ソフトウェア工学 119－2
（1998．5．15）

オブジェクト指向代数仕様におけるモジュール再利用
- 経験レポート -

中島 震

NEC C&C メディア研究所

あらまし　代数仕様言語 CafeOBJ ではモジュールをビルディングブロックとして仕様を作成する。実用規模の問題に対しては、モジュールの抽出、モジュール間の関係設定、モジュールの抽象レベル設定、等を適切に行なう必要があり、そのためのガイドラインあるいはモデリング手法が必要である。CafeOBJ では代数モデルによる性質規範スタイルで仕様を作成することから、モデリングの問題と規範モデルの問題と呼ぶ2つの側面が重要であり、「疑似言語」を明確に意識することで、再利用性に優れたモジュール抽出が可能になる。酒屋在庫問題についての仕様作成経験をもとに具体的に考察する。

Module Reuse in Object-Oriented Algebraic Specifications
- An Experience Report -

Shin NAKAJIMA

C&C Media Research Laboratories
NEC Corporation

Abstract　"Module" is a building block of specifications written in an algebraic specification language CafeOBJ. Since constructing specifications of non-tirivial size is hard, it requires some guideline or modeling method for identifying entities to be implemented as modules, for establishing relationships between modules, and for determining abstraction level that modules provide. Defining some form of "pseude language" is a promising approach for assisiting the process, which is in accordance with the idea of using "property-oriented style" to write specifications in CafeOBJ. We show some results from our experience in constructing CafeOBJ specifications of Sake Warehouse Problem.

1 はじめに

代数仕様言語は、ソフトウエアシステムの構造、機能、などを抽象データ型のモデルである代数モデルに基づいて記述するための形式仕様言語である [5]。特に、OBJ [3] や CafeOBJ [7] ならびに Maude[2][9] といった言語は明確な操作的意味を持つため、ソフトウエア開発上流工程でのラピッドプロトタイピングツールとして利用することができる。理論的な研究の進展と共に、実用規模の仕様を作成することが可能な仕様構築環境の開発が行なわれている [6][18]。

OBJ や CafeOBJ の仕様の基本要素はモジュールである。すなわち、仕様はモジュールの組み合わせからなり、モジュールは仕様再利用の単位である。仕様として記述する問題が小規模な場合や既に「抽象データ」として定義されている場合は、モジュールの中身の記述に専念すれば良い。また、言語要素としてはパラメータ付きモジュールが導入され、汎用のモジュールやパラメータ化したモジュールから、ビルディングブロック的に大規模な仕様を構築する手法が提案されている [4][8]。しかし、実用規模の問題に対しては、モジュールの抽出、モジュール間の関係設定、モジュールの抽象レベル設定、等の課題がある。ソフトウエア工学の立場からは、これらの視点を含めて代数仕様のモジュール再利用を考えるべきである。

本稿では、代数仕様言語 CafeOBJ を用いた仕様作成の経験を通して、仕様の単位であるモジュールの再利用の問題について考察する。以下、CafeOBJ のモジュールの紹介とモジュール再利用の際に考察すべき視点について述べる。次に、シナリオ中心オブジェクト指向モデリング技法を用いてモジュールを抽出し作成した仕様 [11][12] の概略を報告する。最後に、半定量的および定性的な観点から、作成した CafeOBJ 仕様に関するモジュール再利用について考察する。

2 CafeOBJ による仕様

CafeOBJ は、順序ソート代数、並行書き換え論理、隠蔽代数を含むマルチパラダイムの代数仕様言語である [6][7]。モジュールが仕様作成の最小単位となる。

2.1 CafeOBJ のモジュール

CafeOBJ を含む代数仕様言語では、「性質規範スタイル (property-oriented style)」 [16] で仕様を作成する [1]。性質規範スタイルでは、記述対象のソフトウエアシステムが満たす性質や制約条件を記述することで間接的にシステムの振る舞いを規定する。特に、代数仕様言語では、代数モデルを通して間接的に振る舞いを表現する。CafeOBJ では代数モデルを表す手段として「モジュール」を用意している。以下、簡単な例により CafeOBJ のモジュールを紹介する。

モジュールLIST はパラメータ付きで、ジェネリックなリストの振る舞いを規定する。__ はリストの構成子で、|_| は長さを計算する関数、hd とtl は標準的なアクセス関数である。これらの関数の定義は等式により表現する。また、NAT-LIST に示すように、パラメータを実体化することで具体的なデータタイプについてのリストを作成することができる。

```
mod! LIST[X :: TRIV] {
  [ NeList List , Elt < NeList < List ]
  protecting (NAT)
  signature {
    op nil : -> List
    op __  : Elt List -> NeList {id: nil}
    op |_| : List -> Nat
    op hd  : NeList -> Elt
    op tl  : List   -> List
  }
  axioms {
    var X : Elt     var L : List

    eq | nil | = 0 .
    eq | X L | = 1 + | L | .
    eq hd(X L) = X .
    eq tl(X L) = L .
  }
}

mod! NAT-LIST {
  protecting (LIST[NAT]
              *{sort List -> NatList})
}
```

操作的な意味は、等号関係による左辺から右辺

[1] これに対して Z 記法はモデル規範スタイル (model-oriented style) と呼ばれる。

への置換過程として与えられる。たとえば、

```
|(1 2)| => 1 + |(2)|
        => 1 + 1 + | nil |
        => 1 + 1 + 0
        =(*)=> 2
```

のような置換過程で唯一の答えを得る。この過程は、合流性と停止性という仕様実行としての良い性質を持つ。

CafeOBJ では、状態変化を明示的に表現する並行書き換え論理も用いることができる。並行書き換え論理は Maude により、「並行オブジェクト」のモデルとして提案されたものである [9]。以下の例は、Maude の並行オブジェクトモデルにしたがった仕様を CafeOBJ で表現した例である。輸入しているジュール ROOT が並行オブジェクトを模倣する機能を提供している。

```
mod! BUFFER {
  [ BuffTerm CIdBuff ]
  extending (ROOT)
  [ BUffTerm < ObjectTerm ]
  [ CIdBuff  < CId ]
  protecting (NAT-LIST)
  signature {
    op <_:_|_> :
      OId CIdBuff Attributes -> BuffTerm
    op Buff : -> CIdBuff
    op new-buff : OId -> BuffTerm
    op put  : OId Nat -> Message
    op get  : OId OId -> Message
  }
  axioms {
  vars O R : OId  var N : Nat
  var L : NatList

  eq new-buff(O) = <(O : Buff)|(b = nil)> .
  ctrans put(O,N) <(O : Buff)|(b = L)>
  => <(O : Buff)|(b = (N L))> if |L| < 10 .
  ctrans get(O,R) <(O : Buff)|(b = L)>
  => <(O : Buff)|(b = tl(L))> return(R,hd(L))
  if |L| > 0 .
  }
}
```

モジュール BUFFER はクラス Buff を定義する。Buff オブジェクトは、

```
<('b1 : Buff)|(b = (1 2 3))>
```

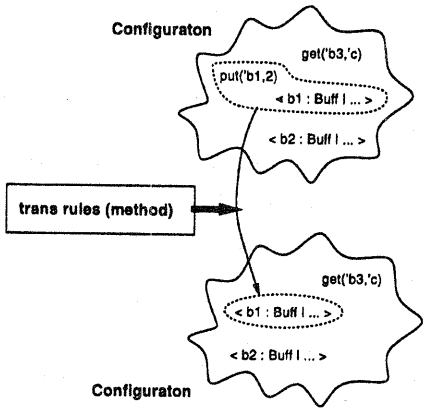

図 1: 並行オブジェクトの実行

のような項により表現する。この例は、(インスタンス) オブジェクトが固有に持つオブジェクト識別子が'b1 で、バッファ本体を示す属性 b の値がNatList のデータ (1 2 3) であることを示す。クラス Buff はオブジェクト生成メソッド new-buff の他に 2 つのメソッドを持ち、その振る舞いを ctrans で示された (条件付き) 書き換え規則で与える。例えば、put の振る舞いは、次のように表現されている。

> 長さが 10 以下であれば引数のデータ
> を属性b の値であるリストに追加する。

この書き換え規則が「発火」して実行されると、その結果として、Buff オブジェクトの新しい項 (=> の右辺の項) を生成する。図 1に、この状況を摸式的に説明する図を示した。

2.2 モジュールの再利用

モジュールが仕様の最小単位 (ビルディングブロック) であること、ならびに、モジュールは性質規範スタイルで記述されること、の 2 つの点から、CafeOBJ モジュールの再利用性に関連して考察すべき点を議論する。

2.2.1 モデリングの問題

仕様として記述したい問題が小規模な場合や既に「抽象データ」として定義されている場合は、先の例のように単独のモジュールで表現す

ることができる。しかし、実用規模の問題に対しては、モジュールの切り出し、とモジュール間の関係が問題となるであろう。この2つの点は互いに強く関連する。モジュールのための「モデリング」技法が必要となる。

代数仕様を作成する場合においても、通常のソフトウエア開発過程と同様に、問題分析、問題領域の実体と関連の抽出、等の「上流」工程が必要である。さらに、作成するモジュールが問題領域とのトレーサビリティが良くてコンパクトに表現できると、一種の「ドメイン知識」として再利用することができる。すなわち、

> 問題領域の実体や関連とのトレーサビリティが良いモジュールの抽出方法

が必要となる。本稿では「モデリングの問題」と呼ぶ。

2.2.2 規範モデルの問題

第2に、CafeOBJによる仕様が性質規範スタイルによる点がある。モデル規範スタイルの形式仕様言語では、言語の側で明確な「モデル」の基準を持つ。例えば、Z記法では公理的集合(Zermelo集合)がモデルであり、公理的集合が提供する概念と直接対応する言語要素(集合、関係、等)を用いて仕様を記述する[14]。一方、性質規範スタイルによる仕様記述では、仕様作成者が記述対象の性質や制約条件を設定する必要がある。仮想計算機や疑似言語のような層を設定する方法が有力である。他モジュールに対する「モデル」の役割を果たすモジュールを、性質規範スタイルで表現した代数モデルとして、作成することに相当する。

この場合、「モデル」のモジュールとそのモジュールを利用して問題レベルの記述を行なうモジュールの間で役割を明確に分離すると見通しが良くなる。同時に、「モデル」モジュールの再利用性を高めることができる。しかし、どのような仮想計算機あるいは疑似言語を設定すべきかは、作成する仕様をどのように使いたいかに依存し、天下り的に決めるわけにはいかない。実際、先のモジュール LIST は、直観的な意味で実行可能となるようなリストを提供することが目的であった。すなわち、

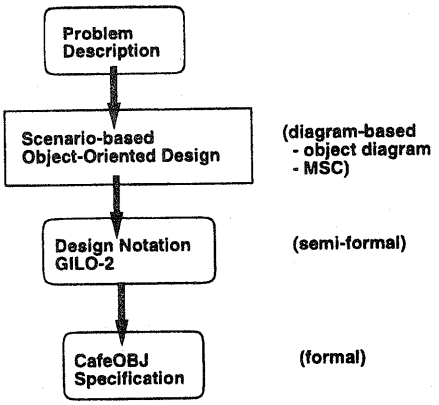

図 2: 仕様作成の過程

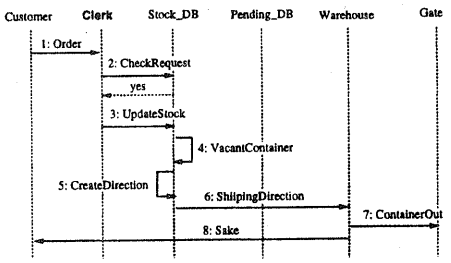

図 3: シナリオの例

> 再利用性の高い規範モデルを持つモジュールの定義方法

が必要となる。本稿では「規範モデルの問題」と呼ぶ。

3 オブジェクト指向代数仕様

モジュール再利用の2つの問題、モデリングの問題と規範モデルの問題について具体的に考察する。従来より設計手法比較の共通問題として用いられてきた「酒屋在庫問題」[15][17]について作成したCafeOBJの仕様[11][12]を例として用いる[2]。

3.1 モデリングとGILO

[2]オブジェクト指向モデリング技法により得た代数仕様というくらいの意味で「オブジェクト指向代数仕様」と称している。

図2に、CafeOBJ仕様作成の過程を示す。与えられた問題記述から、シナリオ中心のオブジェクト指向モデリング技法[1]を用いて、問題領域の実体や関連を抽出する。オブジェクト図やメッセージ系列図(図3)といったダイアグラムを援用して作業を行ない、その結果を、GILOと呼ぶ中間的な設計記法で表現する[10]。CafeOBJの仕様を作成するということは、GILOの記述を、CafeOBJのモジュールに変換することである。GILOの記述が問題領域の実体や関連を反映したものであるため、得たCafeOBJモジュールはトレーサビリティが良いものとなる。すなわち、「モデリングの問題」へのアプローチとなる。

GILOは、シナリオ中心オブジェクト指向モデリングで必須の要素をコンパクトに記述できる表記法を提供する。特に、問題領域の実体を表現するクラス(図3のStock_DB等)、クラスに跨る処理シナリオの流れを明示するコラボレーション、さらに、クラスやコラボレーションの振る舞いを記述するために必要な共通語彙(Container、Stock、等)の3つの表記法がある。ここで、クラスの振る舞いはメソッドの集まりとして表現され、各々のメソッドは事前条件と事後条件の組で表すことができる。コラボレーションはメッセージ系列図を抽象化したものでラベル付き遷移システムで表現する。最後に、共通語彙は等式でモデル化される抽象データである。

この方法では、「規範モデル」に相当するモジュールは、さらに2つに分類することができる。クラスとコラボレーションの計算モデルを表現するために導入する「疑似言語の規範モデル」と共通語彙として抽出した「ドメインの規範モデル」である。共通語彙は、クラスならびにコラボレーションのドメインに関する振る舞いの説明を与えるモジュールである。クラスならびにコラボレーションは、シナリオ中心のオブジェクト指向モデリングという考え方と密接に結び付いてるが、共通語彙はモデリング手法と独立にドメインの語彙として抽出する。すなわち、同じドメインの他の問題や他のシナリオでも利用することを意図しているからである。

以下、作成した2種類の仕様に即して「疑似言語の規範モデルの問題」を議論する。

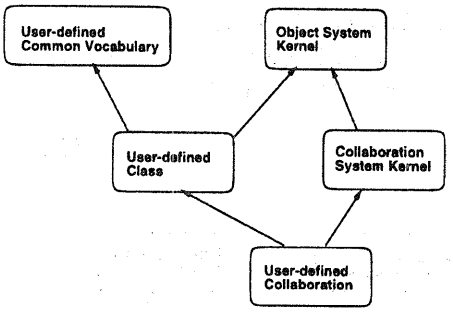

図4: モジュール構成の概要

3.2 CafeOBJ仕様(1)

仕様(1)は、CafeOBJのフルセットの言語仕様を用いたもので、クラスならびにコラボレーションをMaudeの並行オブジェクトでモデル化した。モジュール構成の概要を図4に示す。ここで、Object System Kernelは、図1の並行オブジェクトのメッセージ実行を実現するために必要な規範モデルを提供する[12]。BUFFERが輸入していたモジュールROOTが相当する。

メソッドの一般形は図5の形をしており、複数のメッセージ項(M_i)、複数のオブジェクト項($< O_m : C_m | attr_m >$)が1つのメソッドに関与することができる。図1のConfigurationと呼ぶデータ構造はシステムのスナップショットを表現するもので、ある時点でシステムに存在するメッセージ項とオブジェクト項を管理している。CafeOBJエンジンは、Configurationが持つ項を網羅的に探索し、メソッドとして定義されている書き換え規則の中に実行可能な規則があるか否かを決定する。これはConfigurationに対するAC (Associative-Commutative) マッチング、すなわち、すべての可能なメッセージ項とオブジェクト項の組み合わせを試みることにより行なう。例えば、図1の場合には、2つのメッセージ項と2つのオブジェクト項の間で、モジュールBUFFERが定義する2つの書き換え規則を対象として、実行可能な組み合わせを求める。この例はputメソッドだけが実行される場合に相当する。

次に、Maudeスタイルの並行オブジェクトの上に、コラボレーションの操作的な意味を提供する「疑似言語の規範モデル」のモジュール

$$M_1 \ldots M_n < O_1 : C_1 | \mathrm{attr}_1 > \ldots < O_m : C_m | \mathrm{attr}_m >$$
$$\longrightarrow \quad < O_{i1} : C'_{i1} | \mathrm{attr}'_{i1} > \ldots < O_{ik} : C'_{ik} | \mathrm{attr}'_{ik} > \quad < Q_1 : D_1 | \mathrm{attr}''_1 > \ldots < Q_p : D_p | \mathrm{attr}''_p >$$
$$M'_1 \ldots M'_q \quad \text{if } C$$

図 5: メソッドの一般形

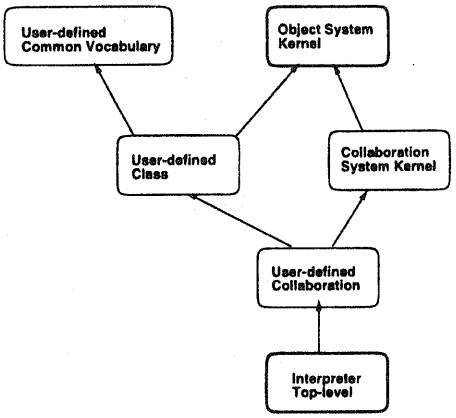

図 6: モジュール構成の概要

を導入する。図 4の Collaboration System Kernal は、並行オブジェクトの考え方でコラボレーションの実体である遷移システムをクラス Machine として実現した。クラス Machine のメソッドを定義することが、問題レベルのコラボレーションの記述となる。

3.3 CafeOBJ 仕様 (2)

仕様 (2) は、AC マッチングならびに trans 規則を用いない版である[3]。仕様 (1) として作成したモジュールをもとに、GILO の記述に直接対応するモジュールをなるべく変更しないで実行機構に相当するモジュールのみを書き直す方針をとった。ただし、前節のような一般的な並行オブジェクトのインタプリタを作成することが難しいため、今回の酒屋在庫問題を実現するために必要な最小限度の機能を作成することとした。具体的には、遷移規則の一般形を制限して高々 2 つのメッセージを受け付けるようにしたこと、同時に複数個のメッセージがひとつのオブジェクトに送られないようにメソッドの書き方を制限したこと、メッセージ項ごとに役割を決めたこと (通常のメッセージ、返却値と同期して実行可能となるメッセージ、返却値の役割を果たすメッセージ)、等の制限を設けた。

仕様 (2) の場合のモジュール構成を図 6に示す。Interpreter Top-level が主な実行制御を司る。合わせて Object System Kernel についても変更を加えた。`Configuration` データ上の AC マッチングにより行なっていた網羅的な探索動作を模倣することである。具体的には、以下のような処理を行なう。

1. `Configuration` に相当するデータを、メッセージ項ならびにオブジェクト項を格納する線形リストで表現する。
2. 線形リストを探索して組になるメッセージ項とオブジェクト項を求める。
3. 実行可能状態の組を実行し、実行結果を線形リストの末尾に追加する。

上記ステップ 2 で、メッセージ項からのレシーバ読みだし、役割に応じた組み合わせ可能性の決定、等の「メタレベル」操作を行なう。

仕様 (2) の場合でも「疑似言語の規範モデル」は、Maude スタイルの並行オブジェクトをシミュレートすることであるが、AC マッチングという並行性を手軽に模倣する機構が使えないために、仕様 (1) の場合よりも「手続き的な」規範モデルとなった。

4 考察

仕様 (1) ならびに仕様 (2) の記述結果の要約を 3 つの表に示した。表 1は、2 つの仕様の規模を示すもので、仕様 (2) の方が規模が大きいことがわかる。また、表 2は、モジュールの役割ごと

[3] trans 規則は全て eq 等式に置き換えた。等式を常に左辺から右辺への書き換えを解釈することでメソッド実現に必要な計算機構を近似的に模倣する。

項目		仕様 (1)	仕様 (2)
モジュール数		47	60
等式の数		89	263
内訳	eq の数	61	263
	trans の数	28	–

表 1: 規模の比較

分類	仕様 (1)	仕様 (2)	GILO
基本機構	13	20	0
共通ライブラリ	4	4	1
ドメイン語彙	11	11	11
クラス	15	15	4
コラボレーション	4	5	2
実行制御	–	5	0
合計	47	60	18

表 2: モジュールの詳細分類

に分類した表で、各々の役割を持つモジュール数を示す。また、各分類ごとに、GILO の記述と直接対応するモジュールの数を示した。最終的に得た CafeOBJ モジュールと問題領域の実体とのトレーサビリィの目安となる。ここで、LIST 等の汎用機能からなる共通ライブラリとドメイン語彙が GILO の共通語彙に相当する。

規模については、仕様 (2) の基本機構 (Object System Kernel) ならびにインタプリタ実行制御 (Interpreter Top-level) のモジュール数が多いことがわかる。その他の分類項目については、モジュール数は同じである。これは、インタプリタ機能のみを「すり替える」という方針を採用したことの現れである。一方、GILO との対応が良いモジュール数は仕様 (1) の場合で約 40% であった。今回の仕様では、ひとつの GILO クラスを 4 つの CafeOBJ モジュールとして実現した。4 つを一塊と見做すと、対応の率は約 60% に向上する。当然のことながら、共通語彙の役割を果たすドメイン語彙は 100% のトレーサビリティがある。「モデリングの問題」に良い効果をあげたことといえる。

総数	新規数	修正数	流用数
60	12	14	34

表 3: 仕様 (2) における再利用

また、他に、ODP トレーダの情報ビューの実行可能仕様を構築した例がある [13]。ここでは、約 40% が標準勧告記載の概念とのトレーサビリティが良いモジュールであり、その他は、実行可能性を導入するために必要であった。以上より、40% から 60% というの値が平均的な数字と考えられる。

表 3は、仕様 (2) のモジュールの中で、仕様 (1) として作成したモジュールの再利用形態について示したものである。総数 60 の内、全く新規に作成したモジュールが 12、役割 (インタフェース) は同じままで記述を修正したモジュールが 14、そのまま流用できたモジュールが 34 である。その結果、

$$\text{流用率} = (\text{流用数})/(\text{総数}) = 34/60 = 56.6\ \%$$

$$\text{修正率} = (\text{修正数})/(\text{新規} + \text{修正}) = 14/26 = 53.8\ \%$$

である。以上より、インタプリタカーネルの実行機構が全く異なるにもかかわらず、約 60% を流用することができた。また、流用できなかったモジュールについても 50% 強は修正で対応することができた。修正対象モジュールの大部分はクラス定義本体、特にメッセージの役割を天下り的に決めたことから生じた修正であった。並行オブジェクトの実行機構である「疑似言語の規範モデル」モジュールの抽出がうまくいったことを示す。

最後に、リフレクション [2] との関連について考察する。仕様 (2) の規模が大きい理由、特に、等式の数が多い理由は、インタプリタカーネルで、`Configuration` が管理する項がメッセージであるかオブジェクトであるかを識別したり、メッセージが 3 つの分類のいづれかであるかを判定するための情報を与えるからである。操作対象の項の構造 (例えば任意の項の最初の引数) をジェネリックにアクセスすることができないため、同等の情報を得るための等式を明示的に仕様として与えた。通常の記述レベルからは得られないメタな情報に相当する。一方、リフレクションの考え方によって、一段下位 (タワーで

は上) の層から見れば、項に関するメタ情報をデータ化することができる。この機能を用いれば、仕様をよりコンパクトに表現することができ、「疑似言語の規範モデル」モジュールの作成が容易になることが期待できる。文献 [2] では書き換え戦略をユーザレベルで変更することが研究の動機であるが、本稿の場合のように、「疑似言語」の新しいインタプリタを構築する際にも有用な機構である。

5 おわりに

中規模の CafeOBJ 仕様を作成した経験から、代数仕様のモジュールの再利用について考察した。モジュール再利用の視点として、「モデリングの問題」と「規範モデルの問題」を設定した。モジュール抽出を行なう作業を支援する中間的な設計記法 GILO を用いて仕様作成を行なった経験から、何らかの疑似言語 (本稿の場合は GILO) を設定する「言語パラダイム」によるモジュールの役割の明確化がモジュール再利用の 2 つの問題について効果があることがわかった。

謝辞

この成果は情報処理振興事業協会 (IPA) が実施している「創造的ソフトウエア育成事業」の一環として行なわれたものである。

参考文献

[1] Carroll, J.M. (ed.) : *Scenario-Based Design*, John Wiley & Sons, 1995.

[2] Clavel, M., Eker, S., Lincoln, P., and Meseguer, J. : Principles of Maude, Proc. 1st Workshop RWL (1996).

[3] Futatsugi, K., Goguen, J., Jouannaud, J-P., and Meseguer, J. : Principles of OBJ2, Proc. 12th POPL, pp.52-66 (1985).

[4] Futatsugi, K., Goguen, J., Meseguer, J. and Okada, K. : Parameterized Programming in OBJ2, Proc. 9th ICSE, pp.51-60 (1987).

[5] 二木 厚吉 : 代数モデルの基礎, コンピュータ・ソフトウエア, Vol.13, No.1 pp.4-22 (1996).

[6] Futatsugi, K. and Nakagawa, A.T. : An Overview of CAFE Specification Environment, Proc. 1st IEEE ICFEM (1997).

[7] Futatsugi, K. and Diaconescu, R. : The CafeOBJ Report, AMAST Series, World Scientific (1998).

[8] Goguen, J. : Principles of Parameterized Programming, in *Software Reusability*, pp.159-225, ACM Press 1989.

[9] Meseguer, J. : A Logical Theory of Concurrent Objects and its Realization in the Maude Language, in *Research Directions in Concurrent Object-Oriented Programming (Agha, Wegner and Yonezawa ed.)*, pp.314-390, MIT Press 1993.

[10] 中島 震 : オブジェクトの集団的振舞いの設計, 情処・ソフトウエア工学研究会 95-6 (1993).

[11] 中島 震、二木 厚吉 : オブジェクト指向代数仕様作成のケーススタディ, Proc. FOSE'96, pp.18-25 (December 1996).

[12] Nakajima, S. and Futatsugi, K. : Object-Oriented Modeling Method for Algebraic Specifications in CafeOBJ, Proc. 19th ICSE, pp.34-44 (May 1997).

[13] 中島 震、二木 厚吉 : CafeOBJ による ODP トレーダ仕様の記述, コンピュータ・ソフトウエア, to appear (1998).

[14] Spivey, J. : *The Z Notation (2ed edition)*, Prentice Hall 1992.

[15] Tamai, T. : How Modeling Methods Affect the Process of Architectural Design Decisions: A Comparative Study, Proc. IWSSD-8, pp.125-134 (March 1996).

[16] Wing, J. : A Specifier's Introduction to Formal Methods, IEEE Computer, pp.8-24 (1990).

[17] 山崎 利治 : 共通問題によるプログラム設計技法解説, 情報処理, vol. 25, No. 9, p.934 (1984).

[18] CafeOBJ ホームページ
http://www.ldl.jaist.ac.jp/cafeobj/index.html.en