ソフトウェア工学 108－9
（1996．3．22）

オブジェクト指向モデル記述言語 Bramble の開発

上田 賀一，　中野 喜之，　金村 星吉，　高橋 大輔

茨城大学 工学部 情報工学科

〒316　茨城県日立市中成沢町 4-12-1

ソフトウェアシステムの開発において要求者と開発者の意思疎通を図るためにはオブジェクト指向によるプロトタイピングが有効である．しかし，分析や設計段階で用いたオブジェクト指向パラダイムの種々の概念を実装段階で素直に記述できるオブジェクト指向言語は少ない．

本研究では，動的振舞いや種々の関係を柔軟に記述でき，一貫したオブジェクト指向開発を可能とするモデル記述言語 Bramble を開発した．本言語は，全ての要素をオブジェクトとして扱い，コピーベースのオブジェクト指向を核とするインタプリタ言語である．また，オブジェクトはメソッドと変数を区別しない属性だけを持ち，文字列だけを基本データ型とする．さらに，言語処理系自体を開発環境や CASE 環境と拡張できるように，グラフィックスインタフェースやデータベースインタフェースの機能をもつオブジェクトも基本オブジェクトとして持たせている．

Bramble : Object-oriented Model Description Language

Yoshikazu Ueda,　Yoshiyuki Nakano,
Seikichi Kanemura,　Daisuke Takahashi

Ibaraki University

4-12-1 Naka-Narusawa, Hitachi, Ibaraki, 316 Japan

In a software system development, a prototyping with object-oriented paradigm is useful for the better understanding between requirers and developers. Most of the current object-oriented programming languages can't represent the concepts of models in object-oriented analysis and design well.

This paper presents a model description language: Bramble which represents the dynamic behaviors of an object and the flexible relationships between objects. Bramble's characteristics are to treat every element as an object, to be copy-based object-orientation, to have attributes not to distinguish methods and variables, and to have an only data type as a string. Furthermore, Bramble includes the several base objects that work as the interfaces of graphics and database in order to extend the interpreter to the development or CASE environment.

1．はじめに

大規模なシステムでは，要求者，開発者間の意味的な隔たりが大きくなり，開発の大きなネックとなっている．そのため，システムの実行可能なモデルを早期に作成し，動作確認をすることで要求を明確にしていくプロトタイピングの必要性が高まっており，オブジェクト指向開発は有効な方法であると言える[1]-[4]．しかし，オブジェクト指向分析や設計で用いられる種々の関係を含んだモデルを変換することなくオブジェクト指向言語で実装することは難しい．それは，分析や設計段階で使える概念が実装段階では使えないからであり，柔軟な記述が可能なオブジェクト指向言語が望まれる．また，オブジェクト指向によるプロトタイピングを有用なものとするためには，動的な振舞いを記述できることも必要である．

本研究では，これらの問題を解決するべくオブジェクト指向モデル記述言語Brambleを開発したので報告する．

2．言語の特徴と仕様

2.1 言語の特徴

本言語の特徴を以下に挙げる．

- 全ての要素がオブジェクト．
  オブジェクトの集合は，その集合で一つのオブジェクトとして定義され，言語自身が振舞うために必要な要素もオブジェクトとして持つ．
- コピー操作を主体としたオブジェクト指向．
  クラス-インスタンスの概念が無い．
- オブジェクトは属性だけを包含．
  オブジェクトは変数とメソッドを区別せず，属性として包含する．
- メソッドもオブジェクト．
  メソッドの役割を持ったオブジェクトをメソッドオブジェクトと呼び，これがオブジェクトの属性となる時，メソッド属性と呼ぶ．

2.2 仕様の概要

本言語の基底概念として，オブジェクトの構造，その操作手順，そしてメッセージ送信プロトコルが定義されている．その定義にそって，本言語の基本的な機能を提供するいくつかのベースオブジェクトが定義される．一つは，オブジェクトの基本的な操作を行うベースオブジェクト Object である．また，オブジェクトが持ち得る文字列の操作の機能を提供するベースオブジェクト String がある．本言語では，文字列が基本的なデータ型である．文字列は，それに送られるメッセージによって，文字列，リスト，整数，メソッドの役割をする．本言語ではオブジェクトの持つメソッドと属性に違いがない．それは，メソッドが文字列のプログラムコードを値として持つ文字列オブジェクトとして定義されているからである．

また、これら以外のベースオブジェクトとして，真偽値を表す True，False，オブジェクトの雛型を表す Template が定義されている．

ベースオブジェクト String のメソッド属性の内，eval は本言語の文法を規定する．メソッド属性にこの eval メッセージを送る(メソッド属性のメソッド eval を起動する)と，eval は，メソッド属性に記述されているプログラムコードを解釈し評価する．

2.3 オブジェクト

- オブジェクトの構造

本言語の基本的な単位はオブジェクトであり，属性名と属性オブジェクトへの参照のペアの集合から成る．メソッドも，プログラムコードの書かれた文字列オブジェクトなので，属性になっている．

全てのオブジェクトは，継承のためのオブジェクト super-objec を属性として参照している．この super-object は，ベースオブジェクト String を継承した文字列オブジェクトで，これを参照しているオブジェクトが自ら参照している属性オブジェクトの内，継承する属性名を値として持っている．

また，全てのオブジェクトはベースオブジェクト Object を継承していなければならない．これは，オブジェクト単体では，属性の追加，継承，属性の探索などの基本的な機能さえ持っていないからである．通常オブジェクトは，オブジェクトの雛型である Template オブジェクトからコピーされることで生成されるが，そのコピーされたオブジェクトは，ベースオブジェクト Object を継承している．

- オブジェクトの操作

本言語ではオブジェクトの操作は，属性の検索，変更，オブジェクトの継承，コピー，比較に分かれる．これらの操作は，ベースオブジェクト Object に find-attribute, add-attribute, inherit, copy, = という名称で割り当てられている．

属性の検索は，属性名を表す文字列を指定して，オブジェクトからそのオブジェクトが参照している属性オブジェクトを取り出す操作である．この操作では，まず検索対象のオブジェクトの属性を調べ，その後，継承しているオブジェクトを継承の優先順

位の高いオブジェクトから順に検索し，最初に一致した属性を目的のオブジェクトとする．

属性の変更は，属性の追加/削除も兼ねている．この操作は，変更(追加/削除)したい属性の名前を表す文字列と，属性として参照したいオブジェクトを指定して行われる．もし対象としているオブジェクトに指定した属性名の属性が無ければ，その属性は新しく作られる．また，参照するオブジェクトがNULL オブジェクトであったなら，その属性は削除される．継承しているオブジェクトに同名の属性があったとしても，この操作は継承しているオブジェクトに影響を及ぼさない．

オブジェクトの継承は，継承するオブジェクトの名前を表す文字列と継承したいオブジェクトを指定して行われる．

また，これらの操作とは別に，オブジェクトの文字列としての操作がベースオブジェクト String にメソッドとして定義されている．

- オブジェクト間の関係

本言語中のオブジェクト同士の関係は，参照関係(図1)と継承関係(図2)という二つの有向グラフで表すことができる．参照関係は，ベースオブジェクト root から始まり，必ず root から全てのオブジェクトに到達可能なグラフである．また，継承関係は，オブジェクト毎に存在し，そのオブジェクトを始点として，ベースオブジェクト Object を終点とするグラフとなる．

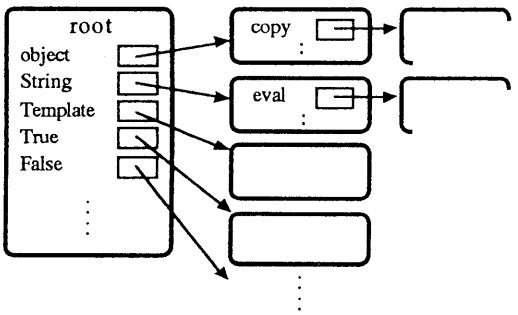

図1．オブジェクトの参照関係

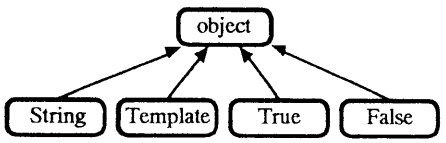

図2．オブジェクトの継承関係

2.4 文字列オブジェクト

文字列は本言語での基本的なデータ単位である．

オブジェクトは，ベースオブジェクト String を継承することにより，文字列オブジェクトとしての機能を持つことができる．

以下に，文字列を記述するための BNF を示す．ここで，“word”は，バッククォート “`”，コロン “:”，左括弧 “(”，右括弧 “)”と，スペース，タブ，改行コードで区切られた文字列である．

```
<string>    ::= " <words> "
              | { <elements> }
              | word
<words>     ::= word
              | word _ <words>
<elements>  ::= <string>
              | <string> _ <elements>
```

構文1．文字列の文法

- 文字列の表記

文字列は，ダブルクォート(")もしくは，中括弧({})で括ることで表される．文字列をリストとみなす時，スペースやタブ，改行を含まない部分，または，ダブルクォートか中括弧で括られた部分を一つの要素とみなす．また，この中括弧はネストすることができる．

- 文字列の演算

全ての文字列オブジェクトは，文字列として，リストとして，整数としての演算をするためのそれぞれのメソッドを持っている．

これらのメソッドは，ベースオブジェクト String に定義されおり，このオブジェクトを継承することによってオブジェクトを文字列オブジェクトとして機能させることができる．

しかし，文字列の演算は全てメッセージ送信によって行われるため，演算間の優先順位がない．よって，明示的にメッセージ送信の順序を決めなければならない．

- プログラムコード

通常の文字列オブジェクトとプログラムコードとしての文字列オブジェクト(メソッドオブジェクト)の，言語構造上の区別はない．しかし実際の記述においては，プログラムコードの記述は，次節に述べる文法に基づいて記述されなければならない．

2.5 文法

本言語のプログラムコードはすべて文字列で表されている.

プログラムコードの単位は, 文(sentence)で, これには以下の4種類がある.

- メッセージ送信文
- オブジェクト
- 代入文
- リターン文

代入文は, 内部で属性の変更(add-attribute)に直され, メッセージ送信される. また, 代入先の属性が, カレントオブジェクトにある場合はその属性が変更され, 無い場合はローカルオブジェクトに追加/変更される. リターン文は, プログラムコードの評価を終了し, その後にあるオブジェクトをその評価の戻り値とする.

リターン文を除くいずれの文も, 戻り値はオブジェクトであり, 括弧 "(", ")" で括ることによってメッセージ送信や代入文の一部として使うことができる.

プログラムコード中では, 文字列オブジェクトを表すために, 文字列の前にバッククォート"`"を付けなければならない.

プログラムコードは, メッセージ送信や文字列オブジェクトの評価(eval,while)によって実行される. プログラムコードをメッセージ送信して評価する時, メッセージ送信されたオブジェクトをカレントオブジェクトと呼ぶ. また, プログラムコードを評価した時のカレントオブジェクトは, そのプログラムコードが書かれていた文字列オブジェクトの評価時のカレントオブジェクトになる. プログラム中で, ベースオブジェクト root の属性名, カレントオブジェクトの属性名, ローカルオブジェクトの属性名を書くことで, その属性が参照しているオブジェクトを表すことができる.

以下に, BNF 記法で記述した本言語の文法を示す. なお, 下記 string は, 文字列の文法によって切り出された文字列であるので, ネストした文字列(e.g. {a b {c d {e} f} g h})も一つの文字列であるとみなされることに注意が必要である.

```
<program>  ::=  <sentences> eol
<sentences> ::= <sentence> ;
              |  <sentence> ; <sentences>
<sentence>  ::= <message>
              |  string <- <object>
              |  ^<object>
              |  <object>
<object>    ::= string
              |  ` string
              |  ( <object> )
              |  ( <message> )
<message>   ::= <object> string
                    <message_param_list>
              |  <object> string <object>
              |  <object> string
<message_param_list>
            ::= string : <object>
              |  string : <object>
                    <message_param_list>
```

構文2. 本言語の文法

- メッセージ送信

プログラムコードの基本的な単位は, メッセージ送信である. このメッセージ送信は, 送信先オブジェクト, メッセージ名, メッセージ実行に必要なパラメータを特定することによって行われる.

メッセージが送信されることにより, 対象となるオブジェクトのメッセージ名と同名のメソッドオブジェクトが起動される. このメソッドオブジェクトの評価値が, メッセージの戻り値となる.

メッセージ送信は次のように記述される.

object message-name tag-name: parameter ...

object は, メッセージ送信先のオブジェクトである. *message-name* は, メッセージ名を表す文字列である. メッセージに必要なパラメータは, *tag-name: parameter* という形式で記述される. *tag-name* はタグ名, *parameter* はパラメータとなるオブジェクトである. これらはメッセージ送信後, 起動されたメソッドのローカルオブジェクトとして *tag-name* という名前で *parameter* で示されたオブジェクトを参照するように設定される.

- ローカルオブジェクト

メソッドオブジェクト実行時にそのメソッド内でのみ使われるオブジェクトは, メソッドオブジェクトの属性であるローカルオブジェクトに登録される. メッセージ送信におけるパラメータは, 属性名をタグ名としてこのローカルオブジェクトに, また, そのメソッドオブジェクトのカレントオブジェクトも, self という属性名で登録される.

2.6 オブジェクトの記述

実際にオブジェクトの作成方法を説明する.

- オブジェクトの生成

一般的なオブジェクトの生成は, ベースオブジェ

クト Template をコピーし得られたオブジェクトに属性を追加する．具体的には，

```
tv <- ( Template copy );
tv add-attribute name:`"channel" object:`"NHK";
tv add-attribute name:`"power" object:`"OFF";
tv add-attribute name:`"volume" object:`"0";
```

とすると，channel , power , volume という属性を持った tv というオブジェクトができる．属性はそれぞれ，NHK , OFF , 0 という文字列オブジェクトを参照している．また，`"NHK" , `"OFF" , `"0" のように，"` 文字列"とすることによって，文字列オブジェクトを生成できる．

- メソッドの作成と追加

メソッドの追加は，他の属性と同じように属性追加(add-attribute)によって行われる．但し追加されるオブジェクトは，プログラム記述のための文法に従って作成されていなければならない．

```
tv add-attribute name:`"on"
                 object:`{ power <- `"ON"; };
tv add-attribute name:`"off"
                 object:`{ power <- `"OFF"; };
tv add-attribute name:`"copy" object:`{
    newobj <- self send-message
                 (i-object find-attribute `"copy");
    newobj.channel <- channel;
    newobj.power <- power;
    newobj.volume <- volume;
};
```

こうすることで，オブジェクト tv に，on , off , copy というメソッドが追加される．ここで，on は，属性の power を ON にするメソッド，off は，属性の power を OFF にするメソッド，copy は，このオブジェクトの属性を含めてコピーする（デフォルトの copy では，浅いコピーしかしない）メソッドである．

このオブジェクトは，以下のようなプログラムで利用することができる．

```
tv on;
tv2 <- tv copy;
tv2 off;
```

- オブジェクトの継承

オブジェクトの継承は，メソッド inherit を使う．例えば，power の ON , OFF ができる電化製品オブジェクト elec を作り，前述の tv を，elec を継承したオブジェクトとしたい時，

```
elec <- ( Template copy );
elec add-attribute name:`"power" object:`"OFF";
elec add-attribute name:`"on"
                 object:`{ power <- `"ON"; };
elec add-attribute name:`"off"
                 object:`{ power <- `"OFF"; };
tv <- ( Template copy );
tv inherit name:`"i-elec" object:elec;
tv add-attribute name:`"channel" object:`"NHK";
tv add-attribute name:`"volume" object:`"0";
```

とする．

3．ベースオブジェクト

ベースオブジェクトは，本言語の基本的な機能を持つオブジェクト群である．ベースオブジェクトには，次のものがある．

- 参照関係の出発点である root
- オブジェクトの対する基本的な操作メソッドを持つ Object
- 文字列，整数に対する基本的な操作メソッドを持つ String
- 2 値論理を表す True, False
- オブジェクトの雛型である Template
- 仮想オブジェクト NULL
- GUI 構築支援のウィジェットオブジェクト群
- Database I/F であるデータベースオブジェクト

3.1 参照関係の出発点（root）

ベースオブジェクト root は，参照関係の出発点であり，本言語内の全てのオブジェクトはこのオブジェクトから到達可能である．よって，root が参照しているオブジェクトは，グローバルオブジェクトとして，本言語内の全てのオブジェクトのプログラムコードから参照可能である．また，ベースオブジェクトは，最初に実行されるプログラムコードを"program"という名前のメソッド属性として持っている．このメソッドには，プログラム実行前にしなければならない，オブジェクトの生成，初期化などを行うプログラムコードが記述される．おな，root は属性の追加が可能である．

3.2 オブジェクトに対する基本操作 (Object)

ベースオブジェクト Object は，本言語のオブジェクトが持つ基本的な機能，属性の検索，追加と変

更,オブジェクトの継承などの機能をそのメソッド属性として持つ.オブジェクトはこのObjectを継承することで,以下の基本的な機能を持つことができるようになる.なお,Objectは変更不可能である.

- 属性の検索 (find-attribute)
- 属性の追加,変更,削除 (add-attribute)
- オブジェクトの複製 (copy)
- オブジェクト同士の比較 (=, ==)
- メッセージの送信 (send-message)

3.3 文字列に対する操作 (String)

ベースオブジェクトStringは,文字列を値として持つオブジェクトである.Stringは,この文字列に対するリスト処理,文字列処理,数式処理,実行という機能を持っている.これら以外に,Stringが持つメソッドを挙げる.

- 実行 (eval, while)
  メソッドevalは,オブジェクトが持つ文字列を本言語のプログラムとして実行する.
  メソッドwhileは,タグdo:で実行する内容を指定し,反復実行する.
- 文字列の比較 (scmp)
- 文字列の表示 (display)

ベースオブジェクト String は,変更不可である.

3.4 GUI構築支援について

本言語では,グラフィックスインタフェースの構築を支援するための機能を持ったオブジェクト群をベースオブジェクトとして持っている.このオブジェクト群はウィジェットオブジェクトセットと呼び,ベースオブジェクトと同列に存在する.

本言語では次の実現を目標としている.

- オブジェクトの状態の表示 (ブラウズ機能)
- 専用エディタによるプログラムの作成
- 作成したプログラムの実行開始指示
- 本言語を使用したアプリケーション開発

ウィジェットオブジェクトセットを構成しているウィジェットオブジェクトは,GUIを構築する部品であり,その雛型である.ウィジェットオブジェクトの種類には,Frame, Lable, Button, Editor, MenuBar, PulldownMenu, Dialog, Canvasがある.

実際に言語上でウィジェットを扱う場合は,ウィジェットオブジェクトに対してウィジェットの生成を要求しウィジェットを得てから,この得られたウィジェットに対してメッセージを送信することにより,ウィジェットのリソースに対する変更などのカスタマイズを行いウィジェットを作成する.

ウィジェットオブジェクトは,Motifツールキットのウィジェットセットを用いて実現している[5].

3.5 Database I/F の機能

本言語では,データベース[6]を対象とした操作のために,以下のインタフェースの機能を持つ.

- オブジェクトの格納
- オブジェクトの取り出し
- オブジェクトの削除
- オブジェクトの保護(セキュリティ機能)
- キーワード,コメント保存のサポートによる再利用の向上

オブジェクトをデータベースに格納する際にキーワード及びコメントを一緒に格納することにより,オブジェクト再利用時の検索が可能となる.

本研究ではリレーショナルデータベースSybaseを使用した.

3.6 その他のオブジェクト

■ 真偽値オブジェクト (True, False)

ベースオブジェクト True, False は真偽値を表すオブジェクトであり,真偽値に関する機能,実行の選択,論理(真偽値)演算をメソッド属性として持つ.

- 実行の選択 (if)
- 論理(真偽値)演算 (and , or , not)

ベースオブジェクト True, False は,本言語内で唯一であり変更は不可である.

■ オブジェクトの雛型 (Template)

ベースオブジェクト Template をコピーすることによって新しい空オブジェクトを生成できる.コピーされたオブジェクトは,継承先を示すオブジェクト super-object を属性として参照していて,ベースオブジェクト Object を,"i-Object"というオブジェクトで継承している.

Template は,変更不可能である.

■ ヌルオブジェクト (NULL)

オブジェクトが存在していないことを表す仮想オブジェクトである.

4. 実装

本言語実装の際に用いたオブジェクトや属性の管理の方法について述べる.なお,実装は,C言語を用いて行なった.

4.1 オブジェクトの管理

オブジェクトは,オブジェクトIDによって管理

される．このオブジェクトIDは，本言語内でユニークな値を持つ整数値である．各オブジェクトIDは，それに対応する構造体を持ち，その構造体の配列をオブジェクトテーブルとした．

オブジェクトテーブルは，参照数，データベース上でのオブジェクトID，属性テーブルを持つ．

オブジェクト管理の概念図を図3に示す．

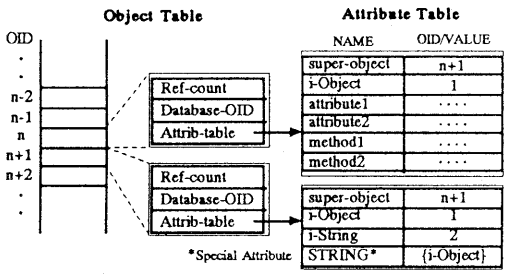

図3．オブジェクト管理概念

■ 参照数

参照数は，そのオブジェクトが他のオブジェクトの属性としていくつのオブジェクトから参照されているかを示す．参照数は，そのオブジェクトが他のオブジェクトから参照されると1増加し，その参照先が変わるか，参照しているオブジェクトが削除されると1減少する．この時参照数が0になれば，そのオブジェクトはシステムによって削除される．

■ 属性テーブル

属性テーブルは，基本的に，属性名とオブジェクトIDの組が要素となっている．本来，オブジェクトの属性となり得るのはオブジェクトだけであるが，実際の本言語の実装において文字列や何らかの特別なデータ，そして，C言語で書かれた組み込みのメソッドを属性として扱う必要がある．そこで，オブジェクトの持ち得る属性に特別属性，ベースメソッドという概念を付け加えた．

- 特別属性

  特別属性は，属性値のメモリブロックポインタと，そのメモリブロックのコピー/解放のC言語の関数から成り立っている．

  この属性は，本言語のユーザが直接扱うことはできず，以下に述べるベースメソッドを通して間接的に扱われる．文字列もこの特別属性として実装されている．

- ベースメソッド

  ベースメソッドとは，特別属性の生成やその操作を記述するメソッドである．ベースメソッドは，C言語の関数として記述されシステムに登録することによって，オブジェクトとして他のオブジェクトから属性として参照することができる．但し，ベースメソッドは完全なオブジェクトではないので，通常のオブジェクトの持っているメソッド (Objectや Stringのメソッド) は evalを除いて使うことができない．

4.2 メッセージ送信

本言語におけるメッセージ送信手順は，メッセージ送信先オブジェクトのメソッド属性に実行に必要なパラメータをセットし，そのメソッド属性にevalメッセージを送ることと定義されている．

1. 送信先オブジェクトのメッセージ名と同名の属性名で参照されるメソッドオブジェクトを特定する．
2. そのメソッドオブジェクトに local という属性を作る．
3. もし，そのメソッドオブジェクトに local というオブジェクトがすでに存在していたなら，その localを prev という属性名で local に追加する．
4. local に self という属性名で送信先オブジェクトを追加する．
5. local に，タグ名という属性名でパラメータオブジェクトを付け加える．
6. この時，パラメータが一つでタグ名が省略された時，属性名は arg にする．
7. メソッドオブジェクトにメッセージ eval を送る．

しかし，この手順を直接実行するのは繁雑であるので，その作業は全てメソッド eval の中に隠蔽されている．つまりユーザは，eval の定める文法のメッセージ書式に従えば，自動的にメッセージ送信が行なわれる仕組みになっている．もちろん全ての手順をユーザが直接実行することができる．

4.3 メソッドの評価

メソッドは，メッセージ eval が送られることによってメソッドオブジェクトのメソッド属性 eval が起動され，その eval によって評価される．

eval は，メソッドの内容を文字列リストとして文字列に切り分け，それぞれを，文法に従って，オブジェクト，文字列オブジェクト，メッセージ文字列，タグ文字列に変換する．文字列をオブジェクトに変換する時，eval は，そのオブジェクトをローカルオブジェクト，カレントオブジェクト，root オブジェクトの順で探索する．それらが，メッセージ送信単位になり次第，eval は前述の手順でメッセ

ージ送信を行なう.

eval は，一番最後のメッセージ送信の戻り値をメソッド評価の戻り値とする.

5．言語比較

本言語と一般的なオブジェクト指向言語である Smalltalk [7], C++, Eiffel [8], CLOS の比較[4]を表1に示す.

5.1 オブジェクト生成

ここで挙げた他の言語は，クラス-インスタンスベースであるので，まずクラスを定義し、そこからインスタンス生成する必要がある．それに対して，本言語では，空のオブジェクトを Template をコピーすることで用意しそれに直接属性を追加していくという方法を取っている.

5.2 メソッド定義

本言語では，メソッドの定義は属性の定義と同じように一つ一つオブジェクトに登録するという方法を取っている．これは，C++やEiffelのようにクラスの中にメソッドを定義するわけではない．また，CLOS のようにあるクラス(型)に対するメソッドを定義していく方法でもなく，クラスに対してメソッドを定義する Smalltalk に近いと言える。

5.3 比較結果

本言語は，コピーベースの概念を採り入れ，オブジェクト構築用のメソッドを Object に持たせることによって，型による決められた役割のあるオブジェクトを扱う従来のクラス-インスタンスベースの言語と比べて，より柔軟なプログラミングができると言える.

6．終りに

モデルベースソフトウェア開発基盤の記述言語を実現するためにオブジェクト指向言語 Bramble を開発した．本言語は，コピーベースのオブジェクト指向の概念を導入したインタプリタ言語であるため，モデルの動的な記述を容易にすることができる．しかし，型付けの概念がないことや，制約記述が書きにくいことなど言語やモデルとしての検証性に欠ける面があり，この点の改善が今後の主な課題となるだろう．また，エディタ，デバッガなどのプログラミング支援ツールを持たせ，開発効率の良いモデル記述環境を構築することも挙げられる.

謝辞

本言語の開発にあたり，ご協力頂いた本研究室の佐藤達也君，細谷伊知郎君，水越 功君に感謝致します.

参考文献

[1] 上田賀一，安間その美，高橋大輔："メタ階層に基づくモデルベースソフトウェア開発基盤の提案"，ソフトウェア工学の基礎 II, pp.11-20，近代科学社 (1996)

[2] 安間その美，高橋大輔，上田賀一："システムプロトタイピング支援のための基盤機構の開発"，情報処理学会ワークショップ論文集，Vol.95, No.1, pp.49-56 (1995)

[3] 所真理雄，松岡聡，垂水浩幸："オブジェクト指向コンピューティング"，岩波書店 (1993)

[4] 本位田真一，山城明宏："オブジェクト指向システム開発"，日経 BP 社 (1993)

[5] Eric F.Johnson, Kevin Reichard (中西隆 監訳):"Motif 1.2 パワープログラミング"，技術評論社 (1995)

[6] 増永良文："リレーショナルデータベース入門"，サイエンス社 (1991)

[7] Bertrand Meyer (二木厚吉 監訳)："オブジェクト指向入門"，アスキー (1990)

[8] Pinson, R.S.Wiener (富士ゼロックス情報システム訳)："Smalltalk：オブジェクト指向プログラミング"，トッパン (1993)

表1．Bramble と他言語の比較

比較内容	Bramble	Smalltalk	CLOS	Eiffel	C++
class-base / copy-base	copy-base	class-base	class-base	class-base	class-base
型付け	無	弱	弱	強	強
Interpreter / Compiler	Interpreter	Interpreter	Interpreter	Compiler	Compiler
オブジェクトの継承	動的	静的	動的	静的	静的
多重継承のサポート	有	無	有	有	有
GUI 構築支援機能の提供	有	有	無	無	無
Database I/F の機能の提供	有	無	無	無	無