ソフトウェア工学 90−14
（1993. 2. 5）

ソフトウェア仕様化／設計法のデータベースの試作

篠原 正紀　　井口 和久　　佐伯 元司

東京工業大学 工学部

あらまし

複雑なシステムを仕様化する際に種々の設計法を用いて複数の観点から仕様化したり，チームで設計作業を行う際に各メンバーが異なる設計法を用いることがある．このような状況を支援するためのツールには，種々の設計法自身をデータとして蓄積し，かつ生成されたプロダクトを統合する機能が不可欠である．本稿では，設計法を統一的に形式化する手法を提案し，その手法に基づいて作業者をガイドしたり，プロダクトや作業履歴を管理するツールについて述べる．設計法固有の種々の図表現を扱うために，設計法，プロダクト作業履歴を管理するサーバー部と，記法を含めたユーザーインターフェースを扱うクライアント部に分けて実現した．

Prototype of a Databese System for Software Specification & Design Methods

Masanori Shinohara　Kazuhisa Iguchi　Motoshi Saeki

Faculty of Engineering, Tokyo Institute of Technology

Abstract

To specify a complex system from multiple view points, and to specify a system in collaborative work, we often use several software specification and design method in developing its software specifcation. Under this situation, we need a supporting tool which can hold various kinds of design methods as data and which integrate the products developed by the methods into one. This paper proposes the technique to formalize design methods, and discusses the tool for navigating designers according their selected methods. It also stores the products and the records of the activities produced throughout their design work. It cosists of a server part and client parts. The server is for managing design methods, produced products, and performed activities. The client part for a design method gives users a user-friendly interface such as graphical representation in the method.

1 まえがき

高品質のソフトウェアを開発するには、効果的にソフトウェアの仕様を設計することが重要である。なぜなら仕様や設計段階は、ソフトウェア開発過程の中では初期の段階に位置しているからである。ソフトウェア仕様の開発をサポートするために、JSD[1]、OOA[2]、SA[3] などの多くのすぐれた仕様化／設計法がすでに研究され、実用化されている。しかしこれらの設計法は、設計対象とするシステムの性質により有効である領域とそうでない領域が存在する。すべての領域に適した万能な設計法を創り出すのはとても困難である。システムを複数の側面から把え各面に適した設計法を選択して記述したりしても、また複数人による仕様化／設計作業環境下で各人が異なる設計法で作業しても最終的に1つの仕様に統合することができれば、より効率的にソフトウェアの仕様化／設計作業を進めることができる。こうした作業を支援するには、従来の支援ツールのように一つの設計法のみに従って設計作業を支援するだけでなく、複数の設計法を統一的に扱い設計法自身をデータとして蓄積するデータベースが必要である。このようなデータベースを含むツールをメソッドベース (Method Base)[4] と呼ぶ。メソッドベースに要求される機能として以下のものがある。

- ユーザは作業に最も適していると考えた設計法を、メソッドベースの中から選択することができる。
- さまざまな仕様で書かれた複数の仕様を、一つの仕様に統合することができる。
- システムはユーザに選択された設計法に従って、仕様の開発をガイドまたはナビゲートすることができる。つまりどのようなアクティビティを実行すべきかを、段階的にユーザに教えることができる。

本研究ではこのような要求を満たすメソッドベースの構築手法を考案し、プロトタイプを作成した。

2 設計法の形式化

複数の設計法を統一的にあつかうためには、設計法を共通の枠組で形式化する必要がある。この枠組の粒度が細かすぎると、それぞれの設計法の全ての概念が個別に存在することになり、Davis の Metamodel[5] のように大きくなり過ぎてしまう。逆に荒すぎると設計法を十分に表わすことができなくなってしまう。いろいろな設計法には共通した概念が存在し、例えば JSD での "Entity" は、OOA での "Object" に対応する。このように様々な設計法が共通に持っている概念を抽出し抽象化することにより、適用範囲が広くかつコンパクトな枠組が構成できる。また共通概念を使うことによって仕様の統合化が行い易くなる。メソッドベースでは個体関連図（以下 ER モデルと略す）を用い、生成物を時間的な振舞い・機能・物理的な構造の3つの視点から把え、共通概念を表す6つの Entity と 21 の Relationship によって枠組を作っている。この枠組のことをメタモデルと呼ぶ。これは設計法をデータとした際のデータ型の役割を果す。メタモデルに従って記述した設計法を、その設計法のオブジェクトモデルという。設計作業はこのオブジェクトモデルに従ってガイドされ、仕様が蓄積される (図 1)。

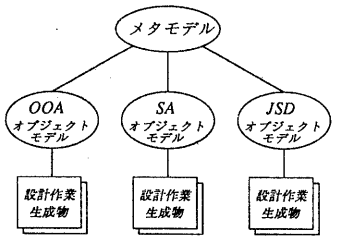

図 1: メソッドベースの論理的なデータ構造

2.1 メタモデル

メタモデルは、設計法による生成物の構造を形式化するためのプロダクトモデルと、設計法の作業手順などを形式化するためのアクティビティモデルの2つから構成され、共に ER モデルで記述されている。詳細は論文 [4] に述べられている。

1. メタモデルのプロダクトモデル

   設計法によって作成される仕様のデータ構造を表現したものである (図 2)。*State*, *Event* は時間的な振舞い、*Process*, *Data* は機能、*Object*, *Association* は物理的な構造を表わす設計法の概念要素を表すメタモデル上の概念（メタ概念）である。メタ概念間の関係（メタ関係）には5つの型がある。*Source/Destination* 関係は、あるインスタンスが他のインスタンスの source または destination であることを示し、*Has_a* 関係は親子などの所有関係、*Define* 関係はある概念が別の概念で定義されていることを示す。*Is_a* 関係は、継承などクラス特殊化の関係概念の一般的表現である。また *Next-to* 関係は、次状態や次に起こる event を表す。

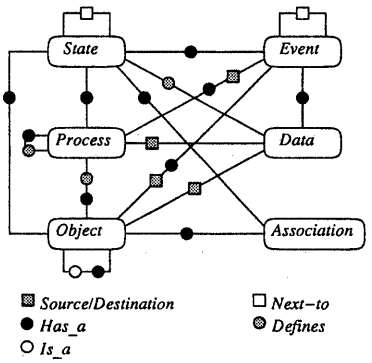

図 2: メタモデル：プロダクトモデル

2. メタモデルのアクティビティモデル

   設計作業の手順と、ある作業ステップにおける入力と出力を形式化するためのモデルである (図 3)。*Precede* が次の作業、*Has* は作業の階層関係、*Input/Output* はその作業での入出力プロダクトを表わす。

設計法を記述するための枠組には、Davis の Metamodel[5]、MetaEdit[6]、ViewPoint[7]、Brinkkemper のモデル [8]、クォーク [9] などがある。しかし [5]、[6]、[8] は、アクティビ

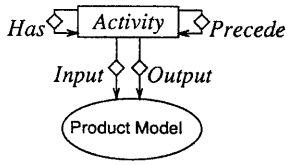

図 3: メタモデル：アクティビティモデル

ティに関するモデルを持っていない。また [7] は形式的な記述がされていない。

2.2 オブジェクトモデル

オブジェクトモデルは、設計法を ER モデルで形式化し、設計法の概念をメタモデルに用意されている共通概念と対応付けることで作られる。

ここで扱われるオブジェクトモデルは、以下の二つのモデルから構成されている。

1. オブジェクトモデルのプロダクトモデル

   設計法の中の生成物のデータ構造を、ER モデルで表現する。図 4は OOA のプロダクトモデルの例である。括弧の中の言葉は、設計法の概念に対応するメタモデルにおける概念を示している。例えば OOA における概念 Object と Subject は、メタモデルにおける共通概念 *Object* と対応付けられている。

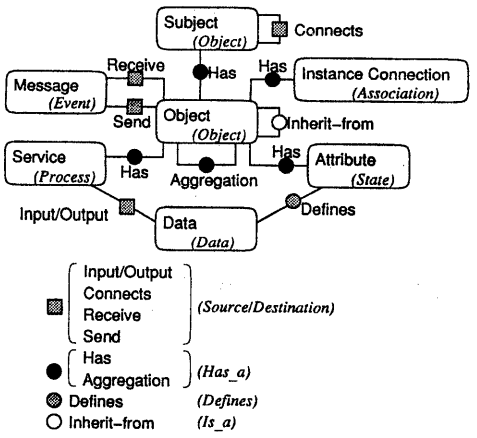

図 4: オブジェクトモデル：OOA のプロダクトモデル

2. オブジェクトモデルのアクティビティモデル

   設計法の中で行われる作業順序や、ある作業ステップで生成されるべきプロダクトを示すために、設計法の手続きを ER モデルで形式化したものである。実際の作業履歴は、この構造に従って蓄積される。図 5は OOA のアクティビティモデルの一部である。例えば Identify Structures のアクティビティでは、precede 関係で連結されている Identify Structures と Define Subjects が次に行うべきアクティビティであり、また Object を入力として、Inherit-form と Aggregation を出力することを示している。

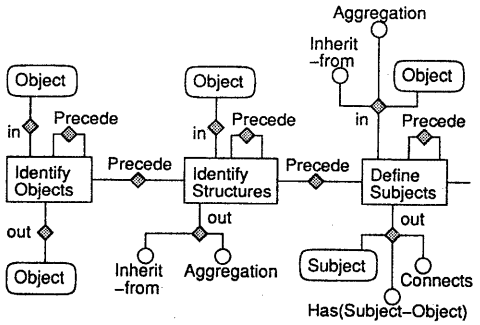

図 5: オブジェクトモデル：OOA のアクティビティモデルの一部

2.3 設計作業におけるデータの蓄積

設計作業では、オブジェクトモデルのアクティビティモデルの手順にしたがって作業を行い、オブジェクトモデルのプロダクトモデルに従った構造の生成物を作り出す。蓄積されるデータは、設計過程の生成物や作業の履歴である。図 6は OOA で記述された仕様の一部で、上図がオブジェクトモデルに従って記述された生成物、下図はそれに対応した OOA の図表現による記述である。

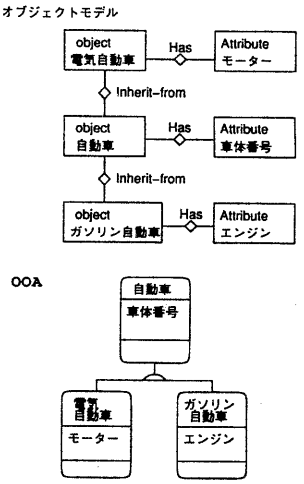

図 6: 設計作業における生成物の例

仕様の統合が可能となるための必要条件は、統合を行いたい仕様間に、名前が同じで、さらにメタモデルにおいて同じメタ概念または同じ型のメタ関係となる生成物が存在することである。例えば JSD で記述された仕様に"自動車"という Entity があり、また OOA で記述された仕様に"自動車"という Object があるとする。それらの生成物は、名前が同じでかつメタモデルにおいて同じメタ概念 *Object* となるので、ここを接点として 2 つの仕様は統合される可能性がある。

3 メソッドベースシステムの論理構造

メソッドベースは前述の要求を満たすために、次の3つの機構から成る。

1. 設計法を形式化し、データとして扱う機構
2. 蓄積された設計法に従って、ユーザをガイドし生成物を扱う機構
3. ユーザが実際に操作し、生成物を入力するインターフェイス

メタモデルに従って記述されたオブジェクトモデルの持つ問題点として、次のようなものが挙げられる。

1. 設計法にはそれぞれ固有の図・式などの表現があるが、それら表現法に関する情報を持っていない。例えば OOA の記法では属性やサービスの記述は、図6のように、属性やサービスを持つオブジェクトの図内に書くことになっている。
2. オブジェクトモデル中のインスタンス間の制約を記述していないため、設計法では矛盾のある生成物を生成できる。例えば、DFD のデータフローは名前の付いた1本の有向枝である。しかし DFD のプロダクトモデルに従うだけでは、図7のように矢印の向きが衝突するような記述が可能となる。元の DFD では表現上このようなフローを書くことはできない。

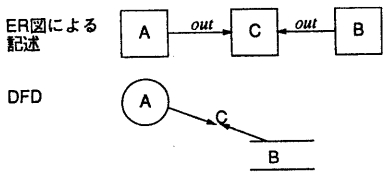

図 7: 矛盾が起こる記述の例

1の問題点は、各設計法固有の図表現情報を扱い、それらの図表現とオブジェクトモデルに従った記述の間とを、相互に変換するインターフェイスを用意することで解決する。

2の問題点はプロダクトモデルの問題点である。これを本質的に解決するにはオブジェクトモデルを拡張し、矛盾した記述が起こらないように制約を加える必要がある。ところがメソッドベースシステムを実働化する上で、プロダクトモデルを繁雑化するのは望ましくない。この問題点も、矛盾した表現をインターフェイスで制限することで解決できる。

以上の理由からメソッドベースシステムは、各設計法固有の情報を扱うインターフェイスを持つ。インターフェイスは本体から分離し、設計法や生成物などのデータを管理するメソッドベース本体と、インターフェイス部分の二つの部分を持ったアーキテクチャとして実現する (図8)。

4 メソッドベースの機能と構造

この章では、実用的なメソッドベースシステムの動作を示す。

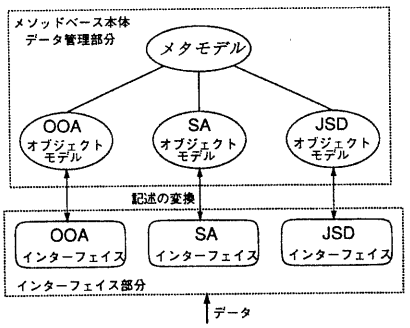

図 8: メソッドベースの物理的構造

4.1 Main Window

メソッドベースを起動すると最初に立ち上がるのが Main Window である (図9)。Main Window は設計法や仕様の追加、削除を行うコマンドメニューと、設計法・仕様・部分仕様の関係を示した Overview Window から成る。ここで部分仕様とは、ある一つの設計法で記述された仕様の一部のことである。Main Window 上で行う操作は以下のものがある。

1. 設計法の追加、削除
2. 仕様の追加、削除
3. 部分仕様の作成、削除、編集を行うエディタの起動

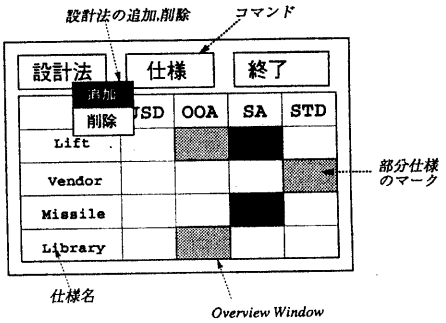

図 9: Main Window の表示例

Overview Window は、仕様がどの設計法によって記述されているかを表示する。マトリックスのそれぞれの行が一つの仕様の構造を示し、最左の項目に仕様の名前が表示される。また仕様を作成するために使われている設計法の項目にはマークが付けられ、その2段階の濃度によって、部分仕様の設計の進捗状況を知ることができる。例えば図9では、Lift の仕様が OOA と SA を用いて記述され、SA による部分仕様が継続中、OOA による部分仕様が完了していることを示している。この進捗状況の設定はユーザが行なう。

4.2 設計法の追加、削除

設計法の追加と削除は、Main Window のコマンドメニュー選択により行う。

- 設計法の追加

  ER モデルで記述された、設計法のオブジェクトモデルのデータを与えることによって設計法を登録する。設計法を追加するコマンドを選択すると、設計法のデータの入力指示が出る。指定した設計法のデータを読み込みデータに誤りがなければ、設計法は登録され、Overview Window のマトリックスに新しい列と設計法の名前が追加される。

- 設計法の削除

  設計法を削除するコマンドを選択した後、削除する設計法の名前を指定する。設計法の名前を指定し、その設計法のデータが存在するならば、Overview Window のマトリックスから設計法の名前とその列が削除される。ただし、その設計法によって記述された仕様が存在する場合には削除できない。

4.3 仕様の追加、削除

仕様の追加と削除は、Main Window のコマンドメニュー選択により行う。仕様の編集は、部分仕様を編集することによって行われる。

- 仕様の追加

  新しい仕様を追加するコマンドを選択した後、仕様名の入力指示が出る。それに従って仕様名を入力すると、Overview Window のマトリックスには新しい行と仕様名が追加される。

- 仕様の削除

  仕様を削除するコマンドを選択すると、削除する仕様の名前を指定するように指示が出る。指定した仕様が存在するならば、Overview Window のマトリックスから仕様の名前とその行が削除される。

4.4 部分仕様の追加、削除、編集

部分仕様の追加、削除、編集を行うには、まず Overview Window のマトリックス内の項目を選択することで、追加・削除・編集を行ないたい部分仕様を指定する (図 10)。追加の場合は白紙の、削除・編集の場合は現在のデータを表示したエディタが起動され、エディタ上で図表現の操作が可能となる。エディタ上で削除のコマンドを指定すると、Overview Window 上のマークは消え、エディタは強制的に終了する。

4.5 Edit Window

Edit Window(図 11) は、部分仕様の新規作成、編集、削除を行うために起動される。ユーザが実際に生成物を入力するためのエディタであり、図表現の部品を選択するコマンド、入力画面となる Draw Window、入力手順のガイドを行う Guide Window から構成される。

4.5.1 Draw Window

図表現の記述は Draw Window 上で行う。扱う部品の数や種類、その操作は設計法によって異なる。部品を選択するコマンドにより扱う部品を決定し、それらを配置、移動、リサイズ、削除することによって設計法固有の記述を作成する。上記の配置・移動・リサイズ・削除は、すべて

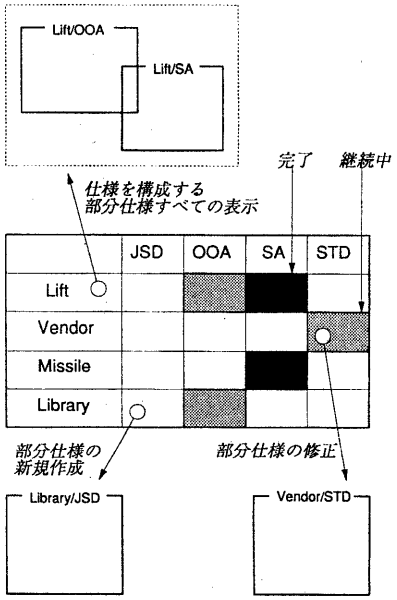

図 10: Overview Window の操作例

共通の操作である。部品の選択、移動の位置やリサイズのスケールの指定にはマウスクリックなどを用いる。

ブラウジングは Edit Window の入力画面上で行う (図 12)。生成物を表示すると、複数の設計法にまたがって記述されている生成物にはマークが表示される。その生成物の表示を選択すると、その生成物と対応関係を持つ部分仕様の Edit Window が起動される。図 12は図書館の仕様の例であり、データの流れを SA で、物理的な構造を OOA で記述している。この例では出版社、人、書庫が対応関係 (点線矢印) を持つ生成物で、それらに網掛けのマークが付けられている。

4.5.2 Guide Window

Guide Window(図 13) では、作業中の設計法のアクティビティモデルを見やすい形式で表示し、設計作業の入力手順をガイドする。ただし、ガイドによる入力手順はユーザの作業を束縛しないので、ユーザは自由な入力を行うこともできる。

アクティビティは、図やテキストの編集作業と対応付けられている。図やテキストの編集を行うと、それがどのアクティビティに該当するかが識別され、必要ならば Guide Window の表示を変更する。例えば図 13の例では、Identify Structures の表示が楕円のマークで囲まれ、現在作業中のアクティビティであることを示している。さらに Identify Structures と Define Subject の表示を反転し次に行うべきアクティビティであることを知らせ、右側にはその出力プロダクトを表示している。また Identify Object の表示を選択し、その出力プロダクトを問い合わせることもできる。

5 メソッドベースの試作

我々はメソッドベースシステムのプロトタイプを、UNIX ワークステーション上のサーバ・クライアントシステムを

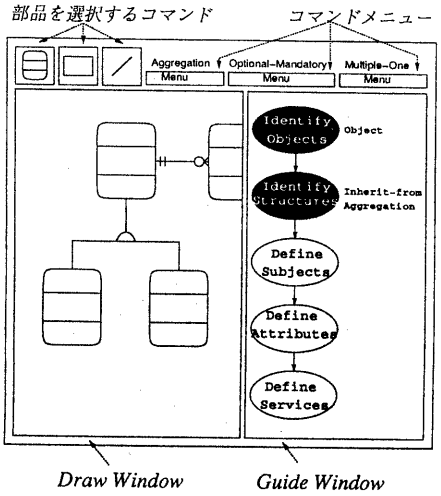

図 11: Edit Window の例

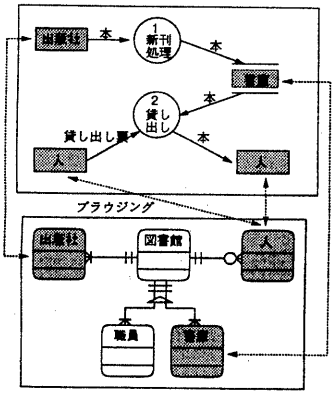

図 12: 複数の Edit Window の表示例

用いて実現した。サーバ・クライアントモデルを用いたのは、3 章で述べたように、データ管理部分とインターフェイス部分とを分離して実現するためである。

5.1 サーバ

サーバ部分は、3 章で述べたメソッドベースの本体、データ管理部分に対応する。蓄積された設計法の管理と、設計法に従ったユーザのナビゲートを行い、また作成された生成物や、実行されたアクティビティの過程を記録する。

5.2 クライアント

クライアントは、3 章で述べたインターフェイス部分に対応し、次の 2 種類から成る。

- 起動用クライアント

  ツールを起動すると最初に立ち上がるクライアントで、4 章における Main Window を実現している。このクライアントでは、設計法の操作、仕様の操作、各

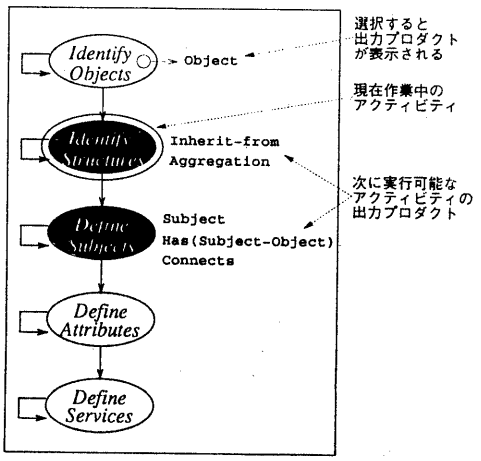

図 13: Guide Window の表示例

設計法の専用クライアントの起動を行うことができる (図 15)。

- 設計法用クライアント

  設計法に適したユーザインターフェイスを実現するクライアントで、個々の設計法に対して専用に設計される。4 章における Edit Window を実現している。例えば OOA のためのクライアントは OOA 図の形式で生成物を表示し、また SA のためのクライアントは、蓄積された生成物からデータフロー図などを表示する。

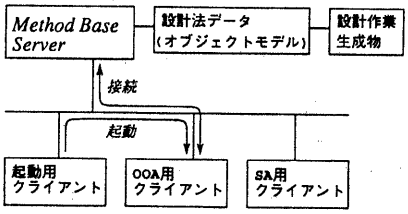

図 14: 試作したメソッドベースシステム

5.2.1 試作

本研究ではサーバ、起動用クライアント、OOA 専用クライアントの試作を行った。図 15は OOA 専用クライアントを実行した例である。OOA の記述は Coad の Object-oriented Analysis[2] に準じている。

OOA 専用クライアントには前述の Edit Window に加え、Subject Window が付属している。これは OOA 専用クライアント固有のウィンドウであり、サブジェクトの構造を表示したり、サブジェクトの関係を入力したりするのに使用する。Subject Window は、Edit Window 上のコマンドで自由に開閉することができる。

OOA 専用クライアントの部品選択コマンドでは、オブジェクトを表す部品・サブジェクトを表す部品・部品間を連結する線の 3 種類から選択して、部品を配置するように

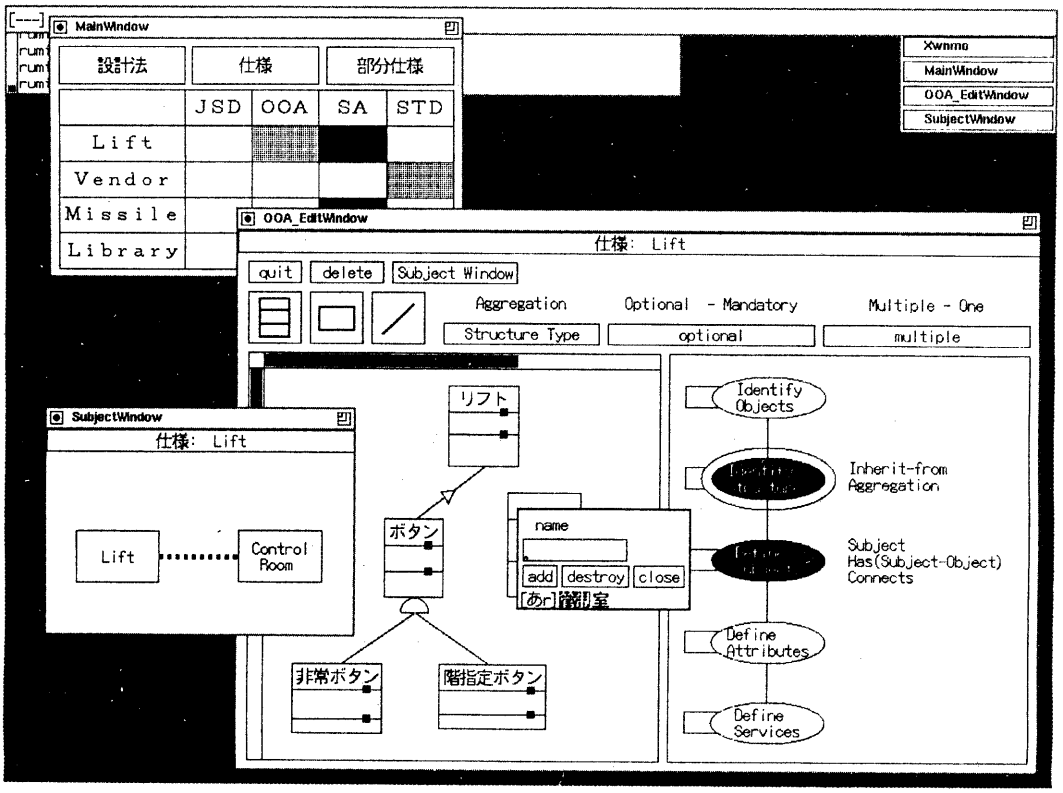

図 15: 試作した OOA 専用クライアントの表示例

なっている。以下ではオブジェクトを表す部品をオブジェクトボックス、サブジェクトを表す部品をサブジェクトボックスと呼ぶ。

Guide Window には OOA のアクティビティモデルが表示される。以下では OOA の作業手順に従って、そこで扱われる記述について述べる。

1. IDENTIFYING OBJECT（オブジェクトの識別）

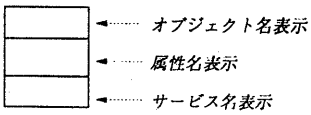

図 16: オブジェクトボックス

部品選択コマンドでオブジェクトボックスのアイコンを選択し、メイン入力画面である Draw Window 上で位置を指定して配置する。配置されたオブジェクトボックスの上段をクリックすると、テキスト入力が可能なウィンドウがポップアップし、その中で名前の設定、修正、消去の操作を行うことができる。

2. IDENTIFYING STRUCTURE（クラス構造の識別）

以下の操作でオブジェクトを関係を表す線で連結し、これによって仕様の構造化を行う。

1. 部品選択コマンドで線のアイコンを選択する。
2. source と destination のオブジェクトを指定する。

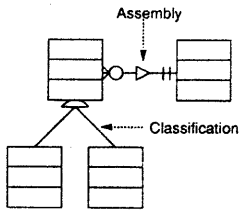

図 17: 構造化の例

オブジェクトの関係の種類は、コマンドメニューによって指定する。構造化のタイプは classification か aggregation かを選択する。オブジェクトの対応関係を指定するコマンドは 2 つあり、それぞれ single と multiple についての関係と、optional と mandatory についての関係を指定する。

連結しているオブジェクトが削除された場合は、これらの構造化に関する記述も同時に削除される。

3. IDENTIFYING SUBJECT（モジュールによるクラスのグループ化）

サブジェクトの配置は以下の手順で行う (図 18)。

1. 部品選択コマンドでサブジェクトボックスのアイコンを選択する。

2. Draw Window 上でオブジェクトをグルーピングする。

3. Subject Window が開いている状態で、Subject Window 上でサブジェクトボックスを配置する位置を指定する。

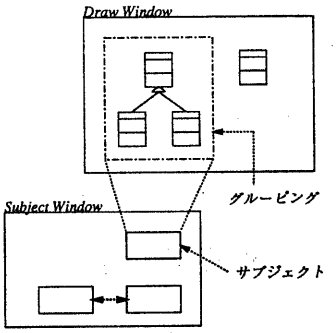

図 18: サブジェクトの作成例

部品選択コマンドで線のアイコンを選択し、Subject Window 上で source と destination のサブジェクトを指定することによって、サブジェクト間にメッセージ連結の線が引かれる。

サブジェクトボックスをクリックすると、テキスト入力が可能なウィンドウがポップアップし、名前と番号の書き込み・修正・消去の操作を行うことができる。またそのウィンドウ中のコマンド選択によって、サブジェクトに含まれるオブジェクトのリストを表示することができる。

4. IDENTIFYING ATTRIBUTE（属性の識別）

配置されたオブジェクトボックスの中段（図16の属性名表示部分）をクリックすると、テキスト入力が可能なウィンドウがポップアップし、属性名の入力・追加を行うことができる。また書き込まれた属性名の表示をクリックすると、属性名の修正・削除を行うウィンドウがポップアップする。

5. IDENTIFYING SERVICE（サービスの識別）

配置されたオブジェクトボックスの下段（図16のサービス名表示部分）をクリックすると、テキスト入力が可能なウィンドウがポップアップし、サービス名の入力・追加を行うことができる。また書き込まれたサービス名の表示をクリックすると、サービス名の修正・削除を行うウィンドウがポップアップする。

6 おわりに

本稿では，種々の設計法を統一的に蓄積し，ユーザが選択した設計法に従って作業を支援するツールについて述べた．今後の課題として，以下のようなことが挙げられる．

1. 種々の設計法に対応したクライアントの実現を行ない，仕様の統合機能についての評価を行なう．クライアントの実現を効率的に行なうために，既存の Graphical User Inteface 作成ツールの使用や，設計法の図表現記述言語の開発なども考えられる．

2. 本システムは，プロダクト（中間生成物も含む）を蓄積するだけでなく，作業履歴も蓄積している．この作業履歴を用いた設計作業の支援方式，例えば仕様変更があった際のプロダクトの変更作業の支援方式などを開発する．

3. グループで作業を行なう際の通信機能や，作業分担，作業計画の管理機能の実現を行なう．ある作業者がプロダクトを変更した際に，影響が及ぶ作業者に対して電子メールで告知したり，作業分担に従ってプロダクトを自動的に配布したりするような機能が必要である．また，メンバによるプロダクトのレビュー作業の支援も必要である．

参考文献

[1] M.A. Jackson. *System Development.* Prentice-Hall, 1983.

[2] P. Coad and E. Yourdon. *Object-Oriented Analysis.* Prentice Hall, 1990.

[3] T.DeMarco. Structured analysis and system specification. *Yourdon Press*, 1978.

[4] 郭文音, 佐伯元司. ソフトウェア仕様化／設計法のデータベース化について. 情報処理学会ソフトウェア工学研究会, Vol. 92, No. 85, pp. 39–46, 1992.

[5] A.K. Jordan and A.M. Davis. Requirements Engineering Metamodel : An Integrated View of Requirements. In *Proc. of 15th COMPSAC*, pp. 472–478, 1991.

[6] K. Smolander, K. Lyytinen, V.P. Tahvanainen, and P. Marttiin. MetaEdit — A Flexible Graphical Environment for Methodology Modelling. In *Proc. of 3rd International Conference CAiSE91, LNCS 498*, pp. 168–193, 1991.

[7] Bashar Nuseibeh and Anthony Finkelstein. Viewpoints: A vehicle for method and tool integration. In *Proc. of 5th Internationnal Workshop on Computer-Aided Software Engineering*, pp. 50–60, 1992.

[8] Sjaak Brinkkemper. *Formalisation of Information Systems Modelling.* Thesis Publishers, 1990.

[9] 鯵坂恒夫. ソフトウェア意味要素の再利用. ソフトウェア再利用技術シンポジウム論文集, pp. 73–82, 1992.

[10] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lonrensen. *Object-Oriented Modeling and Design.* Prentice-Hall, 1991.