値の統一表示の利点を依然として維持する。

【００２８】

図３、図４及び図６～図９は、本発明による各種方法論を示す。説明を簡単化するため本方法論を一続きの動作として図示し説明するが、本発明によると一部の動作が違う順序で起き、及び／又は、他の動作と同時に起きるため、本発明が動作の順番によって限定されないことが本明細書に図示され記述されたことから理解されるであろう。例えば、方法論は、もう一つの方法として一連の相関する状態又はイベントとして状態図等に表現され得る、ということを当業者は理解するであろう。さらに、本発明による方法論を実行するために、図示した動作が全て必要とされるわけではない。

【００２９】

さて図３を参照すると、ダイアグラム３００は、本発明の一態様にしたがったタグ付き型処理を示す。３１０と３２０を進める前に、これらの動作がＥＣＭＡパーティションIIへの変更として提供され得ることに留意する。３１０においては、少なくとも２つの新しい要素型であるＥＬＥＭＥＮＴ＿ＴＹＰＥ＿ＴＡＧＧＥＤとＥＬＥＭＥＮＴ＿ＴＹＰＥ＿ＴＯＰを、タグ付き数値を処理する階層クラス構造の一部として定義する。３２０においては、メタデータ検証ルールは拡張され、ユーザ定義型がトップ（ＴＯＰ）又はタグ付き（ＴＡＧＧＥＤ）型要素から得ることができないことが保証される。上記の通り、ＣＬＩはメタデータを使用して、共通型システムによって定義される型を記述及び参照する。メタデータは、個々のプログラミング言語から独立した方法で保存（「持続」）される。したがって、メタデータは、プログラム（コンパイラー、デバッカー等）を操るツール間及びこれらのツールと仮想実行システム（Ｖｉｒｔｕａｌ　Ｅｘｅｃｕｔｉｏｎ　Ｓｙｓｔｅｍ）の間で使われる共通交換機構を提供する。メタデータトークンは、通常、例えば、メターデータテーブルの行を指定する４バイト数値、又はユーザーストリングヒープ（Ｕｓｅｒ　Ｓｔｒｉｎｇ　ｈｅａｐ）の開始バイトオフセットを記述する。

【００３０】

処理を行う前に、以下のことをＥＣＭＡパーティションIIIへの変更の一部として提供できる。３３０においては、通常、　．ｒｅｆ接尾部を有する全ての命令バリアント（ｖａｒｉａｎｔ）は、タグ付き型数値を処理するために、型トップの数値をスタックにロードする。３４０において、本発明の別の態様では、算術演算を再定義し、タグ付き数値に取り組む。これを実現するために、パーティションIIIのセクション１．５の表を拡張（例えば、オペランドを変更し、型トップ及びタグで演算をし、算術演算の間に検証ルールを拡張する。）。以下の表は、タグ付き型数値上で演算するために拡張されるＥＣＭＡパーティションIIIからの様々な命令を示す。

【００３１】

【表１】

Table 1: Binary Numeric Operations

A's Type	B's Type					
	int32	int64	native int	F	&	O
int32	int32	×	native int	×	& (add)	×
int64	×	int64	×	×	×	×
native int	native int	×	native int	×	& (add)	×
F	×	×	×	F	×	×
&	& (add, sub)	×	& (add, sub)	×	native int (sub)	×
O	×	×	×	×	×	×

【００３２】

【表２】

Table 2: Unary Numeric Operations

Operand Type	int32	int64	native int	F	&	O
Result Type	int32	int64	native int	F	×	×

【００３３】

【表３－１】

Table 3: Binary Comparison or Branch Operations

	int32	int64	native int	F	&	O
int32	✓	×	✓	×	×	×
int64	×	✓	×	×	×	×
native int	✓	×	✓	×	Beq[.s], bne.un[.s],	×

【００３４】

【表３－２】

					ceq	
F	×	×	×	✓	×	×
&	×	×	beq[.s], bne.un[.s], ceq	×	✓	×
O	×	×	×	×	×	beq[.s], bne.un[.s], ceq[2]

【００３５】

【表４】

Table 4: Integer Operations

	int32	int64	native int	F	&	O
int32	int32	×	native int	×	×	×
int64	×	int64	×	×	×	×
native int	native int	×	native int	×	×	×
F	×	×	×	×	×	×
&	×	×	×	×	×	×
O	×	×	×	×	×	×

【００３６】

【表５】

Table 5 : Shift Operations

		Shift-By					
		int32	**int64**	**native int**	**F**	**&**	**O**
To Be Shifted	**int32**	int32	×	int32	×	×	×
	int64	int64	×	int64	×	×	×
	native int	native int	×	native int	×	×	×
	F	×	×	×	×	×	×
	&	×	×	×	×	×	×
	O	×	×	×	×	×	×

【００３７】

【表６】

Table 6: Overflow Arithmetic Operations

	int32	int64	native int	F	&	O
int32	int32	×	native int	×	& add.ovf.un	×
int64	×	int64	×	×	×	×
native int	native int	×	native int	×	& add.ovf.un	×
F	×	×	×	×	×	×
&	& add.ovf.un, sub.ovf.un	×	& add.ovf.un, sub.ovf.un	×	native int sub.ovf.un	×
O	×	×	×	×	×	×

【００３８】

【表７】

Table 7: Conversion Operations

Convert-To	Input (from evaluation stack)					
	int32	**int64**	**native int**	**F**	**&**	**O**
int8 **unsigned int8** **int16** **unsigned int16**	Truncate	Truncate	Truncate	Truncate to zero[2]	×	×
int32 **unsigned int32**	Nop	Truncate	Truncate	Truncate to zero	×	×
int64	Sign extend	Nop	Sign extend	Truncate to zero	Stop GC tracking	Stop GC tracking
unsigned int64	Zero extend	Nop	Zero extend	Truncate to zero	Stop GC tracking	Stop GC tracking
native int	Sign extend	Truncate	Nop	Truncate to zero	Stop GC tracking	Stop GC tracking
native unsigned int	Zero extend	Truncate	Nop	Truncate to zero	Stop GC tracking	Stop GC tracking
All Float Types	To Float	To Float	To Float	Change precision[3]	×	×

【００３９】

尚、表１は２進数値演算、表２は単項数値演算、表３は２進比較又は分岐演算、表４は整数演算、表５はシフト演算、表６はオーバーフロー算術演算を表す。

【００４０】

図４を参照すると、ダイアグラム４００は、本発明の態様によるタグ付き型命令実行の例を示す。以下に記述する命令は、「加算」命令と関連するが、ＥＣＭＡと関連する命令又はその他の標準は、同様に拡張され、タグ付き型処理、又は、タグ付き処理及び非タグ付き処理の組み合わせに適用されることが理解できるであろう。４１０においては、クラス又は関数要素が宣言される。例えば、加算命令（又は他のオペランド）を有するリスト関数は、

【００４１】

【表８】

```
List{
      Add(Top  x);
```

【００４２】

として書くことができる。

【００４３】

４２０においては、タグ付き型及び非タグ付き型を宣言できる。例えば、

【００４４】

【表９】

```
List{
      Add(Top  x);
      Private  Top[ ] values;
      Private  (int,i);
```

【００４５】

である。

【００４６】

４３０においては、タグ付き型演算が実行される。例えば、

【００４７】

【表１０】

```
List{
    Add(Top x);
    private Top[ ] values;
    private (int,i);
      Add (Top x){
                values[i]=x;
                i= I + 1;
      }
}
```

【００４８】

である。

【００４９】

４４０に進み、スタック演算がタグ付き型処理の前、間、及び／又は後に実行される。４５０においては、タグ付き型演算又は関数は終了する。したがって、この例では、最後の例のリスト関数が、

【００５０】

【表１１】

```
List{
    Add(Top x);
    private Top[ ] values;
    private (int,i);
      Add (Top x){
                values[i]=x;
                i= I + 1;
      }
}
```

【００５１】

として現れる場合がある。

【００５２】

図５は、本発明の態様にしたがった１又は複数の実行ルール５００を示す。一態様においては、キャストクラス命令又はルールを５２０に備えることができる。この種類の命令は、オブジェクトをクラスにキャストし、より詳細には図６に関して記述される。別の態様においては、テストクラス命令又はルールを５２４に備えることができる。この種類の命令は、オブジェクト上でテストを実行し、オブジェクトがクラスのインスタンス又はインタフェースであるかどうかを判断し、より詳細には図７に関して記述される。さらに別の態様においては、整数をオブジェクト参照に変換する命令又はルールを５３０に備えることができる。この種類の命令（例えば、タグ命令）は、整数値をオブジェクト参照に変換し、より詳細には図８に関して記述される。さらにもう一つの態様においては、オブジェクト参照を整数に変換する命令又はルールを５３４に備えることができる。この種類の

命令（例えば、非タグ命令）は、オブジェクト参照を整数値に変換し、より詳細には図９に関して説明される。その他の命令又はルールを同様に定義できることが理解されよう（例えば、算術命令）。

【００５３】

図６から図９の説明に進む前に、以下のことを提供して、本明細書に記載されたスタック概念を説明するために次のことを示す。

【００５４】

（スタック遷移図）

スタック遷移図は、命令が実行される前後の評価スタックの状態を示す。以下は、代表的なスタック遷移図である。

【００５５】

【表１２】

…,value 1, value 2→…,result

【００５６】

このダイアグラム例は、スタックが少なくとも２つの要素をスタックにもたなければならないことを示す。定義上、最高の数値（「スタックの最上部」又は「直前にプッシュされた」）はｖａｌｕｅ　２と呼ばれ、（ｖａｌｕｅ２の前にプッシュされた）真下の数値は、ｖａｌｕｅ　１と呼ばれる。（このようなダイアグラムでは、スタックはページに沿って右に増大する。）命令は、スタックからこれらの数値を追い出し、この記述ではｒｅｓｕｌｔと呼ばれる別の数値にそれらを置き換える。

【００５７】

図６は、本発明の態様にしたがったキャストクラスルールを示す。

【００５８】

【表１３】

フォーマット	アッセンブリフォーマット	内容
74<T>	castclass class	Cast obj to class

【００５９】

castclass －　オブジェクトをクラスにキャスト

【００６０】

６１０では、スタック遷移は、キャストクラスのために、以下のように定義される。

【００６１】

【表１４】

Stack Transition:
…,obj → …,obj2

【００６２】

６２０では、キャストクラスは、以下のように定義され、“　”で囲った項目はＥＣＭＡパーティションIIIへの変更を示す。

【００６３】

キャストクラス命令は、オブジェクト（例えば、トップ）をクラスにキャストすることを試みる。クラスは、所望のクラスを示すメタデータトークン（ｔｙｐｅｒｅｆ　ｏｒ　ｔｙｐｅｄｅｆ）である。“クラスのトップのオブジェクトがタグ付きされておらず、Ｔがタグ付きされている場合には、無効例外が投じられる。”スタックのトップのオブジェクトのクラスが、クラスを実行しない場合（クラスがインターフェースである場合）及びクラスのサブクラスでない場合（クラスがレギュラークラスである場合）、無効キャスト

例外（Ｉｎｖａｌｉｄ　Ｃａｓｔ　Ｅｘｃｅｐｔｉｏｎ）が投じられる。
１．　　配列は、システム配列（Ｓｙｓｔｅｍ　Ａｒｒａｙ）から継承される。
２．　　ＦｏｏをＢａｒにキャストできる場合、Ｆｏｏ［　］をＢａｒ［　］にキャストできる。
３．　　上記２の目的のために、ｅｎｕｍｓは、それらの内在する型として扱われる。したがって、Ｅ１とＥ２が、内在する型を共有するならば、Ｅ１［　］を、Ｅ２［　］にキャストできる。

【００６４】
オブジェクトがヌルならば、キャストクラスは成功しヌルを返す。この動きは、以下に記載のｉｓｉｎｓｔとは異なる。

【００６５】
６３０においては、キャストクラスの例外を以下に記載する。
オブジェクトがクラスにキャストできない場合には、無効キャスト例外が投じられる。

【００６６】
クラスを発見できない場合は、型ロード例外（Ｔｙｐｅ　Ｌｏａｄ　Ｅｘｃｅｐｔｉｏｎ）が投じられる。通常、これは、ランタイムにおいてではなく、ＣＩＬがネイティブコードに変換されるときに検出される。

【００６７】
６４０において、キャストクラスの検証可能性を記述する。

【００６８】
コレクトＣＩＬは、クラスが有効なＴｙｐｅＲｅｆ又はＴｙｐｅＤｅｆトークンであること、及び、ｏｂｊがヌル、オブジェクト参照又はタグ付き即時値であることを保証する。

【００６９】
図７は、本発明の態様によるテストクラスルールを示すダイアグラム７００である。

【００７０】
【表１５】

フォーマット	アセンブリフォーマット	内容
75<T>	insist class	obj.がクラスのインスタンスかどうかをテストし、ヌル、又は、そのクラス又はインタフェースのインスタンスを返す。

【００７１】
insist — オブジェクトがクラス又はインタフェースのインスタンスかどうかをテスト

【００７２】
７１０において、以下のように、スタック遷移はキャストクラスのために定義される。

【００７３】
【表１６】

Stack Transition:
…,obj → …,result

【００７４】
７２０において、キャストクラスは以下のように定義され、“　”で囲まれた事項は、ＥＣＭＡパーティションＩＩＩへの変更を示す。

【００７５】

ｉｓｉｎｓｔ命令は、ｏｂｊ（型トップ）がクラスのインスタンスかどうかをテストする。クラスは、所望のクラスを示すメタデータトークン（ｔｙｐｅｒｅｆ又はｔｙｐｅｄｅｆ）である。“スタックのトップにあるオブジェクトがタグなしで、Ｔがタグ付きである場合に、ヌルがスタックにプッシュされる。”スタックのトップにあるオブジェクトのクラスがクラスを実行し（クラスがインタフェースの場合）、又は、クラスのサブクラスである（クラスがレギュラークラスの場合）場合、あたかもキャストクラスがコールされていたかのように、それは、型クラスにキャストされ、結果は、スタック上にプッシュされる。さもなければ、ヌルは、スタック上にプッシュされる。ｏｂｊがヌルの場合、ｉｓｉｎｓｔがヌルを返す。

１．　配列は、システム配列から継承する。

２．　ＦｏｏをＢａｒにキャストできる場合、Ｆｏｏ［］をＢａｒにキャストできる。

３．　２の目的のため、ｅｎｕｍｓは、内在する型として扱われる。したがって、Ｅ１とＥ２が、内在する型を共有する場合、Ｅ１［］を、Ｅ２［］にキャストできる。

【００７６】

７３０において、テストクラスに対する例外は、以下のように記述される。

クラスを発見できない場合は、型ロード例外（Ｔｙｐｅ　Ｌｏａｄ　Ｅｘｃｅｐｔｉｏｎ）が投じられる。通常、これは、ランタイムにおいてではなく、ＣＩＬがネイティブコードに変換されるときに検出される。

【００７７】

７４０において、キャストクラスの検証可能性を記述する。

コレクトＣＩＬは、クラスが、クラスを示す有効なＴｙｐｅＲｅｆ又はＴｙｐｅＤｅｆトークンであること、及び、ｏｂｊは常に、ヌルまたは、オブジェクト参照のいずれか一方であることを保証する。

【００７８】

図８は、本発明の態様による整数からタグ付き型への変換ルールを示すダイアグラム８００である。

【００７９】

【表１７】

フォーマット	アセンブリフォーマット	内容
OPCODE_TAG	Tag[.ovf]	整数iをタグ付きオブジェクト参照に変換

【００８０】

tag[.ovf] ― 整数をオブジェクト参照に変換

【００８１】

８１０において、スタック遷移は、タグクラスに対して以下のように定義される。

【００８２】

【表１８】

Stack Transition:
…,I →…,obj

【００８３】

８２０において、タグは、以下のように定義される。

タグ命令は、タグなし数値ｉ（ネイティブ整数）を（型Ｏの）型トップのインスタンスに変換する。これは、特別タグビットをｉにセットすることによって達成される。

【００８４】

８３０において、タグに対する例外は、以下のように記述される。
数値タグビットが既にセットされている場合に、オーバーフロー例外（Ｏｖｅｒｆｌｏｗ　Ｅｘｃｅｐｔｉｏｎ）が投じられる。
【００８５】
８４０において、タグに対する検証可能性を記述する。
コレクトＣＩＬは、ｉが型ネイティブｉｎｔから出ていることを保証する。
【００８６】
図９は、本発明の態様によるタグ型から整数への変換ルールを示すダイアグラム９００である。
【００８７】
【表１９】

フォーマット	アセンブリフォーマット	内容
OPCODE_UNTAG*	Un Tag	タグ付きオブジェクト参照を 整数iに変換

【００８８】
ｕｎｔａｇ［．ｏｖｆ］　－　オブジェクト参照を整数に変換
尚、ｏｐｃｏｄｅｓ　ＯＰＣＯＤＥ＿ＴＡＧとＯＰＣＯＤＥ＿ＵＮＴＡＧは、有効なＣＬＩ　ｏｐｃｏｄｅｓの範囲から割り当てられるであろう（ＥＣＭＡパーティションIII、セクション１．２．１を参照）。
【００８９】
９１０において、スタック遷移は、タグなしクラスに対して以下のように定義される。
【００９０】
【表２０】

Stack Transition:
…,obj → …,i

【００９１】
９２０において、タグなしは、以下のように定義される。
タグなし（ｕｎｔａｇ）命令は、タグ付きオブジェクト参照をネイティブ整数に変換する。これは、特別タグビットをｏｂｊにアンセット（ｕｎ－ｓｅｔｔｉｎｇ）することによって達成される。
【００９２】
９３０において、タグなしに対する例外は、以下のように記述される。
ｏｂｊがタグ付き数値でない場合に、無効キャスト例外が投じられる。
ｏｂｊがヌルの場合に、ヌル参照例外が投じられる。
【００９３】
９４０において、タグなしに対する検証可能性が記述される。
コレクトＣＩＬは、ｏｂｊが型トップのオブジェクト参照等であり、タグ付き整数を表すことを保証する。
【００９４】
図１０を参照すると、本発明の様々な態様を実行する例示的環境１０１０は、コンピュータ１０１２を含む。コンピュータ１０１２は、処理ユニット１０１４、システムメモリ１０１６及びシステムバス１０１８を備える。システムバス１０１８は、限定はされないが、システムメモリ１０１６を備えるシステムコンポーネントを処理ユニット１０１４に結合する。処理ユニット１０１４は、各種利用可能なプロセッサのいずれかであり得る。デュアルマイクロプロセッサ及びその他のマルチプロセッサアーキテクチャもまた、処理ユニット１０１４として使用され得る。
【００９５】
システムバス１０１８は、メモリバス、メモリコントローラ、周辺バス又は外部バス、

及び／又は、ローカルバスを備える。ローカルバスは、限定はされないが、１１－ビットバス、インダストリアル・スタンダード・アーキテクチャ（Ｉｎｄｕｓｔｒｉａｌ Ｓｔａｎｄａｒｄ Ａｒｃｈｉｔｅｃｔｕｒｅ（ＩＳＡ））、マイクロチャネル・アーキテクチャ（Ｍｉｃｒｏ－Ｃｈａｎｎｅｌ Ａｒｃｈｉｔｅｃｔｕｒｅ（ＭＳＡ））、イクステンドアイエスエー（Ｅｘｔｅｎｄ ＩＳＡ（ＥＩＳＡ））、インテリジェント・ドライブ・エレクトロニクス（Ｉｎｔｅｌｌｉｇｅｎｔ Ｄｒｉｖｅ Ｅｌｅｃｔｒｏｎｉｃｓ（ＩＤＥ））、ベサローカルバス（ＶＥＳＡ Ｌｏｃａｌ Ｂｕｓ（ＶＬＢ））、周辺コンポーネントインターコネクタ（Ｐｅｒｉｐｈｅｒａｌ Ｃｏｍｐｏｎｅｎｔ Ｉｎｔｅｒｃｏｎｎｅｃｔ（ＰＣＩ））、ユニバーサ・シリアル・バス（Ｕｎｉｖｅｒｓａｌ Ｓｅｒｉａｌ Ｂｕｓ（ＵＳＢ））、アドバンスド・グラフィクス・ポート（Ａｄｖａｎｃｅｄ Ｇｒａｐｈｉｃｓ Ｐｏｒｔ（ＡＧＰ））、パーソナル・カード・インターナショナル・アソシエーション（Ｐｅｒｓｏｎａｌ Ｃａｒｄ Ｉｎｔｅｒｎａｔｉｏｎａｌ Ａｓｓｏｃｉａｔｉｏｎ ｂｕｓ（ＰＣＭＣＩＡ））及びスモール・コンピュータ・システム・インタフェース（Ｓｍａｌｌ Ｃｏｍｐｕｔｅｒ Ｓｙｓｔｅｍ Ｉｎｔｅｒｆａｃｅ（ＳＣＳＩ））を含む各種の有用なバスアーキテクチャを使用する。

【００９６】

システムメモリ１０１６は、揮発性メモリ１０２０と不揮発性メモリ１０２２とを備える。起動時等にコンピュータ１０１２内の要素間で情報を転送する基本ルーティンを含む基本入出力システム（ＢＩＯＳ）は、不揮発性メモリ１０２２に記憶される。図によると、制限はないが、不揮発性メモリ１０２２は、リードオンリーメモリ（ＲＯＭ）、プログラマブルＲＯＭ（ＰＲＯＭ）、エレクトロリカリ・プログラマブルＲＯＭ（ＥＰＲＯＭ）、エレクトロリカリ・イレーサブルＲＯＭ（ＥＥＰＲＯＭ）又はフラッシュメモリを含むことができる。揮発性メモリ１０２０は、外部キャッシュメモリとして動作するランダムアクセスメモリ（ＲＡＭ）を含む。図によると、制限はないが、ＲＡＭは、シンクロナウスＲＭＡ（ＳＲＡＭ）、ダイナミックＲＡＭ（ＤＲＡＭ）、シンクロナウスＤＲＡＭ（ＳＤＲＡＭ）、ダブルデータレートＳＤＲＡＭ（ＤＤＲ ＳＤＲＡＭ）、エンハンスドＳＤＲＡＭ（ＥＳＤＲＡＭ）、シンクリンクＤＲＡＭ（ＳＬＤＲＡＭ）及びディレクト・ランバスＲＡＭ（ＤＲＲＡＭ）等の多くの形式で得られる。

【００９７】

コンピュータ１０１２は、着脱式／非着脱式、揮発性／不揮発性のコンピュータ記憶媒体をも備える。図１０は、例として、ディスク記憶装置１０２４を示す。ディスク記憶装置１０２４は、制限はないが、磁気ディスクドライブ、フロッピー（登録商標）ディスクドライブ、テープドライブ、ジャズドライブ（Ｊａｚ ｄｒｉｖｅ）、ジップドライブ（Ｚｉｐ ｄｒｉｖｅ）、ＬＳ－１００ドライブ、フラッシュメモリカード又はメモリスティックのような装置を備える。さらに、ディスク記憶装置１０２４は、記憶メディアを単独で備えるか、制限はないが、コンパクト・ディスクＲＯＭデバイス（ＣＤ－ＲＯＭ）、ＣＤレコーダブルドライブ（ＣＤ－Ｒ Ｄｒｉｖｅ）、ＣＤリライタブルドライブ（ＣＤ－ＲＷ Ｄｒｉｖｅ）、ディジタル・バーサタイル・ディスク・ＲＯＭドライブ（ＤＶＤ－ＲＯＭ）等の光ディスクドライブを含むその他の記憶媒体と組み合わせて備える。ディスク記憶装置１０２４のシステムバス１０１８への接続を容易にするために、インタフェース１０２６等の着脱式又は非着脱式インタフェースが通常用いられる。図１０は、ユーザと、好適なオペレーティング環境１０１０に記述される基本コンピュータリソースとの間を媒介として動作するソフトウェアを記述する。ディスク記憶装置１０２４に記憶されるオペレーティングシステム１０２８は、制御及びコンピュータシステム１０１２のリソース割り当てのために動作する。システムアプリケーション１０３０は、システムメモリ１０１６内に又はディスク記憶装置１０２４上のどちらかに格納されるプログラムモジュール１０３２及びプログラムデータ１０３４を使ってオペレーティングシステム１０２８によってリソース管理をうまく行う。本発明が各種オペレーティングシステム又はオペレーティングシステムの組み合わせによって実行されることが理解できる。

【００９８】

ユーザは、入力装置１０３６を介してコマンド又は情報をコンピュータ１０１２に入力する。入力装置１０３６には、制限はないが、マウス等のポインティング装置、トラックボール、スタイラス、タッチパッド、キーボード、マイクロホン、ジョイスティック、ゲームパッド、サテライトディシュ、スキャナー、ＴＶチューナーカード、ディジタルカメラ、ディジタルビデオカメラ、ウェブカメラ等が含まれる。これら及びその他の入力装置は、インタフェースポート１０３８を介してシステムバス１０１８を使って処理ユニット１０１４に接続される。インタフェースポート１０３８には、例えば、シリアルポート、パラレルポート、ゲームポート及びユニバーサルシリアルバス（ＵＳＢ）が含まれる。出力装置１０４０は、入力装置１０３６として幾つか同じ種類のポートを使用する。したがって、例えば、ＵＳＢポートは、入力をコンピュータ１０１２に与えるために使用され、コンピュータ１０１２からの情報を出力装置１０４０に出力するために使用される。出力アダプター１０４２は、その他の出力装置１０４０の中に、特別なアダプターを必要とするモニター、スピーカー及びプリンターのような出力装置１０４０が幾つかあることを示すために備えられる。

【００９９】

出力アダプター１０４２には、図に示すように、制限はないが、出力装置１０４０とシステムバス１０１８との間の接続手段を備えるビデオカード及びサウンドカードが含まれる。その他の装置、及び／又は、装置のシステムは、リモートコンピュータ１０４４等の入出力機能を備える。

【０１００】

コンピュータ１０１２は、リモートコンピュータ１０４４等の１又は複数のリモートコンピュータとの間の論理的接続を用いてネットワーク環境で動作する。リモートコンピュータ１０４４は、パーソナルコンピュータ、サーバ、ルータ、ネットワークＰＣ、ワークステーション、マイクロプロセッサベースの装置、ピアー装置又は他の共通ネットワークノード等であり、通常、コンピュータ１０１２と関連して説明された多くの要素又は全ての要素を含む。簡潔にするために、メモリ記憶装置１０４６だけをリモートコンピュータ１０４４とともに図示する。リモートコンピュータ１０４４は、ネットワークインタフェース１０４８を介して論理的にコンピュータ１０１２に接続され、次に、通信接続１０５０を介して接続される。ネットワークインタフェース１０４８には、ローカルエリアネットワーク（ＬＡＮ）、広域ネットワーク（ＷＡＮ）等の通信ネットワークを包含する。ＬＡＮ技術は、ＦＤＤＩ、ＣＤＤＩ、Ｅｔｈｅｒｎｅｔ（登録商標）／ＩＥＥＥ １１０２．３、Ｔｏｋｅｎ Ｒｉｎｇ／ＩＥＥＥ１１０２．５等が含まれる。ＷＡＮ技術には、ポイントツーポイントリンク、ＩＳＤＮのような回路スイッチングネットワーク及び変形例、パケットスイッチングネットワーク及びＤＳＬが含まれる。

【０１０１】

通信接続１０５０は、ネットワークインタフェース１０４８をバス１０１８に接続するために使用されるハードウェア／ソフトウェアを参照する。通信接続１０５０は、図を明快にするために、コンピュータ１０１２内部に示されるが、それは、コンピュータの外部にあってもよい。単に例示目的であるが、ネットワークインタフェース１０４８に接続するために必要なハードウェア／ソフトウェアには、標準電話機グレードのモデム、ケーブルモデル及びＤＳＬモデム、ＩＳＤＮアダプター及びイーサネット（登録商標）カードが含まれる。

【０１０２】

図１１は、本発明と相互に作用し合う例示的なコンピュータ環境１１００の概略ブロック図である。このシステム１１００は、１又は複数のクライアント１１１０を含む。クライアント１１１０は、ハードウェア及び／又はソフトウェア（例えば、スレッド、プロセス、コンピューティング装置）であり得る。システム１１００は、１又は複数のサーバ１１３０も含む。サーバ１１３０は、ハードウェア及び／又はソフトウェア（例えば、スレッド、プロセス、コンピューティング装置）であり得る。例えば、サーバ１１３０は、スレッドを格納し、例えば、本発明を使用することによって変換を実行する。クライアント

１１１０とサーバ１１３０間で見込まれる一つの通信は、２又は３以上のコンピュータプロセス間で伝送されるように適合化されたデータパケットの形式で行われ得る。システム１１００は、クライアント１１１０とサーバ１１３０との間の通信を容易にするために使用される通信フレームワーク１１５０を含む。クライアント１１１０は、クライアント１１１０の情報を保存するために使用される１又は複数のクライアントデータ格納部１１６０に接続される。同様に、サーバ１１３０は、サーバ１１３０の情報を保存するために使用される１又は複数のサーバデータ格納部１１４０に接続される。

【０１０３】

上述した内容は、本発明の例を含む。言うまでもなく、本発明を説明する目的で考え付く全てのコンポーネントの組み合わせ又は方法論を記述することはできないが、当業者であれば、さらに多くの本発明の組み合わせ及び置換が可能であることを認識する。したがって、本発明は、添付する特許請求の範囲の精神及び範囲に含まれる全てのそのような変更、改良及び変形を包括的にとらえるつもりである。

【図面の簡単な説明】

【０１０４】

【図１】本発明の態様によるタグ付き型処理システムを示すブロック図である。
【図２】本発明の態様によるタグ付き階層を示すブロック図である。
【図３】本発明の態様によるタグ付き型処理を示すフローチャートである。
【図４】本発明の態様によるタグ付き型命令実行を示すフローチャートである。
【図５】本発明の態様による実行ルールを示す図である。
【図６】本発明の態様によるキャストクラスルールを示すフローチャートである。
【図７】本発明の態様によるテストクラスルールを示すフローチャートである。
【図８】本発明の態様による整数からタグ付き型への変換ルールを示すフローチャートである。
【図９】本発明の態様によるタグ付き型から整数への変換ルールを示すフローチャートである。
【図１０】本発明の態様に適したオペレーティング環境を示すブロック図である。
【図１１】本発明と相互作用するコンピュータ環境の例を示すブロック図である。

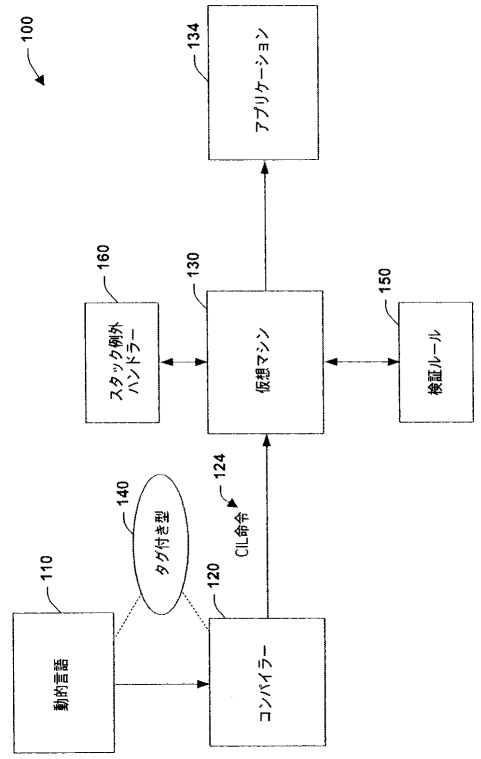
【符号の説明】

【０１０５】

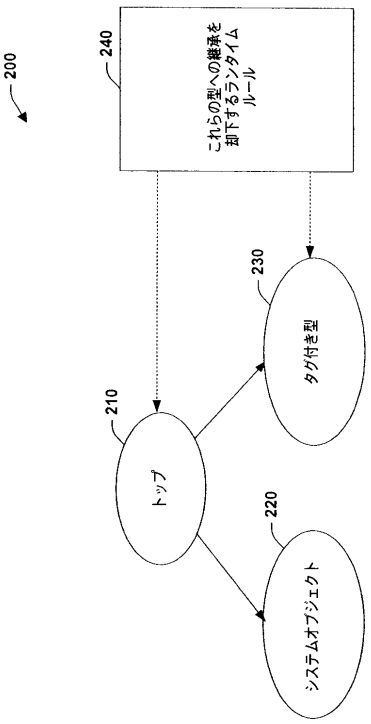
１１０　動的言語
１２０　コンパイラー
１２４　CIL命令
１３０　仮想マシン
１３４　アプリケーション
１４０　タグ付き型
１５０　検証ルール
１６０　スタック例外ハンドラー
２１０　トップ
２２０　システムオブジェクト
２３０　タグ付き型
２４０　これらの型への継承を却下するランタイムルール
５００　実行ルール
５２０　キャストクラス
５２４　テストクラス
５３０　整数をオブジェクト参照に変換
５３４　オブジェクト参照を整数に変換
１０２８　オペレーティングシステム
１０３０　アプリケーション
１０３２　モジュール

１０３４　データ
１０１４　処理ユニット
１０１６　システムメモリ
１０２０　揮発性
１０２２　不揮発性
１０２６　インタフェース
１０２４　ディスクストレージ
１０４２　出力アダプター
１０３８　インタフェースポート
１０５０　通信接続
１０４０　出力デバイス
１０３６　入力デバイス
１０４８　ネットワークインタフェース
１０４４　リモートコンピュータ
１０４６　メモリストレージ
１１１０　クライアント
１１３０　サーバ
１１４０　サーバデータ格納部
１１５０　通信フレームワーク
１１６０　クライアントデータ格納部

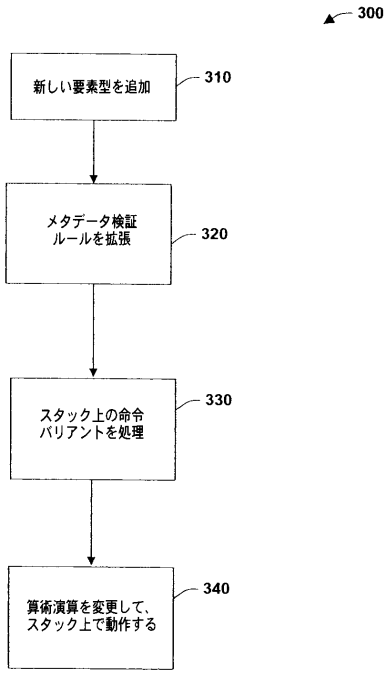
【図１】



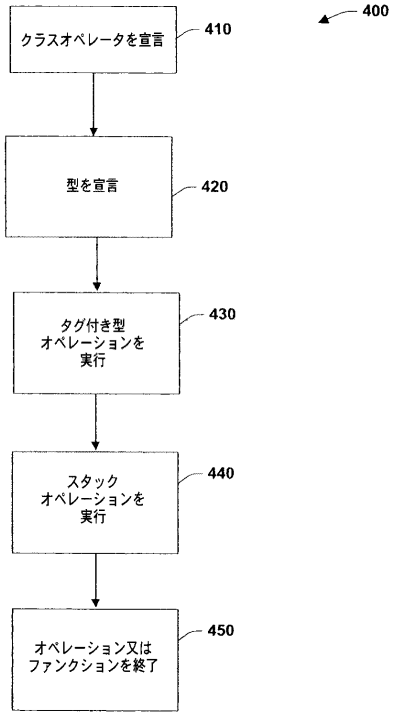
【図２】



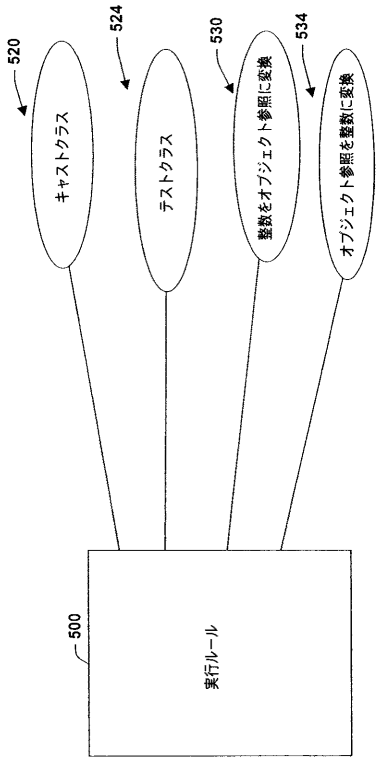
【図３】


300
新しい要素型を追加
310
メタデータ検証
ルールを拡張
320
スタック上の命令
バリアントを処理
330
算術演算を変更して、
スタック上で動作する
340

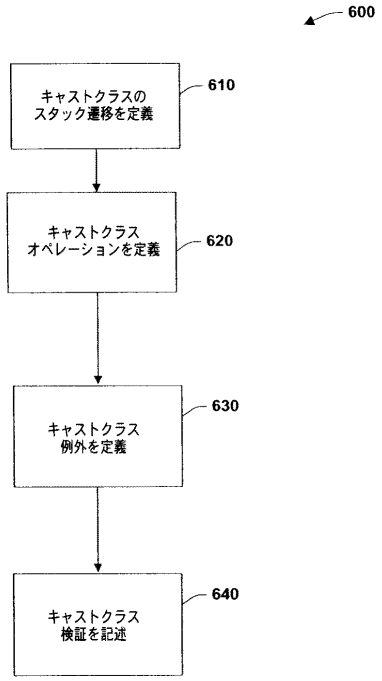
【図４】


400
クラスオペレータを宣言
410
型を宣言
420
タグ付き型
オペレーションを
実行
430
スタック
オペレーションを
実行
440
オペレーション又は
ファンクションを終了
450

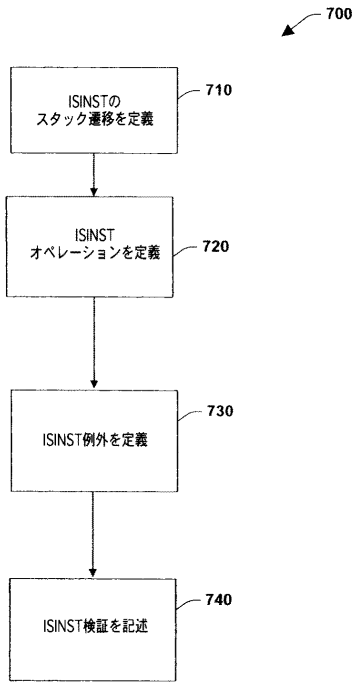
【図５】


520
キャストクラス
524
テストクラス
530
整数をオブジェクト参照に変換
534
オブジェクト参照を整数に変換
500
実行ルール

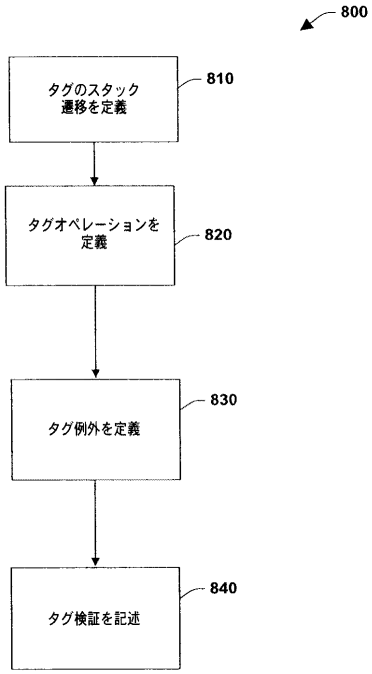
【図６】


600
キャストクラスの
スタック遷移を定義
610
キャストクラス
オペレーションを定義
620
キャストクラス
例外を定義
630
キャストクラス
検証を記述
640

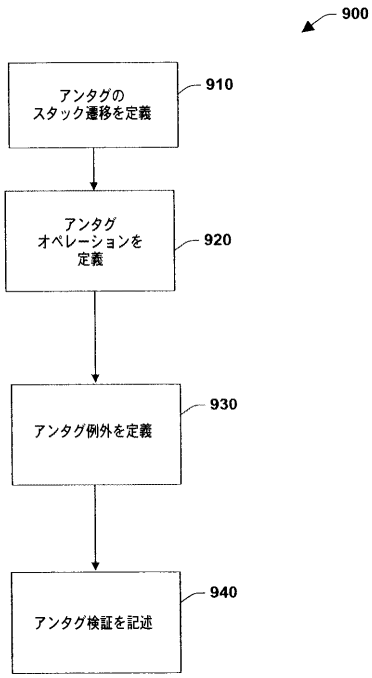
【図７】


700
ISINSTの
スタック遷移を定義
710
ISINST
オペレーションを定義
720
ISINST例外を定義
730
ISINST検証を記述
740

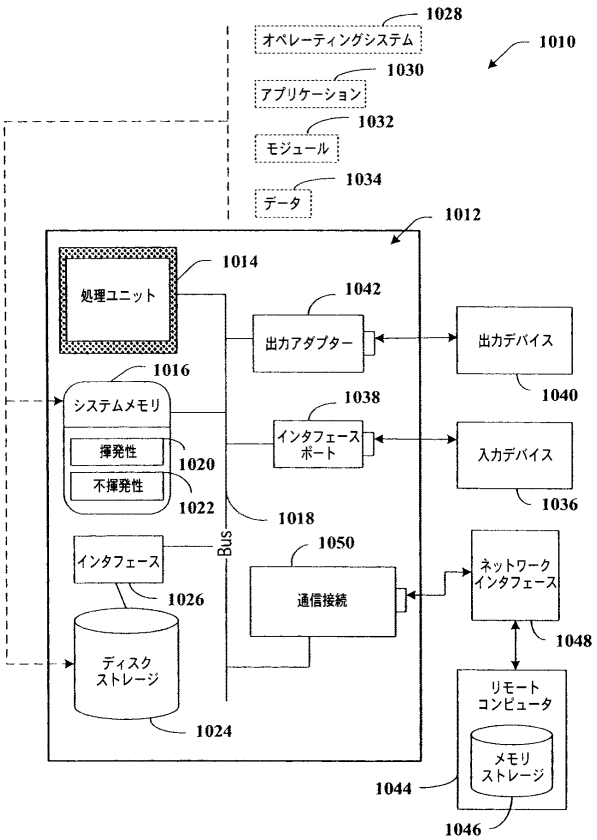
【図８】


800
タグのスタック
遷移を定義
810
タグオペレーションを
定義
820
タグ例外を定義
830
タグ検証を記述
840

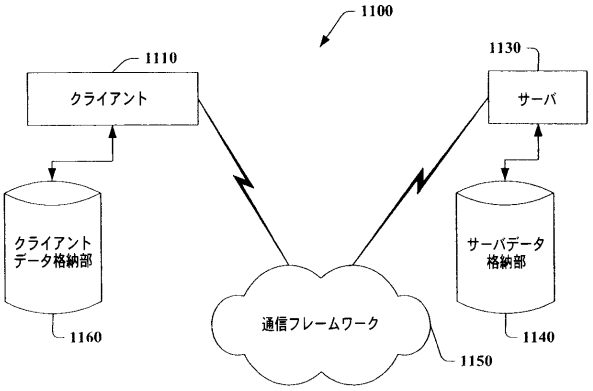
【図９】


900
アンタグの
スタック遷移を定義
910
アンタグ
オペレーションを
定義
920
アンタグ例外を定義
930
アンタグ検証を記述
940

【図１０】


1028
オペレーティングシステム
1010
1030
アプリケーション
1032
モジュール
1034
データ
1012
処理ユニット
1014
1042
出力アダプター
出力デバイス
1040
1016
システムメモリ
1038
揮発性
1020
インタフェース
ポート
入力デバイス
1036
不揮発性
1022
1018
Bus
インタフェース
1026
1050
通信接続
ネットワーク
インタフェース
1048
ディスク
ストレージ
1024
リモート
コンピュータ
メモリ
ストレージ
1044
1046

【図１１】


1100
1110
クライアント
1130
サーバ
クライアント
データ格納部
サーバデータ
格納部
通信フレームワーク
1160
1140
1150

フロントページの続き

合議体
審判長　山崎　達也
審判官　田中　秀人
審判官　石井　茂和

(58)調査した分野(Int.Cl., ＤＢ名)
G06F9/44