ソフトウェア工学 118－3
（1998．3．10）

オブジェクト指向日本語一貫プログラミング環境

加藤木　和夫†　　畠山　正行‡

†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　工学部情報工学科

ドメインユーザ向けにモデル化からプログラミングまでをオブジェクト指向日本語で一貫して記述できる環境を開発した．本環境の特徴は開発各段階の記述にオブジェクト指向表現の相似形の日本語系記述言語を使うことである．一貫した記述過程を支援するために，その使用手順を明確化したNステップ開発手順，及び一貫支援型の連結記述エディタ，トランスレータとリポジトリ，そしてそれらを総合的に扱えるＧＵＩシステムを設計・開発した．本環境を解析シミュレーション分野に使用した結果，ドメインユーザは要求記述時のモデル記述イメージを一貫して持続でき，開発を容易にかつ短期間で行えることを確認できた．オブジェクト指向日本語の記述力等にまだ不足する部分あるが，日本語による一貫プログラミング環境を構築する狙いは実現できた，と言える．

Object-oriented Japanese Consistent Programming Environment

Kazuo Katougi†　　Masayuki Hatakeyama‡

† Hitach Process Computer Engineering, Inc.　　‡ Ibaraki University

We have designed and developed a programming environment that the domain users can consistently describe their software system using a series of the object-oriented Japanese. The most important point is that this environment have the series of the similar Japanese description language. To develop the programming environment, we have designed and implemented the N-steps modeling and developing guidance procedures, a kind of cooperative type describing editor, and the repository. As the conclusion, the purpose of the environment for the consistent programming have been implemented.

１．はじめに

ドメインユーザ向けのプログラミング環境を設計，開発した．ドメインユーザとは例えば素粒子物理学とか，画像処理等といったコンピュータ以外の分野の専門家であって，コンピュータを道具として使う一群のユーザを指すものとする．この様なドメインユーザの多くは自身で自身専用のプログラムを開発しなければならない場合が多い．したがって，使い易いプログラミング環境へのニーズや要求は強く多様である．例えば，作成すべきプログラムに対してドメイン側の知識やアルゴリズムについては完全に記述することはいとわないが，その記述言語はドメインに近い用語或いは自然言語であって欲しいし，プログラミングフリーを望む，といった強い本音要求がある．これに応えようというのが本研究の動機である．

ソフトウェアの非専門家のこの様な要求に対して，ビジネスドメインのユーザにはモデルが提供され解決が図られてきたが，モデルを画一的に規定できない分野のドメインユーザの期待には応えてこなかった．ドメインユーザは自ら対象世界を試行錯誤的にモデル化し，（望まないにもかかわらず）ソフトウェアを作成しなければならない．この観点から見ると，今までの方法は概念モデルの作成からプログラムの実行までのドメインのモデル化過程を支援する開発環境と，その表現言語系に一貫性を持たせていなかった．そこで本研究では，表現言語系の面から一貫した開発（プログラミング）環境を構築することを狙いとした．なお，本環境はドメインユーザの対象分野から考え，当面，中小規模のソフトウェア開発を前提とする．

本環境の狙いを実現する具体的な方法として，モデル記述表現からプログラム記述までずっと切れ目なく相似変換することと，モジュール化技術及びプログラミング技術を併せ持つオブジェクト指向技術を用いることとした．

以下では最初に第２章で筆者等が採用したアプローチ法，第３章では上流の記述言語，第４章では本環境の中核となる日本語プログラミング言語ついて述べる．そして第５章ではエディタ，第６章では本環境の実装と記述事例、第８章と第９章で本環境の考察，評価，結論及び今後の課題について述べる．

２．日本語一貫プログラミングへのアプローチ法

最初のモデル記述からプログラム記述までを相似な日本語系の表現言語を用いて連続的に変換していく方法を日本語一貫プログラミングと呼ぶこととする．以下，その様なプログラミングの言語系と支援環境について述べる．

２．１　日本語一貫記述の言語系

表現言語系の側面から見ると，オブジェクト指向に限らず通常のプログラム言語は，ドメインユーザの専門用語の基盤となっている自然言語である日本語の表現体系（文法）とはかなりかけ離れたものである．そこで，我々は日本語とプログラミング言語との間に一貫してオブジェクト指向の概念を反映した記述言語を取り入れることで，モデル化からプログラミングまでを一貫して記述できる表現言語体系を構築することとした．この様な方針の実現戦略として，我々は以下の様な相似な一貫した系列の記述言語系を考案した．

（ａ）要求記述段階の言語：制限のない自然言語
（ｂ）分析・基本設計段階の言語：論理モデルを記述する「オブジェクト 指向日本語」（ＯＯ－Ｊ：Object-Oriented Japanese）
（ｃ）詳細設計・実装段階の言語：日本語プログラム記述言語「オブジェクト指向日本語プログラム記述言語」（日本語ＳＭＤＬ：Software Modeling Description Language）

尚，これらを総称して「オブジェクト指向記述日本語」（ＯＯＤＪ：Object Oriented Description Japanese）と呼ぶこととする．

２．２　日本語一貫プログラミングの環境

我々は上記言語系の記述を支援する環境として以下の環境を設計，構築した．

（１）Ｎステップ開発手順[1]
（２）ＯＯＤＪの記述を一貫支援するエディタ
（３）日本語ＳＭＤＬトランスレータ
（４）リポジトリ[2]
（５）ＧＵＩ[3][4]

尚，ＯＯ－Ｊは複雑な表現も許す為，解釈は自然言語処理の手法を取り入れず，人間が行うものとする．

３．オブジェクト指向日本語（ＯＯ－Ｊ）

本言語は論理モデルをオブジェクト指向表現するために日本語にオブジェクト指向の概念構成及びその表現力を強化したものである．

（１）クラス概要記述のためのキーワード

キーワードはクラス間関係等を記述する「クラス属性」，及び「属性」と「動作」である．記述

```
クラス　　ボール
    クラス属性：図形の円クラスを継承。
    属性：ボールの位置、大きさ、進行方向
    動作：移動せよ（ ）
          表示せよ（ ）
```

図１　クラス記述例

は自由形式とし，ユーザが入力したデータはマウスにより取り込み，リポジトリへ格納する．クラス記述例を図１に示す．

（２）アルゴリズム記述オブジェクト指向記号

（ａ）インスタンス生成記号（＝＞）

記号〝＝＞〟はインスタンス生成を明示するために導入する．

クラス名 ＝＞ インスタンス名

（例）　ボール　＝＞　スカッシュボール

（ｂ）メッセージ通信記号（←）

オブジェクト指向のアルゴリズムの基本はオブジェクト間のメッセージ通信である．そこで，記号〝←〟を導入する．

オブジェクト ← 振舞い

（例）　スカッシュボール ←移動せよ

本記号はちょうど日本語の「ヲ」または「ニ」に相当する．

（ｃ）プログラムの制御構造用英字キーワード

プログラムの制御構造は日本語表現よりは英字キーワードの方が馴染み易いこともあり，以下の記号を導入する．

連接：□

選択：ＩＦ～ＴＨＥＮ～ＥＬＳＥ～ＥＮＤＩＦ．

（ｉｆ，IF，if　も許す）

繰返し：ＬＯＯＰ（判定条件）｛　～　｝．

（ｌｏｏｐ，LOOP，loop　も許す）

ＯＯ－Ｊによるアルゴリズム記述例を図２に示す．

```
□ボール＝＞スカッシュボール．
□ラケット＝＞ラケット．
□メニュー←プレー条件を入力する
      （制限時間、スピード、得点マックス）．
□スカッシュボール，ラケット←表示する．
□メニュー←開始する．
□ＬＯＯＰ　終了条件（制限時間オーバー）｛
    ＬＯＯＰ　終了条件（ラケット移動あり）
          //　移動は割り込みで判断する．
        {スカッシュボール←移動せよ(速度)．}
  ラケット←移動せよ．
  }
□スカッシュボール，ラケット←消滅させる．
```

図２　ＯＯ－Ｊによるアルゴリズム記述例

４．　日本語ＳＭＤＬ

４．１　日本語ＳＭＤＬの定義

オブジェクト指向日本語プログラム記述言語として日本語ＳＭＤＬを定義・開発した．日本語ＳＭＤＬはＯＯ－Ｊで記述されたクラスの概要，動作のアルゴリズムを更に詳細に記述し，ちょうど詳細設計段階から実装段階の表現言語である．詳細な構文は付録参照．日本語ＳＭＤＬの記述単位はクラスである．クラス記述の構成はクラス定義，属性，動作の３セクションから成る．

４．２　クラス定義セクション

（１）クラス名：

クラス名は日本語名を記述する．対応する英字名も記述すれば，Ｃ＋＋へ変換されたときのクラス名として使われる．英字名を記入しなければシステムが自動的に割り当てる．

（２）関連クラス：

このクラスが関連するクラスを示す．関連クラス名は一覧表として表示される日本語クラス名のメニューの中から選択する．クラス間の関係は汎化，集約，関連の３種類を設け，種別の選択は選択ボタンで行うこととする．

４．３　属性セクション

（１）属性名：

属性名はクラス名同様に日本語名，英字名を記述できる．英字名の記述がなければ自動割当となる．

（２）属性のデータ型：

データ型は基本データ型，配列型とクラス型が

あり，基本データ型には整数型，実数型，論理型，文字型とそのバリエーションがある．また，クラス型は属性として基本データ型とクラス型を内包する．データ型は全て（基本データ型，クラス型）一覧表示されたものから選択する．

（3）属性の有効範囲：

私有，限定共有，共有の3種類を設け，種別の選択は選択ボタンで行う．

4．4　動作セクション

本セクションはオブジェクトの動作を記述する．動作記述にはオブジェクト指向を特長付けるオブジェクト指向記号を導入するが，全体的にはドメインユーザのメンタルモデルと考えられる手続き型のプログラミング記述を踏襲する．また，抽象データ型を実現する為にＧＵＩは外部インタフェースを記述する動作インタフェース部とアルゴリズムを記述する動作記述の本体部から構成し，動作インタフェース部に記述したインタフェースは基本的に他クラスに公開されるものとする．

（1）動作インタフェース部

インタフェース名（メソッド名相当）及び引数名は日本語，英字名の両方が記述でき，英字名が省略されれば自動割当となる．

（2）動作本体記述部

動作本体の部分は抽象的な記述であるアウトライン記述（上位階層）と具象化した日本語プログラム記述（下位階層）の2階層に分けて記述する．

（a）上位階層（アウトライン記述）

1行程度の自然言語またはＯＯ－Ｊ表現による連接（先頭に□記号を付す）で，プログラム手順のアウトライン（プログラムスライスと呼ぶ）を記述する．必要ならその順序は後で入れ替える．プログラムスライスの中身でよく使用するものはコーディングイデオムとして最小部品とし，リポジトリに蓄積できる．なお，上位階層の記述はＣ＋＋変換時はコメントとなる．アウトライン記述例を図3に示す．

```
□ボールの動作範囲チェック
□ボールの反射処理
□移動処理
```

図3　アウトライン記述の例

（b）下位階層（日本語ＳＭＤＬプログラム）

上記アウトラインの1行毎に日本語ＳＭＤＬプログラムを記述する．下位階層のプログラムはトランスレータの対象となる．

（3）オブジェクト指向記号の定義

アルゴリズム記述の基本はオブジェクトへメッセージを送る形式である．そこで，メッセージ送信の構文を以下のように記号化し，オブジェクトとメッセージを分離し見やすくした．

オブジェクト　←　メッセージ

（例）スカッシュボール　←移動せよ［速度］．

なお，ＯＯ－Ｊのインスタンス生成記号に相当するものはない．インスタンスはＣ＋＋と同様に属性に記述されれば生成される．

（4）手続き型プログラミング

動作本体記述の基本形は理解し易さを考慮して制御文，四則演算にはオブジェクトにメッセージを送る形式は採用せず手続き型の考えに従う．

（a）制御文

手順の流れを見やすくするために制御文はＯＯ－Ｊ同様に英文字を使用する．日本語での表現も考えられるが，読み易さからいえば英大文字の混在が認識し易い．

（例）　ＬＯＯＰ　ＩＦ　ＣＡＳＥ

（b）四則演算

手続き型言語を踏襲する．

（例）距離：＝速度＊時間．

日本語ＳＭＤＬの記述例を図4に示す．

```
ＩＦ（　（ 壁←左側 ＞
　（ボールの中心Ｘ＋Ｘ方向＊（半径＋移動））
　ＯＲ
　　（ 壁←右側 ＜
　（ボールの中心Ｘ＋Ｘ方向＊（半径＋移動））
　ＴＨＥＮ　Ｘ方向：＝Ｘ方向＊（－1）．//Ｘの反転
　ＥＬＳＥ　｛ ｝
ＥＮＤＩＦ．
```

図4　日本語ＳＭＤＬの記述例

5．OODJ一貫支援エディタ

5．1　一貫支援型エディタの概要

開発過程を通じて当初のドメインユーザのモデルイメージを一貫して保てるようにプログラム記述の中でクラス単位，メソッド単位，プログラムイデオムの各単位で編集・開発できるエディタが必要である．

そこで要求記述を順次，連続的に日本語ＳＭＤＬプログラムまで書き換えていくことを支援する，連結型のエディタを設計，開発した．エディタは大きく分けてＯＯ－Ｊまでを編集する日本語メモ記述エディタと，それ以降の全ての過程を支援する日本語ＳＭＤＬエディタとから成る．

5．2　日本語メモ記述エディタ

本エディタは日本語及びＯＯ－Ｊ記述を自由形式で記入するエディタである．ユーザは要求記述，あるいはクラス概要記述をこのエディタで行う．

本エディタの特長は通常のエディット機能に加えて，図５に示すようにユーザの記述した日本語の文章の中からクラス名候補，属性名候補，メッセージ名候補をマウスを用いて切り出し，リポジトリ中のデータベース（プールと言う）へ登録することができることである．

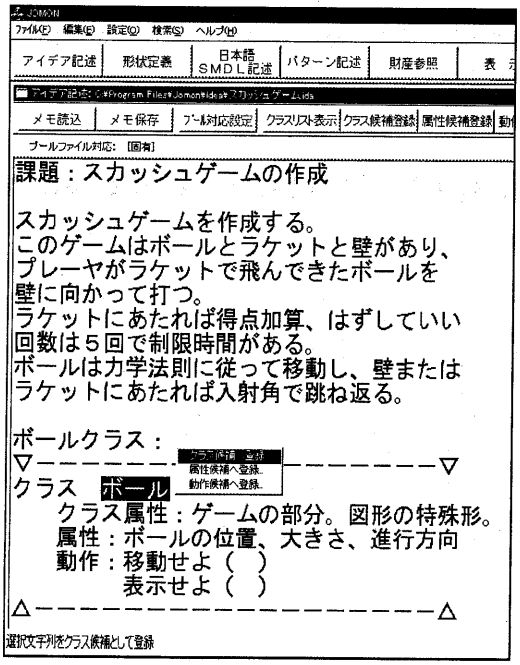


図５　日本語メモエディタ

5．3　日本語ＳＭＤＬエディタ

プールへ切り出したクラス名，クラスの属性名，クラスの動作名候補等は次のステップの日本語ＳＭＤＬエディタを起動したときに，リポジトリから自動的に取り出され，ＳＭＤＬエディタの主画面中に自動的に展開され，クラスフレーム（枠組み）を形成する（図６）．クラス名候補はクラス名の個所に，属性名候補は属性名の個所に各々リスト表示される．

日本語ＳＭＤＬエディタの主画面は，クラス仕様関係，属性仕様関係，動作仕様関係の欄からなる．これら各欄は各々の詳細定義詳細を入力する補助画面が用意されており，記入方式はユーザが画面表示に従って単語をキーイン，あるいはリスト表示から必要事項を選択する方式である．

動作本体を日本語ＳＭＤＬで記述する例は次章の適用例を参照のこと．

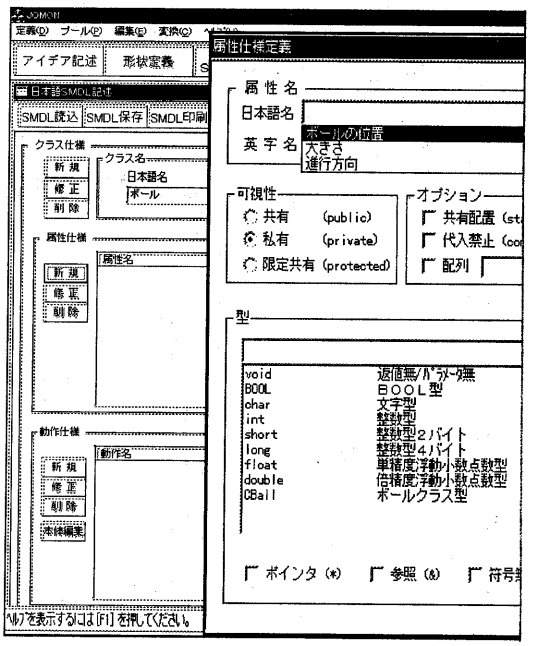


図６　日本語ＳＭＤＬエディタ（自動展開）

6．　実装と記述事例

6．1　実装

本環境自体もオブジェクト指向を用いて開発しており，約８０個のクラスから構成される．基本ＯＳはWindows95／NTで，ＧＵＩフレームワークとしてはＭＦＣを利用して実装している．また，本環境は日本語プログラミング環境としてほぼ同じものが既にＪOMON＋というコード名で実装，試験的に使用中である[4]．

6．2　記述事例

一貫記述プログラミングの記述例として，スカッシュゲームを取り上げる．Ｎステップ開発手順にそって記述例を以下に示す．

（1）自然言語による要求記述

自然言語による要求記述例は図１のエディタ画面中に記述例を上げた．ユーザはこの要求記述をラフスケッチ，ＯＭＴ表記[5]等のダイアグラムを用いて分析し，その結果をオブジェクト指向言語ＯＯ－Ｊで表現する．

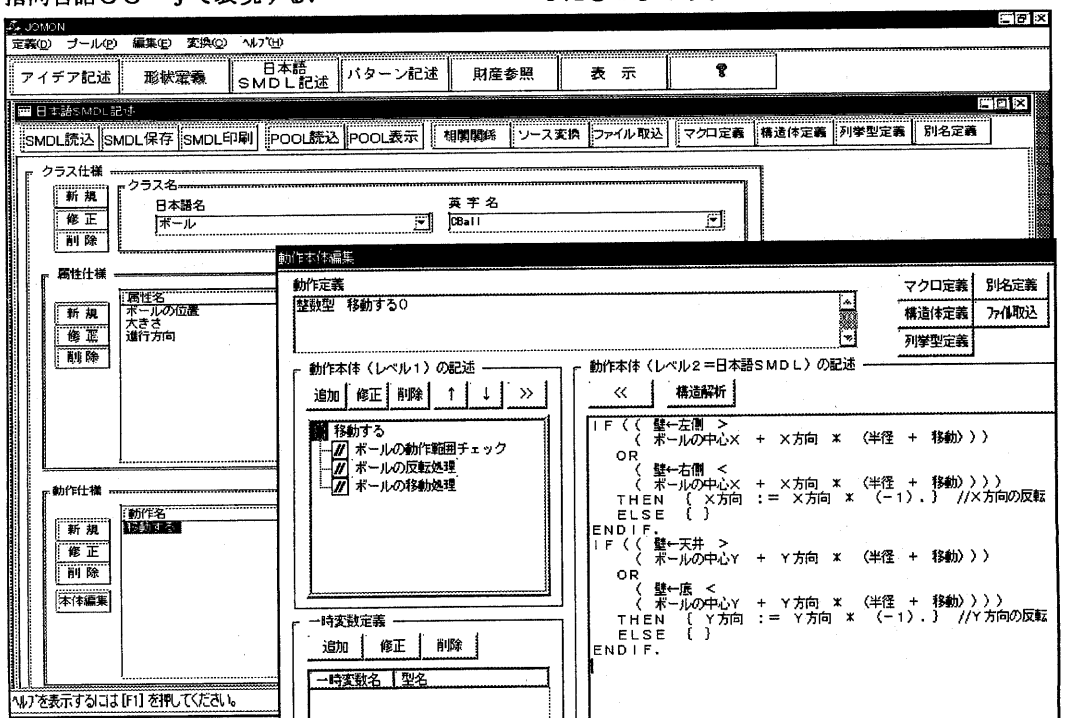
（2）ＯＯ－Ｊによる記述

クラス記述例とアルゴリズム記述例は第３章の図１，図２に示した．ユーザはクラス記述後図５に示すようにマウスでクラス候補等を選択，登録すると，図６の左半分のように日本語ＳＭＤＬエディタの主画面にクラス名等が自動展開される．

（3）日本語ＳＭＤＬによる記述

そこで，ユーザは画面誘導に従ってクラス仕様，属性仕様，動作仕様の詳細を記入する．図６の右半分は属性「ボールの位置」の詳細をリストから選択しているところである．

ユーザは次に動作の本体を日本語ＳＭＤＬで記述する．図７にボールクラスの動作本体記述例を示す．図７の（ａ）画面は日本語ＳＭＤＬエディタの主画面から動作インタフェースとして「移動する」を選択したところで，（ｂ）画面ではその本体を記入するところである．（ｂ）画面の左側は上位階層で「移動する」のアウトラインを記入で，画面の右側はアウトラインの一部である「ボールの反転処理」の詳細を日本語ＳＭＤＬで記入したところである．



（ａ）主画面　　（ｂ）動作本体記述（アウトラインと日本語ＳＭＤＬ）

図７　日本語ＳＭＤＬエディタ（主画面と動作本体の記述）

7. 考察

（1）オブジェクト指向日本語

ＯＯ－Ｊはドメインユーザが日常使用している言語に近い馴染み易い記述形式であるが，この種の記述形式はプログラミング環境で使える記述言語としてはほとんど見かけたことがない．そこで，日本語構造の中にオブジェクト指向の基礎概念を記号化していれることで，用法を制約した自然言語でオブジェクト指向的な記述に慣れ親しむことになる．その結果，要求をオブジェクト指向の世界へ移行するための整理，すなわち，要求記述のどの部分をオブジェクトとし，どの部分をメッセージとするかの考察と分析に記述力の強い自然言語(日本語)の記述力を借りることが出来た．

（2）日本語ＳＭＤＬ

日本語ＳＭＤＬの記述力は動作の概略記述には十分であるが，付録の一覧表からも分かるようにプログラムを記述する上での枝葉的な仕様を現状では省略しており，細かい動作の記述力には不足である．しかし，ユーザの記述試行結果からは日本語化したライブラリが充実してくれば，その引用によりプロトタイプの記述には十分可能な記述力を持つ，という評価を得た．

（3）日本語ＳＭＤＬとエディタ

日本語ＳＭＤＬ用のエディタでは抽象的なアウトライン記述から具象化への展開，具象的なプログラムから抽象化への双方向移行が容易に出来る．すなわち，上位階層でプログラムを部分的に括って日本語で抽象表現をしておいたり，逆に日本語ＳＭＤＬ表現に展開できる．したがって，ユーザは詳細情報の階層的隠蔽／表示／編集が可能となる．また，第３者がプログラムを読む場合，上位階層の１行化日本語記述でプログラムの要旨を把握し，下位階層の対応部分を参照することでアルゴリズム把握が従来よりもはるかに簡潔，容易になる．

（4）Ｎステップ開発手順

本手順はウォーターフォール型のソフトウェア開発手法である．しかし，従来と異なる点は，開発の各段階に適応した相似な系統の限定日本語記述言語を導入して，言語面から連続的にとらえようとするところにある．従来の開発プロセスは要求記述，仕様書，ＯＭＴ表記，C++プログラムと個々にに対応する部分はあるが，記述系や表現に大きなギャップを抱えている．今回はすべてを文章表現の遷移，深化でとらえることで，開発各段階の移行にスムーズ感が得られている．

（5）ＯＯＤＪ一貫支援エディタ

日本語メモエディタと日本語ＳＭＤＬエディタを連結型とすることにより，「モノ」から「オブジェクト」へ，そして「プログラム・モジュール」への一貫したイメージで，かつ必要なだけ前段階へ戻っての改定を行うプログラム作成が可能となった．

8. 結論と今後の課題

本環境によりドメインユーザはオブジェクトを定義する際にＣ＋＋等のプログラム言語を直接使わずに，一貫してＯＯＤＪを用いて記述できる様になった．日本語風の記述のままＮステップを経て順次オブジェクト指向記述に変換していけば違和感少なくオブジェクトが生成される．

現状では本環境を製品としてみるには未だ完成度があまり高くなく，特に辞書やライブラリの整備が進んでいないために，ドメインユーザの負担はメソッドの実装部分の負担軽減にはつながっていない．しかし，日本語インタフェースによるラッパーを用いた再利用でこれを避けることは出来うると考える．

本環境の今後の課題は次のように纏められる．

（1）ユーザインタフェースの高度化

（2）応用分野毎の辞書の充実

（3）ＯＯＤＪの充実

（4）日本語自動実行環境の構想

参考文献

1）加藤木，畠山：オブジェクト指向シミュレーション・モデル記述の開発環境，情報処理学会第９８回ソフトウェア工学研究会，Vol.94, No.43, pp.9-16 (1994)

2）加藤木，畠山，小林：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，情報処理学会第１０１回ソフトウェア工学研究会，Vol.94，No.99，pp.57-64（1994）
3）加藤木，畠山他：シミュレーション向けのオブジェクト生成支援環境の設計と実装，オブジェクト指向'95 シンポジウム，情報処理学会シンポジウム論文集，Vol.95，No.3，pp.205-212（1995）
4）加藤木，畠山他：統合日本語プログラミング環境「ＪOMON＋」，HIPRO技報，第２号，pp.35-40（1997）
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，（株）トッパン）

付録　日本語ＳＭＤＬ構文一覧表

```
<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>]
                [<動作セクション>]
<定義セクション>：：＝"%クラス定義"
                [<クラス特性>]
                [<関連クラス>]
<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>
<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>（制限あり）

<動作セクション>：：＝"%動作"
            <メソッド>の並び
<メソッド>：：＝<文の並び>
<文の並び>：：＝<文>". "<文>". "・・・
                  "RETURN"<文>". "
<文>：：＝<メッセージ文>｜<LOOP文>｜
   <IF条件文>｜<CASE条件文>｜<代入文>
<メッセージ文>：：＝<メッセージ式>
<LOOP文>：：＝ "LOOP"  <条件>
                " {"  <文の並び>  "} "
<IF文>：：＝ "IF"<条件>   "THEN"
                    " {"<文の並び>"} "
              "ELSE"  " {"<文の並び>"} "
             "ENDIF"
<CASE文>：：＝"COND"<  条件 >    " {"
      "CASE"  定数式１
                " {"  <文の並び>  "} "
      "CASE"  定数式２
                " {"  <文の並び>  "} "
                  ・・・
      "} "
<条件>：：＝"<"  <項>  ">"
<代入文>：：＝  <変数>  ": ="  <式>
<式>：：＝（  <メッセージ式>  |
          <２項式>  |  <単項式>  ）

<メッセージ式>：：＝  <項>
      ("<－"  |  "←") <メッセージ>
<２項式>：：＝<項>◎<項>
                          ◎：２項演算子
<単項式>：：＝○<項>     ○：単項演算子
<メッセージ>：：＝
 メッセージ名  " ["  (<項>, ) . . . "] "

<項>：：＝  <基本項>｜
          " ("  <式>  ") "
<基本項>：：＝  <インスタンス変数>  |
              <リテラル>
```