プログラミング 2−17
（1995. 8. 25）

オブジェクト指向分散環境OZ++のプログラミングパラダイム

音川英之(シャープ)* 西岡利博(三菱総合研究所)*
濱崎陽一(電子技術総合研究所) 首藤清法(システム21)*
中川 祐(富士ゼロックス情報システム)* 塚本享治(電子技術総合研究所)

* 開放型基盤ソフトウェアつくば研究室研究員

OZ++は広域のネットワーク上においてオブジェクトの交換と共有と共に、オブジェクトの実行コードを提供するクラスの共有を実現したオブジェクト指向分散環境である。クラスの共有を実現するにはクラスの改変に対して柔軟に対応できる仕組みとクラス名の衝突を回避する仕組みが必要であるが、OZ++ではそれぞれをクラスのバージョン管理とプログラミング時の局所的な名称空間の導入により解決している。本論文では、OZ++のプログラミングシステムの概要と、その上でのソフトウェア開発の方法について述べる。

The Programming Paradigm in OZ++: an Object-Oriented Distributed Environment

Hideyuki Otokawa(Sharp)* Toshihiro Nishioka(Mitsubishi Research Institute)*
Yoichi Hamazaki(Electrotechnical Lab.) Kiyonori Shutou(System Twenty One)*
Yu Nakagawa(Fuji Xerox Information Systems)* Michiharu Tsukamoto(Electrotechnical Lab.)

* Researcher, Tsukuba Laboratory, Open Fundamental Software Technology Project

OZ++ is an object-oriented distributed environment, in which not only objects are exchanged or shared but also classes are shared. To share the classes, it is necessary to deal with modification of them flexibly and to avoid conflicting their names. In OZ++, these problems are solved by managing the versions of classes and by using a local name space. This paper presents the OZ++ programming system and the processes of software development.

1 はじめに

OZ++は、広域ネットワーク上においてオブジェクトの交換と共有を実現したオブジェクト指向分散環境である[1]。さらにオブジェクトが動作するための実行コードを提供するクラスもネットワーク上で共有している。これによりオブジェクトのやり取りを行う上で相手側に必要な実行コードがあるかどうかを考慮する必要がなく、自由にオブジェクトのやり取りを行うことが可能である。またプログラミング時にも自由にネットワーク上で共有しているクラスを利用することができる。ところがクラスの共有を実現するためにはいくつかの問題がある。

一般にソフトウェアは利用されるにしたがって、バグの修正、性能の向上、機能追加等のために変更されるものである。ところが利用頻度の高いものほど、バグ情報や機能の向上の要求等が数多く報告され、変更の機会が増えることになる。一方、オブジェクト指向プログラミングには、クラスライブラリ等によってソフトウェアの再利用を積極的に行うという特徴があるが、再利用しているクラスが変更されると、利用する側のクラスがその影響で正しく動作しなくなる等の障害の発生する恐れがある。このことは広域ネットワーク上でクラスを共有するOZ++では、あるクラスの変更が広範囲に影響を及ぼす可能性があるため、深刻な問題である。OZ++ではクラスのバージョン管理を行うと共に新旧のバージョンの混在を許すことによりこれを解決している。

またクラスを広範囲で共有することにより、クラス名の衝突が起こる可能性が高くなるという問題もある。OZ++ではプログラミング時にシステム全体とは無関係な局所的なクラス名の名称空間を設定することで、他人とのクラス名の衝突を回避している。

さらにOZ++では、クラスの依存関係が明確であると共に、自由に必要なバージョンを選択して開発/実行することが可能であるため、複数人がクラスを共有しながら行う開発作業を支援することも容易である。

本論文では、2節でOZ++のプログラミングに関連するシステムの概要を、3節では実際のソフトウェア開発過程を、4節では複数人による協調開発の支援について述べる。また5節ではプログラミング環境の今後の拡張について述べる。

2 OZ++におけるプログラミング

OZ++では広域ネットワーク上においてクラスを共有し、他人の開発したクラスの利用を促進することで、ソフトウェアの部品化および再利用というオブジェクト指向プログラミングの特徴を有効に利用している。

以下では、OZ++のプログラミングに関連するシステムの構成要素について述べる。

2.1 プログラミング言語OZ++

OZ++言語は、C言語を元にしたオブジェクト指向プログラミング言語であり、オブジェクト指向の要素に加え、並列プログラミングや例外処理機能[3]を導入している。

クラスは多重継承であり、クラスメンバに対してはC++に見られるようなpublic, protected, privateのアクセス制御が可能である。

2.2 クラス管理システム

OZ++では、システム上でのクラスの共有を実現するためにクラスの分散管理を行っている。クラス管理システムでは、クラスにシステム全体で一意に識別可能なID(バージョンID)を付与し、このIDによって各クラスを管理する。

クラスに限らず一般にソフトウェアは、使用頻度の多いものほど作られたままであり続けることは少なく、バグの修正や機能拡張等のために何らかの変更が加えられるものである。ところがクラスを共有している場合、クラスの変更への対応が問題となる。あるクラスに対する変更が即座に完全にシステム全体に反映されてしまうと、そのクラスを利用していた他のクラスが動作できなくなる。このような場合、変更に対処するためには利用している側で再コンパイル等の何らかの対応が必要である。ところが仮に行われた変更がそのクラスの利用側には不必要であったとしても、利用側はそれを拒否することができないことになる。OZ++ではクラスを広域ネットワーク上で共有するため、あるクラスの変更による影響は広範囲に及ぶ可能性があり、このような問題はより深刻である。これを解決するために、クラスの変更をバージョンの更新と位置付け、クラスのバージョン管理の機構を導入すると共に新旧のバージョンの混在を可能にしている。クラスのバージョンは、クラスの機能を次のように三つに分類することによって、三つに分割しそれぞれに異なるバージョンIDを付与している。

1.メッセージパッシングを行うためのインタフェースの部分(パブリックパート)。publicメンバのシ

グネチャに相当する。publicメンバを利用しているクラスに対してこの部分の変更を反映させるためには再コンパイルが必要である。

2. 継承して利用するためのインタフェースの部分(プロテクティッドパート)。protectedのメソッドのシグネチャおよびインスタンス変数の型に相当する。このクラスのサブクラスに対してこの部分の変更を反映させるためには再コンパイルが必要である。

3. 1、2のインタフェースに対する実装部分(実装パート)。インスタンス変数定義もこれに相当する。この部分の変更は再コンパイルを行うことなく反映させることが可能である(後述)。

これらの三つのパートの間には、パブリックパート、プロテクティッドパート、実装パートの順で上下関係がある。各パートは一つの上位パートと複数の下位パートを持つ。下位パートの中のいずれか一つはデフォルトバージョンに指定することが可能である。デフォルトバージョンは利用者がバージョンを明示的に指定しない場合に利用される。またパブリックパートの上位パートとしてルートパートがあり、これはあるクラスのすべてのバージョンを代表するものである(図1参照)。

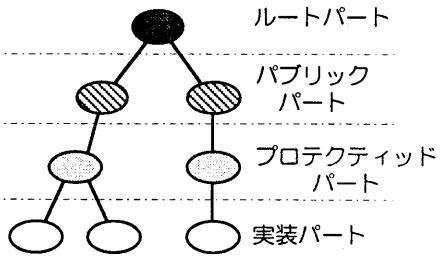

図1: バージョンの構造

2.3 言語処理系

OZ++システム全体ではクラスをIDによって識別するが、プログラミング時のソースコード中ではクラス名を記述できる仕組みが必要である。したがってコンパイラはクラス名をIDに変換する必要があり、そのための変換テーブルとしてスクールと呼ぶものを導入している。スクールはクラス名とIDの2項組のリストであり、クラス名に対するプログラミング時の局所的な名称空間を提供するものである。

オブジェクト指向プログラミングでは、クラスが公開しているインタフェースだけを利用することによって、クラスの実装には依存しないプログラミングが可能である。OZ++ではこの特徴とクラスのバージョン管理の仕組みを組み合わせることによって、プログラム中で使用するクラスの実装パートのバージョンの決定をインスタンス生成時まで遅延させるという柔軟な運用を可能にしている。言語処理系は、パブリックまたはプロテクティッドパートのインタフェースの情報を利用して、インタフェースの利用に関する検査を行う。この検査に利用するの情報を得るために、コンパイラは、実行コードを生成するだけでなく、パブリックおよびプロテクティッドパートのインタフェースに関する情報の出力も行う(図2参照)。したがってOZ++の一つのクラスのコンパイルはクラスのパブリック、プロテクティッド、実装の三つのパート毎に処理を行い、各パートの出力をクラス管理システムに登録する。インタフェースに関する検査はこの情報を取得することによって行う。

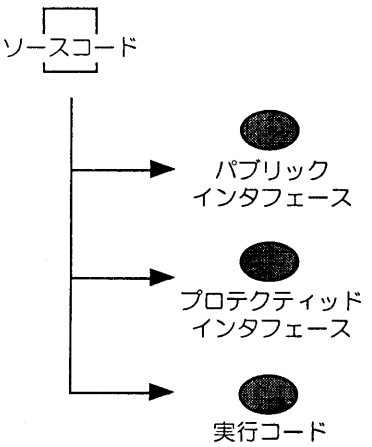

図2: 3つのパートのコンパイル

利用するクラスのバージョンは次のように決定する。

・パブリックパートを利用しているクラスは、パブリックパートのバージョンまでをコンパイル時に決定し、インスタンス生成時に実装パートのバージョンを決定する。

・継承しているクラスは、プロテクティッドパートのバージョンまでをコンパイル時に決定し、インスタンス生成時に実装パートのバージョンを決定する。

インスタンス生成時に行う実装パートのバージョンを決定する処理を、コンフィギュアと呼ぶ。このコンフィギュアの結果の情報(コンフィギュレーション)もクラスの情報の一つとして、クラス管理システムによって管理される。

2.4 ランタイムシステム

OZ++のランタイムシステムは、クラス管理システムから必要な情報をオンデマンドで取得して処理を行う[4]。具体的には、インスタンス生成時にコンフィギュレーションの情報を、メソッドの実行時に実行コードをそれぞれ取得する(図3参照)。

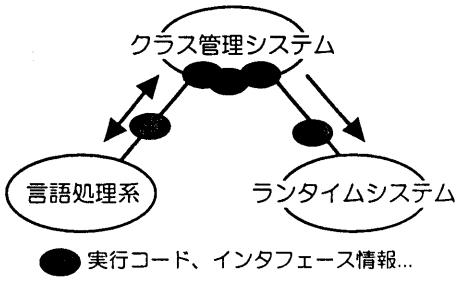

図3: 実行時の構成

2.5 デバッグ環境

一般に分散環境下では不具合が観測された後に同じ不具合が再現する可能性が低いという問題がある。このためOZ++のデバッグ環境では、不具合の発生時点でその原因を追及するための情報をプログラマに提供する[5]。例えば、例外処理が記述されていない例外が発生した場合には、デバッガによってその例外を捕捉し、例外の発生場所の特定や発生した例外の内容のブラウズを可能にしている。

2.6 ワークベンチ

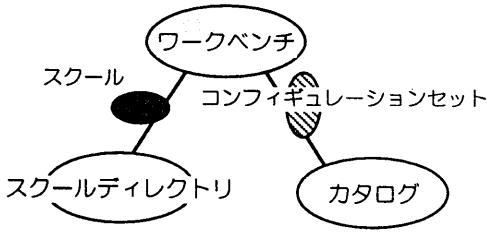

図4: 開発環境の構成

OZ++では、ワークベンチと呼ぶプログラミングのための統合環境を提供している[6]。ワークベンチでは、上で述べたスクールに関する操作、言語処理系の起動、クラスのブラウズ、デバッガの起動等の操作が可能である。

3 OZ++におけるソフトウェア開発

ここでは、OZ++において実際にどのようにソフトウェア開発を行うかについて述べる。また、その中でワークベンチの機能についても述べる。

3.1 スクール

OZ++における一つのソフトウェアは、いくつかのクラスを利用して記述するため、クラスの集まりと考えることができる。またプログラミング時のクラスの名前はスクールで設定するため、スクールにはあるソフトウェアの開発に必要なすべてのクラスが含まれていることになる。つまりソフトウェアの開発時には、開発するソフトウェア単位にスクールを用意することになる。

ここで、開発したソフトウェアを別のソフトウェア開発で利用可能にするためにネットワーク上に公開する場合を考える。OZ++ではこれを公開するソフトウェアに関連するクラスを含むスクールを公開することで行う。ところがここで公開するスクールは、そのソフトウェアの開発時のものをそのまま使用すべきではない。例えば、Aというクラスを公開する場合を考える。ここでAはパブリックインタフェース部分でBというクラスを利用し、また実装部分でCというクラスを利用しているとする。この場合、Aの開発時のスクールには、A、B、Cの三つのエントリが含まれていることになるが、このうちCはAの開発のためにだけ必要なものであり、公開されたAを利用する人にとってはCを直接利用することはないため不要である。次にBは公開するスクールにも含めるべきである。OZ++におけるクラス名はプログラミング時の局所的な名称空間ではあるが、プログラマが付与する名前には特定の意味があると考えられる。したがってインタフェースに表れるクラスについてはその開発者の意図を表現する意味で、公開するクラスと共に含めるべきである。

スクールの公開は、OZ++システムのスクールの管理システムであるスクールディレクトリにスクールを登録することによって行う。

このようにして公開されたスクールを取得することにより、別のプログラマは開発されたソフトウェアを自分の開発作業で利用することができる。

3.2 スクールに関するワークベンチの機能

ワークベンチはOZ++でのソフトウェア開発の中心となるツールである。上で述べたようにOZ++のソフトウェア開発はスクールを単位として行うため、ワークベンチもスクールを開発作業の単位としている。例えば、クラスをブラウズする場合にはあるスクールを指定し、そのスクールを起点としてブラウズを開始する。ワークベンチではスクールに関する次のような機能を提供する。

・スクールブラウザ機能

スクールブラウザにより、指定したスクールに含まれるクラス名と各々に指定されているバージョンをブラウズすることができる。またクラスを指定することでクラスの内容をブラウズし、クラスのバージョンをその中から選択したものに変更することも可能である。

・スクールディレクトリブラウザ機能

スクールディレクトリブラウザにより、スクールディレクトリとワークベンチの間でのスクールの交換が相互に可能である。つまり、開発したクラスを公開するためにワークベンチからスクールディレクトリにスクールを登録することや、公開されているクラスを利用するために、スクールディレクトリからワークベンチにスクールを取り込むことができる(図4参照))。

3.3 コンパイル

コンパイルは、ワークベンチ上でスクールを指定して、コンパイラフロントエンドを起動することによって行う。コンパイラフロントエンドは言語処理系の利用者ビューであると共に、コンパイル時のインタフェースの検査に必要な情報やコンパイルの出力情報をクラス管理システムとやり取りする。
OZ++のコンパイルは2節で述べたように三つのパートに分けて行うが、その各パート毎に次のような制約がある。

1. 各パートのコンパイル

・上位パートのコンパイルが行われていなければならない。例えば、プロテクティッドパートのコンパイルを行うには、パブリックパートのコンパイルが行われていなければならない。

・コンパイラはスクールによってクラス名をIDに変換するため、利用しているクラスがスクールに含まれていなければならない。

2. パブリックパートのコンパイル

・継承しているクラスのプロテクティッドパートのコンパイルが行われていなければならない。

3. 実装パートのコンパイル

・利用しているクラスのパブリックパートのコンパイルが行われていなければならない。

3.3.1 バージョンの更新

クラスのバージョンの更新は、各パートのコンパイル時に指定することによって行う。バージョンの更新が指定されると、指定されたパート以下のバージョンの更新をクラス管理システムに要求する。例えば、パブリックパートのコンパイル時にバージョンの更新を指定すると、クラス管理システムはパブリックパート以下の三つのパートの新たなバージョンを作成し、それぞれに新たなバージョンIDを付与する(図5参照)。

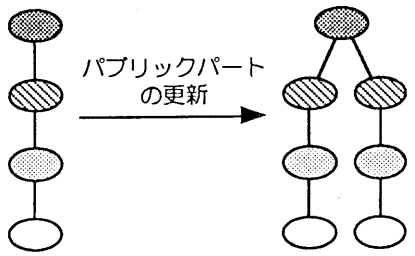

図5: バージョンの更新

3.4 アプリケーションの公開

他人が別のソフトウェアの開発に利用できるようにするためのソフトウェアの公開は、スクールの公開によって行うが、一方アプリケーションとして公開する場合も考えられる。
OZ++におけるアプリケーションの起動は、ランチャと呼ぶツールに、アプリケーションの起動インタフェースとなるオブジェクト(ランチャブル)を登録することよって行う。この時ランチャには、そのランチャブルの生成に必要な情報を渡すことが必要である。OZ++ではインスタンスの生成時には、23節で述べたコンフィギュレーションが必要であり、コンフィギュレーションを明示的に指定しない場合には、クラス管理システムにおいてデフォルトに設定されているコン

フィギュレーションを利用する。したがって利用者は、コンフィギュレーションを指定せずに常にデフォルトのものを利用することで、デフォルトの設定の変更に応じて異なる実装のバージョンを利用したインスタンスを生成することができる。ところが、アプリケーションを公開しようという場合、このように開発者が意図しない任意のバージョンのインスタンスを生成し、利用することは望ましくない場合がある。そのような場合には、ランチャブルだけでなく、そのアプリケーションの動作に必要なすべてのクラスのコンフィギュレーションを指定することが必要である。つまりアプリケーションの公開に必要な情報は、そのアプリケーションの実行に必要なすべてのクラスのコンフィギュレーションの集合で、これをコンフィギュレーションセットと呼ぶ。したがってOZ++におけるアプリケーションの公開とは、コンフィギュレーションセットを公開することであり、具体的にはコンフィギュレーションセットをOZ++のアプリケーション管理システムであるカタログに登録することによって行う。コンフィギュレーションセットの生成およびカタログへの登録はワークベンチからの操作により次のように行う(図6参照)。

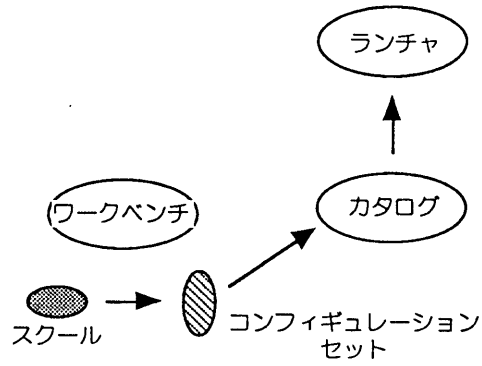

図6: アプリケーションの公開

1. スクールブラウザを起動し、アプリケーションのランチャブルのクラスを選択する。

2. 選択したクラスが利用しているすべてのクラスのコンフィギュレーションをリストアップし、コンフィギュレーションセットを作成する。

3. 生成したコンフィギュレーションセットの内容をブラウズし、必要なら修正を行う。

4. 完成したコンフィギュレーションセットをカタログに登録する。

カタログは専用のブラウザによってブラウズすることが可能であり、またその中のコンフィギュレーションセットをランチャに登録することで実際にアプリケーションを動作させることができる。

4 複数人による協調開発の支援

OZ++の持つ特徴は、複数人でのソフトウェアの協調開発の支援にも有効である。

構造化プログラミング時代のソフトウェア開発では、綿密に設計されたモジュール仕様が各プログラマに渡され、コーディングが行われた。理想的には仕様が十分明確で、あたかも一人で開発しているかのように、プログラミングの段階の他のモジュールとの調整は不要であった。

しかし、顧客の要求が複雑となり、引合から納品までの期間が短縮されるにしたがって、この古い開発手法は破綻してしまう。最新の開発手法では、ラピッドプロトタイピングが用いられ、スパイラル型に開発が進む。そこではデイリービルドが行われ、クラスライブラリとアプリケーションクラスの分離が進んでいる[7]。このような時代のプログラマは、スタブやドライバを用いて他のモジュールの仕様を仮定したままでプロトタイピングし、担当者間で頻繁にネゴシエーションしながら全体の仕様を洗練させ、徐々にモジュール境界や全体仕様を固定し、完成品を作り上げていかなくてはならない。

このような高度化/複雑化したソフトウェア開発では、モジュール間の依存関係が複雑になりやすい。一方で、一日あたり、一人あたりのコーディング量も増加する。このため、コードの行数あたり、およびプログラマ一人あたりが注意を払わなくてはならないモジュール間インタフェースも相対的に増加する。

例えばプログラマAの担当しているモジュールaが、プログラマBの担当しているモジュールbを利用している場合を考える。bのパブリックインタフェースが改変されb'となった場合、Aはaをb'に対応させなければならない。そのためには次のような対応が考えられる。

・bの変更が影響する範囲を調べ、aをb'に対応させるためのコストと現在進行中の作業の優先度を比較し、b'への対応の時期を決定する。

・一定期間bを使い続けるのであれば、デイリービルドやテストの環境をそのように設定するとともに、Bにbを使い続けることを伝える。

このような作業では、プログラマどうしのコミュ

ニケーションを密にするのが最善であるが、分散開発であったり生活習慣の違いなどから地域的/時間的な差が生じ、必要なコミュニケーションができない場合が多くなっている。

OZ++では、これらに対して以下の性質が備わっている。

・各クラスは、他のクラスのどのインタフェースに依存するかを意識してコンパイルしなければならないため、依存性解析の材料は整っている。

・各クラスの異なるバージョンが混在して動作可能で、かつコンフィギュレーションをインスタンス生成時に動的に変更できる性質を持っているため、複雑な結合/試験のための環境の構築が容易である。

具体的には、以下のようなプロジェクト管理オブジェクトの導入を検討している(図7参照)。

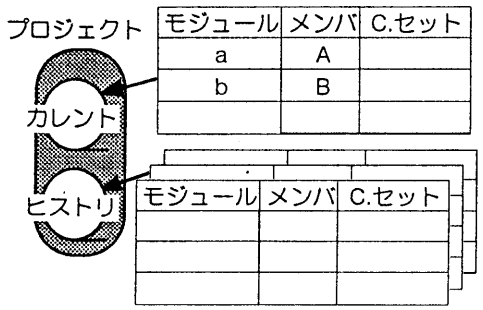

図7: プロジェクト管理オブジェクト

プロジェクト管理オブジェクトは、モジュールの集合と、その進化を管理する。モジュールとは、互いに関連の深いクラスの集合であり、スクールで表現される。またワークベンチからは次のようなモジュールに対する操作を可能にする。

・そのプロジェクトで用いる各モジュールと、その担当メンバのワークベンチを登録/変更/削除する。

・各モジュールごとに、そのモジュールが現在動作するのに必要なコンフィギュレーションセットを設定する。

・任意の時点で、各モジュールと、その登録されているコンフィギュレーションセットの表を作成し、それに番号をつけてヒストリに登録する。これにより、システム全体のバージョンの進化の過程を記録し、いつでも記録された状態を回復できる。

ヒストリによって、CVSのような既存のバージョン管理システムと同等の機能が提供できる。加えて、ワークベンチとの連携により、次のような支援が可能になる。

1. インタフェース変更通知機能

あるメンバのワークベンチから、あるモジュールに含まれているクラスのインタフェースが変更されたことがプロジェクト管理オブジェクトに通知されると、そのインタフェースに依存しているクラスを開発しているメンバのワークベンチに、改変されたクラス名と変更ログが通知される。

2. コンフィギュレーション変更通知機能

あるメンバのワークベンチから、あるクラスのデフォルトのコンフィギュレーションが変更されたことがプロジェクト管理オブジェクトに通知されると、そのクラスを含むコンフィギュレーションセットを登録しているモジュールのワークベンチに、その変更ログが通知される。そのメンバがその変更をそのモジュールのコンフィギュレーションセットに反映させるかどうかは、作業上の都合で選択することができる。

3. 結合/試験支援機能

各モジュールの都合に応じて使用するバージョンを詳細に指定した結合/試験が可能である。また、すべてのモジュールに対して、デフォルトのコンフィギュレーションだけからなるコンフィギュレションセットを適用して試験することや、その結果うまく動作しなかった場合などに、デフォルト以外の指定を含むコンフィギュレーションセットを指定しているモジュールをリストアップするなども容易に実現できる。

5 今後の課題

・モジュールの導入

前節で述べた複数人によるソフトウェアの協調開発を実現するために、ワークベンチにモジュールを導入する必要がある。一方、モジュールはワークベンチ上でのみ存在するものではなく、ソースコード中でもクラス名の一部としてモジュール名

を記述できることも必要である。これによって、プログラマに対して利用しているクラスがどのモジュールに含まれるかを明確にすることができると考えられる。また、スクールブラウザではモジュール単位でのクラス名のブラウズを可能にする等の拡張が必要である。

・実装の変更に対する対応

本論文で述べたように、クラスをインタフェースに基づくバージョンによって管理することにより、実装のバージョンの更新は再コンパイルを行うことなくインスタンスを再生成することで反映させることが可能である。一方、実装の変更は次のように細分化することが可能である。

1. インスタンス変数に関する変更。単なるインスタンス変数の数や種類に関する変更の他、インスタンス変数の値の意味的変更も含む。

2. privateなメソッドのシグネチャの変更

3. privateなメソッドの実装部分の変更

これらのうち1はインスタンスの構造や内容に直接影響を与えるため、インスタンスを再生成することが不可欠であるが、2と3は既存のインスタンスに対する影響を与えないため、インスタンスを再生成することなく変更結果が利用できても良いはずである。例えば、些細なバグの修正や、アルゴリズムの変更の場合には、メソッドの実装部分だけを変更して、即座にその変更結果を利用したいという場面が考えられる。これを実現するには、1を実装のインタフェースと捉えることによって、バージョンを四つのパートで管理する方法と、バージョンのパートは三つのままで特殊な状況では実装パートのバージョンの上書きを許すという方法が考えられる。前者は管理するバージョンに関する情報が膨大になる危険性がある。一方、後者ではバージョンの公開という概念を導入し、公開していないクラスを共有の対象から除外し上書きを許すことによって、実装パートが上書きされた場合に影響の及ぶ範囲を制限することができる。現在、この後者による実現を検討している。

6 まとめ

本論文では、広域のネットワーク上で共有するクラスを利用してソフトウェア開発を行うOZ++システムにおけるプログラミングについて述べた。クラスのバージョン管理を行うことによりクラスの変更に対して柔軟な対応を可能とし、またスクールによってプログラミング時のクラス名の衝突の回避が可能となっている。これらによって安全なクラスの共有を実現している。
現在、ここで述べたバージョン管理を行うクラス管理システムと、それと連携して動作するワークベンチを中心とする開発環境を含むOZ++システム第1版の実装が終了し、その評価および拡張、改良を行っている。今後、4節で述べた協調開発環境の実現を行う予定である。なお、OZ++システム第1版は以下で公開している。

`http://www.ipa.go.jp/OFSTP/obj.html`

謝辞

本研究を通じて熱心な討論をいただいている当プロジェクトのメンバー諸氏に感謝する。

この研究は情報処理振興事業協会(IPA)が実施している「開放型基盤ソフトウェア研究開発評価事業」の一環として行われたものである。

参考文献

1. 塚本他,「オブジェクト指向分散環境OZ++の基本設計」, 情報処理学会研究報告 93-OS-61-3, Aug. 1993.
2. 新部他,「OZ++コンパイラによるクラスの版管理」, 情報処理学会研究報告 94-DPS-66, Jul. 1994.
3. 音川他,「オブジェクト指向分散環境OZ++の言語における例外処理の記述」, 情報処理学会第50回全国大会, Mar. 1995.
4. 濱崎他,「オブジェクト指向分散環境OZ++の実行機構の設計」, 情報処理学会第48回全国大会, Mar. 1994.
5. 吉田他,「オブジェクト指向分散環境OZ++のデバッガ」, 情報処理学会研究報告 95-DPS-71-2, Jul. 1995.
6. 籠他,「オブジェクト指向分散環境OZ++の開発環境ワークベンチ」, 情報処理学会第50回全国大会, Mar. 1995.
7. Jacobsen, I., et al. "Object Oriented Software Engineering", Addison and Wesley, 1992.