ソフトウェア工学 101－8
（1994．11．18）

オブジェクトベースト・リポジトリを用いたオブジェクト生成支援環境

加藤木　和夫 *　　畠山　正行 **　　小林　秀行 **

* 日立プロセスコンピュータエンジニアリング（株）　　** 茨城大学　情報工学科

自然界シミュレーション分野等のエンドユーザ向けに、開発環境を設計し、オブジェクトに基本をおいたリポジトリ、オブジェクトベースト・リポジトリ（OBR）の上に実現した。従来の当該分野のプログラム開発には一般的なプログラミング環境を用いており、シミュレーション開発をスムースに行うための一貫性に欠けるプログラミングを強いられた。そこで、筆者らはシミュレーションの開発環境を構築するに当たり、開発から駆動までの多くの作業過程上で生成される各種のオブジェクトを個々に格納、管理すること及びオブジェクト間の相互関係を管理するOBRを考察、実現した。このOBRを基にオブジェクト指向シミュレーションのモデルの記述から実行プログラムの生成までを一貫して統一的に扱う生成支援環境を設計した。その結果、生成支援環境の実装はまだ完全ではないが、OBRアーキテクチャの適格性を確証できた。

Generation Support Environment on an Object-Based Repository

Kazuo Katougi *　　Masayuki Hatakeyama **　　Hideyuki Kobayashi **

* Hitachi Process Computer Engineering, Inc.　　** Ibaraki University

We have designed and developed a kind of the simulation support system for domain users of the natural phenomena, etc. Simulation users have developed their simulation systems by using ordinary programming systems. Therefore , they have urged to be constrained to the non-consistent programming style. We considered that the problems consist in the management of the objects. To develop the consistent generation support system, we have developed an Object-Based Repository (OBR),which is storing and managing many varieties of the objects and the linking management objects during the developing processes of the simulation systems.We designed and implemented the prototype of this simulation system of the support environment based on the OBR. This support environment supports the generation processes from the stage of the description of the simulation meta-model up to the generations of the execution programs of the object-oriented simulation . As the conclusion, the implementation of the Generation Support Environment is, though, not yet complete, the architecture of this OBR system is confirmed to be valid.

1. 支援環境開発の背景

オブジェクト指向の特長は、「対象世界の「モノ」をオブジェクトとして、モデルの構築から実装、実行までを一貫して統一的に扱える」ところにある［1］［2］。このオブジェクト指向の特長を典型的に応用しているのは、いうまでもなくGUIであり、シミュレーションの世界である。とくにシミュレーションはオブジェクト指向の基礎を築いたSmalltalk、あるいはオブジェクト指向プログラミング言語C++の目的の一つでもあり、応用が図られている。しかし、自然現象等の変化を計算機上で再現実行（駆動）するシミュレーション分野は、「オブジェクト指向に基づくシミュレーションのメリット」は原理的に大きいにも関わらず、従来は応用がほとんどなされていない。筆者らはこの分野にオブジェクト指向パラダイムを適用することで次の結果を得ている［3］［4］。

（1）手続き型モデルより複雑な構成を持つ対象世界をはるかに手軽に構築できる。
（2）対象の動的な変更が非常に容易である。
（3）対象世界の「モノ」を計算機上の「モノ」（実行モジュール）として扱いたい対象に本来の有効性を発揮する。

現状では、一般的なシミュレーションのエンドユーザは対象世界の「分析過程」においてはオブジェクト指向に近いモデリングを行っている。これはそれが自然だからである。しかし、「設計、実装過程」においては慣習的に方法として確立、蓄積された手続き型プログラミングを行っており、オブジェクト指向へのパラダイムシフトに消極的である。この原因としては、自然界等のシミュレーションを行うエンドユーザにとって対象世界のモデルとオブジェクト指向のプログラム及び「モノ」の駆動イメージにギャップがあるためと考えられる。つまり、対象世界のモデル化とオブジェクト指向のプログラムが結びつかない、動作イメージの記述がオブジェクト指向（クラスなどを用いるため）になることで手続き型プログラミングに慣らされてきたエンドユーザには「現状では」逆にイメージし難くなるからである。

一方、オブジェクト指向パラダイムも自然界シミュレーションを扱うエンドユーザに対し、きちんとした環境を用意していないことも挙げられる。従来の開発環境は一般のプログラミング環境を用いており、オブジェクト指向に沿った一貫性のある開発環境とはなっていない。一貫性のある開発環境になり得ないのは、開発途上に生成される各種のオブジェクトを系統的に管理する機構が不備であることがその障害の一つになっているためと筆者らは考える。

更に、シミュレーションの駆動時には様々な数値データが発生するが、このバージョンを系統的に蓄積、管理する機構が用意されていないが、数値データもオブジェクトの一部として捉え、管理する必要性がある。

そこで筆者らは以上のことをふまえ、対象世界のモデル化から駆動するまでを一貫して支援するモデル記述方式を導入し［5］、それを実現する開発支援環境（以降、（オブジェクト）生成支援環境と呼ぶ）のプロトタイプ版を設計、評価した。本環境のねらいはエンドユーザが対象世界をモデル化するときに持つ駆動イメージを、できるだけそのまま実際のシミュレーション駆動まで持っていけるように支援するものである。本稿では生成支援と駆動支援での両環境（総合支援環境）において、オブジェクトの統一的管理機構として導入を図ったオブジェクトベースト・リポジトリ（Object-Based Repository,OBR）［6］のアーキテクチャ及びOBRを用いた生成支援環境の実現方式とその評価について報告する。

2 生成支援環境

2．1 シミュレーションシステム開発の概要

本生成支援環境の対象ユーザはシミュレーションのモデリングが出来るエンドユーザである。そして、オブジェクト指向の概念を理解しており（必要ならば教育を受ける）、手続き型プログラミングの基礎知識は持っている一般的なエンドユーザを前提とする。オブジェクト指向プログラミングには抵抗感を持っているかもしれない。

本生成支援環境の目標は「モノ」（＝オブジェクト）のイメージを保ったまま駆動時の「モノ」へ変換することにある。言い換えると、対象世界をモデル化を経て計算機の世界へ変換する過程、その分析過程、設計過程、実装過程を

違和感なしにスムーズに移行することを支援する。したがって、支援環境の持つべき機能としては、オブジェクト指向に抵抗感なく（あまりプログラミングの詳細などを意識させずに）オブジェクトを生成する過程をガイダンスする機能が上げられる。また、記述したオブジェクトを実行可能なプログラムへ変換する機能も必要である。ここで実行可能なプログラムは普及度合いを考慮してC++とする。また、ユーザはシミュレーションする物体の形状をプログラミングすることに多大な労力を使っているが、これは既に画面を見ながらマウスなどで描けるように実現している［内部資料］。

したがって、筆者らは支援環境の要素として、インクレメンタルな開発方式をガイダンスするN-stepsモデル記述方式及びN-steps記述の最終段階（C++プログラムの前段階）でのシミュレーションモデル記述言語SMDL（Simulation Model Description Language）、及び形状を直接画面上で作成できる形状エディタを既に作成した［5］。以下にその概略を述べる。

（1） N-steps記述方式

N-steps記述方式は、対象世界のモデルを少しずつ変化させるために、変化をインクレメンタルにN段階のステップで記述するものであり、ウォーターフォールモデルの一変形である。各ステップは次のステップを指す。

（a）対象世界のモデルをエンドユーザになじみの深い自然言語（日本語）で記述する。
（b）自然言語で記述したモデルを順次洗練し、SMDL記述にリライトする（N-steps）。
（c）SMDLで記述したモデル（クラス等）はトランスレータにより実行可能な言語C++に変換する。

（2） SMDL

SMDLはオブジェクトベースの仕様を具体的に枠組み（フレームワーク）として示すガイダンスする言語仕様となっている。モデリング段階とオブジェクト（「モノ」）の関係は次のようになる。

（a）対象世界のモデル化：モデルの単位はオブジェクトとする。オブジェクト間の共通の属性、動作はクラスとして抽象化する。クラス同士はいくつかの関係（is-a, is-part-of等）を持つことができ、対象世界の構造を表現できる。
（b）ソフトウェア化：クラスはオブジェクトの共通仕様を記述するフレームワークを意味する。
（c）プログラム化：クラスはプログラム記述上ではモジュールの単位となり、モジュールはコンパイルの最小単位ともなる。
（d）駆動時：オブジェクトは駆動時にクラスから生成（対象世界の「モノ」の再現）し、仮想物体としてシミュレーションの操作単位となる。駆動の操作はメッセージ交換で行う。

SMDLのターゲット言語はC++とする。したがって、C++の仕様がSMDLに一部入ってくるがC++の持つ物理的な情報は極力排除する。 排除の難しい仕様、例えば記憶エリア情報（static等）は角括弧＜＞で封じ込める。SMDLの文法チェック範囲を限定する機能を用いることで、トランスレートする度に順次プログラムを洗練させることができる。

（3） 物体形状のビジュアル製作

物体形状の作成にはユーザがマウスを用いて画面上で形を確認しながら作成できるように、形状エディタを用意した。物体は形を画面上に表示するだけでなく、シミュレーション駆動時にメッセージを受け取り、動作する「モノ」である。つまり、物体は動作を行うメソッドを持っていなければならない。筆者らはこれらの物体のメソッドは別に物体のクラスとして記述し、実行時に物体とリンクする方式を取ることとした。特に、テスト段階においては物体の様々な形状と様々なメソッドが複数組み合わせることができるようにし、確定後は実行速度を重視する方式を考えた。

2．2 生成支援環境の機能

生成支援環境の機能は、N-steps記述のためのガイダンス機能と、自動変換のためのトランスレータ機能及び形状エディタ機能がある。

（1）ガイダンス機能

本環境で管理すべき対象物は個人レベルの生成プロダクトである。個人環境ではツールの強力さ、使い易さが必須であり、そのためのガ

イダンス機能が重要である。

ガイダンス機能には次の４つがある。

A. クラス構造を記述する為の誘導メニュー
B. N-steps記述支援のために、下流モデルを記述中に上流モデルをすぐ呼び出せる機能
C. ユーザの要求するカテゴリに属するメソッドを検索、提示する機能
D. 操作プログラムのための汎用C++プログラムを提示、修正する機能
E. ビジュアルプログラミング機能
（形状エディタではなく、操作プログラムをビジュアルに生成することを目的とする）

（2）トランスレータ機能

本環境の特長としてトランスレータを含むことが上げられる。これはガイダンスと一体化してモデルを作成から実行までスムースに移行する「しかけ」である。トランスレータはモデル記述言語SMDLをC++プログラムへ変換する。

（3）形状エディタ

ユーザは物体の形状をプログラムではなく、形状エディタを用いて生成することができる。形状エディタを用いて作成した物体の動作は、通常のメソッドとして別に記述し、実行時に駆動前処理としてリンクされる。

3．オブジェクトベースト・リポジトリ（OBR）

3．1　OBRの導入

生成支援環境を構築するに当たり、筆者らは次のような２項目にわたるシステムの基盤的機能拡張の必要性を感じた。

（1）多様な「モノ」の蓄積・管理機構

多様なバリエーションの「モノ」を蓄積、管理する機構が必要である。筆者らは２．に述べた支援環境に基づき簡単な例題を記述した結果、良好な評価を得ている［7］。しかし、複雑な対象世界を系統的にかなり長期間に渡って継続して開発して行く場合、N-steps記述では何段階かのモデル記述が存在するために、開発途上ではモデルの定義記述、SMDLプログラム、実行プログラム等の様々なオブジェクトが生成される。これらは系統的に永続格納、管理することができれば、開発作業の中で再使用、或いはモデルの作成順を（順に又は逆順に）辿るとき等に使用できる価値の高いオブジェクトとなる。最近ではこの様な生産物はソフト工学の分野ではソフトウェアリポジトリと呼ばれるシステムにて蓄積、管理されている［7］。

しかし、「オブジェクト指向」に基づくシミュレーションシステムの開発では、次の２点で従来のソフトウェアリポジトリとは異なる要求がある。ひとつは筆者らの目指すシミュレーションのオブジェクト指向は、実世界の「モノ」の完全なモデル化とその実現を目指したモデリング及び実現法を意味しており、このことを筆者らは「オブジェクトベース」と呼んでいる。「オブジェクトベース」では、「属性＋メソッド＝オブジェクト」を一貫して管理の基本単位とし常時例外なくワンセットで扱う。また「オブジェクトの相互関係」を陽に取り扱う。したがって、従来のリポジトリシステムでは対応しにくいと考える。

もうひとつは、シミュレーションシステムであるから、シミュレーションの結果データを数値で持つ。これらの数値データは、駆動テスト中では各種データの組み合わせがあり、テスト終了後の本格駆動でも現象毎のパラメータ等様々なバリエーションがかなりの規模で発生する。また、数値データを出力したオブジェクトとの相互関係も持たなければならない。このようなオブジェクト（数値データ）を管理するにも同じ管理システムで蓄積、管理することが望ましい。これらの管理はOODBを基盤とした管理システムが適切であると考える。また、大規模なシミュレーションの開発、管理を考えれば、OODBで管理することは必要条件であろう。

（2）相互リンク管理機構

「モノ」は相互にリンクする必要がある。「オブジェクトの相互関係」を陽に取り扱うと述べた。これは、例えば物体の形状はプログラムで作成するのではなく直接にマウスで作成するが、この場合、直接生成した形とその物体動作の記述（オブジェクト）をリンクさせなければならない。すなわち、「モノ」同士をリンクする機構、すなわちリンク管理機構が必要である。

以上の（1）、（2）の考察から得られる

結論はOODBを基盤として、開発過程での生成オブジェクト、シミュレーションの結果データ等のバリエーションを持ったオブジェクトの蓄積、管理、そしてオブジェクトの相互リンクの管理、及びシミュレーションの駆動準備までを行うシステムが必要であることが分かる。この様なシステムはソフトウェアリポジトリとOODBMSを統合した「オブジェクトベースト・リポジトリ」として構築すべきであると筆者らは結論した［6］。オブジェクトベースト・リポジトリにおいては、オブジェクトは対象世界の「モノ」であり、モデル記述のレベルでは「対象世界のモノ」を抽象化したクラスであり、プログラムレベルではオブジェクトの共通仕様を記述するクラス（属性＋操作）である。シミュレーション駆動時には仮想物体のプロセス（即ちオブジェクト）となる。また、このリポジトリではモデル記述単位（オブジェクト）をシステムの中でそのまま蓄積、管理、変換の単位として扱う。次にOBRのシステム全体の中での位置づけについて述べる。

3．2　OBRアーキテクチャ

オブジェクトベースト・リポジトリは前節に述べたように管理の基本単位をオブジェクトにおいたリポジトリで、オブジェクト指向シミュレーションを総合的に支援する環境構築のために基盤機構として導入した。ここでリポジトリの管理対象物であるオブジェクトは「目的プログラム（オブジェクトプログラム）」を意味するのではなく、オブジェクト指向におけるオブジェクト（属性＋メソッド）を意味する。

本リポジトリのシステム内における位置づけは図１に示すように、オブジェクトベースト機構［4］と一体になってOSの上に位置し、逆にシミュレーションのアプリケーションは開発から駆動までを支援する環境はこのリポジトリの上に構築される。オブジェクトベースト機構はOODBMSのONTOSを利用して作成され、クラスの定義、オブジェクト及び永続オブジェクトの定義と操作ができる。

本リポジトリはオブジェクトの格納、取り出し、検索、オブジェクト間の関係の管理及びシミュレーション駆動の管理を含む［6］。

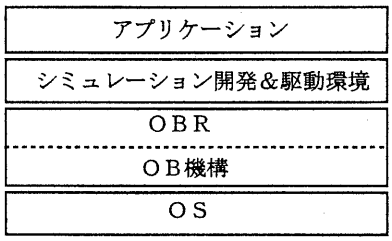

図１　OBRアーキテクチャ

3．3　シミュレーション支援環境とOBR

シミュレーション支援環境はオブジェクトを生成する支援環境（開発環境）、オブジェクトを駆動する駆動環境、そして開発と駆動の間にありテストを支援するテスト環境から成る。OBRはこれら環境から作り出されるオブジェクトを管理する。各支援環境とOBRの関係は図２のように位置づける［6］。

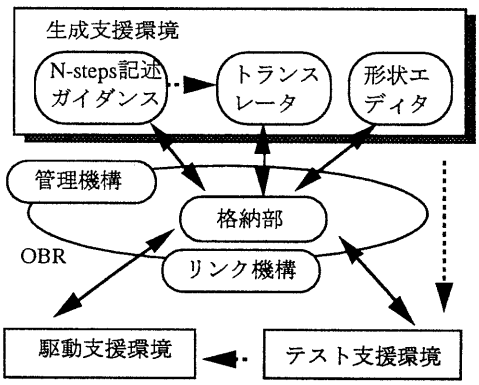

図２　支援環境とOBRの関係

データの格納、取り出し、検索
ライフサイクル

4．　OBRを用いたオブジェクト生成支援環境

従来の一般的な開発支援環境の構築は、本支援環境に即していえばガイダンス機能、トランスレータ、形状エディタの各サブシステムが

システムとしては表面上は連結しているように見えるが、実態は別々に存在し、ファイルを通してデータをやり取りする形態をとる。

しかし、今回の生成支援環境構築に当たっては、OBRを導入することで、その機能であるクラス定義と永続オブジェクトを活用し、ガイダンス機能のシステムとトランスレータを有機的に結合した形態を取れるようになる。即ち、ガイダンスのシステムとトランスレータを一つの＜クラス＞（以下の説明では、OBR上のクラス定義には＜＞をつける）の中のメソッドとすることで、２つのサブシステム間のデータの共通化を図ることができる。また、このクラスのインスタンスを永続オブジェクトとすることで、他のシステムとデータをやり取りするファイルを作成できる。ここでトランスレータの出力するオブジェクトはC++で記述したクラスとなるので、この＜クラス＞はクラスを生成するクラス「メタクラス」的な位置づけとなる。

一方、OBRのリンク管理機構を用いることでトランスレータの生成した永続オブジェクト（テキストファイル）と形状エディタの生成した永続オブジェクト（形状のデータファイル）とがリンクを取れるようになった。これらは従来は別々に存在し、結びつくことはなかった。

以上のような新しい支援環境の実現方法は、オブジェクトベースの考えに基づく生成支援環境の構築に当たり、解決しなければならない次のような要求をも満たしている。

（ａ）モデル記述やSMDLプログラムを平板なテキストではなく、できるだけオブジェクトの形を保存する。そうして、テキスト情報からクラスの属性、メソッド情報をそれぞれの部分で管理、活用したい。

（ｂ）N-steps記法（インクレメンタル開発方式）によるモデル記述の流れを管理する。モデルには上流過程から下流過程まであるので、一つのクラスが変遷過程を辿り、またモデル記述の離合集散が発生する。この過程を簡単に実現したい。

これらの要求はOBRを導入することで解決する。すなわち、（ａ）の解決にはオブジェクト（属性＋メソッド）の形の保存にはテキストを構造化すればよい。すなわち、OBRの＜クラス＞の中にクラスの属性、メソッドを分けて記憶する。更に（ｂ）の解決にはモデル間のポイント、クラス間の関係ポインタを記憶する構造を持つように＜クラス＞を作成すればよいことが分かる。

以上の考えから、OBR環境下においては上記のような＜クラス＞を定義し、そのインスタンスの永続性を用いて各種テキストを作成し、活用することとした。支援環境内にはこの方法に基づいた＜システム定義クラス＞はいくつかあるが、その中で代表的な＜モデル定義＞クラスの位置づけについて図３にその概要を示す。

＜モデル定義＞クラスはOBRの持つクラス定義と永続オブジェクトの定義・操作機構を利用する。すなわち、クラスから永続オブジェクトを生成する機構を利用する。例えば、対象世界のクラスＸを作成するには、ユーザは開発環境のガイダンスにしたがってクラス情報を入力する。開発環境中の＜モデル定義＞クラスはその情報に基づいてインスタンスとしてクラスＸのフォーマット付きのテキストファイルを作成する。このテキストファイルはデータを取り出しやすい構造になっているので、構造に合わせたクラスを作成することで表示などが容易にできる。トランスレータ機能により、C++プログラムに変換され、それが実行プログラムとなる。

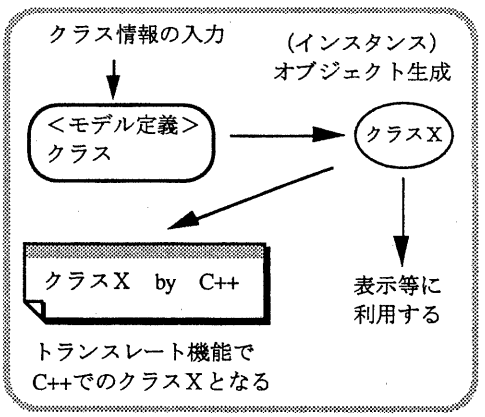

図３　クラスＸの作成例

一方、OBRを導入したことで下記のような生成支援環境に対する新たな要求が生じた。現在、その点での生成支援環境側の設計変更を行い、改訂実装中である。

（a）OBRデータ管理のインタフェース部はライブラリを用いた基本関数を用い、マクロ変換方式を採用する。
（b）OBRがオブジェクトの定義を組み込む際にメソッドのインタフェース部分を中心とした種々の情報を取得する必要がある。このため、特定のコメント形式での明示的な記述やマクロ変換などを用いた上でYaccを利用することで対処する。これにより、リンクされる部品オブジェクト間の属性の数や型の不一致、存在しない属性・メソッドに対し行われようとする不正なアクセス、メソッドの返り値の型・引数の数や型の呼び出し側との不一致などの矛盾を排除する。

5. OBRを用いたオブジェクト生成支援環境の実現例

5．1 モデル定義クラスの概要

＜モデル定義クラス＞は属性のデータ構造部分にはモデル記述プログラム（文字列）とモデル（クラス）間の相互関係のポインタを格納する。メソッド部分にはモデル記述プログラム（文字列）を入力・解析して属性とメソッドのテキスト列に格納するメソッド、C++プログラムを生成するトランスレータの役割を持つメソッド、及び永続性のためのメソッドを定義している。したがって、この＜モデル定義クラス＞から生成された＜オブジェクト＞は次のようになる。

・＜オブジェクト＞：＝
　（＜データ（文字列）＞、＜メソッド＞）
・＜データ＞：＝モデルを記述した日本語列｜SMDLプログラム
・＜メソッド＞：＝データ取込、格納メソッド、トランスレータのメソッド、編集メソッド等（ただし、メソッドは物理的には一カ所）

5．2 ＜モデル定義＞クラスの定義

＜モデル定義＞クラスを擬似的日本語を交えて定義すると次のようになる。

```
＜モデル定義＞クラス（一部分）
    属性：定義セクション
            モデル種別　int
            クラス名　　char*
            クラス関連種別　int
            関連クラス名　　char*
            プロパーティ　　char*
        属性セクション
            属性［配列］
            種別　　int
            テキスト　char*
        メモリ管理セクション
            メモリ管理［配列］
            種別　　int
            テキスト　char*
        操作セクション
            メソッド［配列］
            カテゴリ　int
            メソッド名　char*
            戻りの型　　char*
            引数テキスト　char*
        システムセクション
            上流モデルポインタ
            下流モデルポインタ
            兄弟ポインタ
            形状クラスへのポインタ
    操作：def_class() {....}　// クラス定義
          attr() {....}　　// 属性セット
          method() {....}　// 操作セット
          trans() {....} // C++プログラム生成
          save(file_name) {....}　//格納
          ・・・・・
＜モデル定義＞クラスの使用例
    Model_class model;
      //クラスからオブジェクトを生成する。
    model.attr();　　//データ入力
    model.trans();　//トランスレータの起動
    model.save(file_name);　　//永続的に格納
    model.restore(file_name);　　//取り出し
      （提示一覧表からファイル名を指定）
```

＜モデル定義＞クラスは4．に述べたように丁度クラスを生成する「メタクラス」の位置にある。このクラスのインスタンスは、属性の中に文字列つまりテキストを含み、この箇所にC++プログラムの基になるデータが格納される。OBRへリンクするメソッドsave、restoreを用いるとこのインスタンスは永続性を持つことができる（つまり、テキストファイルになる）。＜モデル定義＞クラス内の相互関係のポインタは、

インスタンスではC++プログラムの中では相互ポインタになるので、丁度メタポインタの立場にある。

6　評価と今後の展望

6．1　評価と考察

本オブジェクト生成環境構築の目的はエンドユーザが対象世界の自然なモデリングとモデリングの最終段階（プログラミング）から駆動までをスムーズに持っていけることを支援することにある。本論文では生成支援環境を構築するに当たり、次の２点の考察及び開発方式の実現を図った。

（1）生成支援環境、及び駆動支援環境と合わせて、そのオブジェクトの蓄積、管理に焦点を当てた考察を行い、OBRの導入の結論を得て、それを設計した。そして、OODBMSとソフトウェアリポジトリの性格を持つOBRの導入により、様々なバリエーションを持つオブジェクトを系統的に蓄積、管理する機構を実現できた。

（2）生成支援環境はこのOBR上に構築することで、新たに今までのプログラミング環境の構築より進んだ方法、すなわちオブジェクトの生成を「メタクラス」的な方法から生成するから導き出す方式を確立した。現在、プロトタイプを制作中である。

本方式は支援環境の実現に＜モデル定義＞クラスのように「メタクラス」的性格を持つクラスを定義することで、以下の特長を新たに獲得した。

（a）クラスから生成されたインスタンスの内部に属性、メソッドがテキストデータとして分けて格納されているので、属性又はメソッドだけを取り出せる。これを利用するとハイパーテキスト風表示を作成しやすい。

（b）クラスから生成されたインスタンスの内部にモデル関連情報を記憶しているので、オブジェクトの生成過程の再現が容易である。

（c）同様に、インスタンスの情報の中からクラス間の構造を容易に取り出せるので、対象世界の表示が簡単に実現できる。

以上のことから、本研究の目的であるオブジェクト生成支援環境の構築実現方式は確立したと言える。

6．2　今後の改良点

＜モデル定義＞クラスの中にポインタ情報を埋め込み、モデルの作成過程を繋げる方式は、小規模システムでは問題ないが、中・大規模システムの場合は、モデル連結用のポインタテーブルを検討しなければならないであろう。また、今後構築過程においては、＜モデル定義＞クラスはガイダンス機能からトランスレート機能までを含むかなりの大きさになるため多くの問題が発生することが予想される。これらはライブラリ化などで対応していきたい。

6．3　展望

評価結果を踏まえ、オブジェクト生成支援環境のプロトタイプを制作中である。また、問題点をひとつひとつ解決することにより、本環境の実現がエンドユーザの快適な環境になるように取り組みたいと考えている。

参考文献

［1］米沢、柴山、「モデルと表現」、岩波書店、1992

［2］所他監修、「オブジェクト指向コンピューティング」、岩波書店、1993

［3］畠山、金子、「オブジェクトベース機構：オブジェクト指向一貫モデリング過程論に基づくシミュレーションの実現」、情報処理学会第１７回プログラミング研究会、1994年6月３日

［4］畠山、金子、「オブジェクトベース機構に基づく数値シミュレーション」、情報処理学会第５１回ハイパフォーマンスコンピューティング研究会、1994年6月17日

［5］加藤木、畠山、「オブジェクト指向シミュレーションモデル記述の開発支援環境」、情報処理学会第９８回ソフトウェア工学研究会、1994年5月25日

［6］M.Hatakeyama, K.Katougi, H.Kobayashi, "An Object-Based Repository",4th International Conference on Database Systems for Advance Applications (DASFAA '95),April,1995.（投稿中）

［7］落水、「ソフトウェア・リポジトリ」、情報処理学会誌、Vol.35, No.2、pp.140-149 (1994)