題目　エンドユーザコンピューティングのための
分散オブジェクト指向設計技法

英文題目　Distributed object-oriented design
for end-user computing

著者名　中所　武司
所属　明治大学 理工学部 情報科学科
ソフトウェア工学研究室

概要

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．この傾向は，情報のグローバル化と情報のパーソナル化という両面から一段と加速されるものと思われる．これに対応するため，業務の専門家が，自ら，業務のモデルから直接情報システムを構築できる技術として，分散オブジェクト指向設計技法を開発する．

まず初年度（平成６年度）には，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パラダイム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）

・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．そして，その実現の第１段階として，非定型業務のモデル化をマクロレベルとミクロレベルの２階層に分けて行う方式とし，マクロレベルでは，既存のオブジェクト指向開発技法のようにオブジェクト間の静的関係を最初に定義するボトムアップ法ではなく，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最初に構築するトップダウン法を規定した．その具体的な開発手順を形式化し，その例題を示した．

次年度（平成７年度）は，これらの技術をモデリング＆シミュレーション機能として実現するアプリケーションフレームワークの設計を行い，上記の基本技術を実現するプロトタイプを開発する．

なお，本研究はエンドユーザのみによる開発・保守を最終目標とするが，マイルストーンとして，開発はシステムエンジニアの支援を受けながらエンドユーザ主体で行い，保守はエンドユーザのみで行うレベルを設定する．

成果の要約（指定項目に関して）

■全体の目標と今年度の達成度
・第１ページの概要に記載.

■用途
・次ページ以降の本文中，第２章に記載.

■従来技術との比較
・次ページ以降の本文中，第３章３.１および第５章５.４に記載.

■今後の課題
＜平成７年度：プロトタイプの開発＞
・アプリケーションフレームワークの設計
・上記の基本技術を実現するプロトタイプの開発.
＜平成８年度：評価改良と機能拡張＞
・特定のアプリケーションを取り上げ，上記のプロトタイプの評価改良.
・実用化のための機能拡張.

■有形成果物（論文を除く）
・下記ドキュメントのみ

■デモンストレーションの可否
・開発プログラム無し（平成７年度プロトプログラム開発予定）

■実用化シナリオ
・平成６年度から平成８年度までの３年間の基礎研究で，プロトタイププログラムの開発による feasibility study を終了後，育成研究を申請し，実用化研究を推進する.

■ドキュメント
・T. Chusho : Modeling of Application Software for End-User Computing as "a Domain Model ≡ a Computation Model," Technical Report of, Dept. of Computer Science, Meiji University (Sep. 1994).
・中所：EAGL第２３回技術交流会配付資料（1995.1.24）
・中所：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（1995.5予定）

1．はじめに

情報システムは，従来，業務の専門家が情報処理の専門家に依頼して開発することが多かった．しかし，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．この傾向は近い将来の多様なネットワークのさらなる普及により，情報のグローバル化と情報のパーソナル化という両面から一段と加速されるものと思われる．

このような分散コンピューティング，エンドユーザコンピューティング [5] という新しい動向に対応して，分散環境下でのオフィス業務（非定型業務）を業務の専門家自身がコンピュータ化するための設計技法を開発する必要がある．

これまでは，ソフトウエア開発技法の研究に関しては，大規模な定型業務（基幹業務）向けソフトウエアの開発技法が中心である一方，エンドユーザは，ＯＡパッケージソフトウェアの利用が中心だった．

本研究では，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 [6] が必要であるという認識から，このような”作りやすさ”と”使いやすさ”を両立させるための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 [12]．オブジェクト指向概念 [4] に基づく分散協調型問題解決モデル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フレームワークを開発する．

最終的には，市場の急速な拡大が予想されるエンドユーザコンピューティングツールとして実用化する．非定型業務を業務モデルのレベルでコンピュータ化できることにより，情報処理に関するインフラ（情報機器・ネットワーク）を十分活用でき，情報（化）社会の進展に寄与できる．さらに若年労働者の減少に対応して，一人あたりの労働生産性を高めることができると思われる．

本報告では，非定型業務のモデル化をマクロレベルとミクロレベルの２階層に分けて行う方式とし，マクロレベルでは，既存のオブジェクト指向開発技法のようなオブジェクトの間の静的な関係を最初に定義するボトムアップ法ではなく，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最初に構築するトップダウン法を確立したので，その内容について述べる．２章では本研究の目的，３章では本研究で基本とする計算モデル，４章ではモデリングプロセスの形式化，５章ではアプリケーション開発の例題について述べる．

2．研究の目的と対象

本研究は，誰が，何のために，どのように利用する技術を開発しようとしているかを明確にしておく．

（1） エンドユーザ

情報（化）社会の到来は，エンドユーザの爆発的増大を招き，それに対応するために新分野の開発方法論が必要であるというのが，本研究の基本的な動機である．そこで，まずどのようなエンドユーザを研究対象とするかを明確にしておく．

一般に，エンドユーザとして少なくとも以下の３種類が考えられる．

（ａ）基幹業務担当者

例えば，銀行のようなユーザ企業において，システム部門に対するエンドユーザ部門に所属する人達で，利用するソフトウエアはシステム部門が開発し，提供してくれる．

（ｂ）業務の専門家

一般にオフィスワーカーといわれるような人達で，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する．

（ｃ）一般ユーザ

例えば，日常生活の中で銀行のＡＴＭを利用するような一般の人達で，将来，マルチメディア時代の主要ユーザとなる．

本研究では，（ａ）と（ｂ）のエンドユーザに

対象を絞るが，特に非定型業務のコンピュータ化という視点では今後急速に増大すると思われる（b）の方により重点を置く．（c）のエンドユーザについては，当面はパソコンを利用する場合も電子メールや文書作成のためのツールとしてアプリケーションパッケージを利用することはあるが，自らアプリケーションソフトウエアを開発することは少ないと思われる．

（2）対象ソフトウエア

ソフトウエア開発の問題領域は多様であり，汎用的な設計プロセスは難しい．例えば，ビジネス分野の基幹業務向きのデータモデル中心の設計法，エンジニアリング分野の状態遷移モデル中心の設計法などのように，設計法が問題領域に依存するのは止むをえない．

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CS-life（Computer-supported Life：コンピュータ支援による良い生活）[9]の実現の観点から，上記の（b）のエンドユーザが主体であり，オフィスでの非定型業務用アプリケーション[8]が中心となるので，分散協調型モデルに基づく設計法を提案する．従って，将来的にはCSCW (Computer-Supported cooperative Work) [17,23]のツールやエージェントシステム[22]も対象となる．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．

（3）開発・保守形態

本研究では，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，

「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，問題領域の分析により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，

「ソフトウェア開発＝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，そのモデル上でのシミュレーションにより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．従来の開発工程に対応させるならば，モデリングが分析／設計工程に相当し，プログラミング／製造工程がシミュレーションに相当する．

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（applications of the end-users, by the end-users, for the end-users）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．

また，開発にあたって，既存の類似システムが存在する場合は，その開発方法論が異なっていてもデータベースやファイルの構造あるいはユーザインタフェースの操作などについて，ある程度の見通しが得られるので，クラスオブジェクトの設計やクラスオブジェクト間の関係の定義を初期の段階で行うことが可能である．しかし，本研究の対象分野は，一般に前例のない新規開発の場合が多く，トップダウンに設計する必要がある．

さらに，受注ソフトに多い大規模なソフトの場合は多人数開発になるので，ウォータフォールモデルあるいはその改良方式を用い，各工程で仕様を検証しながら開発を進めるフェーズドアプローチをとる．それに対し，本研究では，エンドユーザコンピューティングの分野を対象とするため，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あげるプロトタイプアプローチを想定する．

3．計算モデル

3.1 オブジェクト指向概念の定義

オブジェクト指向概念の定義は必ずしも明確ではないが，actorモデル[18]などのメッセージ交換に

よる分散協調型計算モデル，Simula67[14]からSmalltalk80[16]につながる抽象データ型などのプログラミング言語概念[11,21]およびＥＲモデルなどのデータモデルがベースになっている．既存のオブジェクト指向設計技法は，G.Booch[1], P.Coad &E.Yourdon[13], S.Shlaer & S.J. Mellor[26], J.Rumbaugh[25]等，などをはじめとしてデータモデルに重きを置いたものが多い．

本研究では，システム全体の動的ふるまいに着目する立場から，計算モデルとしての位置付けを重視し，分散協調型モデル，抽象データ型，クラスからのインスタンス生成，継承機能付きクラス階層をオブジェクト指向概念の主要なものとするが，詳細は後述する．

3.2 2階層モデル

設計技法をモデリング技法とみた場合，従来方式は3種類のモデルを組み合わせたマルチパラダイム型のものが多い．S.Shlaerらの情報モデル，状態モデル，プロセスモデル，J.Rumbaughらのオブジェクトモデル，動的モデル，機能モデルなどがその例である．これらの技法に共通する特徴は，設計プロセスの初期の段階でオブジェクトおよびオブジェクト間関係の定義を重視することである．そのため，とくにオブジェクト間関係のビジュアルな表記法が豊富に導入されている．

これらのボトムアップ的アプローチは，既に開発運用実績を有する従来の基幹データベースを中心としたビジネスシステムの再構築のような場合には適用可能である．しかし，システム全体のマクロレベルの動的ふるまいの設計法があいまいであることや3つのモデルの統合が不明確などの欠点[15,24]があるため，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に近い非定形業務の新規開発には適さない．

このような分野では，何をオブジェクトにするかを決定するためには，データモデルよりも計算モデルを重視し，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ふるまいを記述することが重要である．本報告では，図1に示すように，著者らがオブジェクト指向を基本としたマルチパラダイム型言語[3,10]の開発時に提案した2階層モデルをベースに，ミクロモデルよりもマクロモデルを，静的構造よりも動的ふるまいを優先する設計プロセスを追求する．

本研究での基本的コンセプトとの関連で2階層モデルを規定すると以下のようになる．

（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．

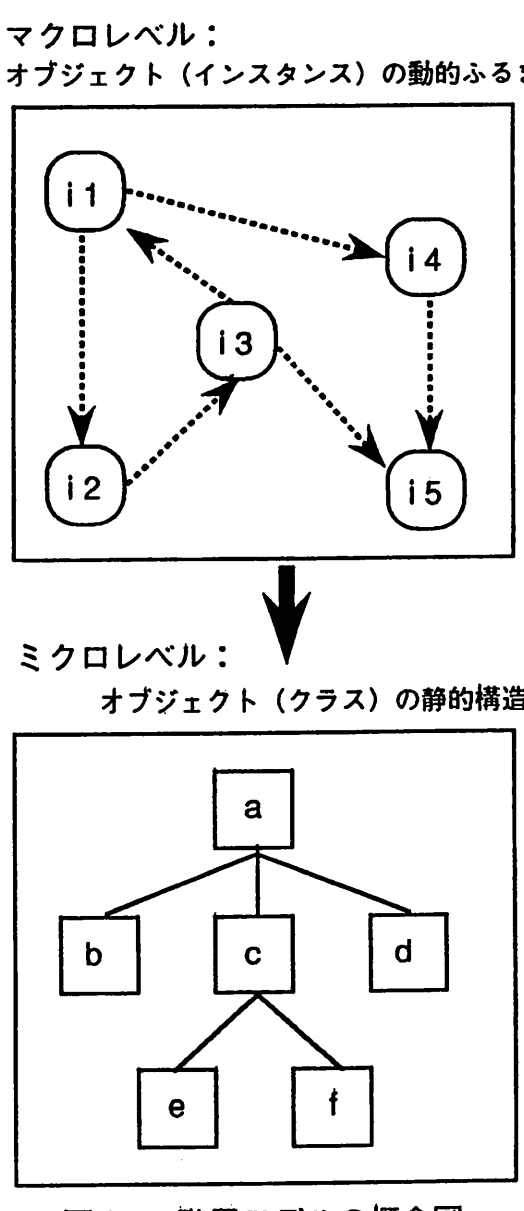

図1　2階層モデルの概念図

3.3 オブジェクト指向概念の発生学的定義

本研究のベースとするオブジェクト指向概念の基本的な要件として，計算モデルの観点から以下の4項目をとりあげる．

●分散協調型計算モデル
●データ抽象化機能
●クラスからのインスタンス生成機能
●クラスの階層化と継承機能

これらのオブジェクト指向概念をより厳密に規定するために，著者らが以前に言語設計時に用いたオブジェクト指向言語モデルの形態形成過程[2]に基づく発生学的定義を以下に示す．即ち，ある概念の存在が前提となって別の概念が導かれるとき，前者が後者よりも先に定義される．なお，以下では特に断わらないかぎりオブジェクトという表現はインスタンスを意味する．

（1）メソッドとメッセージ送信

複数個の自律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という計算モデルの中で，個々のオブジェクトの機能（外部仕様）は，メソッドの集合とする．各メソッドはそれを指定したメッセージの受信によって起動され，所定の機能を果たす．（分散協調型計算モデルの形成）

（2）データ

個々のオブジェクトは，オブジェクトの状態を保持するために，必要ならばデータ（変数）を有する．これらのデータへのアクセスは，それらと同じオブジェクト内のメソッドからのみ可能とする．（データ抽象化機能の形成）

（3）クラスとインスタンス

メソッドが同じで，データの内容が異なるオブジェクトを効率よく作成するために，もとになるオブジェクトを型として，その実体を複数個生成（複製）可能とする．このとき，型となるものをクラス，生成されたものをインスタンスと呼ぶ．（インスタンス生成機能の形成）

（4）クラスメソッドとインスタンスメソッド

メソッドにはクラス用のものとインスタンス用のものがあるので，それぞれクラスメソッドとインスタンスメソッドに分離する．

（5）クラス変数とインスタンス変数

クラスメソッドの操作対象となるデータとインスタンスメソッドの操作対象となるデータを分離し，クラス変数およびインスタンス変数とする．

（6）クラス階層と継承機能

メソッドの集合が少しづつ異なるオブジェクトを効率よく作成するために，クラス階層を導入する．そして，下位のクラスは上位のクラスの性質，即ち，メソッドと関連データを継承できるものとする．（クラス階層と継承機能の形成）

このような言語モデルの概念形成過程において，前のものほどオブジェクト指向概念として本質的な機能と考えられる．ただし，一般には，このような機能に基づく利点はユーザ視点で決まるものであり，その実現のために必要な機能が備わっていれば，すべての機能がなくても「オブジェクト指向」という用語が用いられている．

4．モデリングプロセスの形式化

4.1 ドメインモデルの構築手順

このようなオブジェクト指向の概念の発生学的定義は，もともとオブジェクト指向言語設計時に計算モデルとして導入したものであるが，分散協調型モデルを最初に規定するこの定義順序はモデリングプロセスに対応する[7]．そこで，本研究では，この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2に示す．

分析フェーズで構築されるオブジェクトベースのドメインモデルを以下のように OAM (Object-base 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]へ送信される何種類かのメッセージのうちのn番目の種類のメッセージである．

ここで，以下のような，あるメッセージm[i,j,n]の送信オブジェクトo[i]を求める関数senderとそのメッセージの受信オブジェクトo[j]を求める関数receiver

を導入する．

sender(m[i,j,n]) = o[i]

receiver(m[i,j,n]) = o[j]

ドメインモデルOAMに接する外界を仮想的に第0番目のオブジェクトo[0]とみなすことにより，外界からドメインモデルへのメッセージの集合Minとドメインモデルから外界へのメッセージの集合Moutは，これらの関数を用いて次のように表現できる．

Min = {m | sender(m) = o[0], m∈M}

Mout = {m | receiver(m) = o[0], m∈M}

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m[i,j,n]を受信した後，メッセージの列 m[j,k1,n1], m[j,k2,n2], ...を送信するという関係を次のようなメッセージ変換，

t[r] : m[i,j,n] → {m[j,k1,n1], m[j,k2,n2], ...}

として表現することにすると，

T = {t[r]}

となる．

このメッセージ変換の集合Tは，システムのふるまいの集合であり，Jorgensen と Erickson [20]がオブジェクト指向の統合テストフェーズのために導入したMM-Path (Method/Message path) とASF (Atomic System Function)という概念を包含している．MM-Pathとは，メッセージによって関連付けられるメソッド実行の列である．ASFとは，ある入力イベントに引き続いてMM-Pathの集合が逐次実行され，出力イベントで終了するような入力イベントを意味する．我々は，分析フェーズで同様の概念を用いるが，逐次処理のみならず並行処理も扱う．

要するに，ドメインモデルの構築では，図2に示すように，最初にMinとMoutを決定し，次にメッセージの流れに基づいてO，M，Tを決定する．

4.2 設計モデルの構築手順

このドメインモデルは，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．

（1）オブジェクトの外部仕様

各々のオブジェクトの外部仕様は，そのオブジェクトが受信するメッセージに対応するメソッドの集合で規定できる．今，あるオブジェクトo[j]が受信するメッセージの集合をM(o[j])とすると，次に示すように，M(o[j])はMの部分集合である．

M(o[j]) ⊂ M

そこで，オブジェクトo[j]のメソッドの集合をMD(o[j])とすると；これは以下の手順で求める．

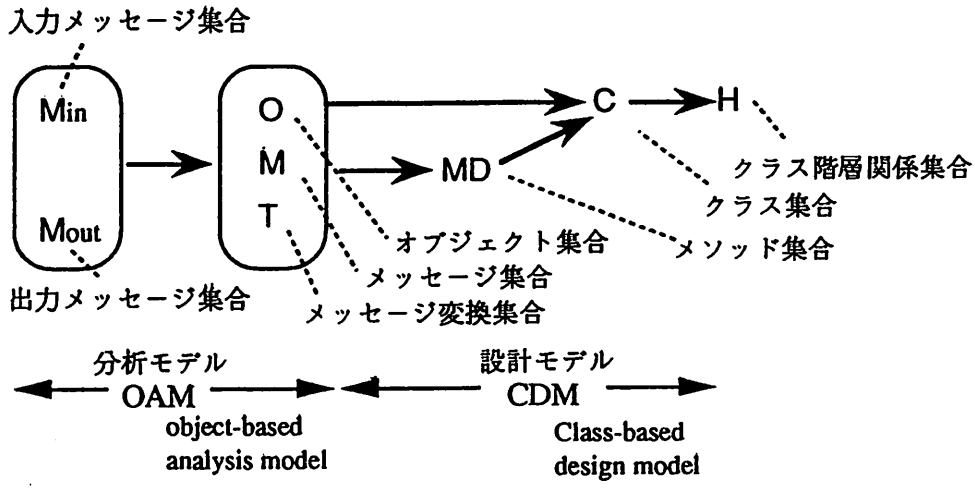

図2 オブジェクト指向概念の発生学的定義に基づくモデリングプロセス

＜1＞M(o[j])をサブセットの集合に分割する．そのとき，一つのサブセットに含まれるメッセージはお互いに機能的に等価となるように分割する．
＜2＞次に，各々のサブセットをMD(o[j])の一つのメソッドに対応させるようにMD(o[j])を決定する．

結局，オブジェクトo[j]は，この等価集合の数に等しいメソッドを有することになる．そして，設計モデルCDMのメソッド集合MDは次のようになる．

$$MD = \cup_j MD(o[j]).$$

（2）クラスの決定

このメソッドの集合がお互いに機能的に等しいようなオブジェクトは同一のクラスから生成できることから，クラスの集合Cは以下の手順で求められる．
＜1＞オブジェクト集合Oをそのサブセットの集合に分割する．そのとき，一つのサブセットに含まれるオブジェクトはお互いに機能的に等価なメソッド集合を持つように分割する．
＜2＞次に，各々のサブセットをCの中の一つのクラスに対応させるようにクラス集合Cを決定する．

即ち，n番目のクラスc[n]から生成されるオブジェクトo[i]からその基のクラスc[n]を求める関数classを，

$$c[n] = class(o[i])$$

とすると，

$$class(o[i]) = class(o[j]) \text{ if } MD(o[i]) = MD(o[j])$$

となるように等価集合を作る．

（3）クラス階層

お互いにメソッドの集合が類似のクラスは，継承機能を伴ったクラス階層を構成しうる．今，クラスc[i]のメソッド集合をMD(c[i])とすると，クラスc[i]とクラスc[j]の間に次のような階層関係が導入される．

$$h[r] : c[i] \rightarrow c[j] \text{ if } MD(c[i]) \subset MD(c[j])$$

即ち，MD(c[i])がMD(c[j])の真の部分集合ならば，次のような操作により，c[i]はc[j]のスーパクラスになりえる．

$$MD(c[j])/new = MD(c[j])/old - MD(c[i])$$

また，MD(c[i])とMD(c[j])が真の共通部分集合を持つならば，この共通の部分集合に対応する新しいクラスc[k]を導入することにより，次のようにc[i]とc[j]に共通のスーパクラスとすることができる．

$$h[s] : c[k] \rightarrow c[i]$$
$$h[t] : c[k] \rightarrow c[j]$$

この時，以下の操作が行われる．

$$MD(c[k]) = MD(c[i])/old \cap MD(c[j])/old$$
$$MD(c[i])/new = MD(c[i])/old - MD(c[k])$$
$$MD(c[j])/new = MD(c[j])/old - MD(c[k])$$

このような操作により，以下のようなクラスの階層関係の集合Hが求まる．

$$H = \{h[i]\} = \cup_i h[i].$$

結局，設計モデルの構築では，図2に示すように，オブジェクト指向概念の発生学的定義に基づいてMD，C，Hを決定する．

5．アプリケーション開発プロセスの例

5.1 概要

新分野に多い分散型アプリケーションソフトウエアは計算モデルもデータモデルも確立していないことが多いので，個々のオブジェクトの詳細な定義に先だって，オブジェクト間の関係を定義する必要がある．さらに，このような分散システムのモデル化にはメッセージフローが本質的なので，オブジェクト間の静的な関係に先だってメッセージの送受信によって規定されるオブジェクト間の動的な関係，即ち，システムのふるまいを定義する必要がある．従ってこのような分野には上記の開発プロセスが適していると考えられる．

そこで，会議開催システムOOO (the object-oriented office system) を例題として，実際的な開発手順を説明する．この例題OOO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と考えてよい．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予定者に送り，出欠の返事を返してもらうという作業あったとして，この業務を自動化するアプリケーションの開発を想定する．

5.2 メタファーベースのモデリング

オフィスの日常的業務（ルーチンワーク）は，

メッセージ駆動型の分散協調型モデルを用いて次のように表現できる．

（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．

マクロレベルでの協調的ふるまいの概要を，図3に示す．本研究では，以下に述べるように，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，メタファーベースのモデル化とはオブジェクトベースまたはインスタンスベースのモデル化であると位置づける．

（1）1：1の擬人化

実世界で一人に一つの業務が割り当てられていれば，ドメインモデルでもその人に一つのオブジェクトを対応させ，その業務を割り当てる．

（2）m：1の擬人化

実世界でm人のグループに一つの業務が割り当てられていれば，ドメインモデルでもそのグループ全体に一つのオブジェクトを対応させ，その業務を割り当てる．

（3）1：nの擬人化

実世界で一人にn種類の業務が割り当てられていれば，ドメインモデルでは，あたかもn人の各々にn種類の業務のどれかが割り当てられていたかのように，そのn種類の業務に応じてn個のオブジェクトを対応させ，各々の業務を割り当てる．

（4）m：nの擬人化

実世界でm人のグループにn種類の業務が割り当てられていた場合も，ドメインモデルでは，あたかもn人の各々にn種類の業務のどれかが割り当てられていたかのように，そのn種類の業務に応じてn個のオブジェクトを対応させ，各々の業務を割り当てる．

例えば図3では，個人に対応するオブジェクトは(1)の例である．事務局に対応するオブジェクトは，実世界での事務局がグループだとすると(2)の例である．会議室予約と備品予約に関しては，実世界で一人がその両方の業務を担当していたとすれば，(3)の例となり，グループで担当していれば(4)の例になる．もちろん，実世界と同じように一つのオブジェクトに両方の業務を割り当てること

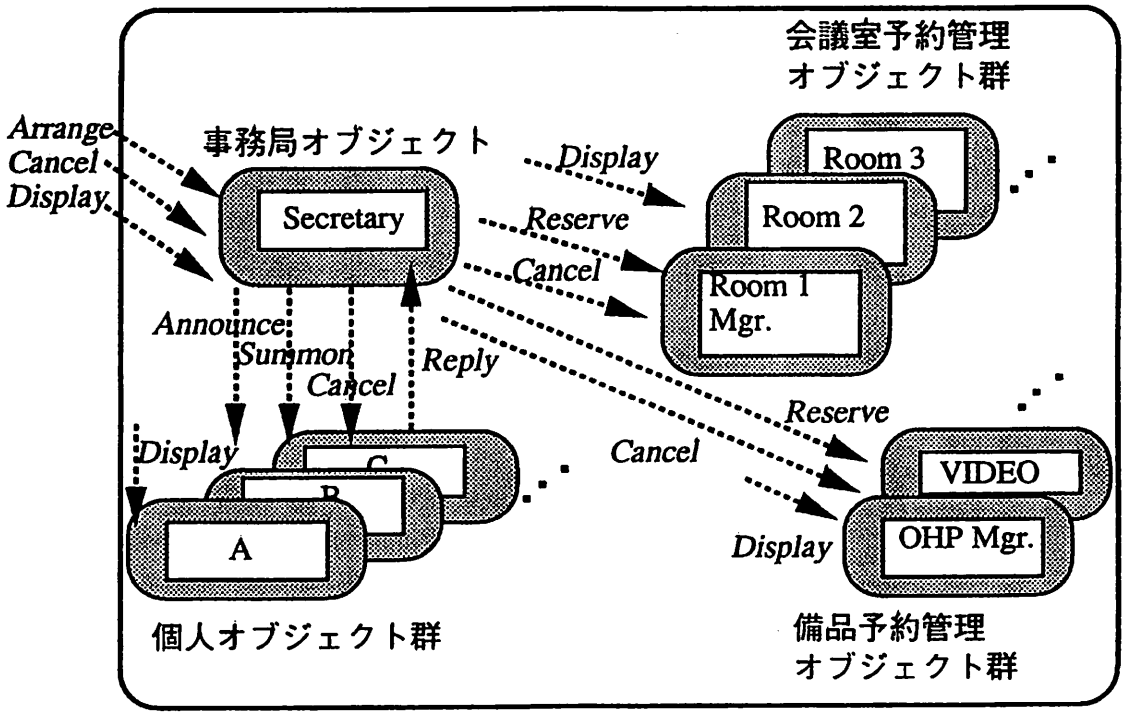

図3 分散オフィスシステムOOOのドメインモデルの概略図

も可能であるが，本報告では，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．

5.3 クラスに基づく設計

上記のインスタンスベースの分析モデルを基にしてクラスベースの設計モデルを構築した．OOOでは，以下の3種類のクラスを定義した．

（1） 事務局オブジェクトのためのクラス
（2）会議室と備品の予約管理オブジェクトにためのクラス
（3）個人オブジェクトのためのクラス

この時点では，クラスの階層化はなされていないが，予約管理オブジェクトと個人オブジェクトにはメソッドに類似性がある．そこで，予約管理オブジェクトのreserveメソッドと個人オブジェクトのannounceメソッドの機能が同じであることに注目し，announceというメソッド名をreserveに変更することにより，個人オブジェクトのクラスを予約管理オブジェクトのクラスのサブクラスとするような階層化が導入された．即ち，図4に示すように，両方のクラスはメソッドの部分集合{reserve, cancel, display}を共有し，個人オブジェクトのクラスは，さらに独自のメソッドsummonを有する．

今回のプロトプログラムでは，ミクロレベルのクラスのインプリメンテーションはC++を用いて行ったが，本研究で対象とするエンドユーザにはC++は複雑すぎる言語であり，もっと高水準でドメインモデルや設計モデルを素直に記述できる言語が必要である．

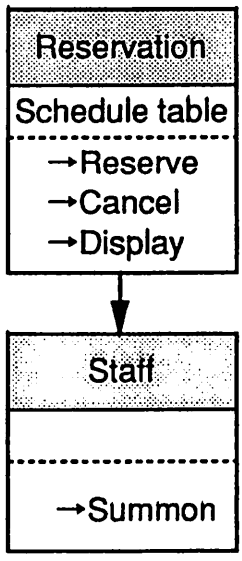

図4 クラス階層の例

5.4 結果の検討

（1）マクロモデル vs. ミクロモデル

最近のソフトウエアの複雑性を克服するために，OOA／OODの分野でいくつかの技法が提案されている．例えば，Cordのサブジェクトという概念によるグループ化，Wirfs-Brock [28] のサブシステムという概念による分割，Jalote [19] のオブジェクトのネスト化，Boochのクラス構造とオブジェクト構造のcanonical formなどである．これらのアプローチは基本的には分割統治の原則に沿ったものであり，大規模ソフトウエアの開発に向いている．一方，本研究で対象とする分散協調型のシステムでは，比較的小規模であるが，動的なふるまいが複雑なものが多い．このような分野では2階層モデルが有効と思われる．

この2階層モデルのマクロモデルにおいて，インスタンスのお互いのふるまいに注目し，クラスの概念は用いないという手順は，クラスの概念を持たないprototype-based languageでプログラムを開発するclass-freee programming [27] と発想は似ている．しかしながら，2階層モデルでは，最終的にはプロトタイププログラムではなく，実用プログラムの開発が目的なので，ミクロレベルでクラスを用いてモデルの厳密な定義を行う．

本報告では研究の主対象であるOA分野の非定形業務の典型的な例を取り上げ，マクロモデルの構築の重要性を示した．特に最初にシステム全体の動的ふるまいを明確にすることが，適切なオブジェクトの抽出に不可欠である．

例えば，従来のデータモデルをベースにしたオブジェクト指向設計技法のように，最初の段階で，実世界（問題領域）に存在するものを「会議室があるから」，「黒板があるから」と次々とオブジェクトにする方法でうまくいかないことは明白であろう．データに注目して「会議室予約ノート」をカプセル化するという方法もプログラミング段階でのデータ抽象化の手法であり，メソッドがデ

ータ操作手続きとして設定されてしまう．

本方式では，システム全体の動的ふるまいの中でのものの役割（機能）に注目する．例題では，予約管理という役割から予約，取消，照会というメソッドを抽出することにより，最終的に会議室予約管理オブジェクトと備品予約管理オブジェクトが同一のクラスから生成可能であることを導いた．さらに個人オブジェクトがその予約管理クラスの特殊化によって得られることも導いた．このようなクラス階層の導出はデータのカプセル化の観点では難しい．

（2）機能指向プロセス vs. データ指向プロセス

現在提案されているＯＯＡ／ＯＯＤ技法では，Rumbaugh のオブジェクトモデルやShlaer の情報モデルのように，最初に実世界あるいは問題領域（ドメイン）からオブジェクトやクラスを抽出するものが多く，オブジェクト指向は，従来の構造化分析などの機能指向アプローチに対抗したデータ指向アプローチに近いととらええられがちである．しかし，分散協調型のシステムでは，マクロレベルでのふるまいの記述が重要であり，オブジェクト指向は機能指向とデータ指向の対立的に止揚されたものと考えられる．即ち，オブジェクトの役割をメソッドの集合として規定する観点では機能指向，オブジェクトをデータのカプセル化とみればデータ指向といえる．

（3）オブジェクトおよびメソッドのメソッドの抽出

実世界からのオブジェクトやメソッドの抽出方法として，問題の仕様から名詞をオブジェクトに対応させ，動詞をメソッドに対応させるものがある．このような方法は，金融システムのように既存のシステムが存在し，データモデルが定義されている場合には，適用しやすい．ただ一般的方法としては無駄が多く，特に分散協調型のオフィスシステムなどにはなじまない．

（4）プロトタイプ方式による拡張性

エンドユーザコンピューティングのためのオフィスシステムのように新規開発が主体の非定形業務は，最初に要求仕様を厳密に規定できないため，プロトタイピング方式の開発にならざるを得ない．このような場合，システムのふるまいが重要なので，既存の方法論とは逆にオブジェクト間の静的関係に先立って動的な関係を定義する必要がある．

そして，プロトタイプの開発のなかで連続的な拡張をしていく必要がある．この場合，変更内容に応じて修正範囲はいろいろであるが，総じて修正個所はわかりやすい．

6．おわりに

まず初年度（平成６年度）には，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，極力プログラミングの概念を排した新しいパラダイムが必要であるという観点から，このような”作りやすさ”と”使いやすさ”を両立させるために，

・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）

という基本コンセプトを設定した．そして，その実現の第１段階として，非定型業務のモデル化をマクロレベルとミクロレベルの２階層に分けて行う方式とし，マクロレベルでは，既存のオブジェクト指向開発技法のようなオブジェクトの間の静的な関係を最初に定義するボトムアップ法ではなく，オブジェクト指向概念を発展させた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最初に構築するトップダウン法を規定した．その具体的な開発手順を形式化し，その例題を示した．

次年度（平成７年度）は，これらの技術をモデリング＆シミュレーション機能として実現するアプリケーションフレームワークの設計を行い，その基本技術を実現するプロトタイプを開発する．

参考文献

1) Booch,G. : Object-Oriented Design with Applications, Benjamin/Cummings (1991).
2) 中所,芳賀:オブジェクト指向型言語と論理型言語の融合方式に関する考察,オブジェクト指向,共立

出版,pp.133-146(1985).
3) 中所,増位,芳賀,吉浦:マルチパラダイム型言語における対立概念の融合方式,人工知能学会誌, 4, 1,77-87(1989).
4) 中所武司:使いやすいソフトウエアと作りやすいソフトウエアーオブジェクト指向概念とその応用ー,電気学会雑誌,Vol.110,No.6,465-472(1990).
5) 中所武司:エンドユーザコンピューティングーソフトウエア危機回避のシナリオー,情報処理, Vol.32,No.8,pp.950-960(1991).
6) 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、啓学出版(1992).
7) 中所：オブジェクト指向概念の発生学的定義に基づくソフトウェア設計技法, 情報処理学会ソフトウェア工学研究会資料, 93-SE-95, 95-7, 1-8, 1993.
8) 中所：wwHww：分散オフィスシステムのためのエンドユーザコンピューティング向きオブジェクト指向モデル、情報処理学会ソフトウェア工学研究会資料, 94-SE-97, 97-5 (Mar.1994)
9) 中所：ＣＳ－ｌｉｆｅ,コンピュータソフトウエア, 11, 6, 1-2 (Nov. 1994).
10) Chusho,T. and H.Haga,H. : A Multilingual Modular Programming System for Describing Knowledge Information Processing Systems, Proc. the 10th World Computer Congress IFIP'86, pp.903-908 (1986).
11) Chusho,T. : What Makes Software Tools Successful?, IEEE Software, Vol.10, No.5, pp.63-65(1993).
12) T. Chusho : Modeling of Application Software for End-User Computing as "a Domain Model ≡ a Computation Model," Technical Report of, Dept. of Computer Science, Meiji University (Sep. 1994).
13) Coad,P. and Yourdon,E. : Object-Oriented Design, Prentice Hall (1991).
14) Dahl,O. and Hoare,C.A. : Hierarchical Program Structures, Structured Programming, Academic Press, pp.175-220 (1972).
15) Fichman,R.G. and Kemerer,C.F. : Object-Oriented and Conventional Analysis and Design Methodologies, IEEE Computer, Vol.25, No.10, pp.22-39 (1992).
16) Goldberg,A. and Robson,D. : Smalltalk-80 : The Language and its Implementation, Addison Wesley (1983).
17) Grudin, J. : Computer-Supported Cooperative Work: History and Focus, IEEE Computer, Vol.27, No.5, pp.19-26(1994).
18) Hewitt,C. and Baker,H. : Laws for Communicating Parallel Processes, Proc. the 7th World Computer Congress IFIP'77, pp.987-992 (1977).
19) Jalote,P. : Functional Refinement and Nested Objects for Object-Oriented Design, IEEE Trans. Softw. Eng., Vol.SE-15, No.3, pp.264-270 (1989).
20) Jorgensen, P. C. and Erickson, C. : Object-Oriented Integration Testing, Comm. ACM, Vol.37, No.9, pp.30-38(1994).
21) Liskov,B. and Zills,S. : Programming with Abstract Data Types, ACM SIGPLAN Notices, Vol.9, No.4, pp.50-59 (1974).
22) Maes, P. : Agents that Reduce Work and Information Overload, Comm. ACM, Vol.37, No.7, pp.30-40(1994).
23) Malone, T., Lai, K. and Fry, C. : Experiments with Oval : A Radically Tailorble Tool for Cooperative Work, Proc. CSCW92, pp.289-297 (1992).
24) Monarchi,D.E. and Puhr,G.I. : A Research Typology for Object-Oriented Analysis and Design, Comm. ACM, Vol.35, No.9, pp.35-47 (1992).
25) Rumbaugh, J. et al. : Object-Oriented Modeling and Design, Prentice Hall (1991).
26) Shlaer,S. and Mellor,S.J. : Object-Oriented Systems Analysis : Modeling the World in Data, Prentice Hall (1988).
27) Smith, R. B. (Moderator) : Prototype-Based Languages:Object Lessons from Class-Free Programming (Panel) : Proc. OOPSLA'94, pp.102-112(1994).
28) Wirfs-Brock, R., Wilkerson, B. and Wiener, L. : Designing Object-Oriented Software, Prentice Hall (1990).