情報処理学会 オブジェクト指向シンポジウムOOPSLA'97
オブジェクト指向最前線, 朝倉書店, pp.128-135, 1997.7



# 「ドメインモデル≡計算モデル」を志向したアプリケーション開発環境M-baseにおける分析・設計技法

松本光由[†] 小西 裕治[†,*] 中所武司[†]

近年，ワークステーションやパソコンの普及およびそれらをつなぐネットワークの普及と共に，業務の専門家が自ら情報システムを構築する必要性が高まっている．このようなエンドユーザコンピューティングの促進のためには，業務の専門家が自ら業務モデルを構築することが，そのまま情報システムの構築につながるような技術が必要であるという観点から，A domain model ≡ a computation model（業務モデルと計算モデルの一致），および，Analysis ≡ design ≡ programming（分析，設計，プログラミングの一体化）　という基本コンセプトを設定した．我々は，これらのコンセプトを達成するためにオブジェクト指向概念に基づくアプリケーション開発環境M-baseの実現を目指している．その主な機能はアイコン操作を基本としたビジュアルなモデリング＆シミュレーションツールにより実現されている．また，スクリプト言語においては，ネットワーク上のノード割り当て方法と並行処理の可否によってオブジェクトを3種類に分類すると共に，大規模化への対応としてオブジェクトのネスト構造を導入した．オブジェクト間の交信は，メッセージセット単位で行う方式にすることにより，全体のメッセージフローの管理を容易にした．

# Analysis and Design of a Software Development Environment, M-base,Based on "a Domain model ≡ a Computation model"

MITSUYOSHI MATSUMOTO,[†] YUJI KONISHI [†,*]
and TAKESHI CHUSHO [†]

Explosive increase in end-user computing on distributed systems requires that end-users develop application software by themselves. One solution is given as a formula of "a domain model ≡ a computation model." This formula implies that one task in a domain model of co-operative work corresponds to one object in a computation model based on an object-oriented model. Application development environment, M-base[1)~3),6)] , supports this formula for co-operative systems such as groupware and work flow systems. At the first stage, the system behavior at a macro level is expressed by using a modeling and simulation tool for constructing a message-driven model while focusing on message flow. At the second stage, static structure and detailed specifications of objects are expressed in a script language. Communication among objects is performed by a set of messages instead of a message, for implementation of flexible work flow.

## 1. はじめに

近年，ワークステーションやパソコンの普及およびそれらをつなぐネットワークの普及と共に，業務の専門家が自ら情報システムを構築する必要性が高まっている．このような新しい動向に対応して，コンポーネントウェアやビジュアルプログラミングに代表されるような新しい開発技法が普及しはじめている．

業務の専門家が自ら作り，自ら使うようなエンドユーザコンピューティングの促進のためには，プログラミングの概念を排した新しいソフトウェアパラダイムが必要であるという認識から，基本的コンセプトとして，

- A domain model ≡ a computation model
  （業務モデルと計算モデルの一致）
- Analysis ≡ design ≡ programming
  （分析，設計，プログラミングの一体化）

を設定した．

"モデリング＆シミュレーション" によってこれらのコンセプトを実現する分散オブジェクト指向設計技法を確立し，それに基づくアプリケーション開発環境M-base[1)~3),6)] を開発中である．エンドユーザコンピューティングの分野を対象としているため，M-base

† 明治大学理工学部情報科学科
Department of Computer Science,Meiji University
* 現在，日立ソフトウェアエンジニアリング（株）勤務
Presently with Hitachi Software Engineering co. Ltd

による開発では，まず核になる部分を試作し，それを改良しながら実用システムに仕立て挙げるプロトタイピングアプローチをとる．

この開発におけるマクロレベルで利用するツールとして，オブジェクト指向概念に基づく分散協調型問題解決モデルを用いて業務モデルの動的な振舞いを表現できるようなモデリング＆シミュレーションツールを試作した．またミクロレベルで利用するツールとして，スクリプト言語[4]の処理系を試作した．

本報告では，2章で本研究の目的と対象，3章でモデリングプロセスの概要，4章で開発環境の概要，5章でモデリング＆シミュレーション，6章でスクリプト言語の特徴，7章でモデリング＆シミュレーションツールとスクリプト言語処理系の連携について述べる．また，開発事例として会議開催システム[3]を取り上げる．

## 2. 研究の目的と対象

### 2.1 エンドユーザ

一般に，エンドユーザの範囲は広いが，本研究では，ユーザ企業のエンドユーザ部門に所属する基幹業務担当者および一般にオフィスワーカーといわれるような人達を対象とする．特に，後者の人達は，業務の専門家として，DB 検索や表計算などに市販のアプリケーションパッケージを利用しており，今後急速に増加すると思われる．

### 2.2 対象ソフトウエア

本研究では，「すべての日常的な業務をコンピュータ化する」，即ち，「日常的業務はマニュアル化でき，マニュアル化できればコンピュータ化できる」という観点から，オフィスでの業務用アプリケーションを中心にグループウェアやワークフローシステムなども対象とする．規模的には，中，小規模のアプリケーションソフトウエアを想定することになるが，ネットワーク接続するによりシステムとしては大規模化することもある．

### 2.3 開発・保守形態

本研究では，基本的コンセプトとして，

「ドメインモデル≡計算モデル」

「分析≡設計≡プログラミング」

をかかげた．これは，問題領域を分析してドメインモデルを構築した時点でソフトウエアの開発を完了させようというものである．即ち，

「ソフト開発＝モデリング＋シミュレーション」

という図式で表現されるように，問題領域のモデルを作成し，そのモデル上でシミュレーションしてモデルの妥当性を検証した後，実用に際しては，そのモデルをインタプリタにより実行するか，あるいは必要に応じて実際のプログラムを自動生成するという方法である．

この時，特定分野向きのコンポーネントウェアを導入して，プログラミングの複雑さを回避することも重要である．

このように最終的にはエンドユーザが自らの業務のアプリケーションソフトウエアを自ら開発し，自ら利用することを目標とするが，その実現に向けての技術課題は多い．そこで，研究のマイルストーンとして，エンドユーザが主体でシステムエンジニアの助けを借りて開発するが，保守はエンドユーザだけで行うレベルを設定する．

## 3. モデリングプロセスの概要

### 3.1 2階層モデル

今後増加していくと思われるエンドユーザコンピューティングの分野に分散オブジェクト指向設計技法を適用するためには，モデリングの初期の段階で，従来の技法であまり明確に示されていないオブジェクト群の動的な振舞いを記述することが重要である．我々は，次に示すようなマクロモデルとミクロモデルの2階層モデルを導入して，基本的コンセプトを実現した．

(1) マクロモデルは，「ドメインモデル≡計算モデル」を実現するレベルであり，オブジェクト指向の分散協調型モデルとして位置づけられる．

(2) ミクロモデルは，「分析≡設計≡プログラミング」を実現するレベルであり，オブジェクト指向のクラス定義として位置づけられる．

### 3.2 ドメインモデルの構築手順

オブジェクト指向の概念の発生学的定義に基づいてモデリングプロセスの形式化を行う．概要を図1に示す．詳細は文献[1]に詳しい．

分析フェーズで構築されるインスタンスベースのドメインモデルを以下のように OAM (Object-based Analysis Model) として表現する．

$$OAM = \{ O, M, T \}$$
$$O = o[i]$$
$$M = m[i,j,n]$$
$$T = t[r]$$

ドメインモデル OAM は，オブジェクトの集合 O，メッセージの集合 M，メッセージ変換の集合 T の3つ組で規定する．o[i] は，ドメインモデルに含まれる i 番目のオブジェクトである．m[i,j,n] は，i 番目のオブジェクト o[i] から j 番目のオブジェクト o[j] へ送信される n 番目の種類のメッセージである．メッセージ変換の集合 T は，あるオブジェクト o[j] があるメッセージを受信した後，メッセージの列を送信するという関係を

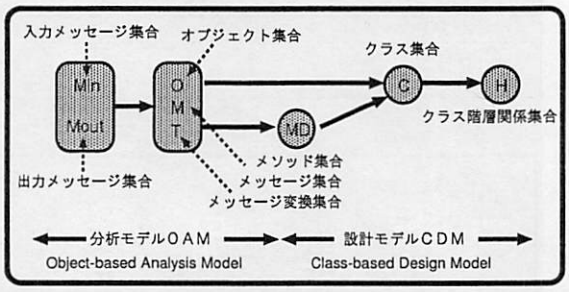


図1 M-baseのモデリングプロセス

t[r] : m[i,j,n] → m[j,$k_1$,$n_1$], m[j,$k_2$,$n_2$], ...
と表現したメッセージ変換の集合である．また，ドメインモデルOAMの外界を仮想的にオブジェクトとみなして，外界からドメインモデルへのメッセージ集合Min，ドメインモデルから外界へのメッセージ集合Moutを決定する．

### 3.3 設計モデルの構築手順

ドメインモデルは，2階層モデルの下位の層に対応する設計モデルに詳細化される．設計モデルはクラスに基づいて構築されるので，以下のようにCDM (Class-based Design Model) として表現する．

CDM = {MD, C, H}

設計モデルCDMは，メソッドの集合MD，クラスの集合C，クラスの階層関係の集合Hの3つ組で規定する．

## 4. 開発環境の概要

**M-base**の開発環境とアプリケーション・アーキテクチャの関係を図2に示す．開発環境は主に次のような構成要素からなる．

- モデリング & シミュレーションツール
- スクリプト言語
- UIビルダ
- コンポーネントビルダ
- 共通プラットフォーム

これらのツールを用いて開発されるアプリケーションは大まかには次の3つの部分から構成される．

- ユーザインタフェース
- モデル
- コンポーネントウェア

モデルはアプリケーションに固有の処理を行う本体である．動的モデル（ドメインモデル）に対応するOAMの部分をモデリング & シミュレーションツールを用いて作成する．静的モデル（設計モデル）に対応するCDMの部分をスクリプト言語を用いて作成する．

ユーザインタフェースはそのアプリケーションのユーザとの対話処理部分であり，モデルと独立させることにより，クライアント／サーバシステムなどのシステム構成，あるいはクライアント端末のマルチプラットフォーム化が容易になる．その構築ツールとしてUIビルダがある．

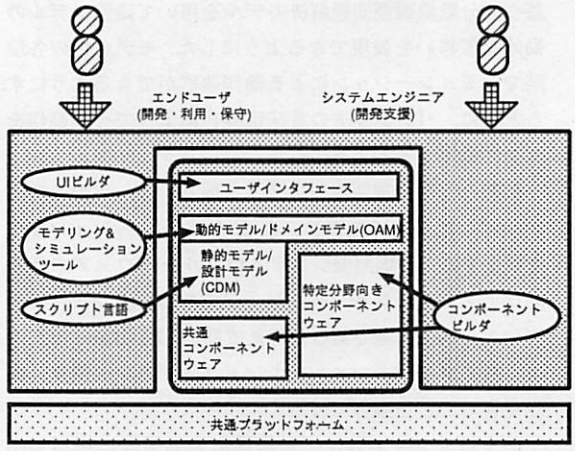


図2 M-baseの開発環境（外側）とアプリケーション・アーキテクチャ（内側）

コンポーネントウェアには分野に共通の基本部品と特定業務分野向きの部品がある．後者が充実してくるとエンドユーザによるアプリケーション構築が容易になる．このようなコンポーネント化を支援するためにコンポーネントビルダを用意する．共通プラットフォームはミドルウェアと基本ソフトウェアの部分で，オープンシステムや部品の流通の観点からは特に分散オブジェクト管理機能などが重要である．

## 5. モデリング & シミュレーション

### 5.1 基本機能

モデリング & シミュレーションツール[6]は，主にマウス操作により動的モデルを構築するツールである（図3）．動的モデルは，オブジェクトとメッセージパッシングで構成される．システムの動的モデルをアイコン表示などで視覚的に判りやすく表し，かつ，オブジェクト指向の概念を素直に表現することを目標としている．

既存のビジュアルプログラミングツールでも，オブジェクト間をリンクで結んでアプリケーションを構築していくものがある．しかし，これらのツールではデータベースからデータを検索し，その結果の表を作成したり，さらにそれをグラフ表示するといった典型的な処理には向いているが，非定型の業務をモデル化の段階から支援してアプリケーションを構築するようにはなっていない．

我々が開発したツールでは，オブジェクト指向概念に

基づく分散協調型問題解決モデルを用いて業務モデルの動的な振舞いを表現できるようにした．モデル化の各段階でシミュレーションによる動作確認ができるようにすると共に，「メソッドの意味定義」によってその動作を詳細に規定できるようにした．

例題として会議開催システムを取り上げ，実際の開発手順にしたがってモデリング & シミュレーション技法を説明する．会議開催システムはグループウェアの代表的なアプリケーションパッケージであるスケジューリングシステムの一種であると考えて良い．**M-base** はこのようなアプリケーションをエンドユーザ自身が開発，保守できることを狙っている．

例えばオフィス等で，事務局に会議開催の指示が出されると，事務局はその会議のために会議室の予約と OHP などの備品の予約を行なうと共に，会議開催通知を出席者に送り，出欠の返事を返してもらう．以上に挙げた業務を自動化するアプリケーションを想定する．

**M-base** におけるモデリングは次の手順で行なわれる．

(1) 外部仕様の作成
  (a) システムの利用者の抽出
  (b) 機能概要の記述
  (c) 入出力メッセージの決定
(2) 動的モデルの作成
  (a) オブジェクトの抽出
  (b) メッセージの抽出
(3) オブジェクト内部の詳細化
  (a) メンバ（属性）の決定
  (b) メソッド（操作）の意味定義の記述
  (c) メソッド内部のスクリプティング

### 5.2 外部仕様の作成

まずはじめにシステムの利用方法を明確にするために，OOSE[7] と同様の一般的な手法をとる．すなわち，システムの利用者（アクタ）を抽出し，各々の外部入力から開始される機能概要（ユースケース）を記述する．そして最後に外部インタフェース，即ち図 1における Min，Mout を決める．

会議開催システムでは，会議開催者，会議参加者が利用者であり，会議の予約，変更，取消の機能が存在し，それぞれの機能概要を記述することになる．また，入出力メッセージについては，会議の予約を例にとると「会議室名と日時を指定する」といったものになる．

### 5.3 動的モデルの作成

このフェーズでは，各シナリオに対するシステムの動的な振舞い，即ち，インスタンスベースのメッセージのやりとりをモデリングしていく．オブジェクトやメッセージの抽出は，以下に挙げる方法によって行なわれる．

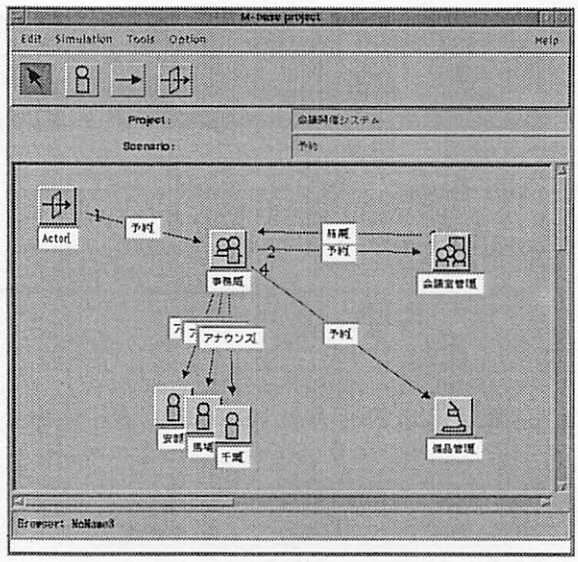


図 3 動的モデル

(1) ひとまとまりの日常的業務が割り当てられる一人または複数の人からなるグループを一つのオブジェクトに対応させる．
(2) 一人又はグループの間で相互の通信手段として用いられている書類，メモ，電話，郵便，口頭連絡などはすべてメッセージとみなす．
(3) 日常的業務における協調作業はメッセージの送受信によって遂行される．

一つの業務を擬人化したオブジェクトに割り当てることにより，メタファベースのモデル化を行う．実世界と同じように一つのオブジェクトに複数の業務を割り当てることも可能であるが，柔軟性と保守性を重視して「1 オブジェクト 1 業務」の割り当てを原則とした．

会議開催システムでは以下の 4 種類のオブジェクトからなる．

(1) 事務局オブジェクト
  会議開催の要求メッセージを受けると，会議室の予約，OHP の予約，その会議のメンバへの会議開催案内などのためのメッセージを各々の担当オブジェクトへ送り，その後，メンバからの出欠通知のメッセージを受け取り，管理する．
(2) 会議室管理オブジェクト
  会議室予約の要求メッセージを受け取り，予約管理する．
(3) 備品管理オブジェクト
  備品予約の要求メッセージを受け取り，予約管理する
(4) 個人オブジェクト

会議開催案内のメッセージを受け取り，出欠通知のメッセージを送る．

この部分で用いるツールは，ドローツール風の視覚的にわかりやすいインタフェースを備えている（図3）．オブジェクトをアイコンにより表示し，メッセージをアイコン間を結んでいる線により表示している．

また，メッセージに付けられている番号はメッセージパッシングされる順番を示している．この順番はメッセージ変換列を表現するために用いる．これに基づき，直列や並列の処理を表現し，スクリプト言語 **Hoop** のメッセージセットを生成する．スクリプト言語については後述する．

**M-base** の対象となるアプリケーションは比較的小規模なものであるため，このような動的モデルに重点を置く．どうしてもモデルが複雑になる場合には，システムエンジニアの支援をあおぐ必要もありうる．

### 5.4 オブジェクト内部の詳細化

基本的には，メンバの決定，メソッドの意味定義の記述，メソッド内部のスクリプティングの3つのステップがある．オブジェクトの詳細を決めていくには図4のウィンドウを用いる．

特にメソッドの意味定義は，マクロモデルからミクロモデルへの橋渡しとして重要である．すなわち，メソッド名だけを用いたシミュレーションによる動作確認には限度があり，メソッド名から内部のスクリプティングに移っていくのには大きなギャップがある．そこで，この問題を解決するために，モデリング時にメソッドの意味定義をできるようにした．

会議開催システムにおける事務局オブジェクトの“予約”メソッドの例を以下に示す．

事務局オブジェクト：会議の予約（日時）
（1） 会議室の予約要求を出す．
- 会議室の予約がとれなかったら，その旨を会議開催者に通知する．

（2） 備品の予約要求を出す．
- 備品の予約がとれなかった場合は，会議主催者にその旨を通知し，会議の取消を行なうか確認する．

（3） 会議案内を出席予定者に送る．

メソッドの意味定義は，次の2つの記述からなる．

基本系列　上の例で，（1）から（3）の各1行目がこれにあたる．基本系列は，メソッドの処理の概要を示すものであり，最初にこれを記述する．

例外処理　上の例では，●印部分がこれにあたり，業務フローにおける例外処理を示す．例外処理は，早い段階で発見でき，気づいた時点で記述しておく．

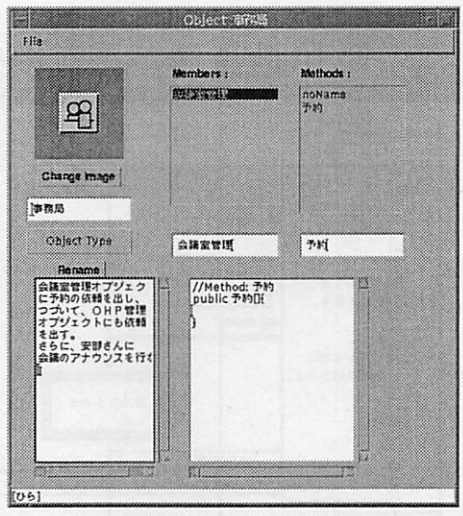


図4　事務局オブジェクトの内部の記述例

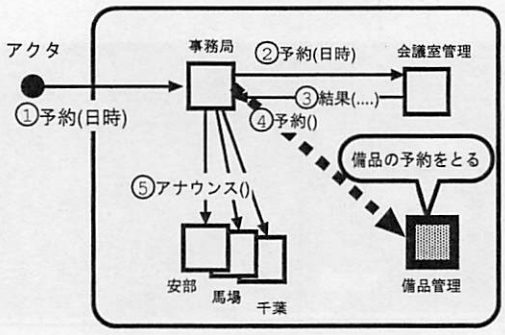


図5　ドメインモデルを用いたシミュレーション

### 5.5 シミュレーション機能

シミュレーションは，作成した業務フローの基本的な動作の確認のために行なう．このシミュレーション時に，メソッドの意味定義をメソッドが起動されるたびに表示することで，視覚的にわかりやすくなる．この表示方法には，次に挙げる2つの方法が考えられる．

（1） ドメインモデルをそのまま用いる方法
（2） 事象トレース図を用いる方法

前者の方法は，ふきだし表示などにより，作成したモデル上で動作の確認が行なえるため，直観的に理解しやすい．図5がその例であり，備品管理オブジェクトの“予約”メソッドを実行中であることを示している．

後者の方法は，システムが動作を開始すると，起動されたメソッドの定義内容が事象トレース図上に表示されていくので，オブジェクト間通信の全体のシーケンスを見渡すことが可能である．図6がその例であり，事務局オブジェクトの会議の予約の処理の中で，会議室管理オブジェクトと，備品管理オブジェクトにメッセージを送り，現在，備品の予約処理を行なっているところである．

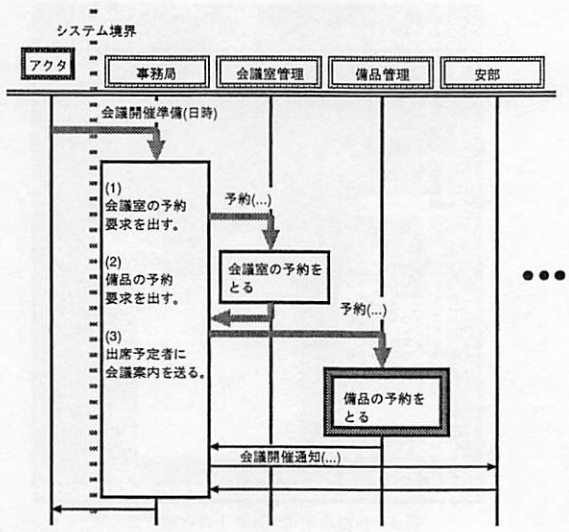


図 6　事象トレース図を用いたシミュレーション

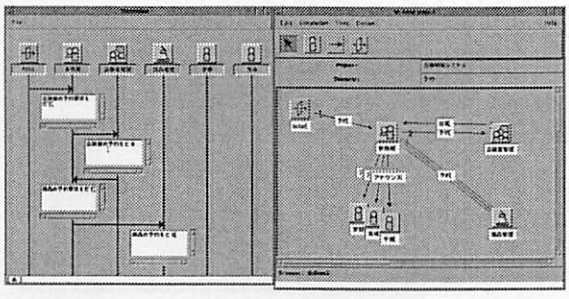

図 7　シミュレーションの画面例

実行済みの業務フローは，太い線で表示されている．

これらの表示方法を併用して使うことにより，メソッドの意味定義の利点を充分発揮できると考えている．この他に，オブジェクトの属性の表示，変更を行なえる機能を実現していく予定である．

実際の実行画面は図 7であり，右側がドメインモデルをそのまま用いる方法であり，左側が事象トレース図を用いたものである．

## 6. スクリプト言語 Hoop

### 6.1 設計思想

**Hoop** 言語[4),5)] は，分析・設計段階で使用することを目的としており，以下のような特徴を有する．

- オブジェクト間のメッセージの流れを記述するメッセージセット
- ネットワーク上でのオブジェクトの柔軟な割り付けを可能とするオブジェクトのネスト構造

### 6.2 分散協調型モデルの表現方法

#### 6.2.1 メッセージセット

**Hoop** のモデルでは，オブジェクトどうしが直接メッセージをやりとりするのではなく，“宛先のオブジェクト”+“メッセージ” をひと組みとし，これをいくつか組み合わせたものをメッセージセットとして，メッセージセットをオブジェクトどうしがやりとりする．すなわちメッセージセットとは，オブジェクト間に伝わるメッセージの流れを記述するためのものである．

メッセージセットを導入することにより，本来メッセージの流れを管理する立場のオブジェクトが，これを指定できるようになる．例えば，ワークフローシステムにおいて全体のワークフローを管理するメタなシステムあるいはメタなオブジェクトが不要になり，純粋な分散協調型モデルを構築できる．

このメッセージセットは，先に述べた OAM モデルにおけるメッセージ集合 M の各要素を時系列的に複数個まとめたものに対応する．

#### 6.2.2 メッセージセットの構文

メッセージセットの構文を以下に示すが，言語の詳細は文献[4),5)] に詳しい．

$$M ::= M' \| (cond) M'$$
$$M' ::= M_s \| M_p \| X$$
$$M_s ::= [M, M, \ldots, M]$$
$$M_p ::= [M|M|\ldots|M]$$
$$X ::= obj.msg$$

obj はオブジェクト， msg はメッセージ， $cond$ は条件， $\alpha\|\beta$ は $\alpha$， $\beta$ のどちらかを意味する．

#### 6.2.3 直列のメッセージセット

メッセージの流れを表現するための従来の方法のひとつとして，アクタモデル[8)] のメッセージでは

[ 宛先のオブジェクト, メッセージ,
　　結果の送り先のオブジェクト ]

のように表現する．**Hoop** では，さらに “結果の送り先に対するメッセージ” も一緒に送る．

例えば，以下のような直列にメッセージを送る場合を考える．

例) ある書類 o を提出する時に，A さん，B さん，C さんにハンコをもらわなければならず，かつ A さん，B さん，C さんの順番でハンコをもらわなければならない．

このような直列のメッセージセットは以下のように記述する．

$$[M_1, M_2, \ldots, M_n] \ (n \geq 1)$$

簡単な例としては

$$[obj_1.msg_1, obj_2.msg_2, \ldots, obj_n.msg_n]$$

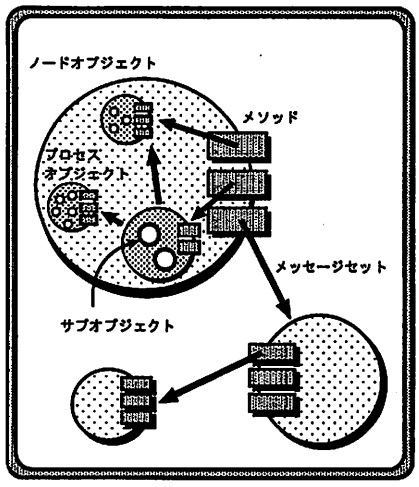


図 8 3種類のオブジェクト

である．オブジェクト $obj_i$ はメッセージ $msg_i$ を受け取るとオブジェクト $obj_{i+1}$ に対して

$$[obj_{i+1}.msg_{i+1}, obj_{i+2}.msg_{i+2}, \ldots, obj_n.msg_n]$$

を送る．

#### 6.2.4 並列のメッセージセット

一方，並列にメッセージを送る場合は以下のように記述する．

例) 会議を開催する時に，Aさん，Bさん，Cさんに出欠の問い合わせをする．ただし，Aさん，Bさん，Cさんのどの順番で返事がかえってきても良い．

このような並列のメッセージセットは以下のように記述する．

$$[M_1|M_2|\ldots|M_n]\ (n \geq 1)$$

簡単な例としては

$$[obj_1.msg_1|obj_2.msg_2|\ldots|obj_n.msg_n]$$

である．あるオブジェクトが上記のメッセージセットを送ると，オブジェクト $obj_1, obj_2, \ldots, obj_n$ にそれぞれメッセージ $msg_1, msg_2, \ldots, msg_n$ が送られる．

#### 6.2.5 ガード

メッセージセットに条件を付加し，メッセージの流れを動的に変化させることができる．**Hoop** では，これをガードと呼ぶ．

ガードを使うのは，以下のような場合である．

例) 会議の出欠をとる時に，出席であれば出席通知，欠席であれば欠席通知を出す．

ガードは以下のように与えられる．

( 条件 ) $M'$

条件が真の時のみメッセージセットが送られる．

簡単な例として，直列のメッセージセットにガードを付加した場合は，

$$(\text{条件})\ [obj_1.msg_1, obj_2.msg_2, \ldots, obj_n.msg_n]$$

のようになる．

### 6.3 分散システムの実行方式

**Hoop** では，分散協調すべきオブジェクト群をネットワーク上のノードへ割り当てる問題や個々のオブジェクトの機能が複雑になった場合のオブジェクト分割問題を解決するために以下の3種類のオブジェクトを導入する (図 8)．

ノードオブジェクト　ネットワーク上のノードに対応するオブジェクトであり，内部にプロセスオブジェクトを持つことができる．基本的に，**Hoop** ではノードオブジェクトを使い記述する．

プロセスオブジェクト　解決すべき問題が複雑な場合に使用する．複数のプロセスの協調により，ノードオブジェクトが果たすべき機能を実現する．内部にサブオブジェクトを持つことができる．

サブオブジェクト　補助的なオブジェクトである．内部にサブオブジェクトを持つことができる．ただし，ノードオブジェクトやプロセスオブジェクトと異なり，逐次処理に限られる．

オブジェクト単位に機能を分割していくときの一番大きな分割の単位がノードオブジェクトである．基本的には，ノードオブジェクトで問題を記述できる．しかし，ノードオブジェクトの機能が複雑な場合は，ノードオブジェクト内をプロセスオブジェクトで分割することにより複雑さを解消する．サブオブジェクトについては，ノードオブジェクトをプロセスオブジェクトで分割しても，まだ複雑である場合に使用する．

## 7. モデリング & シミュレーションツールと言語処理系の連携

### 7.1 スクリプトの生成

会議開催システムにおいて，モデリング & シミュレーションツールで作成されたオブジェクトおよびメッセージ変換列から次のようなスクリプトが自動生成される．

```
class Main {
    public main(String args[]){
        Class_事務局 事務局;
        Class_会議室管理 会議室管理;
        Class_備品管理 備品管理;
        Class_Staff 安部;
        Class_Staff 馬場;
        Class_Staff 千葉;

        事務局 = new Class_事務局 ();
```

```
        会議室管理 = new Class_会議室管理 ();
        備品管理 = new Class_備品管理 ();
        安部 = new Class_Staff();
        馬場 = new Class_Staff();
        千葉 = new Class_Staff();
            :
    }
}

class Class_事務局 {
    public 予約 (Date d){
        [ 会議室管理. 予約 (d), this. 結果 (boolean) ];
        [ 備品管理. 予約 () ];
        [ 安部. アナウンス () | 馬場. アナウンス () |
            千葉. アナウンス ()];
    }
}
```

このスクリプトは Java 言語[9] をベースとした **Hoop** 言語で記述されている．これを実行する処理系は Java で記述されたインタプリタである．ステップごとの実行や属性の変更機能についても今後実装していく．

### 7.2 スクリプトの実行

メッセージセットの実行方式を会議開催システムの例で説明する．前節で示した Class_事務局の "予約" メソッドには次のようなメッセージセットがある．

[ 会議室管理. 予約 (d), **this.** 結果 (**boolean**) ];

このメッセージセットは，会議室管理の "予約" メソッドに渡される．その予約処理の中には **next**(...) 文が記述されており，それを実行した時点で，残りのメッセージセット

[ **this.** 結果 (...) ];

が呼ばれる．

分散システムを実現するために 3 種類のオブジェクトのうち，ノードオブジェクトとサブオブジェクトを実現した．サブオブジェクトを生成するには，プログラム中で次のように宣言する．

**new** クラス名 ();

ノードオブジェクトを生成するには，プログラムの起動時に稼働させたいノード側で次のように指定する．

java hoop.Main -n "ノード名" "ファイル名"
"クラス名" "メソッド名" "引数"

ノード名は，*hostname/name* とする．*name* については任意の名前でよい．ノードオブジェクトの参照が必要な時は，プログラム中で次のように指定する．

**bind** クラス名 () ["ノード名"]

ネットワーク部分の実装は，ORB の技術を使用し，RMI を用いた．

## 8. おわりに

「モデリング & シミュレーション」を支援するアプリケーション開発環境 **M-base** の分析・設計技法について述べた．モデリング & シミュレーションツールとスクリプト言語処理系のプロトタイプを Java 言語を用いて実現した．これらにより業務モデルの動的な振舞いをモデリングことでアプリケーションを構築することが実現可能であることが確認された．

今後は，これらのツールを改良し，統合すると共に，様々なアプリケーションの開発に適用し，実用性を評価する．

## 参考文献


1) 中所武司: M-base:「ドメインモデル ≡ 計算モデル」を志向したアプリケーションソフトウェア開発環境の基本概念, 情報処理学会 ソフトウェア工学研究会資料, No.95-SE-104-4(1995).
2) Takeshi CHUSHO, Yuji KONISHI and Masao YOSHIOKA : M-base : An Application Development Environment for End-users Computing based on Message Flow, APSEC'96, IEEE Computer Society(1996).
3) 中所武司: ソフトウェア危機とプログラミングパラダイム "わかりやすさ" の追求, 啓学出版 (1992).
4) 小西裕治, 中所武司: 分散協調型アプリケーションのためのオブジェクト分析・設計言語 Hoop の設計とその記述実験, 情報処理学会 OO'96 シンポジウム pp.87-94(1996).
5) 小西裕治, 中所武司: オブジェクト分析・設計言語 Hoop における分散協調型モデルの表現法, 日本ソフトウェア科学会 第 12 回大会論文集 pp.197-200(1995).
6) 松本光由, 中所武司：「ドメインモデル≡計算モデル」を志向したアプリケーションソフトウェア開発環境 M-base におけるモデリング & シミュレーション技法, 情報処理学会：ウィンターワークショップ・イン・松山, pp.27-28(1997).
7) Ivar Jacobson,Magnus Christerson,Patrik Jonsson and Gunnar Overgaard: Object-Oriented Software Engineering —A Use Case Driven Approach—,the ACM press(1992).
8) Hewitt, C. and Baker, H.: Laws for communicating parallel processes, Proc. IFIP'77, pp.987-992(1977).
9) Javes Gosling, Henry Mc Gilton : The Java Language Environment A White Paper, ftp://ftp.javasoft.com/docs/whitepaper.A4.ps.tar.Z, (1995).