修 士 論 文

CafeOBJ から Java への変換と同期記述

指導教官　二木厚吉 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

兼　英樹 710026

1999 年 2 月 15 日

目 次

第 1 章

はじめに

要求仕様からコーディングまでのソフトフェア開発のさまざまな局面において形式手法に基づいた活動が盛んに行われている。形式手法の仕様作成段階で用いられる言語は形式仕様言語と呼ばれ、厳密な数学モデルや論理体系に基づいて仕様を記述することができる。これは、より信頼性の高い仕様を作成することができ、仕様の機械的検証も可能であるためソフトウェア開発の効率面でも多くの利点を持っている。

形式仕様言語 CafeOBJ[1][3] において、射影演算を用いたオブジェクト合成はシステムの振る舞いを非常にうまく記述することが出来、その分析も簡単に行うことができる。

このスタイルの仕様は以下の特性を持っている。

1. 単純なプリミティブモジュールから合成されている。
2. 仕様の分析は簡単に行え、CafeOBJ インタープリタのような自動ツールで支援することができる。

また、近年 Java 言語はその言語としてのすぐれた特性、プラットフォーム非依存性により、急速に開発言語として導入されるケースが増えている。しかし、Java 言語は数学的意味の裏付けを持たないため、設計段階で検証を行うことが難しい。

形式言語から java 言語への変換システムが作成されれば、設計段階で検証を行い、その仕様を忠実に反映した Java コードを得ることができるため、システムの信頼性を向上させることができる。

本稿では合成対象のオブジェクトの仕様コードと振る舞い等価 [2] の証明が再利用できる射影演算 [3] を用いたオブジェクト合成において、

形式仕様言語 CafeOBJ 仕様から仕様の構造を保存しつつ、効率的な Java プログラムへ変換する方法を示し、実際に並行動作させることを念頭において同期記述について述

べた。

また、関連研究として、CafeOBJ の仕様の変更がどの範囲まで影響するかを調べ、低コストで仕様を変更できるように考慮した。

一般に、ソフトウェアの要求が変化した場合にはその仕様を変更し、変更された仕様において要求が満たされていることを証明しなおす必要があるが、もし要求の変化がどのコンポーネントに影響し、そのコンポーネントを他のコンポーネントと置き換えた際にどんな性質を検証すれば全体として全ての要求を満たすことができるのかが分かれば大幅なコスト削減が期待できる。

第 2 章

背景知識

本章では、形式仕様の基本概念、および仕様言語、本研究で用いた代表的な仕様言語である CafeOBJ の説明を行なう。

2.1 形式手法

形式手法とは、ソフトウェアの仕様作成から最終的なプログラムコードの作成までを一貫して数学的議論に基づいた形式的な手法で行なう手法である。形式手法はプログラム方法論から生まれたアイデアを利用する方法であり、現在もソフトウェア工学の一分野として研究が進められている。形式手法の一連の工程の中で、その上流工程である仕様記述の段階で、矛盾のない完璧な仕様を作成するための形式仕様の役割は益々その重要性を増している。実際、ソフトウェア開発の上流工程において作成される仕様の品質が、完成したソフトウェアの品質の優劣に大きく影響し、仕様段階での早期のバグ発見は開発効率の向上に繋がる。形式手法の中で、形式仕様を作成する際に用いる形式言語が仕様言語である。

2.2 形式言語

仕様言語とは、前述のように形式手法の上流工程でソフトウェアの仕様を作成する際に用いる形式言語のことである。仕様言語とは、その基盤に必ず厳格な数学的手法を持つという点で、通常のプログラム言語とは性質の異なるものである。仕様言語としては、多種多用な数学モデル、数理体系に基づく言語が多く研究され提案されているが、その中でも、Z[5],OBJ[7],CafeOBJ,VDM,Larch などは実用化され有名である。ここでは、多くの

仕様言語の中から特に、代数的手法に基づき仕様の機械的検証が可能である CafeOBJ について取り上げる。

2.3 OBJ

OBJ は、UCLA で Gouguen によって開発された代数仕様言語 [6] である。OBJ は厳密に抽象データ型の考え方を順序ソート代数と等号論理による数学モデルで裏付けした代数型言語である。順序ソート代数は、抽象代数である多ソート代数を拡張したモデルで、ソートに包含関係を定義できる。また、OBJ は順序ソート代数と符合論理を基盤としている一階の関数型言語としての一面を持ち、仕様をそのままの状態で機械的に実行可能である。また、検証面で多くの利点を持った仕様言語である。

OBJ では、パラメータ化機構が洗練されており、汎用的なパラメータ化モジュールを定義することが出来る。これにより、より抽象的な再利用性の高い仕様を記述することが可能である。また、他のモジュールを輸入でき、モジュールの和をとることができるなど、モジュールを構成する方法も豊富にある。

OBJ を用いて記述を行なう場合は、モデル化の際に集合や類似のものを用いるのではなく、直接振舞をとらえる方程式の集合によって仕様記述を行なう。この記述特性により代数仕様言語は性質指向であると言われる。

代数仕様言語の一番の優位点は、仕様の実行が可能であることである。OBJ でも等式を書き換え規則として解釈する項書き換えシステムにより記号的に実行可能であり、仕様の機械的な検証が可能である。

2.4 CafeOBJ

CafeOBJ[1] は等号論理を拡張した順序ソート書き換え論理に基づいた代数仕様言語である。CafeOBJ は従来の OBJ 言語と比較して、オブジェクト指向プログラミングの技法での記述を可能にするクラス宣言や書き換え規則を導入し、システムの動的な振舞の記述も用位に行なえるように拡張がなされている。また、代数仕様の新しい風潮である Hidden Algebra[2] に基づく記述手法も導入されている。CafeOBJ で記述した仕様は、OBJ の場合と同様に項書換えシステムにより機械的な検証が可能である。CafeOBJ では、module 単位で記述を行ない、module は型宣言などを行なう signature 部と等式やルール宣言を記述する axiom 部からなる。CafeOBJ は、以上のような特徴により、他の形式仕様記述言語に対し、次のような優位点を持つ。

- モジュールのパラメータ化、モジュールの輸入の際の名前の付け替えが可能で、強力なモジュール化機構を持ち、抽象度、再利用性の高い仕様記述が可能である。
- 項書換えシステムにより、仕様をそのまま機械的に実行可能であり、仕様記述段階でプロトタイピング可能である。
- 順序ソート代数に基づき、ソート間に包含関係が定義でき、例外処理や演算の多重継承を自然に扱える。
- 動的なシステムの変化を記述できる書き換え規則を持つ。
- クラス構文を持ちオブジェクト指向モデルの記述にも対応している。
- 演算の引数の位置を自由に指定できる。
- Hidden Algebra に基づいた記述手法が可能である。

2.5 Java

Java[11][12][13] はその技術良し悪しよりも、コンセプトが優れているという点と、インターネットのブームによってネットワークベースのアプリケーションのニーズが高まった事に引きずられて、過去の技術では達成できなかったパラダイムの転換を実現するものとして大きな注目を浴びている。

Java はプログラマから見れば単なるオブジェクト指向型のプログラミング言語と言える。C++、Smalltalk、Lisp などの既存プログラミング言語から良い部分だけをを採用し、またそれらが有していた不都合な部分や、複雑な部分は排除している。Java 言語の構文は既に普及している C や C++のモデルを採用しているため、C や C++を経験しているプログラマであれば比較的容易に修得することができる。また、Java では、実装環境によって不明瞭であった型を定義付けたこと、ポインタの排除、goto 文、関数、多重継承などを排除したことなどにより言語構造をシンプルにする配慮が加えられており、初心者でも比較的バグの少ないアプリケーションの作成が可能である。また、Java ではコンポーネント・モデルも定義されている (JavaBeans)。何年も前からよりオブジェクト化とコンポーネント・モデルによるアプリケーションの再利用の有効性が唱えられていた。しかし、特定プラットフォームのみのサポートができなかったり、定義があいまいで容易な再利用ができない等の理由により現実に利用者がその恩恵を得ることは少なかった。Java はプラットフォームをこえるコンポーネント化を実現する手段としても有効である。また、

Java はインタプリタ型の言語として設計されており、各プラットフォーム上に実装された Java ランタイム・システム上で動作する。各プラットフォームで Java ランタイム・システムがサポートされていればバイトコード (Bytecode) の形態にコンパイルされた Java のアプリケーションは機種に依存する事なく実行可能である。プログラマや利用者は個別のプラットフォーム毎にアプリケーションを用意する必要は無くなり、Java だけを意識して作成すれば良い。

第 3 章

オブジェクト合成

本章では、振舞等価の基本概念、それを用いたオブジェクト合成方法についての説明を行なう。

3.1 振舞等価

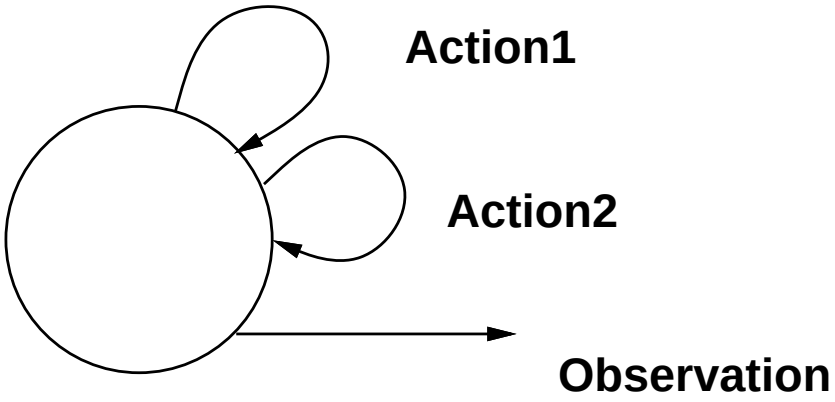


図 3.1: 振舞仕様

システムをブラックボックスと見なし、その振舞いを記述した仕様を振舞仕様と呼ぶ。システムがブラックボックスなので、その状態は観測を通してからしか知ることができない。この観測とは method を繰り返し適用させた後に attribute を適用することである。method を m, attribute を a とすると、
観測は $a\ m_n..m_1\square$ と表現でき、この観測を観測分脈と呼ぶ。

s,s' をシステムの状態、
Allc を観測文脈全体の集合とすると
システム状態間の等価関係 (振舞等価) は

$$s \backsim s' \ iff \ \wedge_{c \in Allc} (c(s) = c(s')).$$

と表現される

振舞仕様は loose semantics に基づいており、これはいくつかのモデルが存在することを示している。

以下に示した add の commutativity を示したいとする。(ここで$'n : Int$ は sort Int の項$'n$ とする)

$$add('n : Int, add('m : Int, init)) = add('m, add('n, init))$$

内容は m と n の加算であるが、左辺と右辺では加算順序が逆になっている。

history を記憶するモデルを history model, 最終状態のみ記憶モデルを cell モデルと呼ぶとする。通常の等式では history model は add の commutativity の性質を持っていない。しかしこの厳しい制限を緩和したい場合、振舞等価が使われる。振舞等価の基本概念は、全ての観測手段を用いても区別できないとき、それは等しいと見ることである。

振舞等価を context の構造を用いた induction で直接証明する方法は context induction と呼ばれる。大きな仕様に対してこの context induction は非常に複雑な証明となる。このような場合、coinduction と呼ばれる方法が用いられる coinduction による証明は以下のステップで行われる。

1. 隠蔽合同関係 R の候補を与える
2. 全ての behavioural operators に関して R が振舞合同であることを証明する
3. R を用いて behavioural property を証明する

3.2 オブジェクト合成

合成対象の各オブジェクトの状態を得るために射影演算 (projection) という演算を定義する。この射影演算は合成対象となっている各オブジェクトに対して定義され、合成されたオブジェクトに対する (操作、属性) 演算が合成対象のオブジェクトに対する (操作、属性) 演算になるように置き換える演算である。射影演算を用いたオブジェクト合成では合

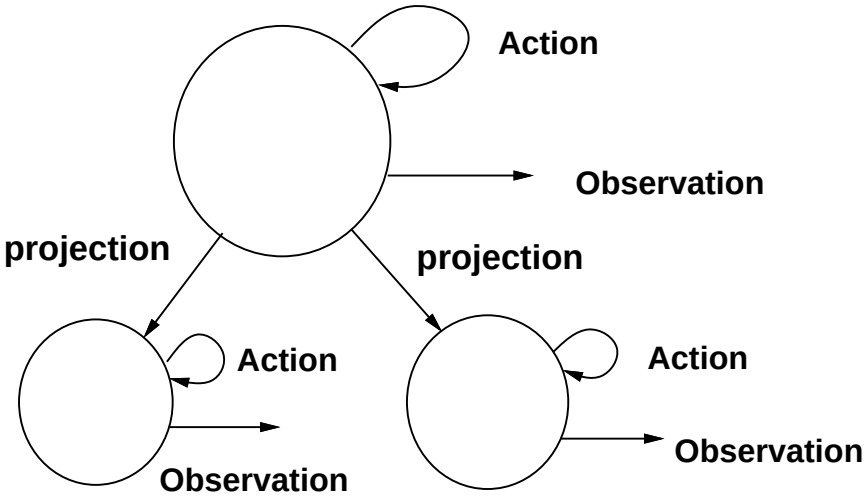


図 3.2: オブジェクト合成

成対象のオブジェクトの記述と振る舞い等価の証明が再利用できるため、検証における手間が大きく省かれる。

定義： $\pi_n : ID_n\ h \rightarrow h_n$は以下の条件のとき射影演算である。

- h は合成されたオブジェクト Oの hidden sort である
- h_nは合成されるオブジェクト O_nの hidden sort である
- ID_nは class n のオブジェクトに対する識別子の集合である
- Oの各 attribute に関して、オブジェクト O_n、オペレータ $f : v_{n_1}...v_{n_m} \rightarrow v(v_{n_m}$は O_nのデータで vは visible sort である)、$a = (\pi_{n_1}; c_{n_1}, ..., \pi_{n_m}; c_{n_m})f$のような visible $O_n - context c_n$が存在する
- Oの各 method $m : h \rightarrow h$ と、合成されるオブジェクト O_n関して、$m : \pi_n = \pi_n; m_n$ のような method m_nの sequence が存在する。
- Oの各定数 $const :\rightarrow h$ と、合成されるオブジェクト O_n関して、$const; \pi_n = const_n$ のような定数 $const_n :\rightarrow h_n$が存在する

合成されたオブジェクトの状態を s,s', 振舞等価を〜とすると、合成オブジェクトでの振舞等価は以下のように表現できる。

$$s〜s'\ if\ \pi_n(S)〜_n\pi_n(s')\ for\ all\ n \in CObj$$

ここで、CObj は合成されるオブジェクトのラベルの集合、$\sim_n$は合成されるオブジェクト O_nの振舞等価,π_nは O_nの射影演算である。

もし、全ての射影演算が behavioural なら

$$s \sim s' = \pi_n(S) \sim_n \pi_n(s') \ for\ all\ n \in CObj$$

となる。

第 4 章

CafeOBJ 仕様からの Java プログラム生成

仕様の構造を保存しつつ、CafeOBJ 仕様から効率的な Java Code への変換システムを提案し、仕様 (CafeOBJ) の変更を体系的に Java のプログラムに反映させることを考える。

ここで扱う CafeOBJ の仕様は projection を用いてオブジェクト合成されたものとする。

$$\text{CafeOBJ} \quad \rightarrow \quad \text{Java}$$

ただし，一般的な CafeOBJ 仕様を対象とするのではなく，射影演算を用いて階層的に合成された振舞い仕様を対象とし，仕様の構造を保存する効率的な Java プログラムへの変換法を目的とする.

Java に変換することで実行効率の向上が期待され、アーキテクチャ非依存で動作させることができるという利点が挙げられる。また、CafeOBJ で作成した信頼性の高いソフトウェアモジュールを Java で利用できる大きなメリットがある。

CafeOBJ でのモジュールの定義は以下のようなスタイルで表現される。

```
module MODULE{
 importing()
 signature{}
 axioms{}
}
```

MODULE がモジュール名である。importing はモジュールの参照関係を記述する部分で、参照方法に対する制約を持たせた protecting, extending, using の 3 種類の参照方法があ

る。signature の内部ではソートの定義と各ソートの包含関係、及びそのソートに対する演算を定義する。axioms の内部は等式の詳細を記述する部分である。OBJ の意味論的裏付けは、操作的には項書換システムによって、宣言的には順序ソート代数によって成されている。つまりここで定義を行った等式について、宣言的には等式そのものとして、操作的には左から右への書き換え規則に従う。宣言的意味論は記述した仕様に対する数学上のモデルを規定する。CafeOBJ ではこの数学的モデルとして抽象データ型による始代数意味論である。始代数モデルは同型で閉じるという意味で一意に定まり、与えられたシグニチャと等式集合から得られる標準的モデルとなる。また始代数には以下のような性質がある。

- 非冗長性：台集合の全ての要素が、与えられた定数・記号関数で表現できる
- 非混同性：その代数で成立する基底等式はすべて与えられた等式から証明できる

参照関係に対する制約は、他のモジュールの参照時のこれらの性質の守られ型によっている。すなわち、protecting では非冗長かつ非混同が守られた形態をとり、extending では非混同のみが守られるように参照され、using では制約の無いものとして定義されている。

projection で合成されているスタイルでは protecting が用いられ、非冗長かつ非混同が守られている。

projection で合成されている CafeOBJ の仕様に対して、Java のプログラムは以下の一対一の対応で変換できる。

CafeOBJ		Java
module	→	class
protecting	→	import
sort	→	型
method	→	method(戻り値なし)
attribute	→	method(戻り値あり)

method への変換は、単純に

`operation` → `method`

とはならず、eqation の意味論を考慮する必要がある。しかし、operation が projection により下位モジュールの operation へ関連付けられている場合は、意味論を考慮する必要はない。

また、operation が attribute の場合は (java の)method の戻り値で表現できる。

4.1 hidden sort の扱い

CafeOBJ の hidden sort はシステムの内部状態を示すものである。Java ではこの内部状態を示すためのオブジェクト（変数など）を作成することで表現する。

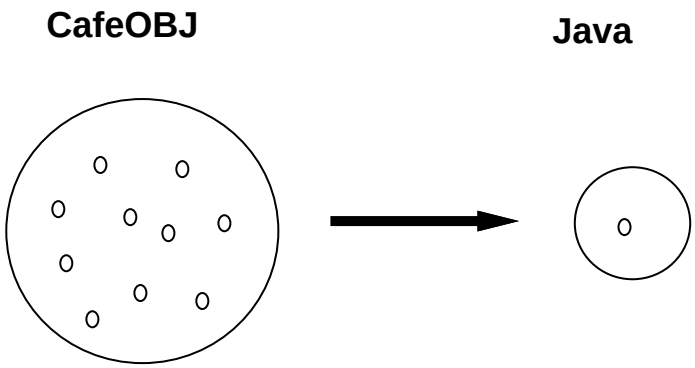


図 4.1: 具体化モデル

変換した Java プログラムは CafeOBJ の hidden sort で表現されている状態をある具体的な状態に決定して変換している。よって、CafeOBJ の仕様で表現されている複数のモデルの中で具体的な１つのモデルを表現していることになる。

CafeOBJ では operator の引数には１つ以下の hidden sort しか書けない（つまり内部状態は１つ以下）ので java に変換する際に問題はおこらない。

例えば状態一つを記憶することができる Cell(図 4.2) では

```
mod* CELL {
protecting(INT)
  *[ Cell ]*

  op undefined : -> Int
  op init-cell : -> Cell      -- initial state
  bop put : Int Cell -> Cell  -- method
  bop get : Cell -> Int       -- attribute
```

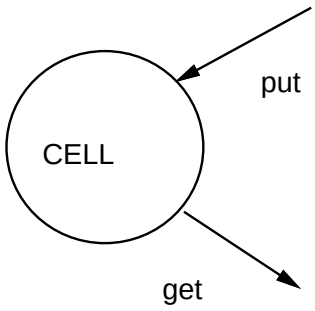


図 4.2: cell

```
  var I : Int
  var C : Cell

  eq get(init-cell) = undefined .
  eq get(put(I, C)) = I .
}
```

signature（ソートの定義と各ソートの包含関係、及びそのソートに対する演算を定義している部分）だけを見ると、put は引数 Int をとって戻り値なし。get は引数なしで戻り値が Int である。

内部状態を表す変数名を cell として、これを Java のスタイルで表現すると

```
int cell
public void put(int x){};
int y;
public int get(){ return y};
```

となる。

内部状態の初期値を（ここでは簡単に）0 とし、put の意味（引数を記憶する）を追加すれば以下の Java プログラムが作成される。

```
public class Cell{
 int cell = 0;
 public void put(int x){
   cell = x;
```

```
  }
 public int get(){
  return cell;
 }
}
```

4.2 projection の扱い

projection は下位モジュールの operator を直接呼ぶことで表現する。より具体的には java のインスタンシエーションを用いて表現する。

例えば次の CafeOBJ 仕様では

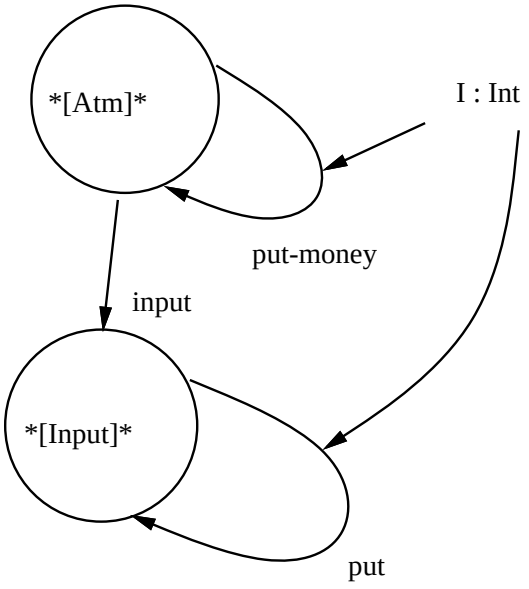


図 4.3: 下位モジュールの呼び出し

```
  *[ Atm ]*
  var ATM : Atm
  var I : Int
  bop put-money : Int Atm -> Atm        -- method
  bop input : Atm -> Input              -- projection
   eq input(put-money(I, ATM)) = put(I, input(ATM)) .
```

projection input は Cell クラスのインスタンス変数 input で表現し、method putmoney ではこの input を用いて下位モジュールにアクセスする。java のプログラムでは以下のようになる。

```
Cell input = new Cell();
public void putmoney(int I){
  input.put(I);
}
```

これまでの例では比較的簡単な変換法則だったが、Stack などのデータ構造を持った表現は直接変換できない。例えば以下のような表現では UId が Stack で管理されており、pop することで検索している。

```
  bop deposit : UId Nat AccountSys -> AccountSys   -- method
  ceq account(U, deposit(U', N, A)) = add(N, account(U, A))
       if U == U' .
  ceq account(U, deposit(U', N, A)) = account(U, A)
       if U =/= U' .
```

これを Java で表現する場合には、以下のように

１、オブジェクトを引数に渡すことで簡単な形にする

```
public void deposit(Counter u, int n){
 u.add(n);
}
```

もしくは、

２、Java の組み込みクラス Hashtable などを用いてデータ構造を定義する

のどちらかの方法を採る必要がある。

```
java.util.Hashtable H = new java.util.Hashtable();
public void deposit(int u, int n){
 Integer uid = new Integer(u);
    if (H.get(uid) == null) {}
    else {
```

```
     Integer I = (Integer) H.get(uid);
     n= I.intValue() + n;
     Integer N = new Integer(n);
    H.put(uid,N);}
 }
```

本研究ではよりシンプルな形で表現できる１の方法を採った。２の場合、オブジェクトを生成する機会が増えるためオーバーヘッドが増加し、可読性も劣っていると考えられる。

これらの規則を用いて変換した例として、銀行の現金自動預金システム (ATM) を用い、変換させたものを付録に表示した。ATM は switch,counter,cell といった基本構造のオブジェクト合成より作成される預金者と預金額のデータベースシステムである。

以下に示す COUNTER(図 4.4) は add で整数を加算して read でその合計を知ることができる。

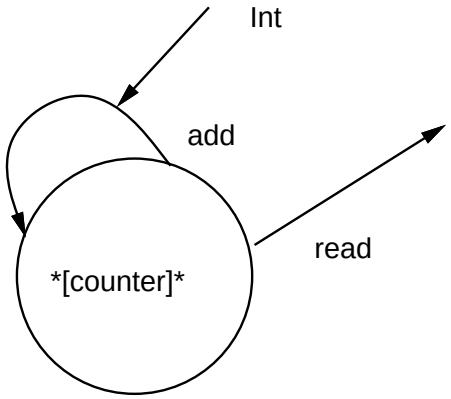


図 4.4: counter

```
mod* COUNTER {
  protecting(USER-ID + INT)

  *[ Counter ]*

  op init-counter : UId -> Counter
  bop add : Int Counter -> Counter
  bop read_ : Counter -> Int
```

```
  var I : Int
  var C : Counter
  var U : UId

  eq read(init-counter(U)) = 0 .
  eq read(add(I, C)) = I + read(C) .
}
```

CafeOBJ

```
public class Counter{
 private int counter = 0;
 public void add(int x){
   counter = counter + x;
  }
 public int read(){
  return counter;
 }
}
```

Java

また、以下に示すSwitch(図4.5)はon,offで状態を変化させてstatusでその状態を知ることができる。

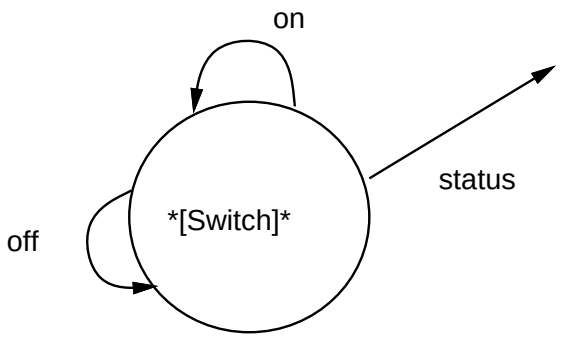


図 4.5: switch

```
mod* SWITCH {
   [ Value ]
  *[ Switch ]*
```

```
  ops on off : -> Value
  op init-sw : -> Switch
  bop on_ : Switch -> Switch
  bop off_ : Switch -> Switch
  bop status_ : Switch -> Value

  var S : Switch

  eq status(init-sw) = off .
  eq status(on(S)) = on .
  eq status(off(S)) = off .
}
```

CafeOBJ

```
public class Switch{
 private boolean state= false;
  public void on(){
   state= true;
  }
  public void off(){
   state= false;
  }
  public boolean state(){
   return state;
  }
}
```

Java

ATM(図 4.6) のような非常にシンプルは仕様では、モジュール構造を保ったまま変換できていることが分かる。

CafeOBJ から Java へのトランスレータを作成する場合、CafeOBJ の仕様が Stack などの基本的なデータ構造を含むかどうかをトランスレータに判断させるのは大変な作業になることが予想されるので、CafeOBJ の仕様をなるべく基本的なライブラリなどを用いて作成し、ライブラリを Java の組込クラスに変換するスタイルが望ましい。

以下に示した最上位モジュールで以下のメソッドを実行させた場合、

```
public static void main(String argv[]){
 ATMTOP att = new ATMTOP();
```

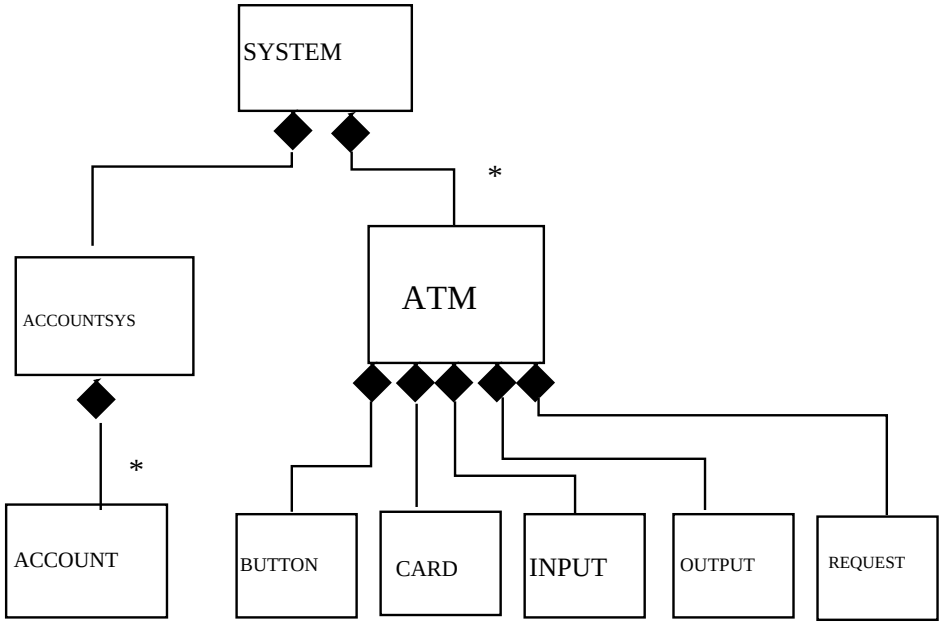


図 4.6: 現金自動預金システム (ATM)

```
 Counter c1 = new Counter();
 ATMClient atc = new ATMClient();
 ATMClient atc2 = new ATMClient();

 att.adduser(c1,120);
 System.out.println(att.balance(c1));
 att.deposit(c1,atc,50);
 System.out.println(att.balance(c1));
 att.withdraw(c1,atc,30);
 System.out.println(att.balance(c1));
}
}
```

以下の結果が出力される。

```
% java ATMTOP
120
170
140
```

内容は、新しくユーザアカウントを作成し、初期状態が 120 とする。次に 50 入金した後、30 引き出すという操作を行っている。

この結果は仕様を満たしていることが分かる。

4.3 変換に関するまとめと考察

CafeOBJ 仕様から仕様の構造を保存しつつ、効率的な Java プログラムへ変換する規則を ATM の例を用いて示した。

CafeOBJ から Java への変換は以下ような対応付けで行われた。

CafeOBJ		Java
module	→	class
protecting	→	import
visible sort	→	型
hidden sort	→	型（private）
method	→	method(戻り値なし)
attribute	→	method(戻り値あり)
projection	→	下位モジュールの operator

if 文について前節で述べなかったが、これも単純な対応付けで表現できる。例えばモジュール ATMSYSTEM の例では

```
    CafeOBJ

  ceq account-sys(ok(A, S)) =
       deposit(user-id(atm(A, S)), get-input(atm(A, S)), account-sys(S))
       if button-status(atm(A, S)) == deposit and-also
          user-id(atm(A, S)) =/= unidentified-user and-also
          get-input(atm(A, S)) =/= 0 .

    Java

public void ok(ATMClient atc){
  if ((atc.buttonstatus() == true)     &&
      (atc.userid() != null)           &&
```

```
    (atc.getinput() != 0)){
      ac.deposit(atc.userid(), atc.getinput());
   }
```

となり、各項が一対一の対応となっていることが分かる。

CafeOBJ の仕様が Stack などのデータ構造を持った表現の場合、直接は変換できなかった。そこで 1,method の引数にオブジェクト識別子を渡す方法と 2,method に新しくデータ構造をかかえこませる方法が考えられた。2, の方法で表現した場合、Accountsystem の Java プログラムは以下のように変換される。

```
import java.util.Hashtable.* ;
public class Accountsystem{

   java.util.Hashtable H = new java.util.Hashtable();
   Counter c1= new Counter();

 public void add(int u, int n){
  Integer uid = new Integer(u);
  Integer N = new Integer(n);
  if (H.get(uid) ==  N) {}
  else {H.put(uid,N); c1.add(n);}
 }
 public void dell(int u){
 Integer uid = new Integer(u);
    if (H.get(uid) == null) {}
    else {H.remove(uid);}
 }
 public void deposit(int u, int n){
 Integer uid = new Integer(u);
    if (H.get(uid) == null) {}
    else {
     Integer I = (Integer) H.get(uid);
     n= I.intValue() + n;
     Integer N = new Integer(n);
    H.put(uid,N);}
    c1.add(n);
 }
              :
              :
```

1, の方法に比べ、オブジェクトを生成する機会が増えるためオーバーヘッドが増加し、可読性も劣っていると考えられる。

また、効率のよいJavaプログラミングに変換するテクニックとして、CafeOBJの仕様で以下のようなスタイルを変換する場合、右辺に

```
projection(hidden sort)
```

がある場合は状態が変わらないことを表しているため、Javaでは表現せず、

```
  eq button(deposit(ATM)) = on(button(ATM)) .
  eq button(withdraw(ATM)) = off(button(ATM)) .
  eq button(request(N, ATM)) = button(ATM) .
  eq button(put-money(N, ATM)) = button(ATM) .
  eq button(take-money(ATM)) = button(ATM) .
  eq button(set-money(N, ATM)) = button(ATM) .
  eq button(put-card(U, ATM)) = button(ATM) .
```

この例の場合以下のように変換される。

```
Switch button = new Switch();
public void deposit(){
  button.on();
}
public void withdraw(){
  button.off();
}
```

第 5 章

同期記述

同期とはあるスレッドが仕様中のデータを他のプロセスが偶然変更しないように、スレッドどうしが協調して動作することを保証することである。これは、あるデータを並行にアクセスできるスレッドの数を制限する関数呼びだしを提供することで実現されている。相互排他制御の場合は、一度に１つのスレッドだけが指定のコード部分を実行することができる。このコード部分はあるグローバルデータを変更したり、デバイスに対して読み込んだり書き込んだりすることが可能である。例えば、スレッド T1 はロックを獲得し、あるグローバルデータを使用しはじめる。ここでスレッド T2 は、T1 が実行しているのと同じコードを実行する前に、スレッド T1 が終了するまで待たなければならない。そのデータを変更するすべてのコードの前後に同じロックを用いることによって、そのデータが矛盾のないことを保証することができる。

Java プログラムでスレッドを使うと同時に２つ以上のことができるようになる。スレッドを使うには以下の３つの理由がある。

1 実際に２つ以上のことを同時にする。

　これはプログラムの一部が何らかのリソースを持ってブロックしても他のスレッドは実行できてブロックしないということである。

2 ある種のプログラムはスレッドに分けた方が書きやすい

　この古典的な例としてはクライアント/サーバのサーバがある。クライアントからリクエストがきた時に、そのリクエストを処理する新しいスレッドを作ることができれば大変に便利である。スレッドを使わない場合は、それぞれのクライアントのリクエストの状態を保持しなければならない。

3 ある種のプログラムは本質的に並列処理である

スレッドを使って書けばそれがわかりやすい。このようなプログラムにはソート、マージ、行列演算、再帰プログラムなどがある。

ATM の例ではクライアント／サーバであるので、２に相当する。

また、複数のスレッド間の関係は「同期」と呼ばれており、複数のスレッドを用いるプログラムはスレッド間で必要とされる同期の種類によって下の４つの難易度に分けられる。

１ 関係のないスレッド
最も簡単なスレッドプログラムは、複数の制御のスレッドが別々のことをしていて相互作用がないプログラムである。

２ 関係しているが同期していないスレッド
この複雑度のプログラムとしては、問題を分割して同じデータ構造の別の部分を処理する複数のスレッドを用いて解くプログラムがある。スレッドは互いに干渉しない。この場合、それぞれの制御スレッドはそれぞれに割り当てられた仕事をするのだが、対象となるデータを共有したりはしない。したがって、データのアクセス時に同期をとる必要はない。がい

３ 相互に排他制御されているスレッド
スレッドが相互作用を行なうもの。ここでは、同じデータ構造の同じ部分を使わなければならない複数のスレッドを用いる。

４ 通信する相互に排他制御されているスレッド
複数のスレッドがデータを互いにやりとりするもの。同じデータを処理する複数のスレッドがあるので、同期させなければならない。３の排他制御だけでなく、データが準備できていないとサスペンドするようなスレッドも存在する。

ATM の例では３にあたり、同時にアクセスされると困るものとしては、例えば複数のクライアントから同時に引き出すと残高がマイナスになってしまう可能性があることが挙げられる。

5.1 相互排他制御

java ではスレッドを使うためにクラスを扱う必要はあるが、一番問題となる同期機構が言語レベルで含まれている。排他制御は synchronized 修飾子を用いることで表現され、

これによってスレッドの実行順序を制御できる。また、変数を volatile と指定することで、メソッドがインスタンス変数を読み書きしてデータを更新しようとする際に、複数スレッドの並行する更新においても矛盾が生じないよう実行系に保証してもらうことができる。

例えば、預金を引き出す withdraw では

```
static void withdraw(Counter U, ATMClient A, int N){
 ats.putcard(A,U);
 ats.withdraw(A);
 ats.request(A,N);
 ats.ok(A);
}
```

synchronized を付け加えて、

```
static synchronized void withdraw(Counter U, ATMClient A, int N){
  .......
}
```

と表現することでコンパイラに「排他制御をしなければならないクラスメソッドの集合にこのメソッドを加えよ」と命じたことになる。クラス全体で１つのメソッドしか動けないので、引き出し操作の競合状態は起こらなくなる。

また、Thread の実装を実行させるには start() を呼ばなければならないため、class の構造に少し手を加える（各 method に run() を加えるなど）必要がある。

このように java での同期記述は比較的簡単に行なえる。しかし、CafeOBJ では実際に同時に動作させることができないため、今のところ、このような簡単な記述で決まったスタイルは存在しない。

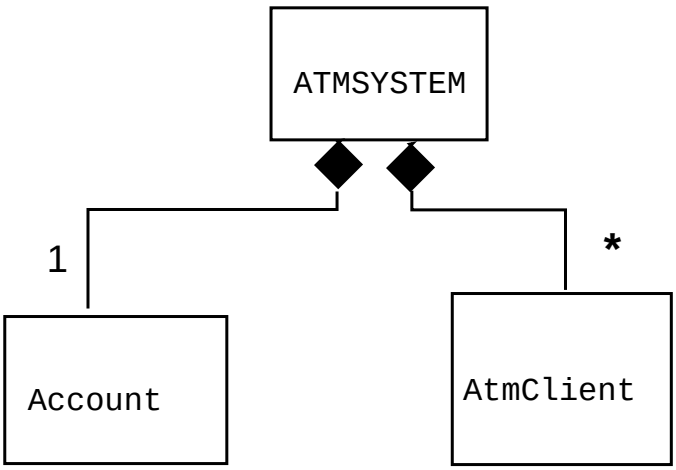
次に synchronized を付加すべき場所について説明を行う。まずはじめに、合成されるオブジェクト間に何の関係もない場合については合成されている各オブジェクトに関する操作演算どうしには並行性があるため、同期させることができる。このため synchronized で排他制御を行う必要はない。しかし、合成されるオブジェクト間に次のような関係がある場合、

- あるオブジェクトが他のオブジェクトの操作演算を呼び出す
- 射影演算の定義が他のオブジェクトの観測演算に依存している
- 複数のオブジェクトの状態を同時に遷移させる

合成されている各オブジェクトに関する操作演算は、一部あるいは全てに関して並行性がないため、synchronized を付加すべきである。

5.1.1 ATM の同期記述

例えばATM（自動預金支払機）のクライアントをスレッドで表現した場合を考える。

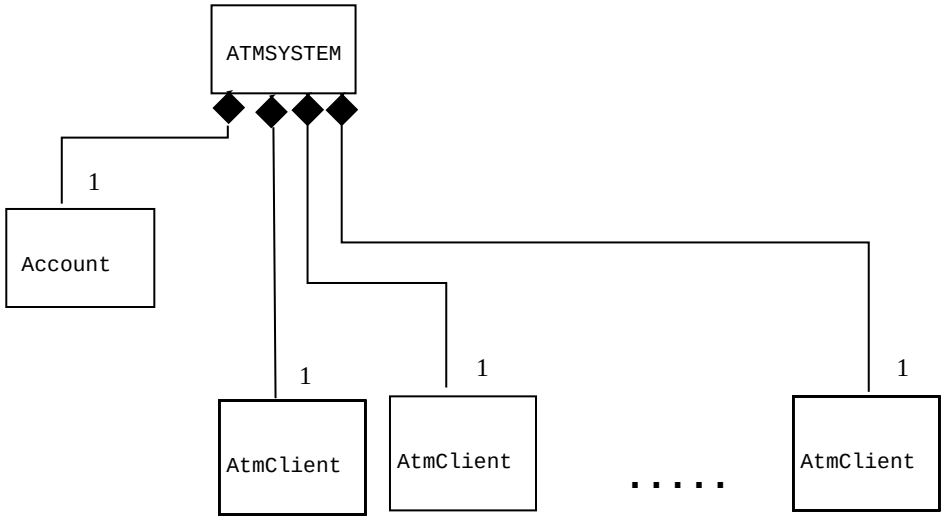



```
ceq account-sys(ok(A, S)) =
 withdraw(user-id(atm(A, S)),
          get-request(atm(A, S)), account-sys(S))
   if button-status(atm(A, S)) == withdraw and-also
      user-id(atm(A, S)) =/= unidentified-user and-also
      get-request(atm(A, S)) =/= 0 and-also
      get-request(atm(A, S)) <= balance(user-id(atm(A, S)), account-sys(S)) .
```

ここでは残高が 0 以下にならないような制限が記述されている

例えば、初期残高 100　として、クライアントを１０個作成し、各クライアントで 15 ずつ引いていく場合を考える。この場合、下記のように排他制御を行わなければうまく動作しない。

```
static Accountsystem ac = new Accountsystem();
 void put_money(int x){
   synchronized(ac){
      ...
   }
 }
```



以下に CafeOBJ で記述した仕様の一部を示す。

```
mod* ATM-SYSTEM {
                :
 bop withdraw : AId System -> System
 bop ok : AId System -> System
 bop account-sys : System -> AccountSys
 bop atm : AId System -> Atm
                :
 eq account-sys(withdraw(A, S)) = account-sys(S) .
 ceq account-sys(ok(A, S)) =
  withdraw(user-id(atm(A, S)),
           get-request(atm(A, S)), account-sys(S))
    if button-status(atm(A, S)) == withdraw and-also
       user-id(atm(A, S)) =/= unidentified-user and-also
       get-request(atm(A, S)) =/= 0 and-also
       get-request(atm(A, S)) <=
         balance(user-id(atm(A, S)), account-sys(S)) .
                :
                :
}
```

この仕様を Java に変換したものは以下の通りである。

```
public class ATMSystem extends Thread {
 static Accountsystem ac = new Accountsystem();
 public void withdraw(ATMClient atc){  atc.withdraw(); }
 public void ok(ATMClient atc){
```

```
// synchronized(ac){
      if((atc.buttonstatus() == false) &&
          (atc.userid() != null)       &&
          (atc.getrequest() != 0)      &&
          (atc.getrequest() <= ac.balance(atc.userid()))) {
      try{sleep(100);} catch(Exception e){}
            ac.withdraw(atc.userid(), atc.getrequest());   }
// }
 }
 ATMClient atc = new ATMClient();
 Counter c1;
public void run(){
 putcard(atc,c1);
 withdraw(atc);
 request(atc,15);
 ok(atc);
}
public static void main(String argv[]){
 ATMSystem ats = new ATMSystem();
 Counter c1 = new Counter();
 ats.adduser(c1,100);
  ATMSystem []at = new ATMSystem[10];
  for(int i=0; i<10; i++){
   at[i] = new ATMSystem();
   at[i].c1 = c1;
   at[i].start();
   }
  try{for (int i=0; i<10; i++) at[i].join();  }
    catch(Exception e){}
  System.out.println(ats.balance(c1));
}}
```

これらを動作させると、synchronize を付けない場合は残高 -50　、付けた場合は 10 で停止する。

この例では static な Accountsystem で排他制御を行なった。これでは複数のユーザが同時に withdraw した場合にはただ一人のユーザしかアクセスできない。

そこで以下のようにユーザー ID で排他制御を行なうことで、より効率の良い並行動作を行なうことができる。

```
 public void ok(ATMClient atc){
   :
 synchronized(atc.userid()){
```

```
    :
      }
 }
```

この場合も、synchronize を付けない場合は残高 -50　、付けた場合は 10 で停止する。

5.2　通信を行う相互排他メソッド

前節のようにあるスレッドが走っている場合にロックをかけるだけでなく、複数のスレッドがデータを互いにやりとりするために、データが作成されると連絡したり、データが準備できていないとサスペンドするようなスレッドを考える。java でこの問題を解決してくれるものに wait と notify を用いた並列プログラムの書き方がある。

wait,notify は、同じクラスの同期メソッドが互いに通信しなければならないときに使われる。１つのスレッドはデータを間欠的に作り、もう一方のスレッドは処理を行う場合（プロデューサ/コンシューマ）などで使われる。これらのスレッドのメソッドは synchronized コードの中でしか呼び出せない。これは、排他ロックを保持している時にしか呼び出せないということを意味する。

しかし、コンシューマがロックを取得しても、バッファに処理すべきものがなかったり、プロデューサがロックを取得しても、バッファに空きがなかったりする場合があるため単純な同期だけでは不十分である。

ロックを取得し、データを動かせるかどうか確かめ、もし動かせなければロックを放棄するというループの中に書いて何度も繰り返して行うこともできなくはないが、この方法では CPU のサイクルの無駄遣いとなるため、wait() と notify() の２つのメソッドを使う方法を用いる。

wait() はこのオブジェクトの変化がほかのスレッドによって通知されるまで（データが用意されるまで）待機し、notify() はオブジェクトのモニタで待機中の 1 つのスレッドを再開する。

ATM の account system と ATMclient はクライアント/サーバシステムである。

ここでは account system を producr,ATMclient を consumer と置く。

Java 風の疑似コードで書くと、

```
// producer thread
enter synchronized code (i.e. grab mutex lock)
```

```
produce_data()
notify()
leave synchronized code (i.e. release lock)

// consumer thread
enter synchronized code
while( no_data )
 wait()
consume_the_data()
leave synchronized code
```

となり、コンシューマは、ループの中で wait することになる。これは、別のコンシューマ（ATMclient）がデータを持っていってしまうかも知れないからで、その場合には、もう一度 wait しなければならないからである。同期するコード部分に入ったり出たりするには、単に synchronized 装飾子をメソッドにつければよい。

また、通常プロデューサは、作ったデータをなんらかの「境界付きバッファ」に入れる。これは、プロデューサがこのバッファを満たしてしまい、バッファに空きがでるまで wait() する必要があるということである。コンシューマはバッファ上のものを取り除いたらプロデューサに notify() して教えてやる必要がある。これを疑似コードで書くと以下のようになる。

```
// producer thread-produces one datum
while( buffer_full )
 wait()
produce_data()
notify()public class Cell{

 private int cell = 0;

 public void put(int x){
   cell = x;
  }

 public int get(){
```

```
  return cell;
 }


//consumer thread-produces one datum
while( no_data )
  wait()
consume_the_data()
notify()
```

Java に変換した際に wait/notify の並列プログラムを表現できるようにするために、CafeOBJ の仕様に wati/notify を使用するようにコメントとして付加させる。

ここでは producer と consumer というコメントを付加するとこで表現することにする。

例えば、以下の ATMClient の deposit button の equation に producer を付加することで

```
 eq button(deposit(ATM)) = on(button(ATM)) .
```

Java 変換の際に、通信する同期排他メソッドとして変換を行う。

```
 eq button(deposit(ATM)) = on(button(ATM)) .   -- producer
```

producer コメントを見つけた場合、変換の際に buffer を作成し、deposit は以下のように変換される。

```
static synchronized void deposit(){
 while( buffer_full )
  wait()
 button.on()
 notify()
}
```

第 6 章

仕様の変更

一般に、ソフトウェアの要求が変化した場合にはその仕様を変更し、変更された仕様において要求が満たされていることを証明しなおす必要があるが、もし要求の変化がどのコンポーネントに影響し、そのコンポーネントを他のコンポーネントと置き換えた際にどんな性質を検証すれば全体として全ての要求を満たすことができるのかが分かれば大幅なコスト削減が期待できる。要求分析をどのように行うのか、それに基づいてどのように仕様化するのか、さらに要求の変化にどのように対応するのか、といった問題を形式的に行おうという研究は、ほとんど行なわれていない。

合成されるオブジェクトの仕様の変更可能範囲を形式的に表現できれば、最小限のコストで仕様を変更できることが予想される。

要求の変更には以下の２つがある。

１, 既存の機能を変更する（例えば、機能の詳細化）

２, 新しい機能を追加する（例えば、インタフェースの追加）

２の場合、さらに新しい機能が既存の機能の組合せで表現できる場合、そうでない場合に分類できる。

要求の変更は通常曖昧性を含む自然言語で行われるが、ここでは代数仕様言語 CafeOBJ で表現することにより、曖昧性をなくす。

本稿で扱う状況では、CafeOBJ での要求の変更は

- method の追加
- equation の変更
- projection の張り直し

により表現するとする。

さらに、適応範囲の拡大方法としてはモジュールのパラメータ化が挙げられる。

6.1 仕様の変更例：ATM

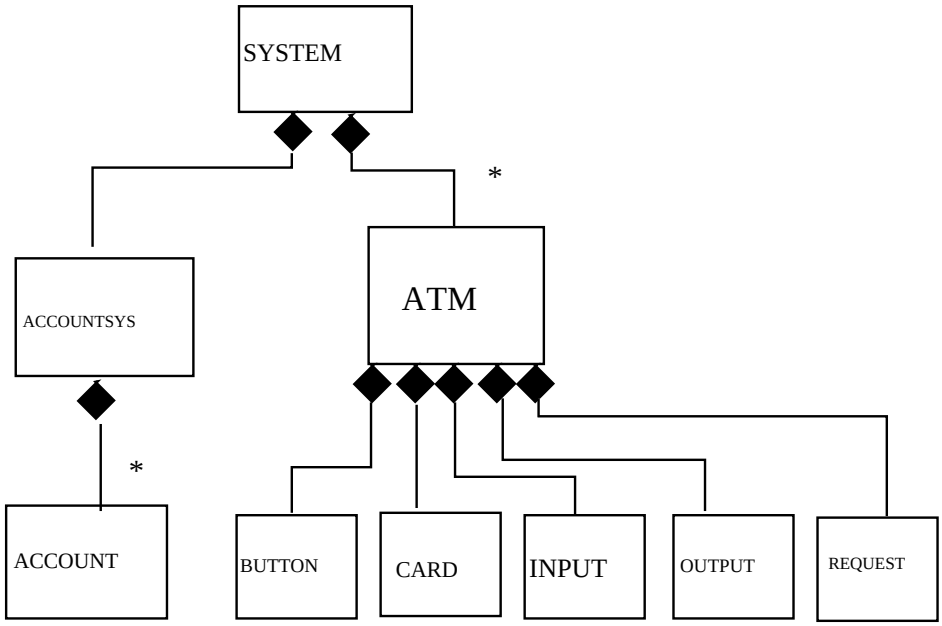


図 6.1: 現金自動預金システム (ATM)

銀行の現金自動預金システムを例題として用いる。

変更前の ATM は user account の追加、削除、user account に対する預入、払戻、残高照会が行なえる。以下に module ACCOUNT-SYSTEM の一部を示す

```
*[ AccountSys ]*
op init-account-sys :          -> AccountSys -- initial state
bop add      : UId Nat AccountSys -> AccountSys -- method
bop del      : UId     AccountSys -> AccountSys -- method
bop deposit  : UId Nat AccountSys -> AccountSys -- method
bop withdraw : UId Nat AccountSys -> AccountSys -- method
bop balance  : UId     AccountSys -> Nat        -- attribute
bop account  : UId     AccountSys -> Account    -- projection
```

ソート名は [ ] で括って宣言され、隠蔽ソートは*[ ]*で宣言される。演算子は op に続けて宣言され、bop に続けて宣言される演算子は振舞い演算として扱われる。

この ATM に他の user account へ送金するという新しい機能を追加する。

method と equation を追加する方法を用いた場合、
変更スタイルには大きく分けて２つ考えられる。

1, 新しいオブジェクトを作成して projection で合成させる

2, 既存の method や equation を変更する

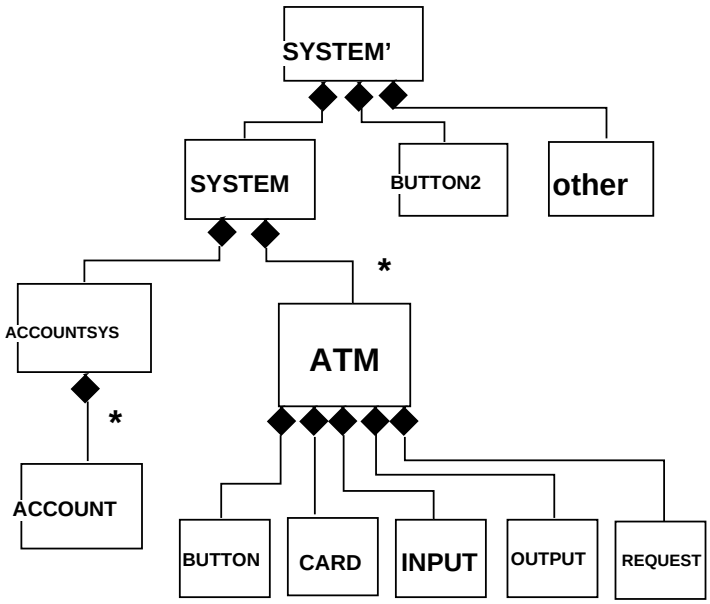


図 6.2: projection で合成させる

1, の場合、最上位のモジュールのみに手を加えればよいので変更は楽だが、機能別に分類されていないため（この場合インターフェイス部は ATM）、設計者の意図に会わない場合がある。そこで、ここでは 2, の方法をとる。

分析の方法としては最下位のモジュールから上位モジュールを分析する方法と、逆に最上位のモジュールから下位モジュールを分析する方法で行なった。

6.1.1 ボトムアップ分析

module ACCOUNT-SYSTEM では、出金と入金という既存の機能の組合せで表現できるため、以下の記述を追加すれば良い。

```
bop transport : UId UId Nat AccountSys -> AccountSys
 eq transport(U, U', N, A) =
     deposit(U',N, withdraw(U, N, A)) .
```

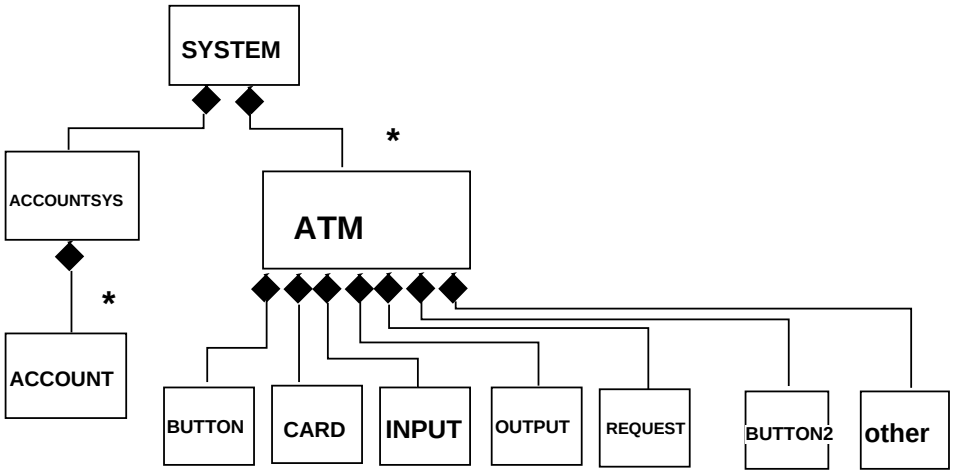


図 6.3: method や equation を変更する

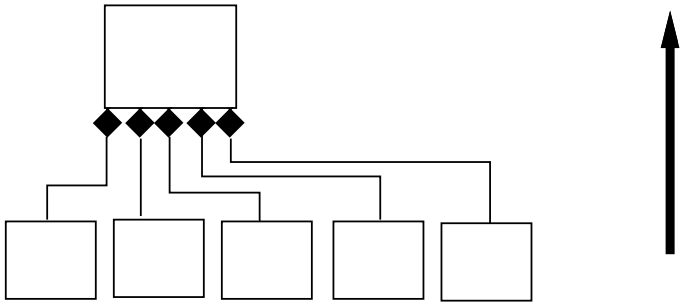


図 6.4: ボトムアップ分析

また、ATM クライアント部では transfer を選択するボタン、送金先の user-ID を入力するインタフェース

```
module button2
module other
```

を追加する。

これらの追加によりインタフェース部の上位モジュール ATM、その上位モジュール System を変更する必要がある

モジュール ATM の変更点

- ATM - > botton2, ATM - > other への projecton を張る

- transfer の選択を行う method、その確認を行う attribute の追加と、それに伴う equation の追加を行う。
- other-usr-id の入力を行う method、その確認を行う attribute の追加と、それに伴う equation の追加を行う。

モジュール SYSTEM の変更点

- method transfer を追加　それに伴い、
  - System $->$ AccountSys　への projecton に関する equation の追加（この場合、eq account-sys(transfer(A,U,S)) = account-sys(S) .）
  - System $->$ ATM　への projecton に関する equation の追加
  - transfer が選択された場合に transfer を実行させるように条件文付きの equation を追加

method を追加する変更は追加したオブジェクトからその method の機能を使用する合成後のオブジェクトまですべての合成オブジェクトを変更する必要がある。

6.1.2　トップダウン分析

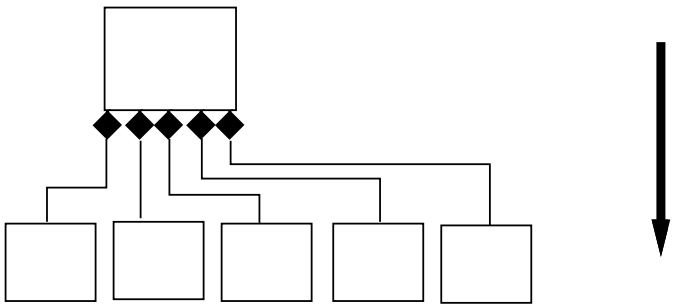


図 6.5: トップダウン分析

方針： projection を辿って、下位のどのモジュールに method や equation を追加すればよいか調べる。

まず transfer 機能を実現するために、最上位のモジュール SYSTEM に対して以下の method,equation を追加する。

```
bop transfer : AId UId System -> System
 eq account-sys(transfer(A,U,S)) = account-sys(S) .
ceq atm(A, transfer(A',U,S)) =   if A == A'
        then  transfer(U,atm(A,S))  else  atm(A,S) .
ceq account-sys(ok(A, S)) =
 transfer(user-id(atm(A, S)), other-id(atm(A,S)),
 get-request(atm(A, S)), account-sys(S))
 if transfer-button-status(atm(A, S)) == transfer
  and-also  user-id(atm(A, S)) =/= unidentified-user
  and-also  get-request(atm(A, S)) =/= 0
  and-also get-request(atm(A, S))
    <= balance(user-id(atm(A, S)), account-sys(S)) .
```

これに伴い、下位モジュールを変更する。
まず、追加した equation の中で module SYSTEM で定義されていない operation を抽出する。この場合、

```
transfer(UId,?,Nat,AccountSys),
other-id(Atm),
transfer-button-status(Atm),
transfer(UId,Atm)
```

の４つの operation である。

いずれも module SYSTEM で定義されている隠蔽ソートを含まないので、projection を辿って下位モジュール ACCOUNT-SYSTEM を見る。

*[AccountSys]*の定義があるため,

```
method transfer(UId,?,Nat,AccountSys)
```

を ACCOUNT-SYSTEM に定義する。

また同様に、projection を辿って下位モジュール ATM を見る。*[Atm]*の定義があるため

```
other-id(Atm),
transfer-button-status(Atm),
transfer(UId,Atm)
```

を ATM に定義する。

equation はユーザが別途定義するがこれらの equation に新たな operation があれば同様の操作を繰返す。

6.2 体系的な仕様変更

仕様を変更する前の CafeOBJ から Java へ変換した結果と、変更後の CafeOBJ から Java へ変換した結果を比較してみる。前節の送金操作の追加の例で見ると、変更前後の CafeOBJ の差分と同様の変更が Java に対しても反映されていることが分かる。また、当然であるが前章で述べたように CafeOBJ を並列分散化した場合にも Java へも体系的な変化が現れる。

以下の送金機能追加の例の場合、

```
bop transport : UId UId Nat AccountSys -> AccountSys
 eq transport(U, U', N, A) =
     deposit(U',N, withdraw(U, N, A)) .
}
```

Java では以下のように表現されて

```
public void transport(Counter u, Counter u, int n){
  withdraw(u,n);
  deposit(u,n);
}
```

CafeOBJ の場合、モジュール SYSTEM の変更点は

- method transfer を追加　それに伴い、
  - System − > AccountSys　への projecton に関する equation の追加（この場合、eq account-sys(transfer(A,U,S)) = account-sys(S) .）
  - System − > ATM　への projecton に関する equation の追加
  - transfer が選択された場合に transfer を実行させるように条件文付きの equation を追加

だったが、Java の場合は projection に関する equation(Java の場合 method) は省略できるため

- transfer が選択された場合に transfer を実行させるように if 文付きの method を追加

となる。

また withdraw,deposit を同時に行うとまずいので、並行動作させる場合は
synchronized を付け加えて、

```
static synchronized void withdraw(Counter U, ATMClient A, int N){
  .......
static synchronized void deposit(Counter U, ATMClient A, int N){
  .......
}
```

とする必要がある。

第 7 章

まとめ

本稿では、射影演算を用いて階層的に合成された振舞い仕様 [2] を対象とし、

CafeOBJ 仕様から仕様の構造を保存しつつ、効率的な Java プログラムへ変換する規則を示し、ATM の例を用いて説明した。変換した Java プログラムは CafeOBJ の仕様で表現されている複数のモデルの中で具体的な１つのモデルを表現している。

また、CafeOBJ で記述された仕様に並行性がある場合について、Java のマルチスレッドを使って実際に並行動作するようなコードに変換する方法を考え、排他制御に関しても考察を行った。CafeOBJ で記述した ATM が実際に Java で並行動作することを確かめた。

また、既存の機能の組み合わせ＋インタフェースの追加において、最上位のモジュールと最下位のモジュールを変更（method,equation の追加）することで変更が必要な範囲を示した。この結果からボトムアップとトップダウンを組み合わせることで、任意のモジュールが変更された場合にその上位のモジュールと下位のモジュールの変更すべき場所が特定できることが予想される。

本研究の今後の課題としては以下のことが考えられる本研究の今後の課題としては以下のことが考えられる

- 変換された Java プログラムが仕様を完全に満たしていることを証明する
  - CafeOBJ の仕様で表現されている複数のモデルの中で具体的な１つのモデルを表現していることは分かったが、そのモデルが仕様を正確に満たしているのかを数学的な証明で行う必要がある。
- より複雑な CafeOBJ 仕様への対応の調査
  - 本研究では比較的単純な基本データ構造を合成して作成した ATM の例を用い

たが、より複雑なモジュールを合成した場合に同様なことが言えるのかを調査する必要がある。

- 同期の記述方法についての考察
  - 本研究で行った同期の記述方法は排他制御を必要以上に行っているため、安全ではあるが実行効率は良いとは言えない。そこで実行効率を考慮した記述方法について考察する必要がある。

第 8 章

謝辞

本研究を進めるにあたり、指導して頂いた二木厚吉教授に深く感謝致します。また、有益な助言をして頂いた渡辺卓雄助教授、緒方和博助手、飯田周作氏に御礼を申し上げます。最後に、研究に関する議論につきあって頂いた言語設計学講座の諸氏に感謝致します。

参考文献

[1] R. Diaconescu, K. Futatsugi. CafeOBJ Report, World Scientific, (1998).

[2] J. Goguen and G. Malcolm. A Hidden Agenda, Report CS97-538, April 1997.

[3] S. Iida, M. Matsumoto, R. Diaconescu, K. Futatsugi, and D. Lucanu. Concurrent Object Composition in CafeOBJ, Reprot IS-RR-98-0009S

[4] P.Borba and J.A.Goguen. Refinement of concurrent object-oriented programs. In S.J.Goldsack and S.J.H. Kent, editors, Formal Methods and Object Technology. Springer-Verlag, 1996.

[5] Roger Duke, Paul King, and Graeme Smith Gordon Rose. The Object-Z specification language: Version 1. Technical Report 91-1, Software Verification Research Center, Department of Computer Science, The University of Queensland, April 1991.

[6] 二木厚吉, 代数モデルの基礎. コンピュータソフトウェア pp4-22 Vol.13, No1 (1996). ソフトウェア科学会 (1996).

[7] Futatsugi,K. An overview of OBJ2, Proc. of Fance-Japan A. I. and C. S. Symposium 86, Institute for New Generation Computer Technology, Japan, 1986.

[8] P.Coad,E.Yourdon 著, 羽生田訳, オブジェクト指向分析 (OOA) 第 2 版, プレンティスホール/トッパン,1993.

[9] S. Ragsdale. PARALLEL PROGRAMMING, McGraw-Hill,Inc. 1991

[10] Bil Lewis, Daniel J. Berg 共著 岩本信一 訳, マルチスレッドプログラミング入門, アスキー出版, 1996

[11] Gosling, Joy, and Steele, The Java Language Specification Addison-Wesley, Reading, Massachusetts, 1996.

[12] Peter van der Linden, just JAVA second edition, Sun Microsystems,Inc. 1997

[13] 戸松豊和 著, JAVA プログラミングデザイン. SOFTBANK BOOKS, 1997

第 A 章

付録

A.1 ATM(CafeOBJ)

```
-- ----------------------------------------------------------
-- Values of SWITCH
-- ----------------------------------------------------------
mod! ON-OFF {
  [ Value ]

  ops on off : -> Value
}

-- ----------------------------------------------------------
-- SWITCH
-- ----------------------------------------------------------
mod* SWITCH {
  protecting (ON-OFF)

  *[ Switch ]*

  op init-sw : -> Switch         -- initial state
  -- switch on
  bop on_ : Switch -> Switch     -- method
  -- switch off
  bop off_ : Switch -> Switch    -- method
  -- observe the state of the switch
  bop status_ : Switch -> Value   -- attribute

  var S : Switch
```

```
  eq status(init-sw) = off .
  eq status(on(S)) = on .
  eq status(off(S)) = off .
}

-- ------------------------------------------------------------
-- User identification
-- ------------------------------------------------------------
mod! USER-ID {
  protecting (NAT)
  [ Nat < UId ]

  op unidentified-user : -> UId
}

-- ------------------------------------------------------------
-- Counter
-- ------------------------------------------------------------
mod* COUNTER {
  protecting(USER-ID + INT)

  *[ Counter ]*

  -- initialize counter with user ID
  op init-counter : UId -> Counter   -- initial state
  -- add a value to the counter
  bop add : Int Counter -> Counter   -- method
  -- read the value of the counter
  bop read_ : Counter -> Int         -- attribute

  var I : Int
  var C : Counter
  var U : UId

  eq read(init-counter(U)) = 0 .
  eq read(add(I, C)) = I + read(C) .
}

-- ------------------------------------------------------------
-- Counter with error
-- ------------------------------------------------------------
mod* COUNTER* {
  protecting (COUNTER)
```

```
  op counter-not-exist : -> Counter  -- error value
}

-- ---------------------------------------------------------------
-- Account sytem
-- ---------------------------------------------------------------
mod* ACCOUNT-SYSTEM {
  protecting (COUNTER* *{ hsort Counter -> Account,
  op init-counter -> init-account,
  op counter-not-exist -> no-account })

  *[ AccountSys ]*

  op init-account-sys : -> AccountSys             -- initial state
  -- add a user account with user ID
  bop add : UId Nat AccountSys -> AccountSys       -- method
  -- delete a user account
  bop del : UId AccountSys -> AccountSys           -- method
  -- deposit operation
  bop deposit : UId Nat AccountSys -> AccountSys   -- method
  -- withdraw operation
  bop withdraw : UId Nat AccountSys -> AccountSys  -- method
  -- calculate the balance of an user account
  bop balance : UId AccountSys -> Nat              -- attribute
  -- get the state of a counter from the state of an account
  bop account : UId AccountSys -> Account   {memo} -- projection

  vars U U' : UId
  var A : AccountSys
  var N : Nat

  eq account(U, init-account-sys) = no-account .
  ceq account(U, add(U', N, A)) = add(N, init-account(U))
       if U == U' .
  ceq account(U, add(U', N, A)) = account(U, A)
       if U =/= U' .
  ceq account(U, del(U', A)) = no-account
       if U == U' .
  ceq account(U, del(U', A)) = account(U, A)
       if U =/= U' .
  ceq account(U, deposit(U', N, A)) = add(N, account(U, A))
       if U == U' .
  ceq account(U, deposit(U', N, A)) = account(U, A)
       if U =/= U' .
```

```
  ceq account(U, withdraw(U', N, A)) = add(-(N), account(U, A))
       if U == U' .
  ceq account(U, withdraw(U', N, A)) = account(U, A)
       if U =/= U' .

  eq balance(U, A) = read(account(U, A)) .
}

-- ---------------------------------------------------------------
-- Trivial module with an element (undefined)
-- ---------------------------------------------------------------
mod* TRIV+ {
  [ Elt ]

  op undefined : -> Elt
}

-- ---------------------------------------------------------------
-- Cell
-- ---------------------------------------------------------------
mod* CELL(X :: TRIV+) {
  *[ Cell ]*

  op init-cell : -> Cell      -- initial state
  -- put the element to the cell
  bop put : Elt Cell -> Cell  -- method
  -- get the element from the cell
  bop get : Cell -> Elt       -- attribute

  var E : Elt
  var C : Cell

  eq get(init-cell) = undefined .
  eq get(put(E, C)) = E .
}

-- ---------------------------------------------------------------
-- ATM identifier
-- ---------------------------------------------------------------
mod! ATM-ID {
--  protecting(NAT *{ sort Nat -> AId })
  protecting(NAT)

  [ Nat < AId ]
```

```
}

-- ------------------------------------------------------------
--  Button
-- ------------------------------------------------------------
mod* BUTTON {
  protecting(SWITCH *{ hsort Switch -> Button,
                       sort Value -> Operation,
                       op init-sw -> init-button,
                       op on -> deposit,
                       op off -> withdraw })
}

-- ------------------------------------------------------------
--  Cell for card information
-- ------------------------------------------------------------
mod* CARD {
  protecting(CELL(X <= view to USER-ID
                  { sort Elt -> UId,
                    op undefined -> unidentified-user })
                  *{ hsort Cell -> Card,
                     op init-cell -> init-card })
}

-- ------------------------------------------------------------
--  Cell for input
-- ------------------------------------------------------------
mod* INPUT {
  protecting(CELL(X <= view to NAT
                  { sort Elt -> Nat,
                    op undefined -> 0 })
                  *{ hsort Cell -> Input,
                     op init-cell -> init-input })
}

-- ------------------------------------------------------------
--  Cell for ouput
-- ------------------------------------------------------------
mod* OUTPUT {
  protecting(CELL(X <= view to NAT
                  { sort Elt -> Nat,
                    op undefined -> 0 })
                  *{ hsort Cell -> Output,
                     op init-cell -> init-output })
```

```
}

-- ------------------------------------------------------------
--  Cell for request
-- ------------------------------------------------------------
mod* REQUEST {
  protecting(CELL(X <= view to NAT
                  { sort Elt -> Nat,
                    op undefined -> 0 })
                  *{ hsort Cell -> Request,
                     op init-cell -> init-request })
}

-- ------------------------------------------------------------
--  ATM client
-- ------------------------------------------------------------
mod* ATM-CLIENT {
-- importing data and the composing objects
  protecting(ATM-ID + BUTTON + CARD + INPUT + OUTPUT + REQUEST)

  *[ Atm ]*

  op init-atm : AId -> Atm              -- initial state
  op no-atm : -> Atm                    -- error
  op invalid-operation : -> Atm         -- error
  -- push the deposit button
  bop deposit : Atm -> Atm              -- method
  -- push the withdraw button
  bop withdraw : Atm -> Atm             -- method
  -- input the request for withdraw
  bop request : Nat Atm -> Atm          -- method
  -- put money
  bop put-money : Nat Atm -> Atm        -- method
  -- take money
  bop take-money : Atm -> Atm           -- mothod
  -- set money for output (system operation)
  bop set-money : Nat Atm -> Atm        -- method
  -- put the bank card
  bop put-card : UId Atm -> Atm         -- method
  -- clear all the informations kept in the atm
  bop clear : Atm -> Atm                -- method
  -- get the user ID
  bop user-id : Atm -> UId              -- attribute
  -- get the money that user input
```

```
bop get-input : Atm -> Nat            -- attribute
-- get the outputed money
bop get-output : Atm -> Nat           -- attribute
-- get the request
bop get-request : Atm -> Nat          -- attribute
-- get the state of the button
bop button-status : Atm -> Operation  -- attribute

bop button : Atm -> Button     {memo} -- projection
bop card : Atm -> Card         {memo} -- projection
bop request : Atm -> Request   {memo} -- projection
bop input : Atm -> Input       {memo} -- projection
bop output : Atm -> Output     {memo} -- projection

var ATM : Atm
var N : Nat
var U : UId
var A : AId

eq button(init-atm(A)) = init-button .
eq button(invalid-operation) = init-button .
eq button(deposit(ATM)) = on(button(ATM)) .
eq button(withdraw(ATM)) = off(button(ATM)) .
eq button(request(N, ATM)) = button(ATM) .
eq button(put-money(N, ATM)) = button(ATM) .
eq button(take-money(ATM)) = button(ATM) .
eq button(set-money(N, ATM)) = button(ATM) .
eq button(put-card(U, ATM)) = button(ATM) .
eq button(clear(ATM)) = init-button .

eq card(init-atm(A)) = init-card .
eq card(invalid-operation) = init-card .
eq card(deposit(ATM)) = card(ATM) .
eq card(withdraw(ATM)) = card(ATM) .
eq card(request(N, ATM)) = card(ATM) .
eq card(put-money(N, ATM)) = card(ATM) .
eq card(take-money(ATM)) = card(ATM) .
eq card(set-money(N, ATM)) = card(ATM) .
eq card(put-card(U, ATM)) = put(U, card(ATM)) .
eq card(clear(ATM)) = init-card .

eq request(init-atm(A)) = init-request .
eq request(invalid-operation) = init-request .
eq request(deposit(ATM)) = request(ATM) .
```

```
  eq request(withdraw(ATM)) = request(ATM) .
  eq request(request(N, ATM)) = put(N, request(ATM)) .
  eq request(put-money(N, ATM)) = request(ATM) .
  eq request(take-money(ATM)) = request(ATM) .
  eq request(set-money(N, ATM)) = request(ATM) .
  eq request(put-card(U, ATM)) = request(ATM) .
  eq request(clear(ATM)) = init-request .

  eq input(init-atm(A)) = init-input .
  eq input(invalid-operation) = init-input .
  eq input(deposit(ATM)) = input(ATM) .
  eq input(withdraw(ATM)) = input(ATM) .
  eq input(request(N, ATM)) = input(ATM) .
  eq input(put-money(N, ATM)) = put(N, input(ATM)) .
  eq input(take-money(ATM)) = input(ATM) .
  eq input(set-money(N, ATM)) = input(ATM) .
  eq input(put-card(U, ATM)) = input(ATM) .
  eq input(clear(ATM)) = init-input .

  eq output(init-atm(A)) = init-output .
  eq output(invalid-operation) = init-output .
  eq output(deposit(ATM)) = output(ATM) .
  eq output(withdraw(ATM)) = output(ATM) .
  eq output(request(N, ATM)) = output(ATM) .
  eq output(put-money(N, ATM)) = output(ATM) .
  eq output(take-money(ATM)) = init-output .
  eq output(set-money(N, ATM)) = put(N, output(ATM)) .
  eq output(put-card(U, ATM)) = output(ATM) .
  eq output(clear(ATM)) = output(ATM) .

  eq user-id(ATM) = get(card(ATM)) .
  eq get-input(ATM) = get(input(ATM)) .
  eq get-output(ATM) = get(output(ATM)) .
  eq get-request(ATM) = get(request(ATM)) .
  eq button-status(ATM) = status(button(ATM)) .
}

-- ---------------------------------------------------------------
-- ATM system
-- ---------------------------------------------------------------
mod* ATM-SYSTEM {
  -- protecting(BOOL+)
  protecting(ACCOUNT-SYSTEM + ATM-CLIENT)
```

```
*[ System ]*

op init-sys : -> System                    -- initial state
-- add an atm to the system
bop add-atm : AId System -> System         -- method
-- delete an atm from the system
bop del-atm : AId System -> System         -- method
-- add an user account
bop add-user : UId Nat System -> System    -- method
-- delete an user account
bop del-user : UId System -> System        -- method
-- put the bank card
bop put-card : AId UId System -> System    -- method
-- request for withdraw
bop request : AId Nat System -> System     -- method
-- put money
bop put-money : AId Nat System -> System   -- method
-- take money
bop take-money : AId System -> System      -- method
-- deposit operation
bop deposit : AId System -> System         -- method
-- withdraw operation
bop withdraw : AId System -> System        -- method
-- push the ok button on atm to complete the operation
bop ok : AId System -> System              -- method
-- cancel the operation of ATM
bop cancel : AId System -> System          -- method
-- get the balance of specified user
bop balance : UId System -> Nat            -- attribute
-- projection operator for AccountSys
bop account-sys : System -> AccountSys  {memo}  -- projection
-- projection operator for Atm
bop atm : AId System -> Atm             {memo}  -- projection

var S : System
vars A A' : AId
var U : UId
var N : Nat

eq balance(U, S) = balance(U, account-sys(S)) .

eq account-sys(init-sys) = init-account-sys .
eq account-sys(add-atm(A, S)) = account-sys(S) .
eq account-sys(del-atm(A, S)) = account-sys(S) .
```

```
eq account-sys(add-user(U, N, S)) = add(U, N, account-sys(S)) .
eq account-sys(del-user(U, S)) = del(U, account-sys(S)) .
eq account-sys(put-card(A, U, S)) = account-sys(S) .
eq account-sys(request(A, N, S)) = account-sys(S) .
eq account-sys(put-money(A, N, S)) = account-sys(S) .
eq account-sys(take-money(A, S)) = account-sys(S) .
eq account-sys(deposit(A, S)) = account-sys(S) .
eq account-sys(withdraw(A, S)) = account-sys(S) .
ceq account-sys(ok(A, S)) =
     deposit(user-id(atm(A, S)), get-input(atm(A, S)), account-sys(S))
     if button-status(atm(A, S)) == deposit and-also
        user-id(atm(A, S)) =/= unidentified-user and-also
        get-input(atm(A, S)) =/= 0 .
ceq account-sys(ok(A, S)) =
     withdraw(user-id(atm(A, S)), get-request(atm(A, S)), account-sys(S))
     if button-status(atm(A, S)) == withdraw and-also
        user-id(atm(A, S)) =/= unidentified-user and-also
        get-request(atm(A, S)) =/= 0 and-also
        get-request(atm(A, S)) <=
               balance(user-id(atm(A, S)), account-sys(S)) .
ceq account-sys(ok(A, S)) = account-sys(S)
     if user-id(atm(A, S)) == unidentified-user or
        (button-status(atm(A, S)) == deposit and-also
            get-input(atm(A, S)) == 0) or
        (button-status(atm(A, S)) == withdraw and-also
            (get-request(atm(A, S)) == 0 or
            get-request(atm(A, S)) >
                     balance(user-id(atm(A, S)), account-sys(S)))) .
eq account-sys(cancel(A, S)) = account-sys(S) .

eq atm(A, init-sys) = no-atm .
ceq atm(A, add-atm(A', S)) = init-atm(A)
     if A == A' .
ceq atm(A, add-atm(A', S)) = atm(A, S)
     if A =/= A' .
ceq atm(A, del-atm(A', S)) = no-atm
     if A == A .
ceq atm(A, del-atm(A', S)) = atm(A, S)
     if A =/= A .
eq atm(A, add-user(U, N, S)) = atm(A, S) .
eq atm(A, del-user(U, S)) = atm(A, S) .
ceq atm(A, put-card(A', U, S)) = put-card(U, atm(A, S))
     if A == A' .
ceq atm(A, put-card(A', U, S)) = atm(A, S)
```

```
      if A =/= A' .
  ceq atm(A, request(A', N, S)) = request(N, atm(A, S))
      if A == A' .
  ceq atm(A, request(A', N, S)) = atm(A, S)
      if A =/= A' .
  ceq atm(A, put-money(A', N, S)) = put-money(N, atm(A, S))
      if A == A' .
  ceq atm(A, put-money(A', N, S)) = atm(A, S)
      if A =/= A' .
  ceq atm(A, take-money(A', S)) = take-money(atm(A, S))
      if A == A' .
  ceq atm(A, take-money(A', S)) = atm(A, S)
      if A =/= A' .
  ceq atm(A, deposit(A', S)) = deposit(atm(A, S))
      if A == A' .
  ceq atm(A, deposit(A', S)) = atm(A, S)
      if A =/= A' .
  ceq atm(A, withdraw(A', S)) = withdraw(atm(A, S))
      if A == A' .
  ceq atm(A, withdraw(A', S)) = atm(A, S)
      if A =/= A' .
  ceq atm(A, ok(A', S)) = clear(atm(A, S))
      if A == A' and-also
         user-id(atm(A, S)) =/= unidentified-user and-also
         button-status(atm(A, S)) == deposit .
  ceq atm(A, ok(A', S)) = set-money(get-request(atm(A, S)), clear(atm(A, S)))
      if A == A' and-also
         user-id(atm(A, S)) =/= unidentified-user and-also
         button-status(atm(A, S)) ==  withdraw and-also
         get-request(atm(A, S)) <=
                   balance(user-id(atm(A, S)), account-sys(S)) .
  ceq atm(A, ok(A', S)) = invalid-operation
      if A == A' and-also
         (user-id(atm(A, S)) == unidentified-user or
         (button-status(atm(A, S)) ==  withdraw and-also
             (get-request(atm(A, S)) >
                  balance(user-id(atm(A, S)), account-sys(S))))) .
  ceq atm(A, ok(A', S)) = atm(A, S)
      if A =/= A' .
  ceq atm(A, cancel(A', S)) = init-atm(A)
      if A == A' .
  ceq atm(A, cancel(A', S)) = atm(A, S)
      if A =/= A' .
}
```

A.2 ATM(Java)

```
public class Switch{
 private boolean state= false;
  public void on(){
   state= true;
  }
  public void off(){
   state= false;
  }
  public boolean state(){
   return state;
  }
}

public class Counter{
 private int counter = 0;
 public void add(int x){
   counter = counter + x;
  }
 public int read(){
  return counter;
 }
}

public class Cell{
 private int cell = 0;
 public void put(int x){
   cell = x;
  }
 public int get(){
  return cell;
 }
}

public class Card{
 Counter card = new Counter();
 public void put(Counter x){
   card = x;
  }
 public Counter get(){
  return card;
 }
}
```

```
public class Accountsystem{

 public void add(Counter u, int n){
  u.add(n);
 }
 public void del(Counter u){
  u= null;
 }
 public void deposit(Counter u, int n){
  u.add(n);
 }
 public void withdraw(Counter u, int n){
  u.add(-n);
 }
 public int balance(Counter u){
  return u.read();
 }
}

public class ATMClient{

Switch button = new Switch();
public void deposit(){
  button.on();
}
public void withdraw(){
  button.off();
}

Cell request = new Cell();
public void request(int N){
  request.put(N);
 }
Cell input = new Cell();
public void putmoney(int N){
  input.put(N);
}
Cell output = new Cell();
public void takemoney(){
  output.put(0);
}
public void setmoney(int N){
  output.put(N);
```

```
}
Card card = new Card();
public void putcard(Counter U){
  card.put(U);
}
public void clear(){
  button.off();
  request.put(0);
  input.put(0);
  card.put(null);
}
public int getinput(){
 return input.get();
}
public int getoutput(){
 return output.get();
}
public int getrequest(){
 return request.get();
}
public Counter userid(){
 return card.get();
}
public boolean buttonstatus(){
 return button.state();
}
}

public class ATMSystem{

 Accountsystem ac = new Accountsystem();
 public void adduser(Counter U, int N){
  ac.add(U,N);
 }
 public void deluser(Counter U){
  ac.del(U);
 }

 public void putcard(ATMClient atc, Counter U){
  atc.putcard(U);
 }
 public void request(ATMClient atc, int N){
  atc.request(N);
 }
```

```
 public void putmoney(ATMClient atc, int N){
  atc.putmoney(N);
 }
 public void takemoney(ATMClient atc){
  atc.takemoney();
 }
 public void deposit(ATMClient atc){
  atc.deposit();
 }
 public void withdraw(ATMClient atc){
  atc.withdraw();
 }
 public void ok(ATMClient atc){
  if ((atc.buttonstatus() == true)     &&
      (atc.userid() != null)           &&
      (atc.getinput() != 0)){
        ac.deposit(atc.userid(), atc.getinput());
     }
  else if((atc.buttonstatus() == false) &&
          (atc.userid() != null)        &&
          (atc.getrequest() != 0)       &&
          (atc.getrequest() <= ac.balance(atc.userid()))) {
            ac.withdraw(atc.userid(), atc.getrequest());
     }
  if ((atc.userid() != null) &&
      (atc.buttonstatus() == true)) {
        atc.clear();
  }
  else if((atc.userid() != null)        &&
          (atc.buttonstatus() == false) &&
          (atc.getrequest() <= ac.balance(atc.userid()) )){
            atc.setmoney(atc.getrequest());
  }
 }
 public int balance(Counter U){
   return ac.balance(U);
 }
}

public class ATMTOP {

ATMSystem ats = new ATMSystem();
public void adduser(Counter U, int N){
```

```
 ats.adduser(U,N);
}
public void deluser(Counter U){
 ats.deluser(U);
}

public void deposit(Counter U, ATMClient A, int N){
 ats.putcard(A,U);
 ats.deposit(A);
 ats.putmoney(A,N);
 ats.ok(A);
}

public void withdraw(Counter U, ATMClient A, int N){
 ats.putcard(A,U);
 ats.withdraw(A);
 ats.request(A,N);
 ats.ok(A);
}
 public int balance(Counter U){
   return ats.balance(U);
 }
}
```