修 士 論 文

形式的オブジェクト指向分析モデルＦＯ∀Ｍの構築法とその支援環境

指導教官　片山卓也 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

古川 順一

1998 年 2 月 13 日

目 次

第 1 章

はじめに

1.1 背景

現在様々なオブジェクト指向方法論が提案されており、大規模なシステム開発に於いても適用され始めている。中でも OMT 法は典型的な方法論であり、オブジェクトモデル、動的モデル、機能モデルの 3 つのモデルを用いてシステムのモデル化を行なう。3 つのモデルを用いればシステムの分析を様々な視点から行なえるが、これら 3 つのモデル間には一貫性が保証されなければならない。しかし OMT 法では図の記述の定義が厳密でないため、3 つの図に分散してしまった情報から分析結果の正しさを確認することは非常に困難である。また、形式的な取り扱いが出来ないために、大規模なシステムでは不可欠な計算機による支援もまた難しい。

これに対して、以上の問題を解決するために提案された「形式的オブジェクト指向分析モデル ($Formal\ model\ for\ Object-oriented\ Analysis\ Model$: FO∀M)」では 3 つのモデルに記述の厳密な定義を与えられており、3 つのモデルの関連が明確にされている。OMT 法よりもより形式的な取り扱いが可能であるため、計算機による支援の幅も広いと考えられる。

1.2 目的

オブジェクト指向方開発でのシステムの構築は、分析・設計などの工程がラウンドトリップ型で行なわれる点で従来の方法とは異なっている。オブジェクトが開発プロセスを通して共通の単位となっているので後工程は前工程の結果を詳細化するプロセスとなる。しかしシステムの詳細化はシステムの各部分で一様に進む訳ではないので必要に応じて前工程に戻って詳細化のプロセスを繰り返さなければならない。これに従えばシステムの分析段階は一番最初に行なわ れ、且つ後の設計段階からも戻って来る部分なので、対象システムの安定した構造を抽出し、モデル化を行なわなければならない。

そこで上記の FO∀M を用いる。FO∀M を用いることにより分析モデルを形式的に取り扱い、構築したモデルの一貫性や整合性が容易に確認出来る。また、システムの安定した構造を形式的に保持して置けるため、オブジェクト指向開発特有のラウンドトリップ型の開発に対しても有効である。

しかし FO∀M は分析モデルの構築方法までは支援していない。そこで先ず分析モデルを構築する際に必要となる関係や規則などを抽出し、それにそった分析支援環境を提案する。

1.3 論文構成

本論文では FO∀M に基づいた分析モデル構築法と、それを支援する為の環境を提案する。

第２章では FO∀M の概念を紹介する。FO∀M では基本オブジェクトモデル、基本動的モデル、基本機能モデルがそれぞれ閉じた概念の下に形式化されており、統合写像を用いて各モデル中の共通概念を対応付けることで３つのモデルが１つに統合される。

第３章では簡単な自動販売機の例を用いて実際に分析モデルを構築して見るこれは分析モデル構築の際に必要なモデル中の要素間の関係を見極めるための為のものである。

第４章では上記例題の分析を通して明らかになった要素間の関係を形式化する。各モデルには構築指標となる「静的関係」と、構築段階で必要な「決定順序関係」が存在する。また、統合写像決定の際にはスコープルールが必要になる。

第５章では４章で明らかにした関係を踏まえて、分析モデル構築を支援する環境の計算機上への実装方針を示す。また、実際に開発言語として JAVA を用いて実装した環境を紹介する。

第 2 章

形式的オブジェクト指向分析モデル（FO∀M）

FO∀M では、システムの構成的側面、動的側面、機能的側面を各々閉じた概念の下に形式化し、基本オブジェクトモデル、基本動的モデル、基本機能モデルを構成する。これは各基本モデルが複数の分析者によって構築されることを可能にする為である。また各々独立に構成された３つの基本モデルは、統合写像によって１つに結びつけられる。

各基本モデルは、その中の要素を識別子集合として表現することで形式化される。例えば基本オブジェクトモデル中に存在するクラスはその識別子集合 **ClassID** に含まれる要素として扱われる。更に統合写像では基本動的モデル及び基本機能モデル中の要素を基本オブジェクトモデル中の要素を用いて表現する。これらの概念を図 2.1に示す。

各基本モデル中で定義される基本集合及び、統合写像で定義される写像を以下に示す。

2.1 基本オブジェクトモデルに於ける識別子集合

- **ClassID**：クラス識別子の集合
- **AttrID**：属性識別子の集合
- **FuncID**：関数識別子の集合
- **LPropID**：リンク属性識別子の集合
- **AssocID**：関連識別子の集合
- **AggrID**：集約関係識別子の集合

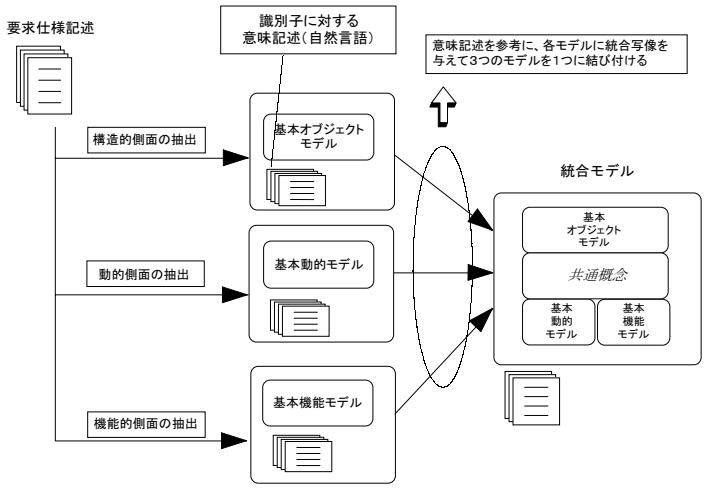

図 2.1: FO∀M の概念

- **InherID**：継承関係識別子の集合
- **LimitID**：限定子識別子の集合

ClassID,AttrID,FuncID,LPropID,AssocID,AggrID,InherID,LimitID 上の値を表現するメタ変数として、$classid, attrid, funcid, lpropid, associd, aggrid, inherid, limitid$ を用いる。

2.2 基本動的モデルに於ける識別子集合

- **StdID**：状態遷移図識別子の集合
- **StateID**：状態識別子の基本集合
- **TransID**：状態遷移識別子の集合
- **EventID**：イベント識別子の集合
- **CondID**：遷移条件識別子の集合
- **ActID**：アクション識別子の基本集合

StdID,StateID,TransID,EventID,CondID,ActID 上の値を表現するメタ変数として、
$stdid, stateid, transid, eventid, condid, actid$ を用いる。

2.3 基本機能モデルに於ける識別子集合

- **ProcessID**：プロセス識別子の集合
- **ActorID**：アクター識別子の集合
- **StoreID**：データストア識別子の集合
- **DataID**：データ識別子の集合

ProcessID,ActorID,StoreID,DataID 上の値を表現するメタ変数として、$oricessid, actorid, storeid, dataid$ を用いる

2.4 統合写像

2.4.1 動的モデルに於ける統合写像

- 状態遷移図 (I_{ST})：**ClassID** → **StdID**
- 状態 (I_s)：**StateID** → Pow(**ClassID**)
- 状態遷移条件 (I_c)：**CondID** → **BExp**
- アクション (I_a)：**ActID** → **AExp**
- イベント (I_e)：**EventID** → Pow(**AttrID**)
- 出力イベント式 (I_{eo})：**TransID** → **EExp**
- **AExp**：アクション式の集合
- **Bexp**：条件式の集合

2.4.2 機能モデルに於ける統合写像

- データ (I_d)：**DataID** → **AttrID**
- プロセス (I_p)：**ProcessID** → **FO**
- アクター (I_{ac})：**ActorID** → Pow(**AOE**) × Pow(**FO**)
- データストア (I_{store})：**StoreID** → Pow(**AOE**)

第 3 章

分析例

FO∀M の概念に従って分析を進める際に、明らかにしておかなければならない事柄が幾つかある。それはモデルの構築方法で、どのモデルをどの様に構築すれば良いかを考えなければならない。そこでこの章では分析対象として適当な例題を取り上げ、その分析を通してモデル構築の際に存在する要素間の関係や性質を見る。これらの関係の形式的な記述は次章で行なわれる。

例として取り上げるのは簡易な自動販売機を取り上げる。これの構造的側面、動的側面、機能的側面中の各視点から分析を行なってみて、決定順序を考える。

3.1 例題の説明

- 自動販売機の仕様

  この自動販売機はタバコを販売するためのもので、販売するタバコの種類は３種類である。購入者は買いたい銘柄に対応したボタンを押すことでタバコが購入出来る。購入者がタバコを購入するために投入した金額は逐次カウントされ、ディスプレーに表示される。タバコを買う毎に入れた金額からタバコの値段が差し引かれて、それに合わせてディスプレーの表示も変化する。

  タバコの購入は値段が十分である限り可能であり、のこり金額を返却して欲しい時、購入者は返却レバーを回す。これによりツリが返却される。

  各ボタンにはライトが付いていて、投入金額がタバコの値段以上になった時点でこのライトが点灯する。タバコが購入出来るなら、このライトは点灯し続け、残りがタバコの値段未満になった時点で消える。

3.2 分析

以上の仕様を持つ自動販売機を分析して分析モデルを構築する。分析に際しては各視点から分析を開始し、その場合の構築順序について考察する。

FO∀M では各側面（構造的側面、動的側面、機能的側面）の分析は独立に行なう。これは各モデルの形式化を各々閉じた概念を用いて行なうことで実現されている。そこで以下で行なわれる分析では、あるモデルを構築中には他の二つのモデルを意識しない。モデル間の関係は統合写像を決定する際にのみ考慮する。

3.2.1 構造的側面の分析

３つのモデルの内、オブジェクトモデルはシステムの構造的側面を表している。オブジェクトモデルを構成する要素には次の様なものがある。

- クラス
- 属性、操作
- 関係（継承、集約、関連）

これらの要素を決定する際の順序は明確ではない。よって実際に各要素の抽出から分析を始めてみて、規定できる順序があればそれを決定して行く。

クラスから

表 3.1: クラスとその意味記述

クラス名	意味
TobaccoVM	タバコ自動販売機
User	タバコ購入者
Return	返却レバー
Disp	金額表示用ディスプレー
Calc	計算機
Button	タバコ選択ボタン
Light	ボタンに付属する購入可能表示用ライト

例題の仕様記述からクラスを抽出してみる。ここではどうやってクラスを抽出するかを論じるつもりはないので、結果だけを表 3.1に記す。

これらのクラスを決定すれば、クラスに属する要素（属性、操作）と関係が決定できる。この２つはどちらからも決定しても良い。関係を決定してからクラスに属する要素を決定しても、その逆であっても本質的な違いはない。今回は関係から決定する。

関係を洗い出すと、表 3.2

表 3.2: 関係とその意味記述

クラス名	クラス名	関係	意味
User	TobaccoVM	関連	購入者は自動販売機にアクセスする
Return	TobaccoVM	集約	返却レバーは自動販売機の部品である
Disp	TobaccoVM	集約	ディスプレーは自動販売機の部品である
Calc	TobaccoVM	集約	計算機は自動販売機の部品である
Button	TobaccoVM	集約	ボタンは自動販売機の部品である
Light	Button	集約	ライトはボタンの部品である

次に各クラスに属する要素を決定して行く。仕様書を元に、各クラスの所持する性質などを抽出する。例えばクラス **Calc** の属性としては、現在の残り金額と、それに対して足す（若しくは引く）金額が必要であり、操作としては足す、引く、初期化するなどが必要となる。他のクラスについても同様に考えて、以下の属性、操作を付加する。

これで一応のオブジェクトモデルが完成した。但し、これで十分であるかどうかはここでは議論しない。

属性、操作から

属性、操作から分析を始める場合、各属性、操作は各々属性集合と操作集合に振り分けられる。次にこれらの集合中の要素をグループ化し、このグループに名前をつける。これがクラス名となる。表 3.4は属性集合を示したものである。これらは全てあるものの性質を表していて、それがなにであるかを分析して共通のもので括る。それがクラスとなる。

属性、操作から分析を開始した場合は、次に行なうことはそれらをクラス単位に分割することである。

関係から

関係を抽出する際には、関係をどの様に定義するかにもよるが、関係は結果的に２つのものの間に張られていなければならない。継承、集約は「何が」「何を」と言うことが分かっていなければ存在し得ないが、関連に関しては抽象的にこれだけを抜き出すことも出来る。但しその場合でも両側に「何か」が存在していることが条件となる。

表 3.3: 属性、操作とその意味記述 (一部)

名前	種類	所属	意味
paid	Attr	TobaccoVM	投入された金額
price	Attr	TobaccoVM	タバコの値段
numT1	Attr	TobaccoVM	タバコの個数１
numT2	Attr	TobaccoVM	タバコの個数２
numT3	Attr	TobaccoVM	タバコの個数３
isEnough	Func	TobaccoVM	投入金額は十分か調べる
ejectT	Func	TobaccoVM	タバコを搬出する
ejectC	Func	TobaccoVM	つりを搬出する
init	Func	TobaccoVM	初期化する
money	Attr	User	所持金
paid	Attr	User	投入した金額
num	Attr	User	購入タバコ数
pay	Func	User	お金を投入する
push	Func	User	ボタンを押す
turn	Func	User	返却レバーを回す
takeT	Func	User	タバコを取り出す
takeC	Func	User	つりを取り出す

例えば自動販売機の例では（半ば強引であるが）「アクセスする」と言う関係だけを定義する。「何が」「何に」アクセスするのかまでは分からないが、取りあえず関係だけを定義しておきたい場合にはこの様に出来る。最終的に「何が」の部分には「タバコの購入者」が当てはまり、「何に」の部分に「自動販売機」が当てはまることが分かった時点で具体的にこれを関連の両側に当てはめる様にする。

関連に付属する要素（リンク属性、限定子）は関連を決定した後でなければ決定出来ない。またリンク属性や限定子は属性、操作を決定しないでいきなり決定できるものではないので、これらの要素は各クラスの属性、操作を決定した後でなければ決定出来ない。

3.2.2 動的側面の分析

仕様から動的な側面を抽出する場合、先ず何についての動作を記述するのかが分かっていなければならない。つまり何に関する状態遷移図なのかは、必ず最初に認識しておく必要がある。

前記仕様に対して、自動販売機の状態遷移図を作成することにする。この時オブジェクトモデルの場合と同様に、各種要素の抽出から分析を開始してみる。

状態から

自動販売機にはどの様な状態が存在しているのかを考慮し、その状態を抽出する。表 3.5に抽出した状態を記す。状態を抽出して次に行なうことは遷移を状態間に張ることである。但し無造作に状態間を結べる訳ではないので、「どの様なときに遷移するか」を考えなければならない。

状態間で遷移が生じる場合はイベントがある場合と、イベントは無いが自動的に次の状態に遷移する場合が考えられ、また遷移を制限する要素として遷移条件を考慮しなければならない。状態遷移図の動作自体がイベントドリブンであることを考えれば、先ず入力イベントを決定し、次に制限がないかを考える必要がある。

例えば状態 **WAIT,COUNT** 間の遷移を考える。カウントした状態へ遷移するためのイベントとして、購入者がお金を投入したことが考えられる。投入された金額は常にカウントしておかなければならないので、この時の遷移の条件はない（常に true である）。以下同様に考えて、遷移とその遷移上の入力イベント、遷移条件を定義する。（表 3.6。なお、各識別子の意味は省略した。）

次に行なう事は、出力イベントの決定かアクションの決定である。今回は自動販売機の状態遷移図のみに注目しているが、実際並行して購入者の状態遷移図を作成したならば、出力イベント名は明らかになっている。

アクションに関しては状態を抽出する際に、ある程度のものが明らかになっている。残りのものはその時に何をすべきかを考慮して決定する。この様にして最終的に決定した状態遷移図の要素を表 3.7に記す。

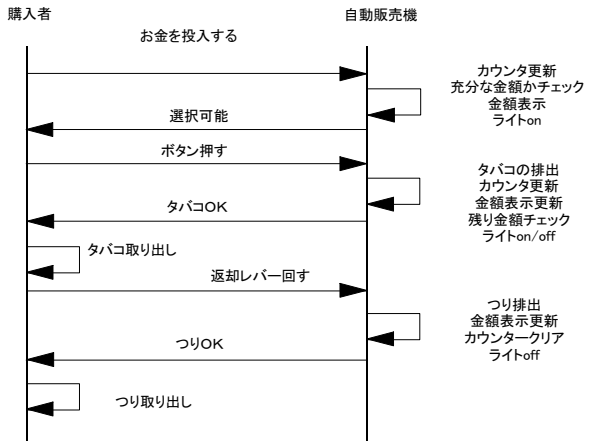

図 3.1: 事象トレース図

イベントから

イベントの抽出から分析を始めるにあたって、事象トレース図を導入する。自動販売機と購入者に関する事象トレース図を図 3.1に示す。

この事象トレース図から、イベントを抽出する。抽出したイベントを表 3.8 に記す。

事象トレース図を用いて、あるイベントを受けとった時に何をするかを決めることが出来る。つまりこれがアクションになり、そのアクションの結果としてまたイベントが発生する。アクションが決定されれば、それに対して何か条件が付随しないかを考える。

例えば図 3.1の事象トレース図で、お金が投入された後、自動販売機側では以下のことを行う必要がある。

- カウンタの更新
- 十分な金額であるかのチェック
- 金額の表示
- ライトの点灯

ここで「十分な金額であるかチェック」が条件分岐になることは容易に分かるので、状態 **CHECK** を作り、投入金額が十分になるまでは選択可能にしない様に出来る。

アクションから

アクションから分析をを始た場合、仕様からアクションの集合を抽出し、このアクションが「どのような時に」起こるのかを考えることになる。例えば「カウンタの更新」が仕様から抽出されている場合、このアクションがどの様な時に起こるかを考えると、先ず自動販売機に対してお金が投入された場合が考えられる。また、カウンタの更新はタバコが買われた場合、にも更新されなければならないだろう。つまり、アクションの次には、そのアクションが実行されるための入力イベントが決定される。また、この時同時に遷移条件も考えることが出来る。カウンタの更新には特に条件が必要でないので、遷移条件は **true** である。

3.2.3 機能的側面の分析

機能的側面に於いて決定する要素は、アクター、プロセス、データストアとデータ（及びそのデータフロー）である。他の２つのモデルと同様、これらの要素に関しても、以下の場合に従って分析をスタートさせてみる。

プロセス

プロセスは、オブジェクトモデルではクラス中の操作として現れる部分である。従ってプロセスを仕様から抽出する場合も、オブジェクトモデル中での操作集合の抽出とほぼ同様の作業になる（表 3.3参照）。実際に自動販売機に対して購入者がお金を投入した場合の機能モデルについて、表 3.9のプロセスが抽出出来る。

プロセスを決定したのちには、そのプロセスに入力されるデータ及びそのプロセスから出力されるデータを決定する。例えば プロセス **count** の入力は購入者に投入されたお金であり、その出力は金額である。その他のプロセスについても同様に、入出力を決定する。（表 3.10）

これらをデータ名を元に繋げば、一連の処理に対するデータフローが決定出来る。また、結合先がプロセスでない部分はデータストアか、アクターがその先のノードにあることになり、この場合例えば **coin** の出力はアクターとして **user** を決定出来る。

3.2.4 統合写像

３つの基本モデルについての分析については上記の通りであるが、FO∀M ではこれら３つのモデルはそれぞれ独立した概念の下で形式化されている。従って３つとも並行して分析を進めることが可能である。従ってこれまでの分析では１つのモデルを構築する際には他の２つのモデルは考慮しなかったが、実際にはメインとするモデルを決定することも出来るだろう。つまり「主導モデル」を考えることが出来る。主導モデルを考えた場合、他の２つのモデルはその主導モデルを考慮しながら作成出来る。以下例として、基本動的モデルを主導モデルとして、基本オブジェクトモデルを決定する。

先ず、自動販売機の動的モデルを図 3.2に示す。図は前に行った動的モデルの分析に基づいている。

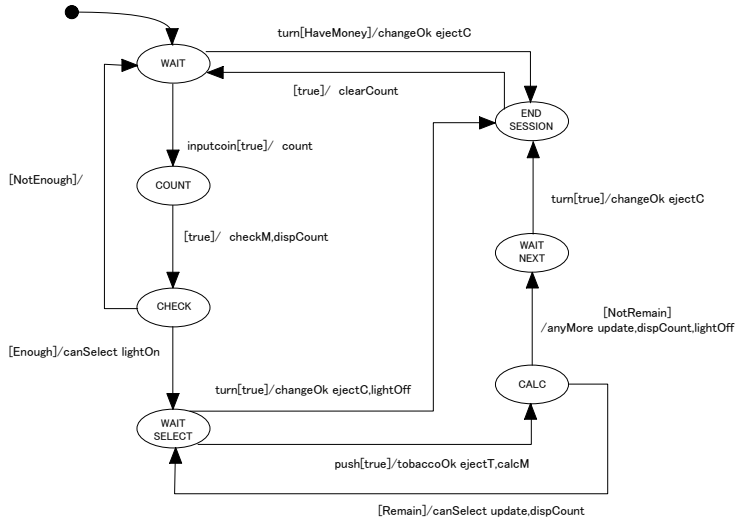

図 3.2: 自動販売機の動的モデル

この基本動的モデルの統合写像を考えることで、基本オブジェクトモデル中に必要な要素を考える。先ず、この動的モデルが自動販売機に対するものであることから、基本オブジェクトモデル中にクラスとして自動販売機と、それにアクセスするべき購入者クラスが必要であることは容易に分かることである。今、動的モデル図中の状態 **WAIT,COUNT** 間の遷移に注目してみると、そこには入力イベントとして **inputcoin**、アクションとして **count** が定義されているのが分かる。アクションである **count** がどの様なものであるかを考えると、これは投入された金額を逐次カウントして、現在の投入金額に足す動作である。つまり基本オブジェクトモデル内の自動販売機クラスには、以下の属性と操作が必要であることが分かる。

また投入金額と言うのは購入者から送られてくるものであり、従ってアクション **count** の統合写像では入力イベント **inputcoin** の写像先である、購入者クラスの属性が用いられているものであると判断できる。

以上の様に主導モデルを決定すれば、そのモデルの統合写像を考えることで他のモデルを決定（または決定補助）出来る。

3.3 まとめ

この章では例題を様々な側面から分析することで、実際にどの様な分析路をとることが可能であるかを見てきた。全体をまとめておくと、次の様になる。

- **基本オブジェクト モデル**

  基本オブジェクトモデルでは、クラス、属性、操作そして関連からの抽出が可能である。クラスから抽出を始めた場合は、次にクラス間の関係か若しくはそのクラスに属する属性、操作の決定が可能にな

る。属性、操作の抽出から始めた場合は、それをカプセル化することが次に来る。最後に関連の抽出から始めた場合は、両側に存在するクラス（のイメージ）を具体化することになる。具体化するとはそのクラスに名前を付けるか、若しくは属性、操作を当てはめて機能的なカプセル化を行うかである。

関連以外の関係（継承、集約）に関しては、関連の様に両側に来るものは分からないが取りあえず継承すると言う事実だけを抽出するのは意味がない。

関連に付属する要素であるリンク属性、限定子は関連の両側に存在するクラスに深く依存する。従って、関連から抽出した場合でも、両側のクラスが何であるかがわからなければ、当然定義は不可能である。更に、それらは実際には両方に共通する属性や操作であったり、多重度を取り除く為に必要な属性名であったりするので、実質的に両側のクラスに対してそれに属する属性、操作がある程度決定されていないと決定出来ない。

- **基本動的モデル**

  基本動的モデルでは、状態、イベント、遷移条件、アクションの抽出からそれぞれ分析を開始出来る。状態から抽出した場合は、次に状態間の遷移が起こる場合のイベントを決定するか、その時のアクション、若しくは遷移条件を決定するかである。イベントから抽出する場合は、分析対象となるシステム全体でのイベントを考える。つまりある状態遷移図での入力イベントは、別の状態遷移図では出力となっているはずである。この様な分析のやり方には、イベントトレース図が有用である。

  遷移条件はシステムで明らかに分岐が必要な場合を抽出することになる。自動販売機の例では「お金が足りない場合」などがそれに当たる。この次に出来ることは、これを足掛かりにして遷移に条件の必要な状態を決定するか、またはその遷移を同じラベル内に存在するであろうイベント、アクションを決定することである。

  アクションから決定する場合は、このシステムがどういうことを行うかを考える、これはちょうどオブジェクトモデルで操作を抽出する行為と同じである。のちの決定は遷移条件の場合と大差ない。

- **基本機能モデル**

  基本機能モデルでは、プロセス、アクター、データスと、データの抽出から分析を開始することが可能である。但しデータフローは名前の通りデータの流れる道筋でしかないので、プロセスなどの要素が両側に定義されていないと定義出来ない。基本オブジェクトモデルでの関連の様には行かない。

- **統合写像**

  統合写像の決定については、前記の通り主導モデルを決定しなければならない。主導モデルを決定した場合でも、その主導モデル自体はある程度構築されていなければならない。例えば主導モデルを動的モデルに設定した場合、あるアクションの写像からオブジェクトモデルの要素を決定する場合は、

そのアクション自体は最低限決定（抽出）されていなければならない。これは機能モデルを主導モデルにとった場合でも同じである。

また主導モデルをどれにとったにしろ、統合写像の決定にはスコープルールを充分に考慮しなければ、基本オブジェクトモデルにおける情報隠蔽が崩れてしまう。このことについては次章で詳しく述べる。

表 3.4: 属性とその意味記述

名前	意味
current	現在の金額
price	タバコの値段
numT1	タバコの個数１
numT2	タバコの個数２
numT3	タバコの個数３
money	所持金
paid	投入金額
num	購入タバコ数
turned	レバーが回されたか
current	現在の金額
val	現在の金額から足す（引く）金額
tName	ボタンに対応するタバコ名
pushed	ボタンが押されたか
isOn	ライトは点いているか

表 3.5: 自動販売機とその意味記述

状態	意味
INIT_VM	初期状態（特別）
WAIT	購入者からのアクセス待ち状態
COUNT	投入金額をカウントした状態
CHECK	投入金額をチェックした状態
WAIT_SSELECT	タバコの購入待ち状態
CALC	残金計算をした状態
WAIT_NEXT	次のタバコ購入待ち状態
END_CONTROL	おしまい

表 3.6: 状態遷移と入力イベント及び遷移条件

from	to	入力イベント	遷移条件
WAIT	COUNT	inputcoin	true
WAIT	END_CONTROL	turn	HaveChange
COUNT	CHECK		true
CHECK	WAIT		NotEnough
CHECK	WAIT_SELECT		Enough
WAIT_SELECT	CALC	push	true
CALC	WAIT_SELECT		Remain
WAIT_SELECT	END_CONTROL	turn	true
CALC	WAIT_NEXT		NotRemain
WAIT_NEXT	END_CONTROL	turn	true
END_CONTROL	WAIT		true

表 3.7: 状態遷移と出力イベント及びアクション

from	to	出力イベント	アクション
WAIT	COUNT		count
WAIT	END_CONTROL	changeOk	backChange
COUNT	CHECK		checkM,dispCount
CHECK	WAIT		
CHECK	WAIT_SELECT	canSelect	lightOn
WAIT_SELECT	CALC	tabaccoOk	ejectTobacco,calcM
CALC	WAIT_SELECT	canSelect	update,dispCount
WAIT_SELECT	END_CONTROL	changeOk	backChange,lightOff
CALC	WAIT_NEXT	anyMore	update,dispCount,lightOff
WAIT_NEXT	END_CONTROL	changeOk	backChange
END_CONTROL	WAIT		clearCount

表 3.8: 自動販売機の入力イベントと出力イベント

名前	入力／出力	意味
inputcoint	入力	お金が投入された
turn	入力	返却レバーが回された
push	入力	ボタンが押された
canSelect	出力	選択可能
tobaccoOk	出力	タバコ搬出
changeOk	出力	つり搬出

表 3.9: 自動販売機のプロセスとその意味

名前	意味
count	投入金額をカウントする
collect	投入金額を貯める
check	十分な金額かチェックする
on_off	ライトの状態を切替える
display	金額を表示する

表 3.10: 自動販売機のプロセスとその入出力

プロセス	入力	出力
count	coin	amount
collect	coin	coin
check	amount	result
on_off	result	light
display	amount	light

表 3.11: 自動販売機クラスに必要な属性と操作（一部）

名前	属性／操作	意味
cAmt	属性	現在の金額
plus	操作	金額を追加する

第 4 章

分析モデルの構築法

第 3 章で行った分析から、分析モデル構築の際には幾つかの関係を明確にしなければならない事が分かる。一つは要素間の静的な関係であり、これはモデルをある程度構築した際の完成度の指標となる部分である。もう一つは動的な関係であり、これはモデル中の要素の決定順序に相当するものである。第 3 章で見たとおり、モデルの構築法には幾通りか存在しているが、どの場合でもある程度共通した順序が存在している。

4.1 オペレータの定義

関係／ルールを定義する前に、モデル取り扱いの便宜をはかるためのオペレータを定義する。これらのオペレータは、以下関係及びルールを定義する際に一貫して使用される。

- **オブジェクト モデルに於けるオペレータ**
  - *Inher :* ClassID → ClassID
    クラス識別子からそのクラスが継承しているクラス識別子を求める。
  - *Aggr :* ClassID → ClassID
    クラス識別子からそのクラスが部品クラスとして用いられているクラス識別子を求める。
  - *Lprop :* ClassID×AssocID → LpropID
    クラス識別子と関連識別子から、そのクラスの関連に付属するリンク属性識別子を求める。
  - *Assoc :* ClassID×AssocID → ClassID
    クラス識別子と関連識別子からそのクラスが関連により繋がれているクラス識別子を求める。
  - *Limit :* ClassID×AssocID → LmitID
    クラス識別子と関連識別子から、そのクラスが関連によって繋がれているクラスに於ける限定子

識別子を求める。

- *Attr :* ClassID∪LpropID → Pow(AttrID)
  クラス識別子及びリンク属性識別子からそのクラス（リンク属性）に含まれる属性識別子の集合を求める。
- *Func :* ClassID∪LpropID → Pow(FuncID)
  クラス識別子及びリンク属性識別子からそのクラス（リンク属性）に含まれる操作識別子の集合を求める。

- **動的モデルに於けるオペレータ**
  - *Input :* TransID → EventID
    状態遷移識別子からその間にあるラベル中の入力イベント識別子を求める。
  - *Cond :* TransID → CondID
    状態遷移識別子からその間にあるラベル中の遷移条件識別子を求める。
  - *Act :* TransID → EventID
    状態遷移識別子からその間にあるラベル中のアクション識別子を求める。
  - *Output :* TransID → ActID
    状態遷移識別子からその間にあるラベル中の出力イベント識別子を求める。
  - *Std :* IDs → STID
    状態遷移図中の各要素の識別子（IDs）からその要素が属する状態遷移図識別子を求める。
  - *DefClass :* STID → ClassID
    状態遷移図識別子からそれに写像されているクラス識別子を求める。
  - *Belong :* EventID → Pow(AttrID)
    イベント識別子からそのイベントの付属属性識別子を求める。
  - *PreS :* TransID → StateID
    状態遷移識別子から、その遷移の前状態の識別子を求める。
  - *PostS :* TransID → StateID
    状態遷移識別子から、その遷移の後状態の識別子を求める。

- **機能モデルに於けるオペレータ**
  - *Data :* SrcID×DstID → DataID
    機能モデル中の要素においてデータフローの前の要素（SrcID）と後の要素（DstID）からその間にあるデータフローに付属するデータ識別子を求める。

- **統合写像に於けるオペレータ（スコープルールに関するもの）**

  - $AsstAttr:$ TransID $\rightarrow$ Pow(AttrID)
    遷移識別子から、その遷移中の要素の統合写像を決定する際に代入属性として有効なものを求める。
  - $RefAttr:$ TransID $\rightarrow$ Pow(AttrID)
    遷移識別子から、その遷移中の要素の統合写像を決定する際に参照属性として有効なものを求める。
  - $RefFunc:$ TransID $\rightarrow$ Pow(FuncID)
    遷移識別子から、その遷移中の要素の統合写像を決定する際に参照操作として有効なものを求める。

4.2 静的な関係

FO∀M の概念に従ってモデルを構築していく過程で、モデルがどの程度完成しているかはこの静的な関係を満たすものがどれだけ存在しているかによる。FO∀M では３つの基本モデルは独立した概念を用いて形式化されているので、静的な関係も各基本モデル中で閉じているものとする。つまりある基本モデル中の要素間の静的な関係は、他の２つの基本モデル中の要素には影響されない。

4.2.1 基本オブジェクト モデルに於ける静的関係

前章の分析から、基本オブジェクトモデル中でもっともアトミックな部分はクラス識別子集合、属性識別子集合、操作識別子集合であると考えられる。その他の要素は全てこれらアトミックな要素間の関係として定義される。

- **属性**
  あるクラスに属する属性は必ず属性識別子集合に含まれていなければならない。
- **操作**
  属性同様、あるクラスに属する操作は、必ず操作識別子集合に含まれていなければならない。
- **関連**
  関連は２つのクラス間に張られるものなので、必ずその両端にはクラスが存在していなければならない。

$$\forall associd\ \exists classid_1, classid_2\ .\ Assoc(classid_1, associd) = classid_2$$

- **継承**

  継承関係も２つのクラス間に定義される関係である。

$$\forall inherid\ \exists classid_1, classid_2\ .\ Inher(classid_1) = classid_2$$

- **集約**

  集約関係も同様に２つのクラス間に定義される関係である。

$$\forall aggrid\ \exists classid_1, classid_2\ .\ Aggr(classid_1) = classid_2$$

- **リンク属性、限定子**

  リンク属性と限定子は関連に付属するものである。

$$\forall lpropid\ \exists associd, classid\ .\ Lprop(classid, associd) = lpropid$$

$$\forall limitid\ \exists associd, classid\ .\ Limit(classid, associd) = limitid$$

4.2.2 基本動的モデルに於ける静的関係

基本動的モデルでは、遷移識別子以外の識別子集合がアトミックであると考えられる。遷移は２つの状態間に張られる関係であり、遷移上のラベルは遷移とイベント、条件、アクションの関係であると見ることが出来る。これを踏まえて以下の形式化を行う。

- **状態**

  ある状態遷移図中で用いられている状態は必ず状態識別子集合に含まれていなければならない。

- **遷移**

  全ての遷移は、２つの状態間に張られていなければならない。（但し自己遷移する場合は、２つの状態は同じ状態になる）

$$\forall transid\ \exists stateid_1, stateid_2\ .\ PreS(transid) = stateid_1 \wedge PostS(transid) = stateid_2$$

$$where\ Std(stateid_1) = Std(stateid_2)$$

- **イベント**

  ある状態遷移図で入力イベントとしてイベントは別の場所で出力イベントとして与えられていなければならない。

$$\forall eventid\ \exists transid_1, transid_2\ .\ Input(transid_1) = eventid \Leftrightarrow Output(transid_2) = eventid.$$

$$where\ Std(transid_1) \neq Std(transid_2)$$

4.2.3 基本機能モデルに於ける静的関係

- **データフロー**

  データフローは基本動的モデルの遷移と同様、必ず２つの要素（プロセス、アクター、データストア）間に張られている。しかしある要素から同じ要素へデータフローが張られることはない。また、データフロー上には必ずデータが定義されていなければならない。

$$
\begin{aligned}
&\forall dataflowid \exists \mathbf{IDs_1}, \mathbf{IDs_2}, dataid. \\
&Src(dataflowid) = IDs_1 \wedge Dst(dataflowid) = IDs_2 \wedge \\
&Data(dataflowid) = dataid \\
&where\ IDs_1 \neq IDs_2
\end{aligned}
$$

4.3 スコープルール

前節で定義した静的関係は、３つの基本モデルの構築方法に関するものであった。次に統合写像を考えるとその構築に際しても何らかのルールが必要となる。

基本動的モデル、機能モデルの統合写像は各モデル中の要素を基本オブジェクトモデル中の要素に写像することにより行われる。オブジェクトモデル中の要素はクラス単位で情報隠蔽さているので、写像先として用いることの出来る属性、操作は自ずと限定される。従って統合写像決定の際には情報隠蔽に沿った写像を与えなければならない。

以下、基本動的モデル、機能モデルの統合写像決定の際に存在するスコープルールを定義する。

4.3.1 オブジェクト モデルの例

先ず例として用いるオブジェクトモデルを図 4.1に示す。

この図に従って動的モデル、機能モデルの各要素の統合写像を決定する。更に一般的なスコープを式として与える。

4.3.2 動的モデルにおけるスコープ

図 4.2は簡単な動的モデルの例である。状態遷移図 $STD\ A$ は二つの状態 $s1, s2$ からなり、その間の遷移には入力イベント inA、遷移条件 $conA$、出力イベント $outA$、アクション $actA$ が存在している。

スコープは、状態遷移図識別子 $STD\ A$ にどのクラスが写像されるかによって異なる。今、図 4.1中のク

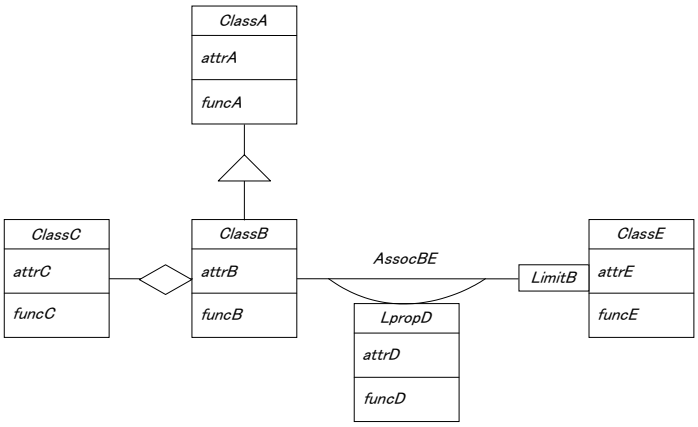

図 4.1: オブジェクトモデルの例

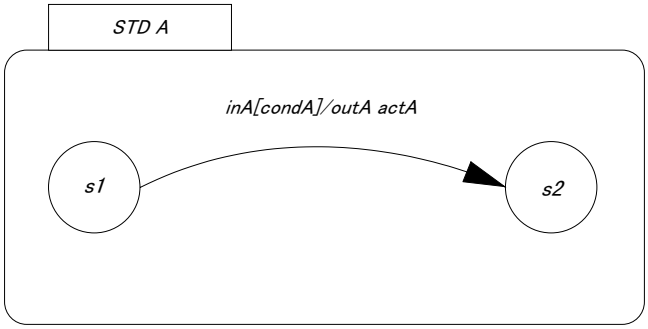

図 4.2: 動的モデルの例

ラス $ClassB$が $STD\ A$ に写像されているとする。つまり、以下が成立しているとする。

$$I_{ST}(ClassB) = STD\ A$$

つまり状態遷移図 $STD\ A$ のデフォルトクラスについて以下が言える。

$$DefClass(STD\ A) = ClassB$$

このときこの状態遷移図中の状態の統合写像について、その写像先として許可されるのはクラス $ClassB$及びそのの構成クラスとなる。従って、状態 $s1, s2$ の写像先として挙げられるのは、

$$ClassB, ClassC$$

である。ここで、状態 $s1, s2$ がどう写像されるかによって、その間にあるラベル要素のスコープが異なってくる。幾つかに場合分けをし、各場合について使用可能となる属性、操作を明らかにする。

1. **代入属性**
   アクション式を構成する時に用いる、代入される側の属性。
2. **参照属性**
   アクション式、遷移条件式及び出力イベント式を構成する時に用いる関数の引数としての属性。
3. **関数**
   アクション式、遷移条件式及び出力イベント式を構成する時に用いる関数。

- $I_s(s1) = ClassB \wedge I_s(s2) = ClassB$
  この時は両方とも $ClassB$に写像されているので、使用可能となる属性、操作は以下の通りである。

  1. **代入属性**
     $attr\,A, attr\,B, attr\,D, Belong(in A)$
  2. **参照属性**
     $attr\,A, attr\,B, attr\,D, Limit, Belong(in A)$
  3. **関数**
     $funcA, funcB, funcD$

- $I_s(s1) = ClassC \wedge I_s(s2) = ClassB$
  状態の変化はラベル中のアクションによって引き起こされるものである。遷移前の状態が $ClassC$に写像されている時（つまりデフォルトクラスを構成するクラスに写像されている時）、その後に来る遷移中のアクションに関連するスコープは上とは異なってくる。この場合はクラス $ClassB$の部品クラス $ClassC$が処理の一部を担っているので、スコープは以下の様になる。

1. **代入属性**

   $attrC, Belong(inA)$

2. **参照属性**

   $attrA, attrB, attrC, attrD, LimitB, Belong(inA)$

3. **関数**

   $funcA, funcB, funcC, funcD$

- $I_s(s1) = ClassB \wedge I_s(s2) = ClassC$
  この場合は一番最初の場合と同じになる。

- $I_s(s1) = ClassC \wedge I_s(s2) = ClassC$
  この場合は２番目の場合と同じになる。

以上の様に、２つの状態間の遷移を考えた場合は最初の状態がどのクラスに写像されているかによってスコープが変わってくる。一般的に考えて動的モデルに於けるスコープルールは次の様にまとめられる。

1. **代入属性**

   - 自分自身のクラスに含まれる属性
   - 自分自身のクラスが継承しているクラスの属性
   - 自分自身のクラスに接続されているリンク属性に含まれる属性
   - その状態遷移中の入力イベントが持つ属性

2. **参照属性**

   - 自分自身のクラスに含まれる属性
   - 自分自身のクラスが継承しているクラスの属性
   - 自分自身のクラスに接続されている関連のリンク属性に含まれる属性
   - 自分自身のクラスが部品クラスの場合、そのクラスを含むクラスの属性
   - 自分自身のクラスに接続されている関連先で指定された限定子
   - その状態遷移中の入力イベントが持つ属性

3. **関数**

   - 自分自身のクラスに含まれる操作
   - 自分自身のクラスが継承しているクラスの操作

- 自分自身のクラスに接続されている関連のリンク属性に含まれる操作

これら動的モデルのスコープルールを、これまでに定義した統合写像及びオペレータを用いて定義する。状態遷移図識別子集合 $STID$に含まれる任意の状態遷移図を $STD\ A$ とし、その中に含まれる任意の遷移の遷移前状態と遷移後状態について以下が成り立つ。

$$\bullet(I_{state}(s1) = ClassB \wedge I_{state}(s2) = ClassB) \vee (I_{state}(s1) = ClassB \wedge I_{state}(s2) = ClassC)$$
$$\Rightarrow (AsstAttr(s1 \rightarrow s2) = Attr(ClassA) \cup Attr(ClassB) \cup Attr(LpropD) \cup Belong(inA))$$
$$\wedge(RefAttr(s1 \rightarrow s2) = Attr(ClassA) \cup Attr(ClassB) \cup Attr(LpropD) \cup LimitB \cup Belong(inA))$$
$$\wedge(RefFunc(s1 \rightarrow s2) = Func(ClassA) \cup Func(ClassB) \cup Func(LpropD)).$$

$$\bullet(I_{state}(s1) = ClassC \wedge I_{state}(s2) = ClassB) \vee (I_{state}(s1) = ClassC \wedge I_{state}(s2) = ClassC)$$
$$\Rightarrow (AsstAttr(s1 \rightarrow s2) = Attr(ClassC) \cup Belong(inA))$$
$$\wedge(RefAttr(s1 \rightarrow s2) = Attr(ClassA) \cup Attr(ClassB) \cup$$
$$Attr(ClassC) \cup Attr(LpropD) \cup LimitB \cup Belong(inA))$$
$$\wedge(RefFunc(s1 \rightarrow s2) = Func(ClassA) \cup Func(ClassB) \cup Func(ClassC) \cup Func(LpropD)).$$

$where$

$ClassA, ClassB, ClassC, ClassE \in \mathbf{ClassID}$

$LpropD \in \mathbf{LpropID}$

$AssocBE \in \mathbf{AssocID} \wedge Assoc(ClassB, AssocBE) = AssocE$

$Inher(ClassB) = ClassA$

$Aggr(ClassC) = ClassB$

$LProp(ClassB, AssocBE) = LpropD$

$Limit(ClassB, AssocBE) = LimitB$

$STD\ A \in \mathbf{STID}$

$s1 \rightarrow s2 \in \mathbf{TransID} \wedge Std(s1 \rightarrow s2) = STD\ A$

$PreS(s1 \rightarrow s2) = s1 \wedge PostS(s1 \rightarrow s2) = s2)$

$s1, s2 \in \mathbf{StateID} \wedge (Std(s1) = STD\ A \wedge Std(s2) = STD\ A)$

$Input(s1, s2) = inA$

$I_{ST}(STD\ A) = ClassB.$

4.3.3 機能モデルにおけるスコープ

プロセス、データ

基本機能モデルの統合写像において、プロセスの写像先は基本オブジェクトモデル中の操作であり、データの写像先は属性である。プロセスとデータの関係を考慮すれば、プロセスに入ってくるデータはプロセスが処理をする時に用いられる属性（参照属性）であり、プロセスから出て行くデータはプロセスにより処理された属性（代入属性）であることが分かる。従って、これらのスコープとして、プロセスは基本動的モデルの場合に定義した「操作」、データは「代入属性」及び「参照属性」に準ずるものとする。

アクター、データストア

基本機能モデルでは、アクター、データストアの写像先として基本オブジェクトモデルの属性と操作の組が与えられる。この場合も基本動的モデルの時と同様、スコープルールを考慮しなければならない。アクター、データストアは基本オブジェクトモデルでは一つのオブジェクトとして実現されているべきものなので、これらの写像先としてはそのオブジェクトが参照可能な属性、操作を用いなければならない。実際に使用可能となる属性、操作は基本動的モデルで定義した「参照属性」「操作」と同様である。

4.4 決定順序関係

前節で定義したのは FO∀M で定義されている静的な関係であり、モデルが最終的に満たしていなければならないものである。しかし実際にモデルを構築する際に「どの様にモデルを構築すれば良いか」と言う動的な部分は FO∀M では定義されていない。分析支援環境を実現するためにはこれらの関係を明確にしておかなければならない。

前章で見たように、分析を行う際には様々な分析路が考えられる。しかし最初に何を決定したかによって、決定の順序はある程度規定できるのも確かである。この節では前章での分析例を元にそれらの順序関係を明確にしてこれを形式化する。

4.4.1 基本オブジェクト モデルのおける決定順序

基本オブジェクトモデルに於いて、最初に決定可能なものにはクラス識別子、属性識別子、操作識別子がある。

クラス識別子から

クラス識別子を抽出した後にそれと関連して行えることは、そのクラス識別子の対応するクラス式を決定することである。また２つ以上のクラス識別子が定義されていて、それらの間に何らかの関係が存在する

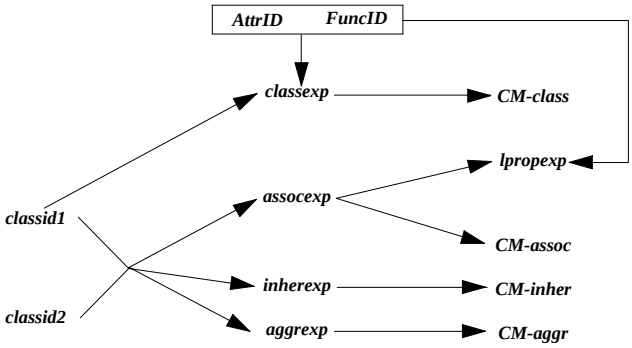

図 4.3: クラスから抽出を始めた場合の決定順序関係

ならば、それをクラス識別子間に定義出来る。定義した関係が関連であったならば、それはリンク属性、または限定子をもつ可能性がある。以上クラス識別子から抽出を始めた場合の順序関係を図示すると、図 4.3 の様になる。

これらより基本オブジェクトモデルに於いて、クラス識別子の定義から始めた場合の決定順序関係を以下に定義する。

- **クラス式**

  クラス式は既に定義されている属性識別子、操作識別子のみを用いて構成される。

$$\begin{aligned} & \{AttrSet, FuncSet\} \prec classexp \\ where \quad & AttrSet \subseteq \mathbf{AttrID}, FuncSet \subseteq \mathbf{FuncID} \\ & classexp = (AttrSet, FuncSet) \end{aligned}$$

- $\mathcal{CM}_{class}[ClassSet]$

  FO∀M で定義されているクラス識別子をクラス式に対応付けるクラス識別子集合上の写像$\mathcal{CM}_{class}[ClassSet]$は上記クラス式が決定されて初めて定義出来る。従って以下が言える。

$$\begin{aligned} & classexp \prec \mathcal{CM}_{class}[ClassSet] \\ where \quad & \mathcal{CM}_{class}[ClassSet](classid) = classexp \\ & ClassSet \subseteq \mathbf{ClassID}, classid_1 \in ClassSet \end{aligned}$$

- **関係**

  クラス識別子の定義から始めた場合、そのクラス識別子と他のクラス識別子の間に関係を定義するには、少なくとも片側に存在するクラス識別子（若しくはクラス式）が定義されていなければならない。例えば継承式を構成する場合には以下が言える。

$$classid \prec inherexp$$

  この場合のクラス識別子（またはクラス式）は継承元、継承先どちらであっても構わない。但し両方とも未定義の場合には継承、集約関係の場合意味がない。

- **リンク属性、限定子**

  リンク属性と限定子は、ある２つのクラス間に関連が生じて初めて定義できるものである。従ってこれらの構成には関連式が定義されていることが必要となる。リンク属性に関して言えば、その内容となるリンク属性式は、クラス式と同様に決定されなければならない。

$$\begin{aligned} & \{assocexp, AttrSet, FuncSet\} \prec lpropexp \\ where \quad & assocexp \ : \ lpropexp \text{ の属する関連} \\ & lpropexp = (AttrSet, FuncSet) \end{aligned}$$

属性／操作識別子から

属性／操作識別子の定義から開始した場合、次に行うことはそれらをクラス式にまとめることである。クラス式にまとめた後はそれを既存のクラス識別子と対応づけるか若しくは別のクラス式（若しくはクラス識別子）との間に関係を定義することが可能である。この場合の関係の決定順序は、上記のクラス識別子の定義から始めた場合の定義に準ずるものとする。関係以外の部分の決定路を図 4.4に示す。

この場合でも、クラス式に関しては前記と同様である。しかし写像 $\mathcal{CM}_{class}[ClassSet]$ に関しては、この段階でまだクラス識別子が決定されていないと言う違いがあるので、その決定が先行する。つまり、

$$classid, classexp \prec \mathcal{CM}_{class}[ClassSet]$$

となる。(実質的には同じ。)

関連識別子から

関係 (継承、集約、関連) の中で継承関係と集約関係については抽象的にそれらを抽出出来ないが、関連についてだけはそれを行える。つまり、両側に存在するクラスが何か分からない状態であっても、関連だけ

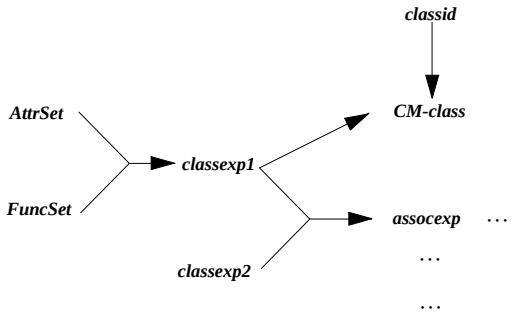

図 4.4: 属性／操作から抽出を始めた場合の決定順序関係

を明らかにしておける可能性がある。この様に関連識別子だけを抽出した場合は、次にそれが対応する関連式を決定しなければならない。関連式についてはクラス識別子から定義を開始した場合に準ずる（つまり、その構造を決定するには両側のクラスが分かっていなければならない)。この時の決定順序関係を以下に定義する。

- **関連式**

$$\{classid_1, classied_2\} \prec assocexp$$
$$where \qquad assocexp = (classid_1(class_2id))$$

- $\mathcal{CM}_{Assoc}[ClassSet]$

$$\{associd, assocexp\}\mathcal{CM}_{Assoc}[ClassSet]$$
$$where \qquad \mathcal{CM}_{Assoc}[ClassSet](associd) = assocexp.$$

4.4.2 基本動的モデルにおける決定順序

基本動的モデルで最初に決定可能なものは、状態識別子、イベント識別子、遷移条件識別子、アクション識別子である。以下、それぞれの場合についての決定順序関係を見る。

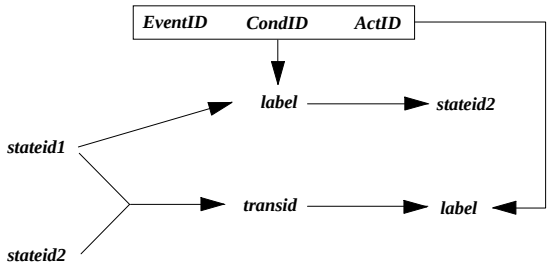

図 4.5: 状態から抽出を始めた場合の決定順序関係

状態識別子から

状態識別子の定義から開始した場合、次に行えることは別の状態との間に遷移を張るか、その状態についての動作を考えることである。ここで言う「動作」とはどの様なイベントを受け取った時に、どの様な条件下で、どの様な出力をして、またアクションを起こすかと言うことである。 今これらをまとめて $label$ としておく。$label$(の一部、または全部) を決定すれば、遷移先の状態が決定できる。状態識別子から抽出を開始したときの順序関係を図 4.5に示す。

以上からこの場合の決定順序を定義する。

- **定義した状態に対して先ず遷移を張る場合**
  - **状態遷移識別子**

    状態遷移識別子は、２つの状態が定義されているときのみ定義可能である。つまり以下が言える。

    $$\{stateid_1, stateid_2\} \prec transid$$

  - **ラベル**

    ラベルは既存のイベント識別子、遷移条件識別子、アクション識別子のみを用いて構成される。また、これらのラベルは定義された遷移識別子に対して定義される。

    $$\{EventSet, condid, acttid\} \prec label$$

    $$transid \prec label$$

$$where \qquad EventSet \subseteq \mathbf{EventID}, condid \in \mathbf{CondID}, actid \in \mathbf{ActID}$$

- **定義した状態に対して先ずラベルを定義する場合**
  - **ラベル**
    上記と同様。
  - **状態識別子／状態遷移識別子**
    ラベルが決定されている場合は暗黙に状態遷移が定義されているに等しい。つまりそのラベル内の動作如何により次の状態が生じるので、後は遷移先の状態識別子を決定すれば良い。従って以下の様に定義する。

$$label \prec stateid_2$$

イベント識別子から

ラベル内要素について、遷移条件識別子、アクション識別子についてはその状態遷移図内で閉じていて、且つ特殊な状態である初期状態が予め定義されているので、この状態に対して上記「 状態識別子から〜」の場合と同様のやりかたをして分析を進めることになる。しかしイベント識別子については状態遷移図に跨って定義されているものであり、ある状態遷移図中で入力イベントとなっているものは、別の状態遷移図中で出力イベントとなっている。

イベント識別子の定義から開始した場合は、先ず単一状態遷移図内だけで定義して、それを元に他の状態遷移図中のイベント識別子を決定していくか、イベントトレース図などを用いて、状態遷移図（若しくは基本オブジェクトモデル中のオブジェクト）間のイベント通信を決定するかである。

4.4.3 基本機能モデルにおける決定順序

基本機能モデルでは、データフロー識別子以外のものから定義を開始出来る。以下各場合について、その場合の決定順序を定義する。

ノードから

ここでノード $(node)$ とは、プロセス識別子かアクター識別子、若しくはデータストア識別子を指す。ノードとなる要素から抽出を始めた場合、次に行えることは別のノードとのデータのやりとりを考えるか、そのノードがどの様なデータを受け取り、どの様なデータを出力するかを考えることである。この場合の順序関係を図 4.6に示す。

以下、各場合についてその決定順序関係を定義する。

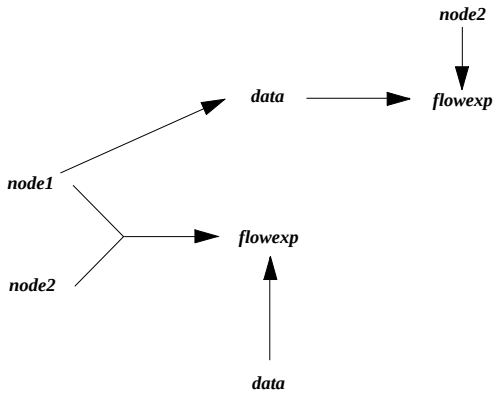

図 4.6: ノードから抽出を始めた場合の決定順序関係

- **フロー式**

  先ず、ノード $(node)$ の定義を形式的に記述しておく。

$$\begin{aligned} & node \in Node \\ where \quad & Node = P \cup A \cup S \\ & P \subseteq \mathbf{ProcessID}, A \subseteq \mathbf{ActorID}, S \subseteq \mathbf{StoreID} \end{aligned}$$

  このノードに対して、そのノードの入力（又は出力）となるデータを決定し、既に定義されている別のノードとの間でデータフロー式を構成することになる。つまり以下の順序関係が成立する。

$$\begin{aligned} & \{node_1, node_2, data\} \prec flowexp \\ where \quad & flowexp = ((node_1, node_2), data) \end{aligned}$$

4.4.4 統合写像における決定順序

前章の分析から、統合写像の決定の仕方には２通り存在していることが分かった。１つは各基本モデルを決めてから統合写像を決定する場合であり、もう１つはある基本モデルを主導モデルとし、その決定内容に沿って他の２つのモデル決定する場合である。特に後者の場合は、主導モデルが基本オブジェクトでない場合、統合写像の決定順序は変化する。

先ず前者の場合でも後者の場合でも、次の決定順序は同じである。つまり基本動的モデル及び基本機能モデル中の要素の統合写像を決定するには、その要素の識別子が定義されていなければならない。

$$itemid \prec Map(itemiid)$$

$where$ $itemid$ ： 基本動的モデル又は基本機能モデル中の要素の識別子

Map ： 要素の統合写像決定

３つの基本モデルを独立して構築した後に、統合写像を決定していく場合は、以下のことが定義出来る。つまりこの時、上で定義したことに加えて、その要素の統合写像を構成する基本オブジェクトモデル中の要素も、統合写像決定より先に決定されていなければならない。

$$OMitem, itemid \prec Map(itemid)$$

$where$ $OMitem$ ： 基本オブジェクトモデル中の要素で、$item$ の統合写像を構成するもの

これに対して主導モデルを決定して、それが基本動的モデル又は基本機能モデルであった場合、以下のことが定義出来る。

$$itemid \prec Map(itemid) \prec OMitem$$

第 5 章

分析支援環境

5.1 支援環境に要求される機能

FO∀M を支援する環境を考えた場合、先ずこれまでにに述べた静的関係、スコープルール、そして決定順序関係を反映したのもでなくてはならない。静的関係は分析モデルを構築して行く際、モデルがどの程度完成したかを示す指標となる。またスコープルールは基本動的モデル、機能モデルの統合写像を決定する際に、基本オブジェクトモデルで実現されているカプセル化を反映させるために用いられる。決定順序関係は、ある要素定義の完成度の指標である静的関係に至るまでの過程を支援するために用いられる。

次に、大規模なシステム分析を想定した場合、一度に仕様中の全ての概念を抽出することは不可能である。従って、モデル構築は段階的に行われ、途中段階のモデル情報を保持出来ることが望ましい。

最後に要素の削除に関して、これには細心の注意が必要である。これは FO∀M に於いて多くの概念が用いられていることよる。

以上から、今回作成される支援環境に要求される機能をまとめると以下のものがある。

- 静的関係を用いて完成度の指標を与える。
- スコープルールを用いて統合写像決定を支援する。
- 決定順序関係を用いてモデル構築の手順を与える。
- 要素の削除によりモデル情報が崩壊しない。
- 途中段階のモデル情報を保持出来る。
- 各モデルに対するバージョン管理が行える。

これらの要求に従い支援環境を設計／実装する。関係／ルールはこれまでに決定したものに従うとし、次節ではまず削除方針を決定する。次に途中段階のモデル情報を保持するための手段としてのデータベースについて解説し、開発言語として用いる Java を取り上げる。最後にバージョン管理について言及する。

5.2 支援環境の設計

5.2.1 削除方針

FO∀M では多くの概念が用いられておりそれを写像により対応付けている。従って写像を決定した後にある要素を削除した場合、その影響が波及してしまうことがしばしばある。例えば以下のような定義がなされていることを仮定する。

$$
\begin{aligned}
& classexp = (attrid, funcid), \mathcal{CM}_{class}[ClassSet](classid_1) = classexp \\
& inherexp = (classid_2, (classid_1)), \mathcal{CM}_{inher}[ClassSet](inherid) = inherexp \\
& assocexp = (classid_1, classid_3), \mathcal{CM}_{assoc}[ClassSet](associd) = assocexp \\
where \quad & classid_1, classid_2, classid_3 \in \mathbf{ClassID} \\
& attrid \in \mathbf{AttrID}, funcid \in \mathbf{FuncID} \\
& inherid \in \mathbf{InherID}, associd \in \mathbf{AssocID} \\
& inherexp \in InherExp(ClassSet), assocexp \in AssocExp(ClassSet).
\end{aligned}
$$

これを図示すると 5.1の様である。

この時、$classid_1$が削除されたとする。もしその影響が写像にも影響するとしたならば、多くの場所で関係が崩れてしまう。この時の様子を図示すると図 5.2に示す。

これはオブジェクトモデル中で、多くの要素が **ClassID** を参照していることによる。また決定順序まで考慮した場合は FO∀M で定義されている静的な関係だけでは不十分で、例えば継承関係は最終的にはクラス識別子に対して定義されるとしても途中過程ではクラス式の間に関係を張りたい場合もあるかもしれない。従って、先ずこれらの問題を解決する為に新規に $classframe$ を導入する。クラスフレームはクラス式の拡張であるが、クラス識別子への参照を含むものとする。つまり以下の様に定義される。

$$(classid, AttrSet, FuncSet)$$

また関係はこのクラスフレームに対して張られるものとする。これによりクラス識別子が削除された場合でも、属性識別子集合と、操作識別子集合の構造は残る。つまり、クラス式に対しても関係が定義出来ている。

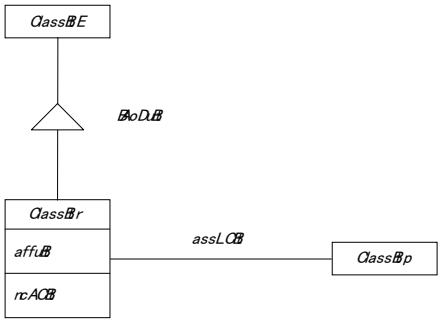

図 5.1: 削除前のオブジェクトモデル

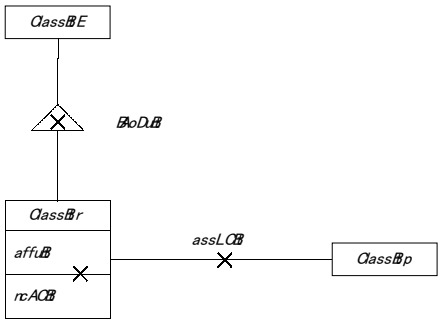

図 5.2: 削除後のオブジェクトモデル

これに付随して、特別な識別子（若しくはその集合）$Undef$を導入する。モデル構築の過程で、未定義のままで残しておいた部分は$Undef$識別子が割り当てられるものとする。例えば上のクラスフレームに於いてクラス識別子を未定義にしておいた時、そこに$Undef$が割り当てられ、実際にはクラス式を宣言したのと同等である。

以上を基にして削除の方針を決定する。今回目標とする支援環境では、以下のルールを適用することにする。

1. ある要素の削除で影響を受けるのは、その要素を直接参照している要素のみ。
2. 削除された部分には$Undef$を割り当てる。
3. 最初に定義する時に未定義のままの残された部分についても$Undef$を割り当てる。

以上のルールに従い、例として基本オブジェクトモデル中の継承関係における削除を考える。継承関係について以下が定義されているとする。

$$
\begin{aligned}
& classframe_1 = (classid_1, attrid_1, funcid_1), classframe_2 = (classid_2, attrid_2, funcid_2) \\
& inherexp = (classframe_1, (classframe_2)), \mathcal{CM}_{inher}[ClassSet](inherid) = inherexp \\
where \quad & classis_1, classid_2 \in ClassSet(\subseteq \mathbf{ClassID}) \\
& attrid_1, attrid_2 \in \mathbf{AttrID}, funcid_1, funcid_2 \in \mathbf{FuncID} \\
& inherid \in \mathbf{InherID}, inherexp \in InherExp(ClassSet)
\end{aligned}
$$

継承関係はクラスフレーム間に張られているものとする。今、クラス識別子集合中からクラス識別子$classid_1$が削除されたとする。この時、$classframe_1$の定義のみが影響をうける。つまり識別子の定義がなされていないことを示す特別な識別子$Undef$を用いて、以下の様になる。

$$classframe_1 = (Undef, attr_1, funcid_1)$$

しかし継承関係自体はクラスフレームを参照しているため、その内容は変化しない。継承関係が従来通りクラス識別子間に定義されるものであるとすれば、当然クラス識別子の削除は継承関係まで波及してしまう。この場合、継承式の構造自体が崩れる為、結局そのクラス識別子が対応付けされているクラス式も分からなくなる。よって統合写像の決定でのスコープルールが変化してしまい、削除影響の波及範囲はかなり広いものとなってしまう。

これらを図示すると、図5.3の様になる。

5.2.2 データベースの導入

前に述べた通り、途中段階での中間的なデータを保持する為の手段としてデータベースを用いる。データベースを用いる事によりデータの永続性が確保でき、またオブジェクト指向開発に特有のスパイラルな開発

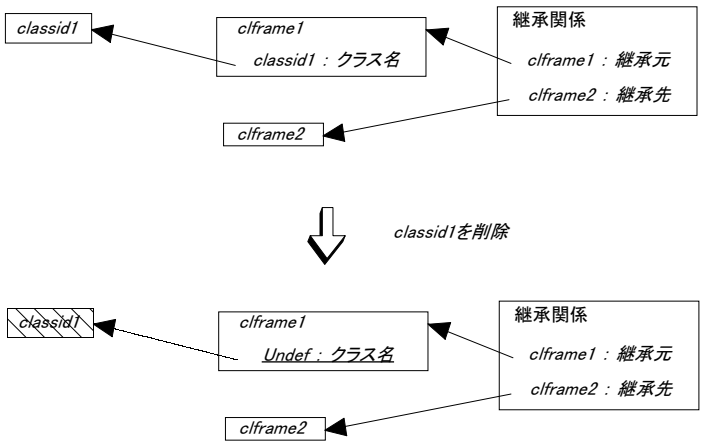

図 5.3: 継承関係における削除例

への対応が可能となる。

今回作成する環境を用いて蓄積されたデータは、他の FO∀M の支援環境（検証支援環境など）でも使用可能であり、更に分析以降の開発プロセスでも用いられるものである。

データベース中でのテーブルの定義

データベース中では、各基本モデルの識別子集合をテーブルとして与える。その際、フィールドとしてはそれらの識別子として必要な情報を定義してゆく。例えば基本オブジェクトモデル中のクラス識別子集合を含むテーブルを定義する際には、テーブル名として **ClassID** を定義し、その要素としてはフィールドとして $name$ を定義した。また継承識別子の様にクラスフレームを参照する様な場合は、その情報を表すフィールドとして $name, src, dst$ を定義する。ここで $name$ は継承識別子名、src, dst は継承元クラスフレーム、継承先クラスフレームを表す。今クラス識別子 $classframe_1, classframe_2$ の間に以下の関係があるとする。

$$inherexp = (classframe_1, (classframe_2)), \mathcal{CM}_{inher}[ClassSet](inherid) = inherexp$$

$$where \qquad inherexp \in InherExp, inherid \in \mathbf{InherId}$$

この時、データベースには表 5.1 の情報が付加される。

他の要素についても基本的にこれと同様の定義の仕方を行う。また、各テーブルには主キー $(Primary\ Key)$ を設定し、表中での重複を防ぐ。例えば上記の継承識別子集合を表すテーブルでは主キーを $name$ に設定する。属性識別子集合の様にその中の識別子名に重複が許されるべきもの（複数のクラスで同じ属性の定義

がなされているかもしれないので）については、主キーとして $name$, 意味をとることにする（つまり識別子名とその意味を取ればテーブルの中で一意に識別出来るので）。

データベースからのデータの削除

削除に関しては前に述べた通りであり、データベース中ではこの削除方針に従った削除を行わなければならない。データベース中の、他の要素から参照される様なテーブル（つまりそれ自身主キーをもっており、それが他の表の外部キーとなっている場合）の中には上記で定義した特別な識別子 $Undef$ を入れておき、ある要素が削除された場合は $Undef$ を参照する様な仕組みが必要となる。また更新に関しては、主キー中のデータが変更されれば、それを参照する外部キー中のデータも更新されなければならないが（更新の $Cascade$ 操作）データベースソフトによってはこれをサポートしないものも存在するので、開発言語側でカバーする必要がある。（例えば **Oracle** では $Cascade$ は削除の時のみ使用できる。また **Postgres** ではキーの概念自体がまだ実現されていない。）

5.2.3 バージョン管理

モデルを構築して行く過程で、モデル構築中に仕様の一部が変化したり、モデル情報の大部分を書き換えてしまわなければならない場合が存在するかもしれない。 この様な場合に備えて、支援環境ではバージョン管理機構を導入する。

先ずバージョンは各基本モデルについて付けることが出来るものとし、管理の構造としては木構造を適用する。これを図 5.4に示す。

最初、何もバージョニングを行わない場合には $Version$ 0 のまま進行する。以下、その子孫を $1 \cdots m$ の用に置く。新たにバージョンを作った場合、子には親のバージョンに含まれている情報がコピーされるものとする。但し、飽くまでコピーであって継承ではない。親の要素が変えられたとしても、子は影響を受けないものとする。また、親のバージョンが消去された場合は、その子孫は全て消去されるものとする。

上記は各基本モデルについてのバージョン管理の方針であるが、統合写像についてもバージョン管理が出来るものとする。構造は上記と同じものを採用するが、ポリシーは多少異なる。統合写像を構築する際には、基本オブジェクトモデルと基本動的モデル（若しくは基本機能モデル）から、任意のバージョンを取ってきて、この間に統合写像を構築できるものとする。（図 5.5）

表 5.1: データベース中の継承関係テーブル

name	src	dst	意味
inherid	$classframe_1$	$classframe_2$	（継承関係の説明）

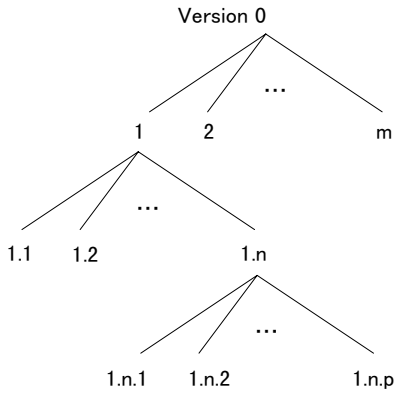

図 5.4: バージョン管理木

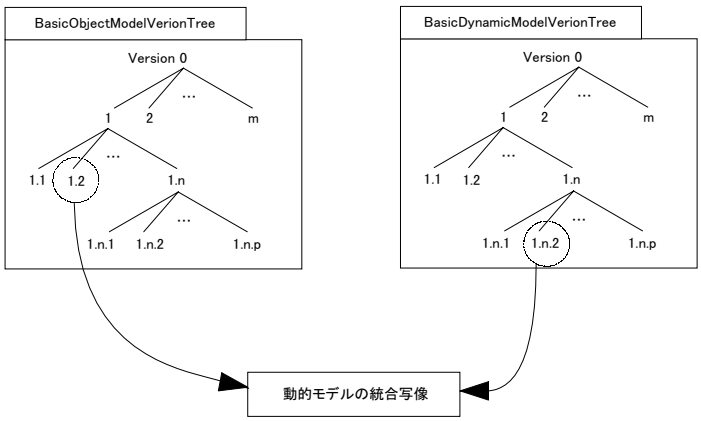

図 5.5: 動的モデルの統合写像構成例

但し統合写像の構成における１つのバージョンは基本オブジェクトモデル、基本動的モデル、基本機能モデルの各々のあるバージョンの組で識別するものとする。また、上記の各基本モデルのバージョンを選択出来るのは、$Version\ 0$ の直下の子ノードのみであるとする。例えば $Version\ 0$ の子として $Version\ 1$ を作った場合、この段階では統合写像の構築に使用する基本モデルのバージョンを選択出来るが、それ以下の子ノード (例えば $Version\ 1.1$ など) に使用される基本モデルのバージョンは $Version\ 1$ の時に選んだものを用いる。但し、子バージョンを作った時点で親バージョンの情報がコピーされるのは基本モデルの場合と同様である。

5.2.4 開発言語

開発言語として、Java を使用する（JDK1.1 ベース）。Java を用いることの利点には以下がある。

- JDBC API によるデータベースとの結合性
- 分散開発の可能性
- マルチプラットホーム対応

JDBC($Java\ DataBase\ Connectivity$)API は各データベースソフトベンダーから提供される。JDBC を用いた場合、データベースに対する操作は基本的に **SQL** 文を発行することにより行われる。また同じような操作を繰り返し行う様な場合には、**SQL** 文にパラメータを渡すやり方も提供されている。

5.3 Java を用いた支援環境の構成例

前節で決定したことを元に、支援環境を構築する。支援環境の全体像を図 5.6に示す。

ユーザーインターフェースからは先ず編集するモデル、及びそのバージョンが選択される。その情報をもとに JDBC を通してデータベースに接続し、必要なテーブル情報を取ってくる。$Converter$はデータベース中のデータを Java 側で使用出来る形にするためのものである（又はその逆)。

モデル情報の編集の際には決定順序関係を考慮する。実際見てきた様に各モデルに対して複数の決定順序が考えられたが（例えば基本オブジェクトモデルではクラス識別子から決定しても、属性／操作識別子から決定しても良かった)、環境側ではこれらに広く対応するようにする。具体的には $Undef$を用いて静的関係を依然満たしていないものを通知することで対応する。更に、例えば基本オブジェクトモデルでの関係の様に、クラス識別子に対してもクラス式に対しても定義するかもしれないものは $ClassFrame$ を導入することで対処する。ユーザーがアイテムを削除（更新）した場合はその情報を元に適切な **SQL** 文を生成し、これを **JDBC** を介してデータベースに送る。

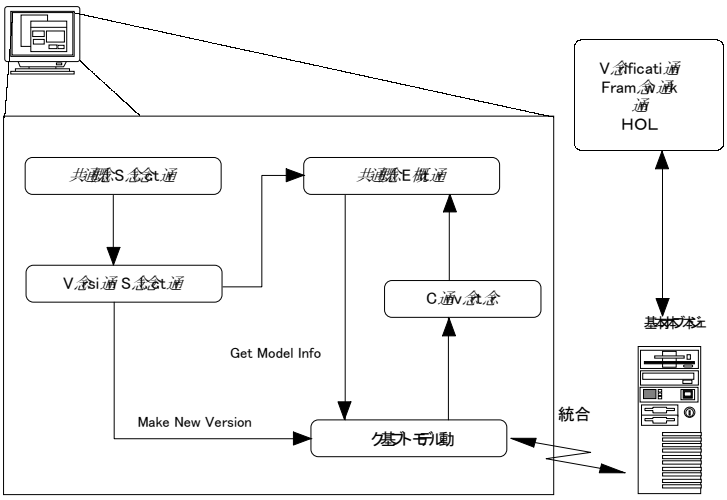

図 5.6: 支援環境の全体像

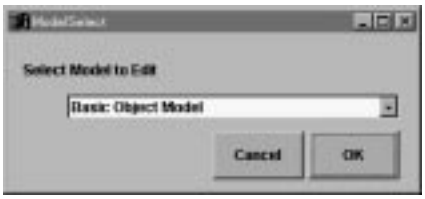

図 5.7: モデル選択用 GUI

例えば基本オブジェクトモデル中の継承識別子集合の編集（読み込み）をする場合を考え る。先ず上記に従い、$Model\ Selector$を用いて基本オブジェクトモデルを選択する。$Model\ Selecter$の様子を図 5.7に示す。

次に基本オブジェクトモデルについて、定義されているバージョンの中から用いるものを$VersionEditor$を用いて選択する（図 5.8）。

最後に、このバージョンに対応する基本オブジェクトモデル中の継承関係の編集を選択する。静的関係より、継承関係の定義の最終目標は選択された２つのクラスフレーム間に関係を定義することである。また決定順序関係より、継承関係の定義には少なくとも一方のクラスフレームが定義されていないとだめなので（つまり$Undef, Undef$間では意味がないと言うこと）、継承先、継承元が共に定義されなかった場合はエラーとなる。継承関係の編集画面の様子を図 5.9に示す。

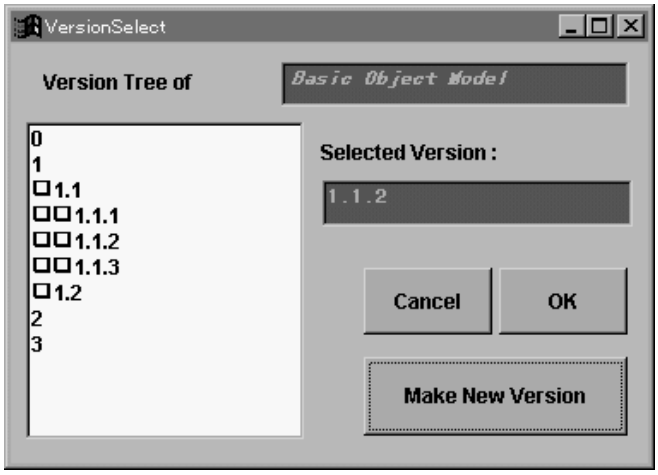

図 5.8: バージョン操作用 GUI

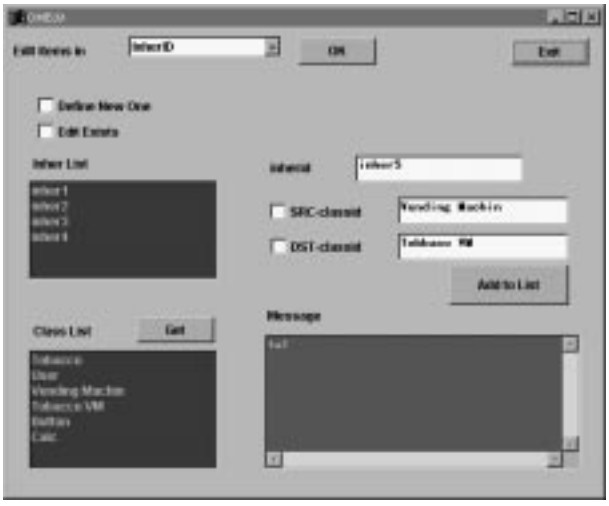

図 5.9: 継承関係編集用 GUI

第 6 章

終りに

6.1 まとめ

本論文では、先ず形式的オブジェクト指向分析モデル (FO∀M) の概念について説明し、次に FO∀M を支援する環境を構築する準備として、自動販売機を例にとり各視点からの分析を試みた。この分析結果から分析モデルを構築する際には幾つかの関係やルールを明確にしなければならないことが明らかになった。それはモデルの完成度を示す静的関係、統合写像決定の際に考慮しなければならないスコープルール、モデル構築の指針を与えるための決定順序である。4 章ではこれらを抽出し形式化を行なった。

実際に支援環境を構築するに当たって、上記の関係／ルールを適用すると同時に途中段階の情報を保持する手段として関係データベースを導入し、データベース中で FO∀M の概念をどのようにテーブル化すべきかを説明した。また要素を削除する場合の方針と、データベースを用いた場合のこの削除方針の実現法を述べ、バージョン管理を導入する際の方針を与えた。最後に開発言語として **Java** を用いた場合の、支援環境の構成例を示した。

6.2 今後の課題

支援環境を完成させ、実際に分析を行なってみて支援環境の評価を行なう必要がある。

データベースソフトに関して現在は **Postgres** を使用しているが、性能を向上させるために市販のもの（**Oracle** など）への移行を検討中である。また開発を継続するに当たってデータベース中のテーブル定義やプログラミングに関する部分を文書化する必要がある。

謝辞

本研究を進めるに当たり終始御指導賜りました片山卓也教授、鈴木正人助手、青木利晃氏に深く感謝致します。また、数々の助言を頂きました片山研究室の皆様に感謝致します。

参考文献

[1] 青木利晃： オブジェクト指向方法論のための形式モデル, Master's thesis, Japan Advanced Institute of Science and Technology, 1996.

[2] 青木利晃, 石田至, 古川順一, 片山卓也： オブジェクト指向分析モデルにおける一貫性検証のための公理系の実装, ソフトウェア科学会 第14回全国大会, p465–468.

[3] Ramgaugh,J., Blaha,M., Premerlani,M., Eddy,F. and Lorensen,W. : Object-Oriented modeling and design, Prentice-Hall International, 1991.

[4] Jacobson,L., Christerson,M., Jonnson,P., Overgaard,G, : Object-Oriented Software Engineering, Addison-Wesley, 1992.

[5] Winskel,G. : The Formal Semantics of Programing Languages, The MIT Press, 1993.

[6] 山本順一, 大須賀昭彦, 本位田真一： 代数仕様技術によるオブジェクト指向分析設計の検証支援, 情報処理 Vol.36 No.5 p1070–1079, 1995.

[7] Bernd B., Jim,B., Jeffry,J., and Jeff,S. : Object-Oriented System Modeling with OMT, OOPSLA '92 pp359–376 1992