ソフトウェア工学 98－2
（1994．5．25）

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

加藤木　和夫
日立プロセスコンピュータエンジニアリング（株）

畠山　正行
茨城大学　情報工学科

自然界シュミレーション分野等のエンドユーザは、分析段階ではオブジェクト指向パラダイムに近い概念で自然にモデリングを行っているが、設計段階・実装段階ではエンドユーザ側及びオブジェクト指向の両方に多くの障壁が存在するために、手続き型言語を用いている。そこで我々は、実行可能なC++プログラムに変換できるモデル記述言語MDLと、ガイダンス付きのN-steps方式のモデリング変換を支援する環境を設計、評価した。簡単な例を試みた結果、良好な結果を得た。また、モデルオブジェクトを「もの」としてその構築全過程を一貫して扱い、オブジェクトを生成するための環境を構築する、という目標に役立つ指針やノウハウを数多く得ることが出来た。

Generation Support Environment for Describing Object-Oriented Simulation Model

Kazuo Katougi
Hitachi Process Comuter Engineering,Inc.

Masayuki Hatakeyama
Ibaraki University

End users concerned with such fields as the simulation for the natural world can model naturally during the analysis phase using the conception close to the object-oriented paradigm. As for the design and the implement phase, end users use a procedual programming style. Since many difficulties exist in both sides of users and the object-oriented methodlogy.Thus, we designed and evaluated a model description language, MDL,which can convert pseudo-code into the runnable C++ program. We also designed and developed the environment for supporting modeling transformation through N-steps procedures with guidance. A simple try resulted in favorable one for user.

1．はじめに

オブジェクト指向の最もそのオブジェクト指向的な特長は「オブジェクト指向が、常に”もの”に一対一対応させたまま、人間の認識モデリングから、概念・論理・再構成・再現モデルを構築でき、それをそのまま実装・駆動まで行える」という点である。

これは我々が以前から「オブジェクトベース」（下記注）として主張し・実現してきた点でもあり［1］［2］、また、最近、同様な記述がみられるようになって来た［3］［4］。

このようなオブジェクト指向が持っている特長を最も典型的に応用できる分野の一つは、自然現象等の変化を計算機上で再現駆動するシミュレーション分野である。この分野は計算機で”もの”を直接扱い、対象としている世界をモデル化してコンピュータ内で再現（シミュレーション）することにより作業・仕事を進める。即ち、この分野の「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おいて大きいものが十分予測される。我々の研究グループ内でも以下の少なくとも３つの大きなメリットを持ち得ることが既に実績として実証された［1］［2］。

（1）手続き型モデルでは扱えないほどの複雑な対象世界が扱える。

（2）対象の動的な変更が非常に容易である。

（3）物を”もの（実現モジュール）”として扱いたい対象に有効である。

現状の動的なシミュレーションはそのほとんどが分析過程（概念モデリング・論理モデリング）までは自然言語ベースのオブジェクト指向に近いモデル化を行っている。しかし、設計過程（再構成モデリング）及び実装過程（再現モデリング、プログラミング）では殆ど全てが手続き的なモデル化及び手続き型言語（例外的に少数のシミュレーション専用言語）を用いて実現している。この原因はひとつには過去の経緯（計算機の能力の低さ、抽象度の高い対象をコンピュータ上に実現する理論の発達の遅れ）から手続き型のアーキテクチャが慣習的・実質的に使われてきているからである。

また、もうひとつには手続き型のモデリング及びプログラミングがある意味で既に十分確立されており、全ての面にわたって異なる概念であるオブジェクト指向シミュレーション方式［2］への移行は必要性があるのか、というユーザの意識の問題点がある。

一方において、オブジェクト指向モデリングパラダイムの概念体系がそれほど普及していないこと、モデリング方法そのもの及び言語やデータベース等のオブジェクト指向側の内部に関しても、まだ解決すべき課題が残っていることがあげられる。

更にもう一つの問題点は、エンドユーザがオブジェクト指向に基づくシミュレーションを実現しようをする際、モデリング過程の前半の抽象度の高い部分と後半の下流部分（実装過程）に近い部分とにギャップがあることである。

その原因の第１はエンドユーザにとって、プログラム表現のオブジェクト構造をどう規定及び表現すべきかが、見えないということにある。第２にはその駆動機構に基づく振舞いがイメージとして見えないところにある。

（注）オブジェクトベースとは、常時＜オブジェクト＞＝＜データ構造＞＋＜手続き＞であるとの定義を、自然界シミュレーションにおいてモデル化から駆動まで一貫して使用することを明確にするために導入した用語で、本来はオブジェクト指向と同義語である。

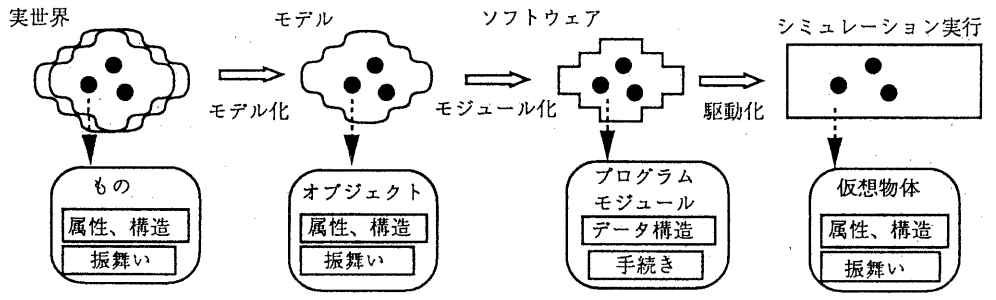


図１　モデリングとオブジェクト

もちろんなぜそう表現することがオブジェクト指向プログラミングになるのか、なぜそのように駆動するかについてのメンタルモデルがエンドユーザに構成されていないことも原因である。通常、エンドユーザはそのモジュールの駆動状態をイメージしつつ、モジュールをチェックしながら順次プログラムモジュールを作り上げて完成度を高くしていく。このため、駆動イメージとモジュールの関係が見えていないとオブジェクト指向プログラムは作るのに非常な困難を伴うことになる。

このような現状を解決するため、本研究の全体計画・最終目的は、オブジェクト指向モデリング分野からの発展状況と、現実のシミュレーション分野との状況とのギャップを埋めるため、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支援環境を構築することであるとした。すなわち、エンドユーザがオブジェクト指向シミュレーションを行う作業において、対象世界のモデル化を行い、それを自然言語から論理的な表現へ、そしてオブジェクト単位（プログラムモジュール）での表現・記述に変換し、それをC++にトランスレートし、出来上がったシミュレーションプログラムを起動・駆動するところまでを一貫して支援するシステムである。

言い換えると、この生成支援環境は一貫したモデリング過程のオブジェクト起動までの全過程を支援するものである。本稿では、その基礎的な考え方とモデルを記述していく際に最も重要となるモデル記述の方式、モデルを記述する言語、支援システム等の概要、及び現時点での評価について述べる。

2．モデル記述の方式概要

2．1　対象ユーザ、支援環境の条件

導入するモデル記述について、対象とするユーザ及び支援環境の持つべき環境は次のように考える。まず対象ユーザは以前からシミュレーションをやっていて、本来的に「自然なモデリングをやれる」実力があり、オブジェクト指向を概念的には理解している。しかし、実装過程では手続き型言語のFORTRAN、またはC言語等のプログラミングスタイルに慣れ切ってしまっていて、オブジェクト指向のプログラミングについて知識的に不足し、心理的にも抵抗感が大きいユーザを想定する。

一方、支援環境が満たすべき条件としては、分析過程（モデリング過程の認識モデリング、概念モデリング、論理モデリング）、設計過程（再構成モデリング）、実装過程（再現モデリング）等を順を追って変換（モデリング）する実際的な過程を、エンドユーザに違和感なしに意識せずにやらせるための支援システム機能が必要である。これはオブジェクトベースのプログラミングに関して不慣れなエンドユーザに、分析過程から直ちにオブジェクト指向プログラムを作成させるのではなく、N回のステップを繰り返し徐々にオブジェクト指向プログラムへ変換させる様に支援システム全体を構成することである。

この支援環境は、本来モデリング過程図に含まれている、よりきめ細かい詳細なモデリングステップ（５つのモデリング段階よりもモデリング落差が一桁少ない物をこう称する）を環境という形式を採用することにより、具体的にかつインタラクティブに柔軟にエンドユーザに自然に実現させていく方式そのものであり、モデリング過程図の実現を支援する環境の実現そのものであるといえる。

さらに、支援環境はエンドユーザに対するガイダンスの役割を果たすように構築する。少なくとも手続き型のモデル化と言語に慣れたエンドユーザがオブジェクトを基本に据えるモデリングとプログラミング方式になれるには相当の時間と努力が必要であり、そのためのガイダンス機能及び典型例・凡例表示機能を持たせる必要があると考える。また、「もの」が徐々に計算機上のオブジェクトのイメージに変換されていく状態を、ユーザの目に見えるような仕組みも支援する必要があろう。

これらの支援環境を構築する為に次のような記述方式を導入した。

2．2　モデル記述言語と支援環境

シミュレーションの対象世界を良く知るエンドユーザがオブジェクト指向の知識と多少のプログラミング経験があれば、対象世界をオブジェクトベースで記述できるような環境として、モデル記述言語（Model Description Language；以下、

MDL）と、その支援環境を導入する。エンドユーザは分析過程ではOMT記法［5］等の図式言語を用いて対象世界を分析し、本モデル記述言語で設計過程でのモデル記述を行う。

MDLを用いたモデルの記述方式はエンドユーザが使いやすいように、はじめに自然言語（日本語）でモデルを記述し、部分毎に順次MDLに洗練していくN－steps方式とする。エンドユーザは前のステップの記述を参考に次のステップの改訂を行う。このため、オブジェクト指向の枠組みの詳細、オブジェクト指向プログラミング言語（C++）の詳細をはじめから知る必要はなく、スムーズに実行可能なプログラムの記述に移行できる。図2にこの記述方式を示す。

MDLの仕様はオブジェクトベースモデルの自然な表現、すなわち記述から第3者が対象世界をイメージできることを目的とし、対象世界を自然にモデル化できるようにオブジェクト指向の概念を基本とする。また、対象世界をイメージしやすいように、キーワードを日本語表現とすることと、まだ厳密な文法を必要としない部分をあいまいな、あるいは抽象的な記述ができることとする（MDLの文法チェックを行わない部分を明確に表示する等の手段を採る）。

一方、この記述方式を支援する環境としては次の仕様が必要になる。

・一貫モデリング過程、オブジェクト指向の枠組み、及びMDLの記述等をガイダンスする機能
・MDLを実際に稼動するプログラム言語へ変換するトランスレート機構
・洗練の各ステップを蓄積し、いつでも前のステップへの記述を参照できる機能

等である。

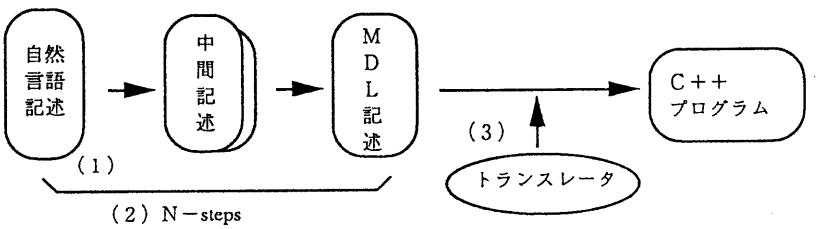
3．モデル記述言語の言語仕様

3．1　概要

MDLはエンドユーザにオブジェクトベースの概念の具体的作業をガイダンスする仕様とする。モデリングの各段階とオブジェクト、クラスの関係を示すと次のようになる。

モデル化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。クラス同士は何種類かの関係を持つことができ、対象世界の構造を表現できる。ソフトウェアの表現としては、クラスはオブジェクト間の共通仕様を記述すると同時に、対象世界の静的なクラス間構造を記述できる枠組みとなる。また、クラスはプログラム記述上ではモジュールの単位となる。モジュールはコンパイルの最小単位ともなる。また、オブジェクトは駆動時（実行時）にクラスから生成（再現）し、仮想物体としてシミュレーションの実行単位となる。駆動時の動作はオブジェクト同士のメッセージ交換で行う。

MDLのターゲット言語はC++とする。したがって、C++の仕様がMDL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。排除できない物理情報、例えば記憶エリア情報（static等）は、角括弧＜＞で封じ込める。



（1）対象世界のモデルをエンドユーザに馴染みの深い自然言語で記述する。
（2）自然言語で記述したモデルを順次洗練し、MDL記述にリライトする（N－steps）。
　　リライトは洗練化の規則を定めておき、それによりハンド変換を行う。
（3）MDLで記述したモデル（クラスなど）はトランスレータにより実行可能な言語C++へ変換される。

図2　記述方式

またMDLの文法チェック範囲を限定することで、トランスレートする度に順次プログラムを洗練させていくことができるようにする。

3．2　クラス仕様

オブジェクトの共通仕様を記述するクラスは、次の５つのセクションからなる。

（1）クラス定義セクション：クラス及びクラス間の構造等を記述する。
（2）属性セクション：クラスのもつ属性を記述する。
（3）記憶域管理セクション：属性の初期化を行う（モデル化に必要ではなく、次のプログラミング段階で必要なセクション）。
（4）動作セクション：クラスのメソッドを定義する。メソッドは振舞いを実現する関数である。メソッド記述は当面C++で行う。
（5）振舞いセクション：駆動時に仮想物体がどの様に振舞うかを記述する。当面コメントとする。

（1）　クラス定義セクションの仕様

本セクションにはクラスの性質、クラス間の関連情報を記述し、次の３つのパートからなる。

・モデル種別：ソフトウェア・ライフサイクルのどの段階のモデル化かを記述する。種別には再構成モデル、再現モデルがある。認識モデルはOMT等で記述する。
・クラス特性：クラスのもつ特性を記述する。特性はクラスがどのようなタイプのモデルかを表わすもので、アプリケーションが設計時に決める。クラスのグルーピングを可能にし、対象世界の構造をクラス表現したときのクラスの役割を明瞭にする。
　（例）　図形型、入出力型、ビュー型、情報モデル型等
・関係クラス：このクラスが関係するクラスを示す。クラス間の関係（汎化、集約、関連）は明示的に記述する。関係するクラスは関係の種別とクラス名を書く。関係の種別としては汎化（継承）を示す「is_a」、集約クラスを示す「a_part_of」、関連するクラスを示す「assoc」の３種類がある。「assoc」はクラス間の強い結びび付けを表現する。プログラミングレベルで言えばポインタ参照によるリンクである。
・持続性：クラスには持続性を指定できる。シミュレーションのオブジェクトはクラスから生成後、持続性を持たせることができる。オブジェクトを「もの」として静的に保存するのに必要である。

便宜上（プログラミング及び実行効率を上げるため）、プライベートな属性へのアクセスを許可する＜friend＞宣言を用意する。

（2）　属性セクション

本セクションにはクラスの属性を記述する。セクションはソフトウェアでの属性の有効範囲を定義する３パートからなる。

・私有：局所的属性を示す。有効範囲はクラス内に留まり、オブジェクト固有の属性を指定する。C++のprivateに相当する。
・限定共有：局所的属性を示す。有効範囲は階層構造の自己と子供のクラスであり、オブジェクト固有の属性を指定する。C++ではprotectedに相当する。
・共有：全クラスから見える（基本的には抽象データ型の考えに基づき本属性は使用しないことが望ましい）。　有効範囲とは別に物理的な場所を共有化する為に＜static＞属性を用意する。

（3）　記憶域管理セクション

属性の初期化及びヒープエリアの解放を行う。C++のコンストラクタ、デストラクタである。当面の記述はC++に準じる。

（4）　動作セクション

オブジェクトの動作を記述するセクションである。動作は機能単位（メソッドという）に細分化し、各機能は手続き的に記述する。モデルの種別が再構成モデルの場合はメソッドの外部インタフェースを記述する。手続き部分にはコメントを記述する。再現モデルの場合は即実行できるC++で記述する。メソッドインタフェースは基本的に他クラスに公開されており、C++のpublicとなる。

メソッドには差分プログラミングとしてメソッドを再定義できる（override）機能、ポリモフィズムプログラミングとしてメソッドの多重定義、仮想関数プログラムを作成できる機能がある。また、メソッドの種類は物理的なものとしてインラインかどうかの区別がある。インライン関数は＜inline＞をメソッド定義に付加する。

再構成モデルのメソッドインタフェースは次の形にする。メソッドにはシミュレーション実験をし易くするために動作等を分類したカテゴリ特性を付加できる。メソッドのパラメータには入出力の区別（in は入力 を表し 、outは 出力を表す）を指定できる。また、他言語とリンケージを取るためのインタフェースを指定できる。

（インタフェース）*メソッド名　リターン型*
メソッドパラメータ　メソッドカテゴリ
（記述例）　move void （in speed） move

（5）振舞いセクション

本セクションは仮想物体の駆動中の振舞いを定義する。当面コメントとする。

以上の各セクションの構文規則を図3に示す。

3．3　実現の状況

現在、MDLの言語仕様をほぼ決定し、サンプルを用いて、N－stepsの考えに基づいて中間レベルの記述およびMDLでの記述を行って評価中である。MDL記述はトランスレータ仕様をもとにハンドにてC++へ変換した。サンプルとしては、「もの」が相互に作用する例を取り上げた。

4．　記述例

MDLの記述例として物体の相互作用を伴うシミュレーション、スカッシュゲームを取り上げる（図4）。スカッシュゲームはオブジェクトとしてボール、ラケット、壁がある。ボールはラケットか壁にあたると反発し跳ね返る（相互作用）。ラケットはゲーム者がキーで操作し、ラケットがボールとあたると点数は加算される。試合は設定したタイムで終了する。

記述にあたっては対象世界のモデル記述と、シミュレーションで実験したい内容及び手順（オブジェクトの起動順序など）を明快に分離するために、オブジェクトベースで記述するモデル系とシミュレーションのライフサイクル「生成・動作手順・消滅」を記述する操作手順系に分けて記述した（図5）。

```
＜モジュール＞：：＝＜クラス＞
＜クラス＞：：＝" class"＜クラス名＞
                ＜定義セクション＞
               [＜属性セクション＞]
               [＜記憶域管理セクション＞]
               [＜動作セクション＞]
               [＜振舞いセクション＞]
＜定義セクション＞：：＝" %クラス定義"
                ＜モデル種別＞
               [＜クラス特性＞]
               [＜関係クラス＞]
               [＜持続性＞]
＜モデル種別＞：：＝" 再構築モデル" |
                  " 再現モデル
＜クラス特性＞：：＝＜任意の文字列＞
＜関係クラス＞：：＝＜汎化クラス＞|
                  ＜集約クラス＞|＜関係クラス＞
＜汎化クラス＞：：＝" is_a" ＜上位クラス名＞
＜集約クラス＞：：＝" a_part_of"
                  ＜集約クラス名＞
＜関係クラス＞：：＝" assoc" ＜関連クラス名＞
＜持続性＞：：＝" persistent"
＜属性セクション＞：：＝" %属性"
                   [＜私有属性＞]
                   [＜限定共有属性＞]
                  [＜共有属性＞]
＜私有属性＞：：＝" 私有" ＜宣言＞
＜限定共有属性＞：：＝" 限定共有" ＜宣言＞
＜共有属性＞：：＝" 共有" ＜宣言＞
＜宣言＞：：＝C＋＋の＜宣言＞
＜記憶域管理セクション＞：：＝
      " %記憶域管理" [＜コンストラクト＞]
                  [＜デストラクト＞]
    ＜コンストラクト＞と＜デストラクト＞は
  C＋＋に準ずる。
＜動作セクション＞：：＝" %動作"
              ＜メソッド＞の並び
 モデル種別が再構築モデルの場合
       ＜メソッド名＞　＜戻りの型＞　" ("
 ＜引数の並び＞　" )"
＜振舞いセクション＞：：＝" %振舞い"
                   ＜コメント＞
     コメントはセミコロン以降である。
```

図3　MDLの構文規則（一部）

モデル系は図形と動作メソッドのクラス群（ボール、ラケット、壁、そしてその上位の基本図形群）等のクラスからなる。

図６にモデル系の中間ステップを記述した例とMDLの記述例（一部）を示す。クラス「ボール」はクラスの性格として、図形型であり基本図形の円クラスを継承する。オブジェクトの持続性はこの例ではなくともよい。クラスの属性としてはボールの位置、大きさ、進行方向を持ち、メソッドとしては「与えられたスピードで移動する」を持つ。これを中間レベルで記述すると図６（ａ）のようになる。これをMDL仕様に沿うようハンド変換すると、図６（ｂ）のMDL記述となる。

一方、操作手順系はスカッシュゲームに現れるオブジェクトの生成、ゲームの手順に合わせたオブジェクトの動作手順、そしてオブジェクトの消滅からなるプログラムである。この手順系はシミュレーションの種類に応じて凡例的に作成できる部分でもある。

５．評価と考察

本研究の目的はエンドユーザが対象世界の自然なモデル及びメンタルイメージを出来る限り壊さないでモデリングの最終段階（プログラミング及び駆動）まで持っていけるような環境を構築することにある。本稿ではそれらの実現に関わる問題の初期段階の一部として、次の二点についての研究・開発を行った。

（１）Ｎ－stepsの記述方式を設定し、設計過程と実装過程をガイダンスするメニューおよび凡例参考表示による方式を用意する。これにより、自然言語を用いた要求記述から、最終成果物であるプログラムに至るまでのモデリング過程が容易に実施作業できるようになることを狙った。

（２）Ｎ－stepsのモデリングの最終記述として、オブジェクトベースのモデル記述言語（MDL）を用意する。このMDLはC++言語に変換される。これはモデリング過程の最終記述が実際のプログラムに変換されることを狙った。

記述例においてはそれに沿ってハンド変換を試み、一応良好な結果となることを確かめた。

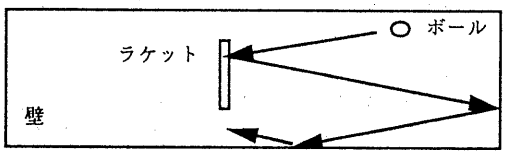


図４　スカッシュゲーム

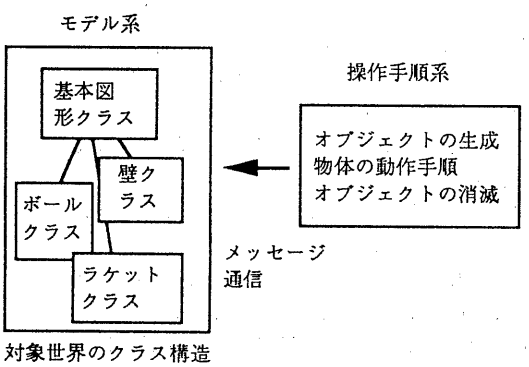


図５　記述構成

```
class　ボール
    クラス定義：図形型、持続性なし
                円（Circle）クラスを継承する
    属性：　ボールの位置、大きさ、進行方向
    初期値：　ボールの位置、大きさ、進行方向
    動作：　移動（移動スピード）
                ；ボールの動作範囲チェック
                ；ボールの反転処理
    振舞い：ボールは移動しながらラケットと壁
        と相互作用する
```

（ａ）　中間レベルの記述

```
class Ball              // ボールクラス
  %クラス定義
      モデル種別：実現モデル
      クラス特性：図形型
      関係クラス：is_a  Circle
                  <friend> InteractEnv
      ；持続性：none
  %属性
      限定共有：int x, y, r, dirX, dirY
  %記憶域管理
      初期化：Ball (int initx, nt inity,
                int initr, int dx, int dy)
  % 動作
      move void, (in int speed ) move
  %振舞い
      ；ボールオブジェクトは移動しながらラ
      ；ケットと壁と相互作用する
```

（ｂ）　MDLによる記述

図６　記述例

記述結果の考察として次の点があげられる。

（1）N－stepsのガイダンス過程

エンドユーザのモデリング作業としては、分析されたモデル記述をオブジェクトベースの概念に沿ってガイダンスされながら部分的に洗練することが出来る点が評価できる。しかし、MDLにユーザが慣れた場合（ガイダンスを必要としない）はそれに対応したガイドが必要になるであろう。ガイダンス実装に対しては、部分的に洗練されていく様子をビジュアル化して進捗状況をユーザに見せる工夫の要求があった。

（2）MDL

言語仕様はオブジェクトベースの基準に照らし属性、動作セクションを一まとまりのオブジェクトとして（ガイダンスされながら）記述できる点は評価される。しかし、記憶域管理セクション及び動作セクションの一部は、現状ではプログラミングの詳細な知識が必要であり記述しにくい点が残る。記憶域管理の記述そのものが、MDLの「出来る限りオブジェクト指向のプログラミングの技巧に関する記述をエンドユーザにさせない」という方針は必ずしも充分実現されたとは言えない。今後、エンドユーザとの議論の中で解決策を見い出す必要があろう。動作セクションの書き難さ解消の一つの方法としてはクラスライブラリの充実を考えている。

その他、実行時の操作手順系の標準化のためには、クラス名、メソッド名等のメニュー表示情報が必要であり、実行時に操作手順系とモデル系が情報を交換しやすい機構を統合的に扱う機構が必要であろう。

以上のような結果から本研究の目標である、「”もの”に一対一対応させたオブジェクト」の定義・設計・実現に関しては、簡単な例についてはMDLの実現に依って一応の水準に達したと判断し、順次複雑な対象に関してのテスト・評価を続けている。但し、オブジェクト間の相互関係（静的及び動的な相互関連と交互作用等）の記述に関してはまだ不十分なので今後充実させていきたい。

6．展望

評価結果を踏まえ、MDLトランスレータを含むシステムの設計を行っている。このシステムの中核に、N－stepsガイダンスを実現するためにモデル記述の各段階を管理する機構、MDLをC++へ変換するトランスレータ、および操作手順系の標準化のために実行時に操作手順系とモデル系が情報を交換しやすい機構等を扱う統合的な開発支援環境として、オブジェクトベース・リポジトリの概念を導入し、設計を進めている。現状、概念設計は完了し、並行してプロトタイプを作成中である。オブジェクトベースリポジトリの実現のために最初はオブジェクト指向データベースOntosを使用する予定である。

本システムは現在エンドユーザの意見を取り入れながら詳細な反復設計中であるので、総合評価はまだ出ていないが、ある程度の水準以上のものは出来上がるであろう。但し、一応の使い易さとモデリングの各詳細ステップを踏んでいけるようなメニューを出せるガイダンスシステムのバージョンは本年度に稼動を始めると考えているが、決定バージョンが完成する時期はだいぶ先になるでろう。

オブジェクトベース概念の自然なプログラミング支援環境を目標にした本研究は詳細設計付近にある現段階においては一応の成果を見たと言えよう。しかし今後、実際のシステム環境の構築過程において多くの問題が多数発生すること予想されるので、今後はそれらを充分ロングレンジで予測し、予め対処しつつ今後のオブジェクトベース生成支援環境の構築に取り組みたいと考えている。

参考文献

［1］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

［2］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

［3］米沢、柴山、「モデルと表現」、岩波書店、1992

［4］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

［5］Rumbaugh, J. et. al., ”Object-Oriented Modeling and Design”, Prentice-Hall, 1991（邦訳「オブジェクト指向方法論　OMT」羽生田監訳、トッパン）