ソフトウェア工学　135−2
（２００１．１１．２１）

ユーザインタフェースの代数的仕様記述と仕様からのプログラム生成

池田瑞穂　中村岳志　高田喜朗　関 浩之

奈良先端科学技術大学院大学 情報科学研究科

代数的仕様記述の部分クラスである抽象的順序機械型仕様 (ASM 仕様) を用いたユーザインタフェース（UI）の形式的仕様記述法とそこからのプロトタイプ生成法を提案する．まず，個々の UI モジュールがメッセージ送受信によって非同期に動作するような，簡潔な多プロセスモデルである抽象的ウィンドウシステム（AWS）モデルを導入する．次に，AWS に基づき UI の ASM 仕様を記述する方法について述べる．また，Java を実装アーキテクチャとした，UI の ASM 仕様を実装するための枠組を提案する．この枠組に従って ASM 仕様を Java プログラムに変換するコンパイラを作成した．コンパイラを用いて仕様記述例をコンパイルした結果についても述べる．

Algebraic Specification of User Interface and Its Automatic Implementation

Mizuho Ikeda　Takeshi Nakamura　Yoshiaki Takata　Hiroyuki Seki

Graduate School of Information Science
Nara Institute of Science and Technology

In this paper, we propose a formal description method of user interface (UI) based on a subclass of algebraic specifications called abstract sequential machine (ASM) specifications and also discuss a prototype generation method from an ASM specification. We first introduce an abstract window system (AWS), which is a simple multiprocess model where each UI module behaves asynchronously by sending or receiving messages. Based on AWS model, ASM specifications of some UI are presented. We also describe experiments with a compiler which translates an ASM specification into a Java program.

1　まえがき

近年, ソフトウェアシステムにおけるユーザインタフェース (UI) 部分の比重が高まっており, UI 設計のための方法論の確立が重要となっている [7]. ソフトウェア設計における形式的手法は, 仕様の厳密な記述, 段階的詳細化, 形式的検証が可能であるといった利点をもつが, これを UI 設計に用いる場合には, 次の要件を満たすことが求められる.

(1) 従来のソフトウェア設計と異なり, UI 設計では, 設計対象の振舞いや入出力関係に関する設計 (機能設計) に加えて, 画面上のレイアウトなどプレゼンテーションレベルでの詳細設計が必要である. 両者の設計にふさわしい仕様記述法をそれぞれ用意し, 特に機能設計レベルでは, プレゼンテーションレベルとは独立に閉じた仕様を記述できることが望ましい.

(2) UI 設計では，上流工程において, 最終成果物のユーザビリティ等を正確に予想することは困難である. 従ってプロトタイプによる評価を通し, 設計を逐次改善していく必要があり, そのためには, 仕様からのプロトタイプ自動生成が可能であることが望ましい.

本稿では, 代表的な形式的記述法の一つである代数的仕様記述法 [6] を用いた UI 設計法を提案する. 代数的仕様記述法は, 等式論理に基づき仕様の意味が簡明に定義される, 要求定義レベルからプログラムに対応する具体化レベルまで任意の抽象レベルで記述が行なえるなどの利点をもつ. 代数的仕様記述法の部分クラスとして, 抽象的順序機械型代数的仕様 (以下, ASM 仕様と略記) が知られている [11]. 一つの ASM 仕様は, 状態遷移関数, 状態成分関数の 2 種類の関数の宣言[1]および, 公理の集合からなる. 各公理では, ある状態遷移関数の適用により, 各状態成分関数の値がどのように更新されるかを, 遷移前の状態成分関数の値の組合せによって定義する. ASM 仕様は, 代数的仕様記述法一般の利点に加えて, 以下のような特長をもつ.

- 左線形無あいまいな項書換え系の部分クラスと見なせることから無矛盾性が保証される. 同様に, 完全性の

[1] 一般には, これに加えて補助関数を許す.

検査も容易である [11].

- 状態遷移関数を手続き, 状態成分関数を変数に対応づけることにより, 効率の良いプログラムへの変換が可能である [10]. この特長は, 上記要件の 2. に対して有効である.
- 仕様の形式的検証を行なう際に, 状態を表す項に関する構造的帰納法を用いるなど, 検証の方針が立て易い.

本稿では, UI の代数的仕様記述に先だって, 抽象的ウィンドウシステム (AWS) と呼ばれるモデルを提案する (2 節). AWS は, 個々の UI モジュールがメッセージ送受信によって非同期に動作するような簡潔な多プロセスモデルである.

次に, AWS に基づいた UI の ASM 仕様の記述法を述べる (3 節). そこでは, 個々の UI モジュールは一つの ASM として定義される. AWS 自身も, 個々の UI モジュール, およびメッセージ処理用キューを状態成分としてもつ一つの ASM として定義される. 提案する記述法に基づいた ASM 仕様の記述例を示す.

4 節ではまず, Java を実装アーキテクチャとし, UI の ASM 仕様を実装するための枠組を提案する. 一般に, UI の効率的な実装のためには, 多くの UI 部品がライブラリとして提供されていることが望ましい. 以降, 現在設計の対象となっている UI を, 「設計対象 UI」と呼ぶ. 概念的には, UI 部品および設計対象 UI のどちらも, AWS における UI モジュールであり, 代数的仕様においては, それぞれ一つの ASM 仕様によってその振舞いが定義される. 一方, UI 部品と設計対象 UI は, 実装において次のように扱いが異なる.

- 設計対象 UI の ASM 仕様を記述し, それをコンパイラに与えることにより, プロトタイプを生成する.
- UI 部品については, その ASM 仕様と実装の両方をあらかじめライブラリとして与えておく.
  - UI 部品の ASM 仕様は, その部品の利用者 (UI の設計者) へのインタフェースとして利用され, また, 設計対象 UI の ASM 仕様と合わせて, 閉じた ASM 仕様を形成する.
  - 一方, UI 部品の実装は, 設計対象 UI の ASM 仕様のコンパイル時にリンクされる.

  UI 部品の ASM 仕様においては, 必ずしもプレゼンテーションレベルの詳細すべてが記述されている必要はない. ASM 仕様の特長として, 状態成分関数ごとに独立に公理を記述できる. 従って, 仮にプレゼンテーションレベルの意味を ASM 仕様に追加したい場合でも, 必要な状態成分関数とその意味を定義する公理を追加するだけでよく, 既に記述した部分を変更する必要はない. さらに, 部品ライブラリは固定のものではなく, 必要に応じて随時追加すれば良い. 以上は, 上記要件の 1. に対する解決策となる.

図 1: SetValue の表示例

4 節では以上の方針に基づき, ASM 仕様から Java プログラムへのコンパイラについて述べる. 従来のコンパイル法 [10] と異なり, 本コンパイル法では, 各 ASM を Java の一つのクラスに対応させることにより, 複数の ASM を含む仕様（ある ASM が, 別の ASM の状態成分になっている場合も含む）を同時コンパイルすることが可能である. 試作したコンパイラを用いて, 3 節で示した仕様記述例をコンパイルした結果についても紹介する.

最後に, まとめと今後の課題を 5 節で述べる.

代数的仕様記述法を用いた UI 設計とプロトタイプ自動生成の先行研究に [3] がある. しかしそこでは, 副作用を利用した仕様記述を行なっており, 代数的仕様記述法の利点である意味定義の簡明さが失われている.

2 抽象的ウィンドウシステム

UI の形式的手法を用いた設計において, モジュール単位での仕様作成・修正・再利用などが容易に行なえるように, 抽象的ウィンドウシステム（AWS）というモデルを導入する. AWS は, 一般的なオブジェクト指向ツールキット[2]で構成される GUI アプリケーションを単純化したものであり, 個々の UI モジュールがメッセージ送受信によって非同期動作するような多プロセスモデルとなっている. AWS 上の各 UI モジュールをオブジェクトと呼ぶ.

以降, 図 1 のような簡単な UI モジュール（SetValue[3]）を例に説明する. この UI は, 以下のような機能をもつ.

- 内部にカウンタを 1 つもち, 4 つのボタン up, down, reset, ok と入力促進文, 現在のカウンタ値を表示する.
- ウィンドウ起動時, カウンタ値の初期値を表示し, ボタン up (down) をクリックすると, カウンタ値に 1 加算（減算）した値を表示する. ボタン reset のクリックによりカウンタ値は初期値にリセットされる.
- ok ボタンのクリックにより, この UI を生成した親オブジェクトのコールバックメソッドを呼び出す. このとき, その引数としてカウンタ値が親オブジェクトに渡される. そして, 各 UI 部品とともにウィンドウが消滅し処理が終了する.

SetValue オブジェクトは, 2 個の Label オブジェクトと 4 個の Button オブジェクトを構成分子として含んでいる. しかし AWS 上では, これらの構成分子も独立したオブジェクトとして扱われる. すなわち, 各 Label・Button

[2] X toolkit[5], Swing[8] など.

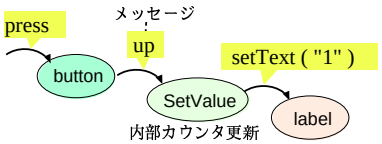

図 2: オブジェクト間通信

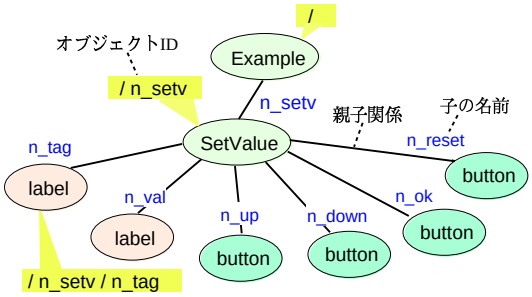

図 3: オブジェクト間の親子関係とオブジェクト ID

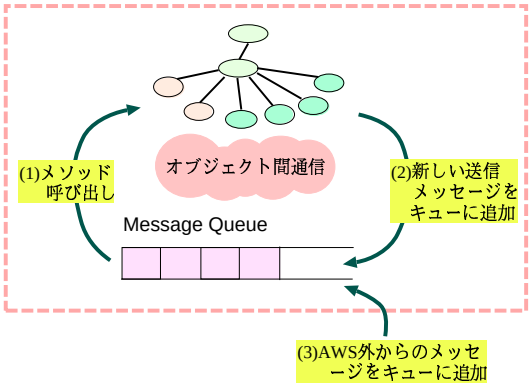

図 4: 抽象的ウィンドウシステム

オブジェクトも直接 AWS の管理下にあり，SetValue オブジェクト自身は，カウンタ値を管理する機能と，自分の構成分子であるオブジェクトと通信する機能だけを持つ．ユーザの操作を表すイベント，および，オブジェクト間の通信は，AWS のメッセージ配送機能を介して行われる．これを，SetValue 上の up ボタンが押された場合を例に説明する．up ボタンの押下は，AWS 外から Button オブジェクトへの press メッセージの送信として表される．AWS がこれを Button オブジェクトに配送する．press メッセージを受信した Button オブジェクトは，予め指定されたコールバックメッセージを SetValue オブジェクトに送信する．コールバックメッセージの受信により，SetValue オブジェクトは up ボタンが押されたことを知り，内部カウンタ値を増加させる．また，カウンタ値を表示している Label オブジェクトに，表示内容の更新を要求するメッセージを送信する（図 2）．

各オブジェクトの仕様は，

(1) そのオブジェクトが受け付けるメッセージ（メソッドとも呼ぶ）の集合，
(2) 各メッセージの受信（メソッド呼出しとも呼ぶ）の際に新たに送信するメッセージ（一般に複数），および，自身の状態変化の内容

から成る．(1) には新しいオブジェクト（その仕様のインスタンス）を生成するメッセージ（生成メッセージ）が含まれる．生成メッセージが送信された場合，AWS は新しいオブジェクトを生成する．生成メッセージの受信（生成されたオブジェクトが行う）に対しても (2) を記述でき，他の生成メッセージを送信するよう記述されていた場合は連鎖的にオブジェクトが生成される．オブジェクト o が生成メッセージを送信してオブジェクト o' を生成したとき，o を o' の親，o' を o の子と呼ぶ．

メッセージの送信先オブジェクトを指定するために，親オブジェクトは子の生成時に名前を指定し，以降，子にメッセージを送信する際はその名前を送信先として指定する．この名前は兄弟（同じ親に対する子）間で一意であればよい．一方，オブジェクトが親，自分自身にメッセージを送信する際は，それぞれ特別な名前 “**parent**”，“**self**” を送信先として指定する．オブジェクト間の通信は親子間でのみ行える[3]．AWS は，オブジェクト間の親子関係と，親が子に付けた名前とを組み合わせて，階層型ディレクトリにおけるパス名と同様な識別子を使って各オブジェクトを一意に識別する（図 3）．この識別子をオブジェクト ID と呼ぶ．

AWS は，各オブジェクトおよびメッセージ処理用キューから構成される（図 4）．AWS は以下の機能を持つ．

(1) キューの先頭メッセージを取り出し，送信先オブジェクトに対してメソッド呼出しを行う．
(2) メソッド呼出しの結果送信されたメッセージをキューに追加する．このとき，送信先に指定された名前をオブジェクト ID に変換する．
(3) AWS 外からのメッセージをキューに追加する．

3 UI の代数的仕様記述

3.1 代数的仕様

代数的仕様とは，3 項組 (S, F, AX) である．ここで，

- S はソートの有限集合．

[3]もし親または子以外のオブジェクトにメッセージを送信したい場合は，自分から相手までの，親子関係を辺とするパス上の各オブジェクトにメッセージを中継してもらう必要がある．

- F は S-シグニチャ, すなわち, S に属するソートの上の関数記号の宣言の有限集合.
- AX は公理の有限集合. ここで公理とは, あるソート $s \in S$ の項の順序対 $l = r$ である.

以下の例および実装したコンパイラへの入力では, 仕様記述言語 LOTOS[12] 中で用いられている, データ型記述言語 ACT-ONE[4] の構文に準拠して仕様を記述する (図 8 参照). $SP = (S,F,AX)$ を代数的仕様とする. F から生成される項の集合上の AX のすべての公理を満たす最小の合同関係を, SP の指定する合同関係といい, $\equiv_{SP}$ で表す.

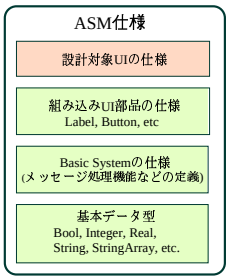

図 5: AWS の代数的仕様の構成

3.2 抽象的順序機械型代数的仕様

代数的仕様 $SP = (S,F,AX)$ が次の条件をすべて満たすとき, SP を ASM 仕様と呼ぶ.

(1) S は, 抽象的状態を表すソート state を含む.

(2) F の関数は, 次のように分類される.

(2.a) 状態遷移関数: 順序機械の状態を遷移させる関数. state 型の引数を高々1 個持つ. 関数値のソートは state である.

(2.b) 状態成分関数 : 各状態における状態成分を表す関数. state 型の引数をちょうど 1 個持つ. 関数値のソートは state 以外である.

(2.c) 補助関数 : 引数のソート, 関数のソートのどれも state 以外の関数.

(3) AX に現れる公理は, 次の条件をすべて満たす.

(3.a) 公理の左辺は線形で, かつ重なりを持たない [9].

(3.b) 公理の右辺に表れる変数は, その左辺にも現れる.

(3.c) 各関数の意味を定義する公理は以下の形式である. ここで, T, T' を状態遷移関数, O を状態成分関数, A を補助関数とする. さらに, s を state 型の変数とし, $x_1, x_2, \ldots, x_n$, $u_1, u_2, \ldots, u_m$ を state 型以外の変数とする.

形式 1 $O(T(s,u_1,\ldots,u_m),x_1,x_2,\ldots,x_n) = \cdots$

形式 2 $T'(s,x_1,x_2,\ldots,x_n) = \cdots$

形式 3 $A(x_1,x_2,\ldots,x_n) = \cdots$

形式 1 および形式 3 の右辺に状態遷移関数が現れてはならない. したがって, 状態遷移後の状態成分関数の値は, 遷移前の状態における状態成分関数や補助関数の値と, 状態遷移関数の state 以外のソートの引数のみに依存して定まる.

3.3 抽象的ウィンドウシステムの代数的仕様

抽象的ウィンドウシステム (AWS) の代数的仕様は, 次の 4 つの部分から構成される.

(1) 整数 (Integer), 文字列 (String) などの基本データ型の仕様.

(2) 2 節の末尾で述べた AWS の各機能を実現する部分 (BasicSystem と呼ぶ) の仕様. メッセージを表すデータ型やメッセージ処理用キューの仕様も含まれる.

(3) Label, Button などの組み込み UI 部品の ASM 仕様.

(4) 設計対象オブジェクトの ASM 仕様. この部分を設計者が記述する.

2 節で述べた例 (SetValue オブジェクト) のカウンタ値を保持する機能などは, カウンタ値を表す状態成分関数を宣言し, 各状態遷移の際のその成分の増減を公理で与えることで, ASM 仕様で容易に表すことができる. 次に, メッセージ送受信に関する部分を考える. 本仕様記述法では, メッセージ受信 (メソッド呼出し) を, そのオブジェクトの状態遷移として表現する. そこで, あるオブジェクトがメッセージ受信の際に新たにメッセージを送信したい場合, そのオブジェクトは他のオブジェクトに行って欲しい状態遷移を指定する. それに対して AWS が, (メッセージ処理用キュー内のメッセージの処理を表す状態遷移を行ったとき) 相手オブジェクトの状態を遷移させる.

メッセージの表現法 以降, 他のオブジェクトに行って欲しい状態遷移を指定したものをメッセージと呼ぶ. メッセージは, 送信先オブジェクトの名前, 状態遷移関数名, および任意個の引数から構成される. 例えば, 親オブジェクトから n_val という名前の Label オブジェクトに setText という状態遷移を要求する場合, このメッセージは

msg(n_val, Label_setText, "hello")

という項で表される. このメッセージは, オブジェクト n_val の抽象的状態を, 状態 s から状態 setText(s, "hello") に遷移させるよう要求している. ここで, n_val はソートが name である定数, Label_setText はソートが meth である定数である[4]. オブジェクトの名前や関数名を表す定数は, 公理で用いた分だけ自動的に宣言されると考える. ま

[4] 関数を値として扱えるよう代数的仕様を拡張する方法も考えられるが, ここではより単純に, 関数名を表す定数を各状態遷移関数ごとに用意することにする. (3.3.2 節参照)

た，関数 msg の引数の数や第 3 引数以降のソートはメッセージごとに変化するが，これも，適切な引数ソートを持つ msg が自動的に宣言されると考える．

AWS がこのメッセージを処理する（送信先オブジェクトの状態を遷移させる）際には，送信先オブジェクトを識別するため，送信先を名前ではなくオブジェクト ID で表す必要がある．上記メッセージの送信者である親オブジェクトの ID を j としたとき，送信先オブジェクトの ID は

child(j, n_val)

という項で表される．BasicSystem は，メッセージ処理用キューにメッセージを追加するとき，第 1 引数をオブジェクト ID に変換してから追加するよう設計されている．第 1 引数がオブジェクト ID であるメッセージは，

xmsg(child(j, n_val), Label_setText, ”hello”)

のように関数 xmsg を使って表す．

BasicSystem は，補助関数 apply を使って，このメッセージを状態遷移関数に対応づける．上の例の場合，この対応づけは以下のような公理によって行われる[5]．

forall o: oid, s: str, lbl: Label
ofsort Label
apply(xmsg(o,Label_setText,s),lbl) =
setText(lbl,s);

この公理は，状態 lbl である Label オブジェクトにメッセージ xmsg(o,Label_setText,s) を適用すると，この Label オブジェクトの状態が setText(lbl,s) に遷移するということを表している．この公理は状態遷移関数 setText の宣言（つまり引数と関数値のソート）のみから作ることができる．これらの公理の宣言も自動的に行われると考える．

3.3.1 BasicSystem の仕様

AWS は，各オブジェクトとメッセージ処理用キュー（キューと略記）を状態成分として持つ ASM である．すなわち AWS の抽象的状態は，全オブジェクトの状態およびキューの状態の組である．AWS の仕様中，メッセージ処理機能などの部分を BasicSystem と呼んでいる．BasicSystem の代数的仕様を図 6 に示す．ただし，メッセージを表すデータ型やキューの仕様は省略している．またこの他に，前述した，自動的に宣言されると考える関数（関数名を表す定数や msg など）や公理が加わる．BasicSystem の状態成分関数および状態遷移関数を以下に説明する．

- comp: オブジェクト ID が j であるオブジェクトの現在の状態を表す状態成分関数．
- mq: キューの現在の状態を表す状態成分関数．

[5]この公理は，3.2 節の形式 3 に当てはまらないが，xmsg の第 2，第 3 引数を取り出すような関数と if-then 関数を使えば，形式 3 の形にできる．ここでは読みやすさのため本文中のように書いた．

```
type basicsystem is messagequeue
  sorts state
  opns next  : state              → state
       call  : state, message     → state
       enq   : state, message     → state
       shift : state              → state
       init_state                 → state
       comp : state, oid          → obj
       mq    : state              → mqueue
  eqns
    forall s:state, m:message, j:oid
    ofsort state
      next(s) = if isempty(mq(s)) then s
        else next(call(shift(s),head(mq(s))));     (*B1)
    ofsort obj
      comp(call(s, m), j) = if receiver(m) eq j
          then apply(m, comp(s, j))
          else comp(s, j);                         (*B2)
      comp(enq(s, m), j) = comp(s, j);             (*B3)
      comp(shift(s), j)  = comp(s, j);             (*B4)
    ofsort mqueue
      mq(call(s, m))  = enqueue(mq(s),
        out(apply(m, comp(s,receiver(m))),
                                receiver(m));      (*B5)
      mq(enq(s, m))   = enqueue(mq(s), m);         (*B6)
      mq(shift(s))    = dequeue(mq(s));            (*B7)
      mq(init_state)  = empty_queue;               (*B8)
endtype
```

図 6: Basicsystem の代数的仕様

- next: キュー内のメッセージを，キューが空になるまで処理する．下記の call などを使って，形式 2 の公理（3.2 節）で意味定義される（公理 (*B1)）．
- call: 1 つのメッセージを処理する．すなわち，メッセージで指定された状態遷移関数を m，引数として与えられたデータを a とすると，送信先オブジェクト o の状態を $m(o,a)$ に遷移させる（公理 (*B2)）．同時に，out($m(o,a)$) をメッセージ処理用キューに追加する（公理 (*B5)）．out は，各オブジェクトが持つ，状態遷移時に送信するメッセージ列を表す状態成分関数である．関数 enqueue の第 3 引数は，第 2 引数であるメッセージ列の送信者の ID を表す．ここでは，メッセージ m の受信者が新しいメッセージ列の送信者である．送信者の ID は，メッセージの送信先をオブジェクト ID に変換するために用いられる．
- enq: AWS 外からのメッセージをキューの末尾に追加する（公理 (*B6)）．
- shift: キューの先頭要素を削除する（公理 (*B7)）．
- init_state: 初期状態を表す定数（公理 (*B8)）．

3.3.2 組み込み UI 部品の仕様

組み込み UI 部品の仕様例として，Button オブジェクトの ASM 仕様を図 7 に示す．ASM 仕様上では，組み込み UI 部品や設計対象オブジェクトなどの区別はないので，図 7 を設計対象オブジェクトの仕様記述例と考えてもよい．

AWS 上のオブジェクトの代数的仕様は，以下の条件を満たす ASM 仕様である．これらの制約は，4 節で述べるプロ

```
type Button is Widget
  sorts button
  opns Button  : str, meth   → button
       press   : button      → button
       destroy : button      → button
       out     : button      → mlist (* output message *)
       callback: button      → meth
  eqns
    forall b:button, g:meth, x:str
    ofsort mlist
      out(Button(x, g))    = [ ];
      out(press(b))        = [msg(parent, callback(b))];
      out(destroy(b))      = [ ];
    ofsort meth
      callback(Button(x, g))  = g
      callback(press(b))      = callback(b)
endtype
```

図 7: Button オブジェクトの ASM 仕様

トタイプ生成において, ASM 仕様とプロトタイプ間で名前の対応関係などを単純にするためのものであり, ASM 仕様の形式的意味を変更するものではない.

- ASM 仕様の名前 (キーワード type の右に書かれた Button) と同じ名前の状態遷移関数は, オブジェクトの初期状態を表す.
- 前述のように，out は状態遷移時に送信すべきメッセージ列を表す状態成分関数である．メッセージ列を $[m_1, m_2, \ldots]$ などと書く.
- destroy は画面上の表示を消す状態遷移関数である. ASM 仕様では, out 以外の状態成分関数がすべて未定義の状態に遷移するように定義する.
- ASM 仕様 A 中で状態遷移関数 m を宣言した場合, m の関数名を表すソート meth の定数 A_m も宣言されているとみなす. (定数 A_m はメッセージを構成する関数 msg や xmsg の第 2 引数に用いられる.)

仕様では, ユーザからの直接操作と他オブジェクトからのメッセージ受信に対してそれぞれ状態遷移関数を用意し, 公理を記述する. Button オブジェクトの仕様では, press 関数が前者, Button, destroy 関数が後者にあたる.

3.4 UI の仕様記述例

2 節で述べた SetValue の ASM 仕様を図 8 に示す．この UI の抽象的状態を表すソート名は setvalue である.

状態遷移関数として, 初期状態を表す SetValue, 4 つのボタンのクリックをそれぞれ表す up, down, reset, ok, および自分を消滅させる destroy が宣言されている. SetValue(x, g, v) は,「ラベルとして x が画面上に表示され, コールバックメソッドとしてメソッド g が指定され, カウンタ値が v である」ような状態を表す.

状態成分関数として, コールバックメソッドを表す callback, 現在のカウンタの値を表す get, カウンタの初期値を表す getinit, 送信すべきメッセージ列を表す out が宣言されている.

(*S1)–(*S4) は状態成分関数 out に関する公理である. 公理 (*S1) は，この UI は初期状態 SetValue(s, g, v) において, Button 型の 4 つのオブジェクトおよびカウンタ値を表示するための 2 つの Label 型オブジェクトの合計 6 つの部品を生成するためのメッセージを送信することを表す. 公理 (*S2) では，状態遷移 up が起こると, 遷移前のカウンタ値 get(s) に 1 加えた値を引数として, Label 型オブジェクト n_val に対してメソッド setText を送信すること，公理 (*S3) では，状態遷移 ok が起こると, 遷移前の値 get(s) を引数として，親オブジェクト **parent** に対してメソッド名 callback(s) のメッセージを送信すること，公理 (*S4) では，状態遷移 destroy が起こると，各 UI 部品のオブジェクトに対してそれらを消滅させるためのメッセージを送信することを記述している．公理 (*S5)–(*S8) は，状態成分関数 get に関する公理である．これらの公理により get の値は，初期状態 SetValue(s, g, v) においてはその入力パラメタ v に初期化され，状態遷移 up が起こると，遷移前の値に 1 を加えた値 (公理 (*S6))，状態遷移 reset が起こると，初期値 getinit(s)(公理 (*S7)) に変更され，一方，状態遷移 ok が起こっても状態遷移前と値が変わらないこと (公理 (*S8)) が記述されている.

4 コンパイラの実現と変換例

4.1 プロトタイプ生成の方針

本仕様記述法で導入した AWS モデルは，一般的なオブジェクト指向ツールキットに基づくものとなっており，現在利用可能なツールキットの多くは，プロトタイプを実装するためのアーキテクチャとして使用できると考えられる. 今回は可搬性の高さなどから，Java[2] および Swing ツールキット [8] を実装アーキテクチャとして選んだ．図 9 は, 3 節で述べた代数的仕様と，生成されるプロトタイプとの対応関係を示している.

今回作成するプロトタイプ生成用コンパイラは，設計者が記述した設計対象 UI の仕様を入力とし，それの実装となっているような Java プログラムを出力するものである. 具体的には，一つの ASM 仕様が Java における一つのクラスに，各状態遷移関数がそのクラスのメソッドに，各状態成分関数がそのクラスのデータメンバに，それぞれ変換される．状態遷移関数・状態成分関数の変換方法は [10] と同様である．ただし，状態遷移時に送信するメッセージ列を表す状態成分関数 out は，データメンバに変換せず，後述のような特別な扱いをする.

設計対象 UI 以外の部分は以下のように扱う.

基本データ型は，Java の int 型や String クラスなどのデータ型・クラスに対応づける．つまり，設計対象 UI の

```
type SetValue is frame
    sorts setvalue
    opns  SetValue              : str, meth, int   → setvalue
          up, down, reset, ok   : setvalue         → setvalue
          destroy               : setvalue         → setvalue
          callback              : setvalue         → meth
          get                   : setvalue         → int
          getinit               : setvalue         → int
          out                   : setvalue         → mlist
    eqns
        forall s: setvalue, x: str, g: meth, v: int
        ofsort mlist
            out(SetValue(x, g, v)) =
              [ msg(n_val, Label_Label, String(v)),
                msg(n_up, Button_Button, "up", SetValue_up),
                msg(n_ok, Button_Button, "ok", SetValue_ok),
               ··· ];                                          (*S1)
            out(up(s)) =
              [msg(n_val,Label_setText, String(get(s)+1))];    (*S2)
            out(ok(s)) =
              [msg(parent,callback(s),get(s))];                (*S3)
            out(destroy(s)) =
              [ msg(n_up, Button_destroy),
                msg(n_ok, Button_destroy),
               ··· ];                                          (*S4)
        ofsort int
            get(SetValue(x, g, v))    = v ;                    (*S5)
            get(up(s))                = get(s) + 1 ;           (*S6)
            get(down(s))              = get(s) - 1 ;
            get(reset(s))             = getinit(s) ;           (*S7)
            get(ok(s))                = get(s) ;               (*S8)
            getinit(SetValue(x, g, v))   = v ;
            getinit(up(s))               = getinit(s) ;
            ··;
        ofsort meth
            callback(SetValue(x, g, v))   = g ;
            callback(up(s))               = callback(s) ;
            ··;
    endtype
```

図 8: ASM 仕様記述例：SetValue

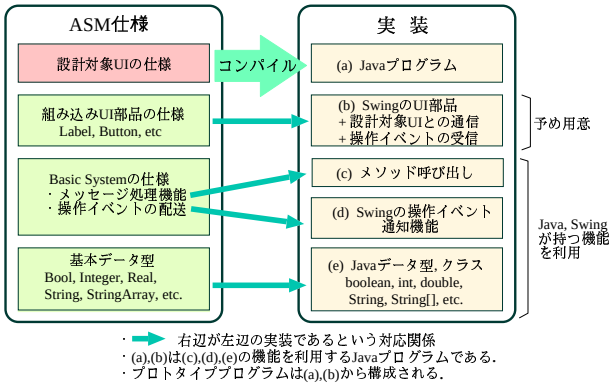

図 9: 代数的仕様と生成されるプロトタイプとの対応関係

仕様中に現れる基本データ型の演算に対して，Java のデータ型・クラスで定義された演算がその実装になっていると考え，単純にこれらの演算への変換を行う．

BasicSystem の機能のうち，オブジェクト間のメッセージ送受信機能は，Java での通常のメソッド呼出しによって実現する．例えば「SetValue オブジェクトが状態遷移 up を行った際，名前が n_val である子オブジェクトに setText メッセージを送信する」という仕様は，Java による実装においては，「SetValue クラスのメソッド up の中で，データメンバ n_val が指すオブジェクトのメソッド setText を呼び出す」という形で実現される．具体的には，状態遷移関数 up の実装であるメソッド内で，out 以外の状態成分に対する更新を行った後，out で指定されたメッセージ列の送信を表すメソッド呼出し列を実行する．子オブジェクトの生成も，親オブジェクトのメソッド内で，Java での通常のオブジェクト生成を行うことで実現する．作成された子オブジェクトへの参照を，親オブジェクトがデータメンバとして保持する．一方，親へのメッセージ送信を行うために，各オブジェクトは自分の親オブジェクトを知っておく必要がある．そこで実装上では，各生成メソッドは，仕様で宣言された引数に加えて，親オブジェクトへの参照を引数として受け取ることにし，親が子の生成メソッドを呼び出す際に自分への参照を渡すようにする．

AWS 外からのメッセージの受信，すなわち操作イベントの受信は，各組み込み UI 部品の実装において個別に実現する．すなわち，Swing が提供する操作イベント通知機能を使って，仕様に書かれた動作を実行するメソッドが操作イベント発生時に呼び出されるようにしておく[6]．

各組み込み UI 部品について，Java における一つのクラスの形でコンパイラ提供者が予め実装しておく．設計対象オブジェクトとのメッセージ送受信を行えるよう，設計対象オブジェクトと同様に，各状態遷移関数をメソッドとして，メッセージの送信を送信先オブジェクトに対するメソッド呼出しとして，それぞれ実装する．通常は，Swing で提供されている UI 部品クラスを利用し，メソッドの引数の型を仕様上の状態遷移関数の引数のソートに合わせるためのコードや，上述の，操作イベント通知に対して対応するメソッドを呼び出すためのコードなどを加えることで実装が行われる．

4.2　コンパイラの概要

4.1 節の方針に基づくコンパイラを Java プログラムとして実装した．字句解析部および構文解析部の作成には，字句解析器生成ツール JLex および構文解析器生成ツール Cup[1] をそれぞれ用いた．コンパイラプログラムのサイズ

[6] 今回作成したプロトタイプ生成用コンパイラへの入力では，Swing が提供する操作イベント通知機能と ASM 仕様上の状態遷移関数との対応を記述できないため，操作イベントを直接受信するオブジェクトは組み込み UI 部品に限られる．

```
public void ok() {
  int   prev_get = this.get;
  int   prev_getinit = this.getinit;
  Method  prev_callback = this.callback;

  this.get     = prev_get;
  this.getinit = prev_getinit;
  this.callback = prev_callback;
  try {
    Object[] args = { new Integer(prev_get) };
    prev_callback.invoke(parent,args);
  } catch (InvocationTargetException e) {
    throw new RuntimeException(e.getMessage());}
    catch (IllegalAccessException e) {
    throw new RuntimeException(e.getMessage());}
}
```

(a) メソッド ok

```
public SetValue(Object parent,
  String arg0,Method arg1,int arg2) {
    this.parent   = parent;
    this.get      = arg2;
    this.getinit  = arg2;
    this.callback = arg1;
    Container  cp = this.getContentPane();
    cp.setLayout(new FlowLayout());
    n_tag = new Label(this,arg0 + ":");
    cp.add(n_tag);
    n_val = new Label(this,String.valueOf(arg2));
    cp.add(n_val);
    n_up = new Button(this,"up",up);
    cp.add(n_up);
    n_down = new Button(this,"down",down);
    cp.add(n_down);
    n_reset = new Button(this,"reset",reset);
    cp.add(n_reset);
    n_ok = new Button(this,"ok",ok);
    cp.add(n_ok);
    ...
```

(b) 生成メソッド SetValue の一部

図 10: コンパイル結果：SetValue.java（一部）

は，JLex, Cup への入力である構文規則および意味動作の記述が約 700 行，コンパイラ内部で扱うデータ構造（クラス）の定義が約 1,400 行，コード生成部などが約 650 行である．

4.3 仕様例のコンパイル結果

3.4 節で示した仕様記述例（図 8）をコンパイルし得られた Java プログラムから特徴的な部分を抜粋し紹介する．コンパイル結果の一部を図 10 に示す．図 10(a) は，状態遷移関数 ok に対応するメソッドのプログラムコードである．ASM 仕様上での callback という状態成分関数は，実装上では `Method` クラス[7] のデータメンバ `callback` で表される．

親オブジェクトに対するメッセージ msg(**parent**, callback(*s*), get(*s*)) の送信は，親オブジェクトを記憶しておくための `Object` クラスのデータメンバ `parent` を用いて，

```
Object[] args = { new Integer(prev_get)};
prev_callback.invoke(parent,args);
```

と表される．ただし，`prev_get`, `prev_callback` は状態遷移前の get, callback の値を保持する変数で，このメソッドの冒頭でそれぞれ get，callback の値が代入されている．図 10(b) は，初期状態を表す状態遷移関数 SetValue に対応するメソッドのプログラムコードの一部である．メソッド内の初めの 4 行は，SetValue を左辺に含む形式 1 の各公理 (3.2 節) に基づいて，状態成分を表す各データメンバに初期値を代入している．5 行目以降は，out の仕様に基づいて，子オブジェクトの生成を行なっている．同時に，生成されたオブジェクトをウィンドウ上の構成分子として登録する作業も行っている．

5 あとがき

本研究では，AWS モデルに基づいた UI の ASM 仕様の記述法を提案し，UI 部品間の関係や画面遷移，データの扱いなど，UI の設計内容を明確に記述できることを示した．また，その ASM 仕様を Java プログラムに変換するコンパイラと，コンパイル例を示した．

今後の課題として，UI の ASM 仕様の形式的検証が挙げられる．また，今回考慮しなかった UI 部品（ラジオボタン，コンボボックスなど）を用いた仕様記述について検討し，それに対応するようコンパイラを改善する予定である．

参考文献

[1] Appel, A. W.: *Modern Compiler Implementation in Java*, Cambridge University Press, 1998.

[2] Arnold, K. and Gosling, J.: *The Java Programming Language*, Addison-Wesley, second edition, 1998.

[3] Cabrera, M., Torres, J. C. and Gea, M.: Towards user interfaces prototyping from algebraic specification, in *Design, Specification and Verification of Interactive Systems'99*, pp.67–83, Springer-Verlag, 1999.

[4] Ehrig, H. and Mahr, B.: *Fundamentals of algebraic specification 1*, Springer Verlag, 1985.

[5] Flanagan, D. ed.: *X toolkit intrinsics reference manual*, O'Reilly, third edition, 1992.

[6] Gognen, J. A. and Malcolm, G. eds.: *Software Engineering with OBJ*, Kluwer Academic Press, 2000.

[7] Newman, W. M. and Lamming, M. G.: *Interactive System Design*, Addison-Wesley, 1995.

[8] Walrath, K. and Campione, M.: *The JFC Swing tutorial*, Addison-Wesley, 1999.

[9] 二木厚吉, 外山芳人: 項書き換え型計算モデルとその応用, 情報処理, Vol.24, No.2, pp.133–146, 1983.

[10] ル一光, 栗屋英司, 関浩之, 藤井護, 二宮清: 抽象的順序機械の形で記述された代数的仕様からプログラムへの変換について, 信学論 (D-I), J73-D-I, No.2, pp.201–213, 1990.

[11] 杉山裕二, 谷口健一, 嵩忠雄: 基底代数を前提とする代数的仕様, 信学論 (D), J69-D, No.4, pp.324–331, 1986.

[12] 高橋薫, 神長裕明: 仕様記述言語 LOTOS, カットシステム, 1995.

[7] java.lang.reflect.Method クラス.