情報処理学会オブジェクト指向シンポジウムOO'96
オブジェクト最前線，朝倉書店，pp.87-94，1996

9617

分散協調型アプリケーションのための オブジェクト指向分析・設計言語 Hoopの設計とその記述実験

小西裕治† 中所武司†

グループウエアやワークフローシステムなどの分散協調型のアプリケーションソフトウェアを効率良く開発することを目的として，分析・設計段階で使用するオブジェクト指向言語 **Hoop** を開発中である．言語の基本機能として，オブジェクトを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，従来のひとつのメッセージではなく，メッセージセット単位でおこなう方式にして協調作業のための業務の委託方法に柔軟性を持たせた．さらに，分散協調型モデルの構築を形式化された手法でおこなうための手順を提示すると共に，グループウエ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を **Hoop** で記述し，記述の容易性，拡張性を確認した．

Design of an Object-Oriented Language for Analysis and Design of Cooperative Systems, and Its Feasibility Study

Yuji Konishi† and Takeshi Chusho†

The object-oriented language, Hoop, is used at the analysis and design phases for developing cooperative systems such as groupware and workflow systems. This language provides three types of objects in relation to network nodes assignment and concurrency, and supports the nested structure of objects. Communication among objects is performed by a set of messages, not by a message, for implementation of flexible work flow. Finally, writability and extensibility of this language are confirmed by feasibility study.

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．

本研究では，エンドユーザコンピューティング促進の一環として，グループウエア[1]やワークフローシステムを効率良く開発するために，「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エア開発環境 **M-base**[2] を開発中である．

その主な機能は，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ふるまいを表現できるようなモデリングツールとそこで用いるスクリプト言語である．

本報告では，スクリプト言語 **Hoop** の設計とその記述実験について述べる．**Hoop** は，分析・設計段階で使用するための言語であり，以下の概念をもとに設計されている．

- 上流工程で使用し，「分析 ≡ 設計 ≡ プログラミング」を目標とする．
- オブジェクト指向の基本概念を素直に表現出来る．
- 自立的なオブジェクトの並行処理 (concurrency) を前提とし，ネットワークを通してメッセージのやりとりが出来る．

2. 設計思想

オブジェクト指向の概念としては，以下の4点などがあげられる．

- 分散協調型モデル
- データ抽象化

† 明治大学理工学部情報科学科
Department of Computer Science, Meiji University

- クラスからのインスタンス生成
- クラス階層と継承

Hoop では，“モデリング重視”の立場から，オブジェクト指向の概念の中でも，特に分散協調型モデルを重視する．“一般に，ソフトウェア技術は「モデリング&シミュレーション」技術である．従来のプログラミングではシミュレーション可能なモデルを作るために論理を駆使してきたが，そのためには熟練技術が必要であった．...対象モデルを計算モデルに変換するという作業は残った”[4]という観点から，モデルの表現能力がソフトウェアを作る際の重要な要因であると考えた．

オブジェクトにメッセージを送ることによって計算を抽象化しようとする[5]オブジェクト指向プログラミング言語としては，ABCL/1[6]，ConcurrentSmalltalk[7]などがある．これらの言語ではメッセージパッシングの機能によって豊かな表現力を得ている．

Hoop では，オブジェクト指向の特長のひとつである強力なモデル化の能力を使い，対象世界を分散協調型モデルで表現する．オブジェクト指向のメッセージパッシングの概念を拡張しメッセージを組み合わせることによって，グループウエアやワークフローなどに必要な機能を実現した．

分散協調型アプリケーションの開発を目標とした**Hoop** のおもな特徴は，以下のようなものである．

- 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．
- 分散協調型モデルの構築の手順を示す形式化された分析・設計手法．

3. Hoop の分散協調型モデル

3.1 3種類のオブジェクト

分散環境下では，グループウエアやワークフローなどの協調型アプリケーションを構成する個々のオブジェクトは，基本的にはネットワーク上の任意のノードで稼働することが望ましい．そこで，ノードへの柔軟な割り付けを可能とするために以下の3種類のオブジェクトを導入した(図1)．

(1) 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．

(2) プロセスオブジェクト
解決すべき問題が複雑すぎる場合に使用する．並行処理可能な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．

(3) サブオブジェクト
補助的なオブジェクトであり，ひとつのプロセスオブジェクトの中で逐次的に実行される．内部にサブオブジェクトを持つことができる．

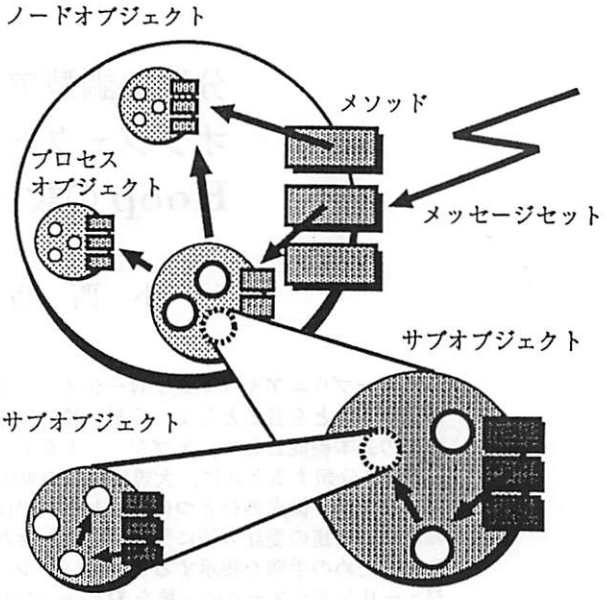

図1 3種類のオブジェクト

3.2 メッセージセット

オブジェクト指向における分散協調型モデルは，“複数個の自立的機能を持つオブジェクトがお互いにメッセージをやり取りしながら協調して問題解決にあたる”ものである．

Hoop のモデルでは，“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，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．

オブジェクトどうしが単一のメッセージをやりとりする従来方式のかわりに，このような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，その流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．

3.3 メッセージセットの構文

$$M ::= M' \| [cond] M'$$
$$M' ::= M_s \| M_p \| X$$
$$M_s ::= \{M_1, M_2, \ldots, M_n\}$$

$M_p ::= \{M_1|M_2|\ldots|M_n\}$
$X ::= (obj, msg)$

obj はオブジェクト
msg はメッセージ
$cond$ は条件
$\alpha\|\beta$ は α, β のどちらか

メッセージセットの基本単位は,
(obj, msg)
であり, obj に msg が送られる.

3.4 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ーモデル[8] がある. このモデルでは, メッセージを送る際に必要となる主な要素は

- 宛先のオブジェクト
- 要求するオペレーション
- 結果の送り先のオブジェクト

等である. **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. これは

$\{(obj_1, msg_1), (obj_2, msg_2)\}$
obj_1 は, 宛先のオブジェクト
msg_1 は, 宛先のオブジェクトに対するメッセージ
obj_2 は, 結果の送り先のオブジェクト
msg_2 は, 結果の送り先に対するメッセージ

と表すことが出来る. このようにオブジェクトとメッセージの組み (obj, msg) を複数組み合わせたものが, **Hoop** の直列のメッセージセットの基本的な考え方である.

直列にメッセージを送るのは, 以下のような場合である.

例 1) 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.

例 2) A さんは書類 o1 をもらうと, 書類 o2 を作り, B さんに送る. B さんは書類 o2 をもらうと, 書類 o3 を作り, C さんに送る.

例 1 の場合は共通の書類 o が順番にまわって行き, 例 2 の場合は次々に書類を作って行く.

直列のメッセージセットは以下のように与えられる.

$\{M_1, M_2, \ldots, M_n\}\ (n \geq 1)$

簡単な例としては

$\{(obj_1, msg_1), (obj_2, msg_2), \ldots, (obj_n, msg_n)\}$

である. オブジェクト obj_i はメッセージ msg_i を受け取

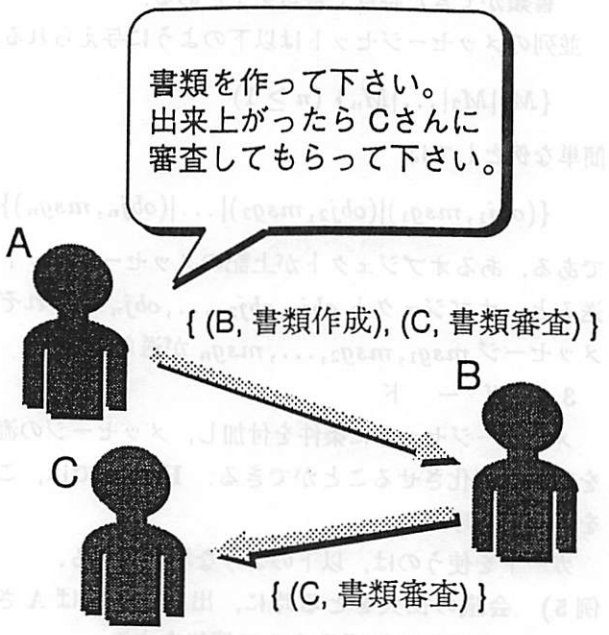

図 2 直列のメッセージセットの例

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.

例えば, A さんが B さんに, 「書類を作成して, C さんの審査を受ける」ように指示した場合のメッセージセットの流れを図 2 に示す. B さんは, A さんからメッセージセット

$\{(B, \text{書類作成}), (C, \text{書類審査})\}$

を受け取り, 書類作成後, C さんにメッセージセット

$\{(C, \text{書類審査})\}$

を送る.

3.5 並列のメッセージセット

あるオブジェクトがオブジェクト obj_1 にメッセージ msg_1, オブジェクト obj_2 にメッセージ msg_2, ..., オブジェクト obj_n にメッセージ msg_n を送りたいとする. しかも, メッセージの送られる順番は問わない時は一度に送れれば便利である.

並列にメッセージを送るのは, 以下のような場合である.

例 3) 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.

例 4) A さんには書類 o1 の作成, B さんには書類 o2 の作成, C さんには書類 o3 の作成を頼み, 三つの

書類ができた時点で書類をまとめる.

並列のメッセージセットは以下のように与えられる.

$$\{M_1|M_2|\ldots|M_n\}\ (n \geq 1)$$

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.

3.6 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 5) 会議の出欠をとる時に, 出席であればA さんに, 欠席であればB さんに連絡をする.

例 6) 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.

例 5 の場合は, 条件により宛先のが変化し, 例 6 の場合は, 送られるメッセージが変化する.

ガードは以下のように与えられる.

$$[\text{条件}]\ M'$$

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.

4. モデル化の手順

モデル化の手順を示すための例題として, 文献[6] でオブジェクト指向モデル化が試みられている“在庫管理問題”を取り上げる.

4.1 在庫管理問題

以下に, 在庫管理問題を示す. 図 3はそのモデルを図式化したものである.

「ある酒類販売会社の倉庫では, 毎日数個のコンテナが搬入されて来る. その内容はびん詰めの酒である. 倉庫係は, コンテナを受け取りそのまま倉庫に保管し積荷表を受付係に手渡す. また, 受付係からの出庫指示によって内蔵品を出庫することになっている. 内蔵品は別のコンテナに詰め替えたり, 別の場所に保管することはない. 空になったコンテナはすぐに搬出

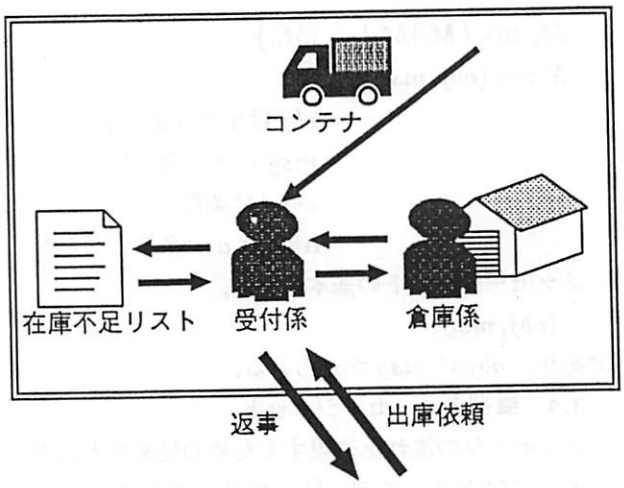

図 3 在庫管理問題

される.

受付係は出庫依頼を受け, そのつど倉庫係に出庫指示書を出すことになっている. ただし, 1 件の依頼では, 1 銘柄のみに限られるものとする. そして, 在庫が無いか数量が不足の場合には, その旨依頼者に連絡し同時に在庫不足リストに記入する. また, 空になる予定のコンテナを倉庫係に知らせることになっている.

受付係の仕事のための計算機プログラムを作成せよ.」

4.2 着 目 点

在庫管理問題の以下の点に着目した.

(1) 受付係が仕事の中心になっており, 受付係が倉庫係などに指示をする.

(2) 受付係が出庫依頼を受け取った場合, 受付係は倉庫係に出庫依頼のメッセージを送り, 次に倉庫係は受付係に在庫の不足のメッセージを送り, 最後に受付係は依頼者に在庫不足のメッセージを送る, といったようなメッセージの流れが出来る.

この例からもわかるように, 全てのオブジェクトが, 同じような仕事をする (図 4(a)) よりも, むしろ, 外部とのやり取りをするオブジェクト (以下, 受付オブジェクト) がいて, 受付オブジェクトが他のオブジェクトに仕事を頼む (図 4(b)) ようにモデリングされる場合が多いと考えられる.

仕事の依頼は,

受付オブジェクト → 他のオブジェクト

という単純な流れではなく, 倉庫係, 在庫不足リストオブジェクトなどは協調して問題を解決するので, オブジェクト間を次々とメッセージが伝わって行く. つまり, 仕事の依頼は

受付オブジェクト → 他のオブジェクト$_1$ → 他

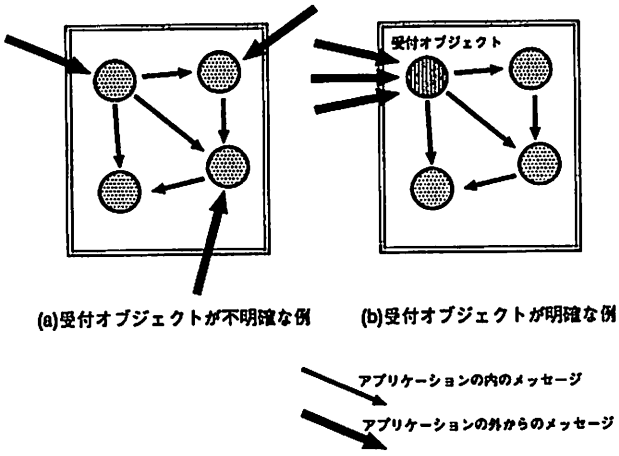

図4 分散協調型モデル

のオブジェクト$_2$→ ...

のように，複数のオブジェクトの間を流れていく．例えば，受付係が出庫依頼を受け取ったが在庫が不足している場合，受付係 → 倉庫係 → 受付係 → 依頼者とメッセージが伝わる．

以上のことをまとめると

(1) 全てのオブジェクトは同じ立場ではなく，外部からのメッセージを受け，他のオブジェクトに仕事を依頼する受付オブジェクトと，受付オブジェクトからの依頼を遂行するオブジェクトに分けられる．

(2) 受付オブジェクトから仕事が開始され，オブジェクト間に次々とメッセージが伝わることによって仕事が進められる．

となる．

4.3 モデル化

以上のような考察に基づいて構築したモデル化の手順を以下に示す．この手順の形式化の詳細は文献[2]に詳しい．

(1) オブジェクトの抽出

メッセージのやり取りをするものをオブジェクトとする (図 5)．

(2) 受付オブジェクトの抽出

抽出したオブジェクトの中から，外部とのメッセージのやり取りをするものを，受付オブジェクトにする (図 6)．

(3) 受付オブジェクトが行う依頼の抽出

外部からのメッセージを受け取った後，他のオブジェクトに対して行う依頼を取り出す (図 7)．“obj$_1$ に対して msg$_1$ を頼んだ後，obj$_2$ に対して msg$_2$ を頼み，...” のような記述である．

(4) 依頼からメッセージセットへの変換

“obj$_1$ に対して msg$_1$ を頼んだ後，obj$_2$ に対して msg$_2$ を頼み，...”(図 8) という依頼を以下のようなメッセージセットに変換する．

$$\{ (obj_1, msg_1), (obj_2, msg_2), \ldots \}$$

(5) 個々のメッセージの作成

msg$_1$, msg$_2$, ... を実装する．

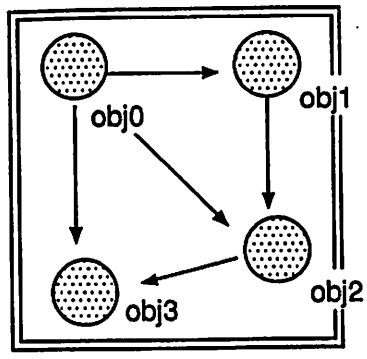

図5 オブジェクトの抽出

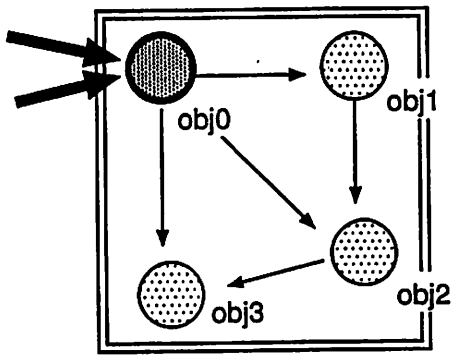

図6 受付オブジェクトの抽出

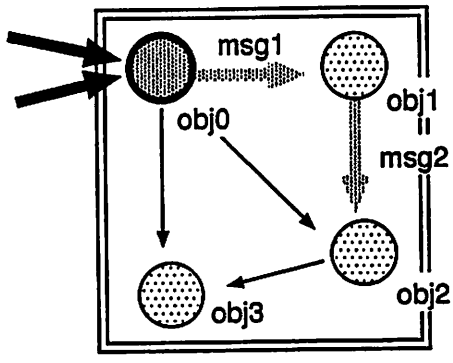

図7 受付オブジェクトが行う依頼の抽出

図8 依頼からメッセージセットへの変換

5. 例題への適用

モデル化の手順を，本システムが主要なターゲットと

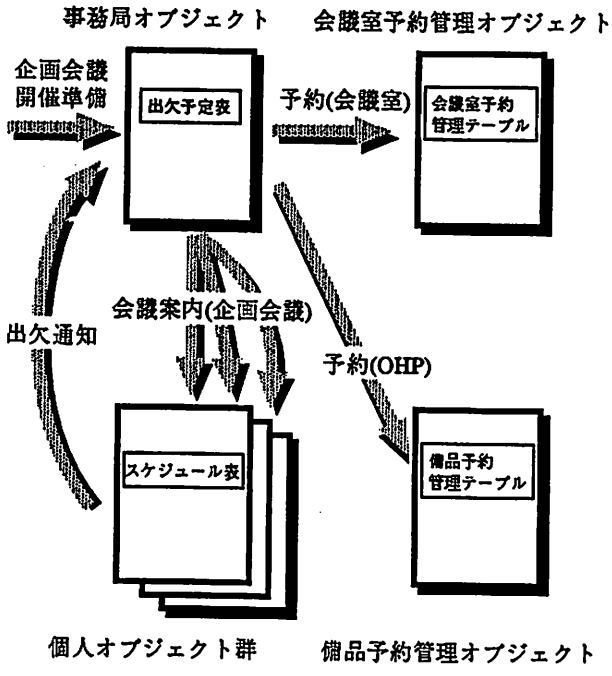

図9 会議開催事務処理

しているエンドユーザコンピューティングの分野の例題に当てはめ，有用性を確認する．例題として，会議開催事務処理問題[4]を取り上げる．

この例題は，グループウエ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．現在，例えばオフィスで，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という業務を自動化するアプリケーションの開発を想定する．

5.1 オブジェクトの抽出

“会議開催事務処理問題”は以下の4種類のオブジェクトからなる(図9)．

(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) 会議室予約管理オブジェクト
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．

5.2 受付オブジェクトの抽出

この例では，会議開催要求のメッセージを外部から受け取っている事務局オブジェクトが受付オブジェクトにあたる．

5.3 受付オブジェクトが行う依頼の抽出

依頼は，以下の3点である．

- 会議室予約管理オブジェクトへ会議室の予約
- 備品管理オブジェクトへOHPの予約
- 会議のメンバへの会議開催案内

5.4 依頼からメッセージセットへの変換

受付オブジェクトが行う依頼をメッセージセットに変換する．

5.4.1 会議室の予約

会議室予約管理オブジェクトに会議室の予約のメッセージを送れば良いので，

{ (会議室予約管理, 会議室の予約) }

となる．

5.4.2 OHPの予約

備品管理オブジェクトにOHPの予約のメッセージを送れば良いのだが，OHPの予約ができなかった場合は，事務局オブジェクトに貸出不可のメッセージが送られるようにする場合を考える．

貸出不可のメッセージについては，“OHPの予約ができなかった場合”という条件があるので，ガードを付加すれば良い．また，事務局オブジェクト → 備品管理オブジェクト → 事務局オブジェクトとメッセージが次々と伝わって行くので，直列のメッセージセットを使えば良い．したがってメッセージセットは次のようになる．

{ (備品管理, OHP予約),
[OHPの予約不可] (事務局, 貸出不可) }

5.4.3 会議開催案内

個人オブジェクトに対しては，会議案内を送り，返事として出欠通知をもらわなければならないので，

{ (個人, 会議案内), (事務局, 出欠通知) }

を送れば良い．

個人オブジェクト全員に対する会議案内メッセージの到着の順番は関係ないので，並列のメッセージセットとして送れる．

{
{ (個人$_1$, 会議案内), (事務局, 出欠通知) } |
{ (個人$_2$, 会議案内), (事務局, 出欠通知) } |
⋮ ⋮

{ (個人 $_n$, 会議案内), (事務局, 出欠通知) }
}

5.5 個々のメッセージの作成

メッセージセットの中に含まれていたメッセージは,

- 会議室の予約
- OHP の予約
- 貸出不可
- 会議案内
- 出欠通知

であるので, それぞれを作成すれば良い.

5.6 適用結果

ここで提示したような手順を使うことによって, M-base の 1 オブジェクト 1 業務の原則のモデル化がおこなえることを示した. また, メッセージセットによ[illegible], メッセージで表現できるオブジェクト間の協調関係をシンプルにモデル化出来た.

このような方式により, 例えば, "すでに予約されている会議室を譲ってもらう" という問い合わせ処理を事務局オブジェクトに追加するような拡張も容易に出来る. 一例としては以下のようである.

{ (会議室予約管理, 会議室の予約),
[すでに予約済] {(事務局, 予約の返事),
(予約者, 変更依頼) }

となる. メッセージの変更, 追加は

- "予約の返事" の追加
- "変更依頼" の追加
- "会議開催準備" の修正
- "会議室の予約" の修正

[illegible]修正内容が容易に確定するという点で, 拡張も容易で[illegible]る.

6. 分散実行問題への適用

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以下のように解決する.

Hoop では, オブジェクト単位に機能を分割していき問題を解決する. 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になりすぎる場合は, ノードオブジェクト内を複数のプロセスオブジェクトに分割することにより複雑さを解消する. ノードオブジェクトの複雑さが会議開催事務処理問題程度である場合, ふたつの実現方法が考えられる.

ひとつは, 例えば会議室予約オブジェクトが複数のアプリケーションで共有される場合, それを別のノードオブジェクトにしておく方法である. この場合, 実際には複数のノードオブジェクトが同一のノードで稼働することも可能である.

もうひとつは, 会議開催事務処理のようなアプリケーション全体をひとつのノードオブジェクトとみなす方法である. この場合, 事務局オブジェクトをノードオブジェクトとし, その他のオブジェクトをノードオブジェクトの内部のプロセスオブジェクトとする.

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.

複雑なオブジェクトでも, 会議開催事務処理問題程度に分割できれば, **Hoop** の分散協調型モデルで表現可能である.

7. プログラム例

Hoop の言語仕様は Java[9] をベースにコンパクトなものにする方針である.

以下に, 会議開催問題を記述したプログラム例を示す. 事務局オブジェクト (office) が個人オブジェクト (staff) に対して会議案内 (arrange()) を送り, 返事として出欠通知 (replyYes() もしくは replyNo()) をもらう場合を考える.

会議案内には, 会議の日付が引数として与えられている. 返事のメッセージにはガードが付加されており, 出席 (true), 欠席 (false) によって, 事務局オブジェクトに返される返事が異なるようになっている. 個人オブジェクトは, 会議の日付 (date) が日曜 (SUN) でなければ出席するとする. 事務局オブジェクトは返事により, 出席, 欠席の人数 (yes, no) を数える.

```
class Office extends Object {
  int yes, no;
  Staff staff[ ];
  public Office()[ ]
  {
    yes = 0;
    no = 0;
    staff = new Staff[3];
    staff[0] = new Staff("abe");
    staff[1] = new Staff("baba");
    staff[2] = new Staff("chiba");
    [ self.arrange() ];
  }
```

```
  public arrange()[ ]
  {
    Date date;
    date = new Date(2, 29);
    [ { staff[0].reserve(date),
        { [ ok == true ] self.replyYes(ok) |
          [ ok == false ] self.replyNo(ok) }} |
      { staff[1].reserve(date),
        { [ ok == true ] self.replyYes(ok) |
          [ ok == false ] self.replyNo(ok) }} |
      { staff[2].reserve(date),
        { [ ok == true ] self.replyYes(ok) |
          [ ok == false ] self.replyNo(ok) }} ];
  }
  public replyYes(boolean ok)[ ]
  {
    yes = yes + 1;
  }
  public replyNo(boolean ok)[ ]
  {
    no = no + 1;
  }
      :
}

class Staff extends Object {
  string name;
  public Staff(string n)
  {
    name = n;
  }
  public reserve(Date d)[ o.reply(boolean ok) ]
  {
    [ o.reply([ d.week() ] != SUN) ];
  }
      :
}
```

8. おわりに

Hoop におけるモデル化の手順，ならびに，モデル化の際に使用されるメッセージセット，3種類のオブジェクトについて説明をした．今後は，**Hoop** のプロトタイプを実装し，アプリケーションへの適用実験をおこなう．

現在**Hoop** を含む**M-base** プロジェクトでは，「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の研究開発をおこなっているが，**Hoop** を**M-base** の主要技術であるコンポーネントウエアおよびビジュアルプログラミングツールで用いるスクリプト言語として位置付けている．

参考文献

1) J. Grudin: Computer-Supported Cooperative Work: History and Focus, *IEEE Trans. Computer*, Vol. 27, No. 5, pp. 19-26 (1994).
2) 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エ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).
3) 小西裕治，中所武司：オブジェクト指向分析・設計言語 Hoop における分散協調型モデルの表現方法, 日本ソフトウェア科学会 第12回大会論文集, pp. 197–200 (1995).
4) 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).
5) 石川 裕，所 真理雄：オブジェクト指向並行プログラミング言語, 情報処理, Vol. 29, No. 4, pp. 325–333 (1988).
6) 柴山悦哉，松田裕幸，米澤明憲：並列オブジェクト指向言語 ABCL によるプログラミング オブジェクト指向 解説と WOOC'85 からの論文, 共立出版 (1985).
7) 横手靖彦，所 真理雄：並行オブジェクト指向言語 ConcurrentSmalltalk, コンピュータソフトウェア, Vol. 2, No. 4, pp. 582–598 (1985).
8) 米澤明憲：ACTOR 理論について, 情報処理, Vol. 20, No. 7, pp. 581–589 (1979).
9) J. Gosling, H. McGilton: The Java Language Environment: A White Paper, *ftp://java.soft.com/docs/whitepaper.A4.ps.tar.Z* (1995).