記　号　処　理　76－1
（１９９４．　１１．　１８）

クラス定義空間の多重化機能を実現したオブジェクト指向 Scheme

前畑 淳也　碓崎 賢一
九州工業大学 情報工学部

本論文では，クラス定義空間の多重化によりマルチメディアシステムの開発を支援するオブジェクト指向処理系について述べる．マルチメディア作品が多数作成されるようになると，それらを部品として，統合・再利用したいという要求が生じる．マルチメディア作品開発にはモデル化を容易とするオブジェクト指向技術が適しているが，従来のオブジェクト指向処理系では作品群を統合しようとするとクラス名が衝突するという問題が生じる．この問題を回避するため，複数のクラス定義空間を提供できるオブジェクト指向処理系の機能と実現方式を示す．提案方式では，クラス空間の多重化によって生じるオーバーヘッドを除去できる実現方式をとっており，処理効率を従来のオブジェクト指向処理系と遜色ないものにしている．

Object-Oriented Scheme with Multiple Class Definition Spaces

Jun'ya MAEHATA and Ken'ichi KAKIZAKI

Faculty of Computer Science and Systems Engineering
Kyushu Institute of Technology

This paper describes an object-oriented Scheme with multiple class spaces for multi-media application development. Generally, object-oriented techniques are used to develop multi-media applications, because it can model everything easily. When many multi-media applications exist, developers will want to reuse and integrate them to develop a new multi-media application. However, in a conventional language, the problem "class name collision" may occur, when we develop a new application by using some existing applications. To avoid the problem, we introduce an object-oriented Scheme with multiple class-definition-spaces. In this implementation, there is little overhead to realize multiple class-definition-space, so the performance is the same as conventional ones.

1 はじめに

近年，コンピュータの処理速度やコンピュータネットワークのデータ伝送速度の向上により，動画や音声などのマルチメディアデータを扱え，流通できるハードウェア環境が整いつつある．しかしながら，マルチメディアの扱いを支援するソフトウェア的なプラットフォームが存在していないために，作成されたマルチメディア作品を効率的に流通させることができないという問題が生じている．

3 次元グラフィックスなどに代表されるマルチメディア作品を作成するに当たって，オブジェクト指向の概念は対象のモデル化の容易さや対話性の向上に適している．このため，マルチメディア作品はオブジェクト指向に基づき開発されると考えられる．オブジェクト指向によって開発された作品群が流通しはじめると，優れた作品を統合・再利用して，新しい作品を容易に作成しようという要求が高まってくる．しかしながら，すでに作成された作品を部品として統合，再利用を行うことを支援するオブジェクト指向処理系は存在しない．

本論文では，マルチメディア作品の開発に対する要求に応えるために，LISP 系の Scheme [Cli91] を基本とし，クラス定義空間の多重化機能を持ったオブジェクト指向のマルチメディア用プラットフォーム [前畑 94] を提案し，その機能と実現方式について述べる．

2 クラス名の重複

複数作品の統合・再利用を行おうとすると，それぞれの作品で定義されているクラス名が重複する可能性がある．図 1 では作品 1 と作品 2 を統合して，作品 3 を作成しようとしている．楕円形はクラスを表している．それをつなぐ線はクラスの継承関係を示し，左側はスーパークラス，右側はサブクラスである．この例では，作品 1 にも作品 2 にも C という名前を持つクラスがあり，統合しようとすれば，クラス名が重なってしまうという問題が生じる．

従来のオブジェクト指向言語では，それぞれのクラスの名前はユニークなものでなければならない．したがって，マルチメディア作品の統合・再利用を行うためにはクラス名を変更する以外方法はない．名前の変更により作品の統合再利用を行うと，同じ作品でありながら各開発者ごとにバージョンの異なる作品が作成される．作品の流通を促進し，作品を部品として統合・再利用を行うことを考えると，同じ作品でありながら多数のバージョンが生じた作品の流通はその利用者に混乱を招く．したがって，作品を構成するクラスの名前を変更などによりバージョンを変化させず，オリジナルのまま作品を流通させることや修正できることが重要となる．

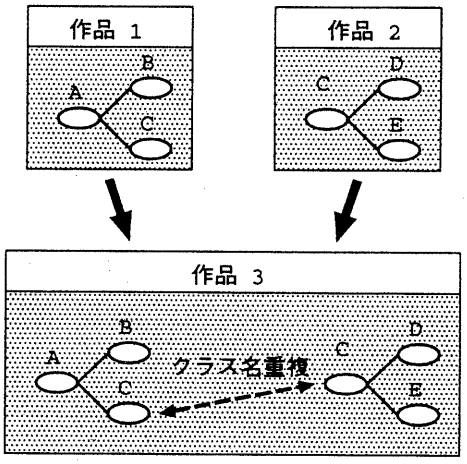

図 1: クラス名の重複

したがって，このような問題を解決するために，開発者がクラス名の重複に注意を払う必要のない機能を持つ，オブジェクト指向処理系を必要とする．

3 クラス定義空間多重化

3.1 ワールド

2 章の問題が発生する原因は，オブジェクト指向処理系のクラス定義空間が 1 つしか存在しないことにある．したがって，クラス定義空間を複数持たせることでこの問題を解決する方式を提案する．

提案方式ではワールドと呼ぶクラス定義空間を導入する．ワールドは，従来のオブジェクト指向の概念でいうクラス定義空間を 1 つだけ含んだものである．1 つの作品の中にはワールドをいくつか含むことができる．ワールドが異なれば同じ名前を持つクラスが同一作品に存在することができる．ワールドは，ソフトウェア部品としての特徴を持ち，入れ子構造に多重化することで，複数作品の統合と再利用を可能にしている．

Common Lisp [Ste90] では，作品の部品化が行えるように，パッケージという機能が備えられている．パッケージが異なると，名前が重なっても対象となる実体が異なるようになっている．また，パッケージ間で同一の実体に対して同じ名前をつけることも可能である．提案するワールドは，このパッケージの機能と同様な目的で，複数の開発者による開発を支援する．

2 つの作品をワールドを用いて統合した作品を図 2 に示す．図中に書かれている矩形がワールドを意味している．

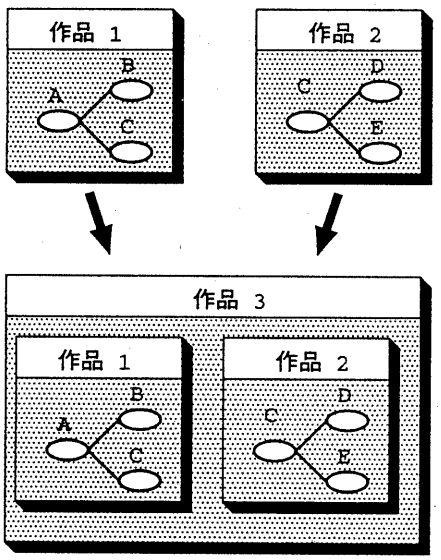

図 2: 作品の統合

ワールドを組み合わせると，ファイルのディレクトリ構造に似た階層構造を構成できる．ワールドの作成は必ず土台となるワールドを元に作成される．この場合，元になるワールドを**親ワールド** (スーパーワールド)，作成されるワールドを**子ワールド** (サブワールド) と呼ぶ．同じ親ワールド内に定義された複数の子ワールドを**兄弟ワールド**と呼ぶ．親ワールドの親ワールド，つまり，親ワールドを再帰的にたどる経路を祖先経路と呼ぶ．逆に，子ワールドを再帰的にたどる経路を子孫経路と呼ぶ．

3.2 クラス定義の選択

あるワールドで利用できるクラス定義は，そのワールド内に存在するクラスが最も優先されて選択されるようにしている．そのワールドにないクラスを利用しようとした場合には，親ワールドを再帰的にたどることにより利用するクラス定義が決定される．これにより親ワールドで定義されたクラスを共有し再利用できるようにしている．逆に子ワールドのクラス情報は参照しないようにしているため，ワールドはその子ワールドの内容に影響されることはない．このようにしているのは，子ワールドにおいて親ワールドの情報を変更したとしても，その変更が親ワールドには影響を与えないようにし，各子ワールドで独自の機能を定義できるようにするためである．また，兄弟の個別のクラス定義も影響しないようになっている．この仕組みにより，ワールド間のクラスの再利用を行うことが可能となり，ワールドごとの独立性も保てるようになる．

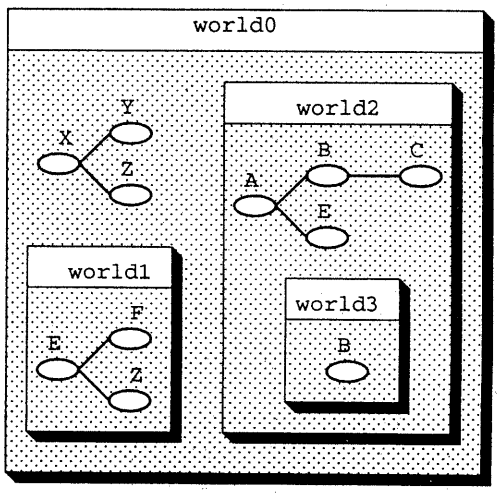

図 3: 3 層構造のワールド

図 3 に示す world1 から利用できるクラスを図 4 に示す．この場合は，はじめに world1 の 3 つのクラスが優先して利用される．続いて親ワールドの world0 のクラスが利用されるが，すでにクラス名 Z が world1 に存在するので，world1 から world0 のクラス Z が利用されない．クラスが破線で書かれているのはクラスの実体が存在しているにもかかわらず，利用されないことを示している．

図 3 に示す world2 から利用されるクラス定義を，図 5 に示す．world2 と world1 では同じクラス名が存在しないので，world2，world1 すべてのクラスを参照することができる．

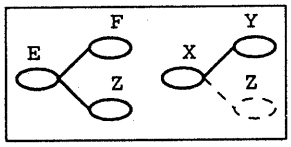

図 4: world1 から利用できるクラス定義

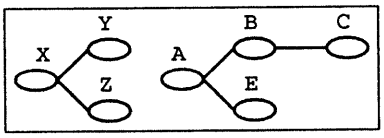

図 5: world2 から利用できるクラス定義

3.3 クラスの解釈

クラス定義空間を多重化することで生じる，従来のオブジェクト指向では定義されていない概念の解釈法について示す．

3.3.1 クラスの再定義

親ワールドですでに定義されているクラスを子ワールドで再定義して，その機能を拡張した場合のサブクラスへの影響について考える．図 3 の world2 と world3 の関係において，world3 では world2 のクラス B を再定義している．説明のために再定義されたクラス B を B' と表現する．world2 から利用できるクラス定義を 図 6 に示す．破線で表されたクラスは利用できないことを示している．クラスの継承関係を示す破線はクラス継承の候補を示す．この時，子ワールドにおいて再定義したクラス B のサブクラス C の解釈が，次に示す 3 種類考えられる．

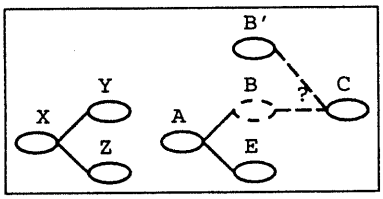

図 6: 再定義したクラスのサブクラスの問題

1. 子ワールドでクラス B' を再定義した時点で，親ワールドにおいて定義されているクラス B のサブクラスはすべて利用できなくなってしまう．つまり，クラス C は存在しないと解釈する．
2. クラス C のスーパークラスは，新しく子ワールドに定義されたクラス B' であると解釈する．
3. クラス C のスーパークラスは，親ワールドにもともと存在するクラス B であると解釈する．

1 の解釈では，再定義したクラスのサブクラスを考慮することがないので単純である．しかしながら，複数のマルチメディア作品を統合した場合に，あるクラスが偶然重複したために，そのサブクラスがすべて利用できなくなるというのは不都合が多い．

2 の解釈では，スーパークラスが変更されてもサブクラスが今まで通りに利用できる．それだけでなく，スーパークラスの修正がサブクラスに反映できるという特徴がある．しかしながら，この特徴のためにクラスが変更されたとき，インスタンス変数の名前変更や削除があると，サブクラスでそのインスタンス変数を利用していたメソッドがある場合に，エラーが生じるという安全性の問題がある．

3 の解釈では，クラス C は子ワールドに新しく定義されたクラス B' を参照していないため，クラス B' の性質は継承されない．このために，継承機能に制約が生じるものの，1 の解釈のような不都合や 2 の解釈のような安全性の問題は生じない．

このような特徴の比較から，提案システムでは異なるワールドのクラスの再定義は 3 の解釈を採用することにする．したがって，world3 で利用できるクラス定義は図 7 に，クラス C のスーパークラスは world2 で定義されているクラス B になる．

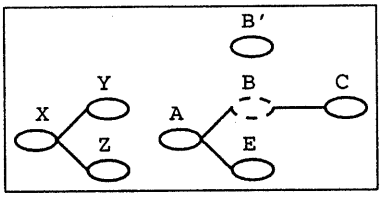

図 7: world3 から利用できるクラス定義

3.3.2 メソッド追加

3.3.1 節で示した親ワールドのクラスの機能拡張は，再定義したクラスにしか反映されていない．これではオブジェクト指向の継承という機能を犠牲にしており，オブジェクト指向の有効性を制限するものとなっている．この問題を解決するために，メソッドの追加によるクラスの機能拡張方式を提案する．

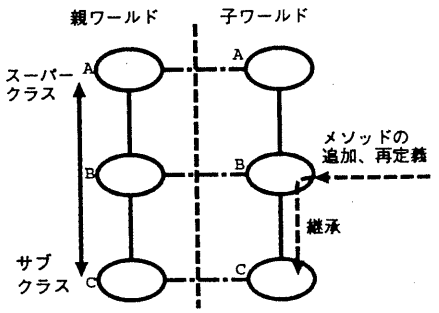

図 8: メソッド追加

LISP 系のオブジェクト指向言語の多くは，クラスの定義とメソッドの定義を分離して行うことができるため，すでに定義されているクラスに改めてメソッドを追加することが行えるという特徴がある．したがって，クラスの機能拡張を考える際に，インスタンス変数などの変更を必要としない場合には，メソッドの追加だけで賄うことができる．メソッドの追加は，3.3.1 節で問題としているクラスの内部構造の変更を伴わないために，サブクラスのメソッドも今まで通り利用することができる．したがって，本方式では，親のワールドで定義されているクラスの拡張機能を，子ワールドでのメソッドの追加で行えるようにしている．

図 8 では，親ワールドのクラスが仮想的に子ワールドに写像されていることを示している．中央の縦に引かれた破線は親ワールドと子ワールドとの境界を示している．親ワールドと子ワールドのクラスを結ぶ破線はクラスの写像関係を示している．クラス B に追加されたメソッドはサブクラス C にも継承されることを示している．このように，オブジェクト指向の特徴的な機能である継承を有効に利用できる方式となっている．

なお，親ワールドのクラスに対して子ワールドで行ったメソッド追加は，親ワールドおよび，兄弟ワールドに予想されない影響 (副作用) を起こしてはならないので，図 9 における world0 に対する world1, world2 のメソッド追加は親ワールドに対して影響を及ぼさないようにする．world1 と world2 の間にある実線は，world1 と world2 との境界を示し，互いに利用できないことを示している．この例のように異なるワールドから，同一のクラスに対するメソッドの追加を互いに独立させるための機構は，4.2.2 節で詳しく述べる．

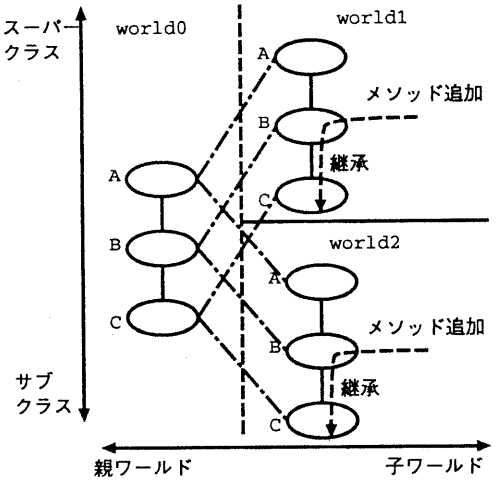

図 9: メソッド追加の影響

3.3.3 動的メソッド探索

はじめに，従来のオブジェクト指向言語でのメソッド探索について述べる．スーパークラスに定義されているメソッドはサブクラスで再定義できる．この場合，サブクラスに対するのメッセージ伝達では，再定義されたメソッドが呼び出される．対象となるクラスそのものにメソッドが定義されていない時には，そのスーパークラスからメソッドを探索しようと試みる．そして，最初に見つけられたメソッドが起動される．

提案方式では，親ワールドで定義されているメソッドも子ワールドで再定義することができる．この場合，メッセージ伝達は，クラスの継承方向に再定義されたメソッドを優先的に選択して実行する方式と，ワールドの多重化の方向に再定義されたメソッドを優先的に選択して実行する方式の 2 つの方法が考えられる．

ワールドの多重化の方向優先の探索方式は、図 10 に示すように，対象となるクラスからメソッドの探索を行う．そのクラスに存在しない場合には，そのクラスの仮想的な写像元のクラスからメソッドの探索を行う．そのようにして，写像元のクラスがなくなるまで親ワールド方向に探索を行う．それでも存在しない場合には，元のワールドに戻り対象となるクラスのスーパークラスから探索を試みる．そこに

もない場合には再び親ワールドからの探索を試みる．このようにして，最初に見つけられたメソッドを起動する．したがって，メソッド m を起動したい場合は，world0 のクラス B に存在するメソッドが起動される．

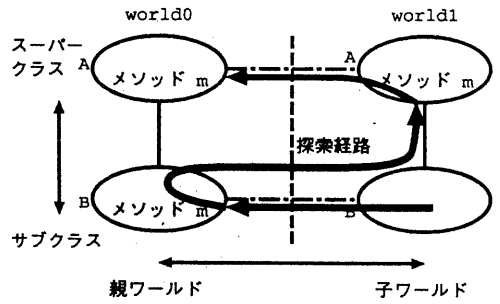

図 10: ワールド多重化方向優先探索方式

クラスの継承の方向優先の探索方式も，図 11 に示すように，はじめは対象となるクラスからメソッドの探索を行う．そのクラスに存在しない場合には，同一のワールド内のスーパークラスをたどって，メソッドの探索を行う．そのワールド内に存在しない場合には，親ワールドのクラスの中からメソッドの探索を試みる．このようにして，最初に見つけられたメソッドを起動する．したがって，メソッド m を起動した場合は，world1 のクラス A に存在するメソッドが起動される．

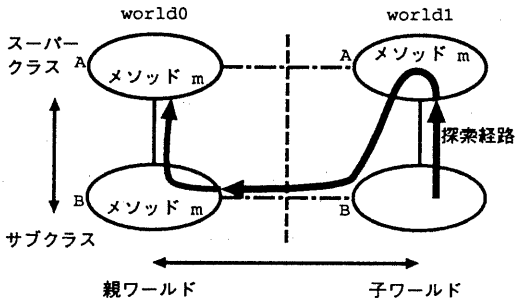

図 11: クラス継承方向優先探索方式

提案方式ではこの 2 つの探索方式から一方を選択するために，メソッドの再定義がどのような場合に行われるかを考える．スーパークラスに定義されるメソッドがサブクラスで定義されることについて考える．この方法による再定義は，再定義されるワールドだけでなく，そのワールドを親ワールドとするすべての子ワールドに波及されることを期待している．一方，親ワールドに存在するメソッドを子ワールドで再定義する場合には，兄弟ワールドには反映されない．したがって，この方法によるメソッドの方がより局所的であると考えられる．

したがって，提案方式ではより局所的なメソッドを実行できるようにするために，クラス継承方向優先の探索方式を採用する．

4 実現方式

提案方式の基本的なオブジェクト指向の実現に関しては TAO [大里 89] のそれを参考にしている．

4.1 ワールドの実現

ワールドごとに異なるクラス定義空間を提供するためにワールドベクタと呼ぶワールドが持つ情報を管理するベクタを用意する．ワールドベクタには各ワールドに固有の情報であるクラスの情報や，親ワールドや子ワールドに対してのリンク情報を格納する．

ワールドベクタは図 12 のような要素を持つ．

クラスベクタポインタ
ワールド名
スーパーワールドベクタポインタ
サブワールドベクタ表
原型クラスベクタ表
複写クラスベクタ表
クラスキャッシュ表

図 12: ワールドベクタの構造

提案方式では，クラス定義空間の多重化機能を実現するためのワールドもオブジェクトとして扱う．ワールドをオブジェクト指向の枠組の中で扱えるようにすることにより，操作性の統一と柔軟性を実現している．

4.2 クラスの実現

メソッドやインスタンス変数などのクラスの情報を管理するためのベクタを用意し，そのベクタをクラスベクタと呼ぶ．クラスベクタを参照することにより，クラスの情報のすべてを取得することができ

る．クラスベクタには，親ワールドのクラスやメソッドの再定義とその解釈法を実現するために，**原型クラスベクタ**と**複写クラスベクタ**の 2 種類のクラスベクタに分類される．

クラスベクタは図 13 のような要素を持つ．

ワールドベクタへのポインタ
スーパークラスへのポインタ
サブクラス表
原型ポインタ
クラス名
インスタンス変数表
インスタンス変数の高速探索表
インスタンス生成用データ表
メソッド表
メソッドキャッシュ表

図 13: クラスベクタの構造

4.2.1 原型クラスベクタ

原型クラスベクタは，一般的なオブジェクト指向処理系のクラス情報を管理する構造体と同じように利用される．クラスを定義すると，**原型クラスベクタ**が作成され，対象となるワールドのワールドベクタに登録される．作成時にクラスベクタに格納される情報はワールドベクタへのポインタ，スーパークラスへのポインタ，クラス名，インスタンス変数表のみで，残りの要素は `nil` に初期化される．残りの要素は後述する別の操作により具体化される．

4.2.2 複写クラスベクタ

親ワールドのクラスを子ワールドで利用する時に子ワールドに作成されるクラスベクタを，**複写クラスベクタ**と呼ぶ．複写クラスベクタは，親ワールドに影響しない独自のメソッドの定義を子ワールドで可能にするために作成される．

複写クラスベクタは親ワールドのクラスベクタを複写して作られる．原型クラスベクタと複写クラスベクタは，クラスの原型ポインタの値で区別することができる．原型クラスベクタではこの領域は `nil` であり，複写クラスベクタでは複写の元となったクラスベクタを指すポインタが格納されている．原型クラスベクタと複写クラスベクタは，図 14 に示すようにインスタンス変数表，インスタンス変数の高速探索表，インスタンス生成用データ表，メソッドキャッシュ表は共有する形を取る．複写クラスベクタのメソッド表のスロットは，はじめ `nil` に初期化されている．複写クラスベクタに対してメソッドが定義されると，図 15 に示すように，メソッドは複写クラスベクタが持つメソッド表に登録される．この時点から，メソッドキャッシュ表はそれぞれのクラスベクタで独自に管理される．これは，ワールド特有のメソッドを他のワールドに影響を及ぼさないためである．

複写クラスベクタは，子ワールドが作成された時点では生成されない．子ワールドで親ワールドに定義されているクラスのインスタンスが初めて生成された場合と，子ワールドで親ワールドに定義されているクラスに対して，初めてメソッドの追加や再定義を行った場合に作成される．この生成方法により，提案方式によるメモリ使用量の増加を抑えている．

4.3 メソッドの定義

定義されたメソッドは対象となるクラスベクタ中に含まれる**メソッド表**に登録される．登録の方法は対象となるクラスベクタの種類により操作が異なる．

対象となるクラスベクタが原型クラスベクタの場合に，メソッド表を持たない時はメソッド表を作成する．メソッド表がすでに存在するならば，そのメソッド表に登録するようにしている．一方，複写クラスベクタへの登録であった場合にメソッド表が存在すれば，そのメソッド表に登録するようにしている．メソッド表が存在しなければ，原型クラスベクタとメソッドキャッシュ表を共有しているので，複写クラスベクタは共有しているメソッドキャッシュ表の複製を新たに構成し，それにリンクを張るようにしている．メソッドキャッシュ表を原型クラスベクタと共有していると，子ワールドのみで利用されるべきメソッドが親ワールドに反映してしまうからである．その後に，メソッド表を新しく作成してメソッドをメソッド表に登録する．

4.4 インスタンス生成

インスタンスを生成する場合には，インスタンスの情報を格納したインスタンスベクタが作成される．インスタンスベクタの内容は，生成元となるクラスベクタへのポインタとインスタンス変数を格納している．

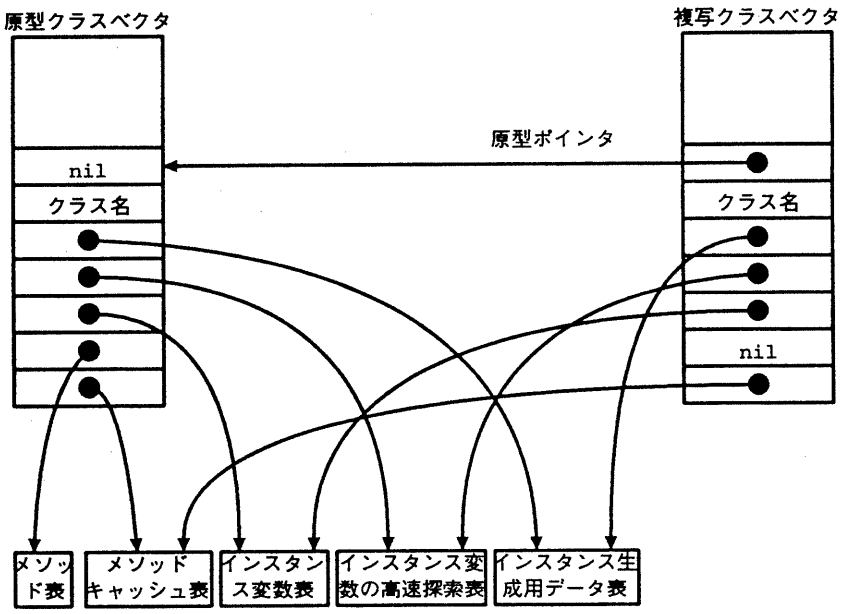

図 14: 複写クラスベクタ生成時に共有される表

インスタンスベクタは，その原型となるインスタンス生成用データ表をコピーして作成される．しかしながら，クラスを定義した時点では，インスタンス変数表しか存在していないので，クラスベクタにはインスタンス生成用データ表とインスタンス変数の高速探索表は作成されていない．クラスベクタに対して，初めてインスタンスが生成される時点でスーパークラスが再帰的に調査され，使用されるインスタンス変数の情報が集められる．この情報に基づき，インスタンス変数の高速探索表と，インスタンス生成用データ表が作成される．2 個目以降のインスタンスが作成される場合には，インスタンス変数の高速探索表やインスタンス生成用データ表を作成する必要がないので，高速にインスタンスを生成できるようにしている．

4.5 メッセージ伝達

インスタンスにメッセージが送られると，対象となるクラスベクタのメソッドキャッシュ表から対応するメソッドが起動される．

メソッドキャッシュ表にメソッドが登録されていない場合，そのクラスベクタが原型クラスベクタならば，その原型クラスベクタのメソッド表からメソッドの探索を試みる．存在しない場合にはさらにスーパークラスをたどってメソッドの探索を行う．クラスベクタが複写クラスベクタならば，その複写クラスベクタが存在するワールド内のスーパークラスをたどって，ワールド独自のメソッドの探索を行う．そのワールド内に存在しないならば，さらに親ワールドを再帰的にたどることにより，メソッドの探索を行う．メソッドが見つかった場合には，メソッドをコンパイルした後，対象となるメソッドキャッシュ表へ登録し，メソッドを実行する．このように，1 度起動されたメソッドは，メソッドキャッシュ表に登録されるため，メッセージ伝達が高速化される．

親ワールドで定義されているクラスのメソッドが子ワールドで利用される場合には，子ワールドには複写クラスベクタが作成される．子ワールドでメソッドの再定義や追加が行われない限り，原型クラスベクタのメソッドキャッシュ表を共有することができる．さらに，メソッドキャッシュ表が複数の子ワールドの複写クラスベクタから共有されている場合，いずれかの子ワールドで利用されたメソッドは，共有されているメソッドキャッシュ表に書き込まれるため，他の子ワールドで利用する場合にもメソッド探索が高速化されるという利点がある．

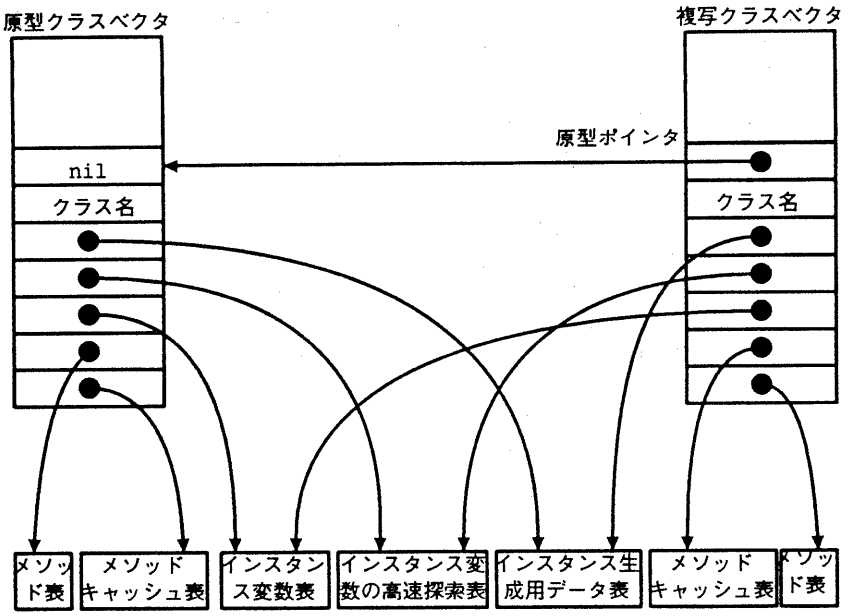

図 15: 独自メソッドにより，共有されないメソッド表とメソッドキャッシュ表

メソッドキャッシュ表が複数のクラスベクタから共有されている状態を図 16 に示す．この例では，world1 および world2 での複写クラスベクタにはメソッドが定義されていないので，原型クラスベクタとメソッドキャッシュ表を共有することができる．メソッドキャッシュ表を共有できるために，いずれかのクラスベクタがメソッドキャッシュに新しいメソッドを登録すると，そのメソッドキャッシュ表を共有しているすべてのクラスベクタに波及して，2 回目以降の同じメソッドに対するメッセージ伝達が高速化されることになる．複写クラスベクタがメソッド表を持つようになると，複写元のクラスベクタのメソッドキャッシュ表を共有できない．複写クラスベクタにメソッド探索が依頼されると，ワールド内で定義されているメソッドを優先的に探すようにし，書き込まれるメソッドキャッシュ表は，それぞれのワールドの複写クラスベクタの持つメソッドキャッシュ表となる．

5 考察

5.1 処理速度

動的結合を用いたオブジェクト指向言語の問題点は，メソッド探索の要するコストが大きいことである．このため，メソッドを高速に探索できるようにメソッドキャッシュを用いるのが一般的である．

この提案システムでもメソッドキャッシュの考え方を採り入れている．しかしながら，提案方式は複数のクラス定義空間を持つことから，キャッシュに登録されていない時のメソッドの探索空間は，クラスの継承方向とワールドの多重化の方向の，2 次元的空間からとなる．このことは従来のオブジェクト指向言語にはない探索空間の広さとなる．したがって最初のメソッド探索のコストは，従来のオブジェクト指向言語に比べて大きなものとなるが，1 度起動されたメソッドの探索コストは，メソッドキャッシュ表の効果により従来のオブジェクト指向言語と変わりない．

このことから，一般にメソッドは繰り返し利用されることを考えると，提案方式は従来のオブジェクト指向言語と遜色ないものとなっている．

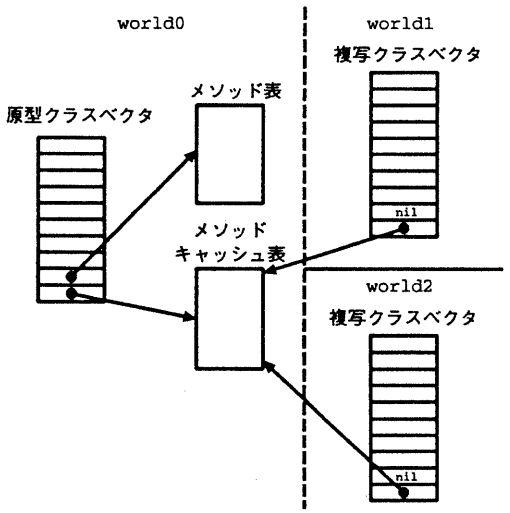

図 16: メソッドキャッシュの共有

5.2 メモリ効率

提案方式ではクラス定義空間であるワールドを多重化させることで，作品の統合・再利用を実現している．このために，ワールドごとにクラスを管理するクラスベクタのメモリ使用量の増加が考えられる．しかしながら，複写クラスベクタごとにはインスタンス変数表，インスタンス生成用データ表，インスタンス変数の高速探索表を独自に持たせる形にはしておらず，また，メソッドキャッシュ表なども原型クラスベクタと共有させている．複写クラスベクタの作成時期も，メソッドの追加やインスタンスの生成の時でないと作成されないようにしている．このように，メモリ消費量を抑えるように効率の良い実現方式を採用している．

5.3 他言語との比較

C++[Str91] を用いた作品の統合についての比較を行う．

C++ には，LISP 系の言語のようにインタプリタが装備されていない．このため，対話的にクラスやメソッドの追加修正を行うことができないため，開発時にモデルの変更や機能の変更をした場合，その効果を容易に確認することはできない．また，C++ は一系統のクラス定義空間しか提供できないため，マルチメディア素材の統合を行ってクラスの重複が起こった場合，開発者がクラスの重複が起こらないようにクラスの構成を変更しなければならない．したがって，クラスの集まりであるマルチメディア素材を統合することは困難であるといえる．

LISP 系の Scheme を基本とした本提案方式を用いれば，開発を容易にすることはもちろん，開発者がクラスの重複といった問題を気にせずに統合することも可能である．

6 まとめ

クラス定義空間の多重化機能を持つオブジェクト指向言語を提案し，その機能と実現方式について述べた．提案した方式は，独立して作成された複数のマルチメディア作品を統合できるため，オブジェクト指向によるマルチメディア作品が広く作成され流通するようになるにつれて，重要な役割を担うものと考えられる．

謝辞

議論に参加していただき有益な御助言をいただいた，田中賢一氏，松並勝氏をはじめとする砒崎研究室の諸氏に感謝します．

参考文献

[Cli91] Clinger, W. and Ress, J.: "Revised^4 Report on the Algorithmic Language Scheme", AI Memo 848b, MIT (1991).

[Ste90] Steele Jr., G. L.: "Common Lisp: The Language Second Edition", Digital Press (1990).

[Str91] Stroustrup, B.: "The C++ Programming Language, 2nd Ed.", Addison Wesley (1991).

[前畑 94] 前畑淳也, 砒崎賢一："クラス定義空間の多重化機能を持つオブジェクト指向マルチメディアプラットフォーム", 1994 年情報学シンポジウム, pp. 109–118 (1994).

[大里 89] 大里延康, 竹内郁雄："複合パラダイム言語 TAO におけるオブジェクト指向プログラミングとその実現", 情報処理学会 論文誌, Vol. 30, No. 5, pp. 596–604 (1989).