エージェント指向言語 Flage(2) − 言語仕様*

3B−2

大須賀 昭彦† 田原 康之† 粂野 文洋‡ 本位田 真一†
新ソフトウェア構造化モデル研究本部
情報処理振興事業協会 (IPA)

1 はじめに

本稿では，マルチエージェントに基づく仕様記述言語 Flage の言語仕様について述べる．Flage は，柔軟なソフトウェア構造を記述するための仕様記述言語で，メタ構造や場の概念によってソフトウェアの協調的な生成・変更過程を記述できる点に特徴がある．

2 言語の概要

図1に Flage のモデルを示す．Flage では，書換論理 [2, 3] などと同様に，系をエージェントの多重集合と捉え，エージェントの挙動を多重集合上の並列書換えとして定式化する．各エージェントは，属性とメソッドの定義を持ち，メソッドの呼びだしは非同期メッセージ交換によって行われる．各エージェントはメタ 0, 1, 2, ... レベルといった階層構造を持っており，メタ i ($0<i$) はメタ $i-1$ レベルのすべての定義や状態を属性として保持する．系には場の概念が存在し，エージェントは自分がどういった場に属しているかを認識している．1つのエージェントは (0 個以上の) 複数の場に属することができ，場への出入りも可能である．メッセージの交換は，同じレベルのエージェント間で，1対1または場へのブロードキャストとして行われる．すべてのエージェントを要素とする特別な場としてノードが存在するので，ノードを通じてすべてのエージェントにメッセージをブロードキャストすることも可能である．本稿では，エージェントのメタ i ($0\leq i$) レベルからみた $i+1$ レベルを (相対的に) メタと呼び，逆に $i+1$ レベルからみた i レベルをベースと呼ぶ．

3 言語の詳細

3.1 エージェントの表現

各レベルの個々のエージェントは以下のような項によって表現される．

$$\begin{array}{lll}\{\ aid & | & \langle\textbf{attribute}\rangle\ attr,\ldots,attr;;\\ & & \langle\textbf{method}\rangle\ mtd,\ldots,mtd;;\\ & & \langle\textbf{spec}\rangle\ spec,\ldots,spec;;\\ & & \langle\textbf{field}\rangle\ fid,\ldots,fid;;\ \ \}\end{array}$$

ここで，*aid* はエージェントの識別子，*attr* は属性名と値の対，*mtd* はメッセージパターンと手続きの対，*spec* は関数の代数的な定義，*fid* はエージェントが属する場の名前である．言語の構文規則を図2に示す．例えば，手続き proc の定義において，<= はメッセージの送信を，

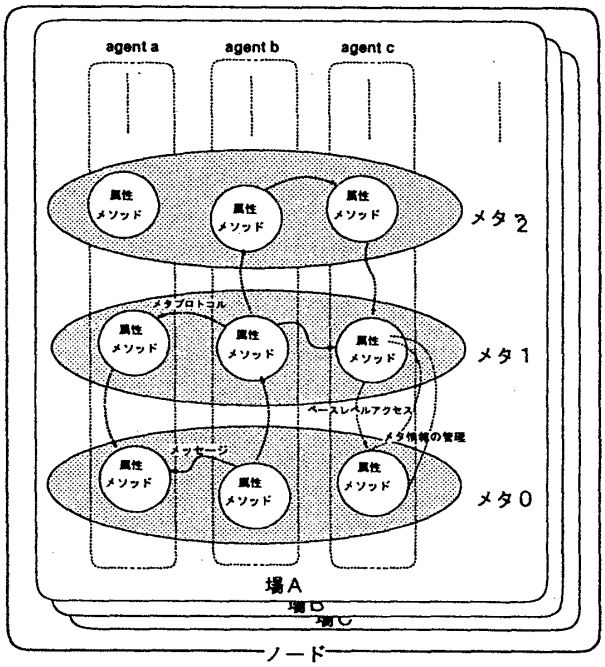

図 1: Flage モデル

! は呼びだし元への値の返却を，= は属性値の変更を表す．ここで，*term-with-atid* は定数として属性名を許した拡張項である．

3.2 メタ構造

各エージェントはメタ 0, 1, 2... といったレベルを持つ．エージェント *aid* のメタは識別子として m(*aid*) を持ち，その属性にベースの記述や状態が保持される．これにより，ベースの記述や状態がメタではデータとして扱えるため，プログラムの柔軟性に関する記述が可能となる．さらに m(*aid*) のメタ m(m(*aid*)) においては，*aid* に関する柔軟性の修正や変更などが記述できる．ベースを変更するための組込み関数として，ベースレベル・アクセスが用意されている．表1にベースレベル・アクセスの一部を示す．

3.3 通信メカニズム

メッセージの送信方法には，相手先の識別子を指定した送信と，場へのブロードキャスト，ノードへのブロードキャストがある．これらのメッセージは，すべてメタ間で受渡しされる．これにより，メッセージに対するメソッドの不備や予期せぬメッセージの存在をメタで検出し，メッセージの処理を拒否したり，ベースのメソッドの生成・適合化を行ったり，メッセージキューを書き換えて他のメッセージ処理を優先したりなどの対処を行うことができる．メタでメッセージを交換したり，エー

*Agent oriented language Flage (2) − Language design: Akihiko Ohsuga, Yasuyuki Tahara, Fumihiro Kumeno, Shinichi Honiden

†(株) 東芝より出向

‡(株) 三菱総合研究所より出向

```
agent-exp ::= "{" aid "|"
  [ "⟨attribute⟩" attr "," ... "," attr ";;" ]
  [ "⟨method⟩" mtd "," ... "," mtd ";;" ]
  [ "⟨spec⟩" spec "," ... "," spec ";;" ]
  [ "⟨field⟩" fid "," ... "," fid ";;" ]   "}"
aid ::= symbol | "m(" aid ")"
attr ::= atid ":" val
atid ::= symbol
val ::= term
mtd ::= msg-pttn ":" proc
msg-pttn ::= "(" mid [ term...term ] ")"
mid ::= symbol
proc ::= dest "<=" msg
   | "!(" dest "<=" msg ")"
   | atid "=" "(" dest "<=" msg ")"
   | atid "=" term-with-atid
   | "!" term-with-atid
   | proc ";" proc
   | "if" term-with-atid "=" term-with-atid
     "then" "(" proc ")" [ "else" "(" proc ")" ]
dest ::= aid | fid | bool-formla | fid ":" bool-formla
fid ::= symbol
msg ::= "(" mid [ term-with-atid...term-with-atid ] ")"
spec ::= sort-decl | op-decl | var-decl | eqn-decl
sort-decl ::= "sort" sid...sid
op-decl ::= "op" opid...opid ":" sid...sid "->" sid
var-decl ::= "var" variable...variable ":" sid
eqn-decl ::= "eq" term "=" term
```

図 2: Flage の構文規則

ジェントの生成を依頼するためのプロトコルとして，メタ・プロトコルが用意されている．表 2にメタ・プロトコルを示す．

3.4 場

場はエージェントが協調する際の範囲を与える概念で，通常はその名前によって識別される．各エージェントは自分がどのような名前の場に属しているかを知っており，メタは自らこの情報を書き換えることで，新しい場に参加したり，場から抜けだすことができる．すべてのエージェントを含む特別な場としてノードが存在し，以下の方法によって，ノード内のエージェントから動的に場を構成できる．*bool-formula* (論理式) で場を指定した場合，その時点で論理式を満足するエージェントに対するブロードキャストができる．また，*fid*:*bool-formula* なる形式で場を指定した場合，その時点で論理式を満足するエージェントから構成される場に *fid* という名前をつけて固定することができる．例えば，ノード内の全エージェントへのブロードキャストは，`true<=`*msg* と指定すればよい．

3.5 仕様の記述と検証

spec には関数定義だけでなくプログラムの抽象的な性質も記述できる．Flage では，任意の抽象レベルで書かれた記述に対して数学的な意味が与えられるので [1]，柔軟な制御が正しいことを検証する際や，エージェントがマイグレーションして新しい環境への適合が必要になった際などに，こういった記述を利用できる．

4 記述例

図 3にメッセージ処理の簡単な例を示す．メタエージェント `m(ex)` は，ex に送られてきたメッセージを mque (キュー) に加え，mcnt (メッセージ数) に 1 を足す．もしメッセージ数が 0 でなければ，関数 `sel-msg` が現状で最適なメッセージを選択し ex 上で実行する．このように，メタにおいてはメッセージの取捨選択などの処理が容易に記述でき，不要と判断したメッセージを送り主に返送することなども可能である．

表 1: ベースレベル・アクセスの例

呼びだし形式	意味
(get-attr *atid*)	ベースの属性 *atid* の値を参照
(req-add-attr *atid val*)	属性 *atid* : *val* をベースへ追加
(req-del-attr *atid*)	ベースの属性 *atid* を削除
(req-mod-attr *atid val*)	ベースの属性 *atid* の値を *val* に変更

表 2: メタ・プロトコル

メッセージ	意味
(send-msg *id msg frm to*)	*id*, *msg* を *frm* から *to* へ送信
(reply *id val*)	*id* に対して *val* を返却
(receive-msg *id frm*)	*id* の受理確認を *frm* へ送信
(broadcast *msg fid*)	*msg* を *fid* へブロードキャスト
(gen-agent *aid*)	エージェントのコピー *aid* を生成

```
{m(ex)|
  <attribute>
    mque:(), mcnt:0, selm:();;
  <method>
    (send-msg I M F ex):
       mque=((I M F) mque); mcnt=mcnt+1;
       F<=(receive-msg I F),
    (); if (mcnt=\=0)=true then
      ( selm=sel-msg(mque); mque=del(selm,mque);
        mcnt=mcnt-1; (exc-mtd selm) );;
  <spec>
    sort Que Msg,
    op   sel-msg:Que->Msg,
    var  Q:Que,
    eq   sel-msg(Q)= ... (get-attr ... ) ... ;; }
```

図 3: メッセージ処理の記述例

5 おわりに

本稿では Flage の言語仕様について述べた．今後は，さまざまな柔軟性の記述に Flage を適用し，記述性や検証性を評価していく．

謝辞 本研究は，産業科学技術研究開発制度「新ソフトウェア構造化モデルの研究開発」の一環として情報処理振興事業協会 (IPA) が新エネルギー・産業技術総合開発機構から委託をうけて実施したものである．

参考文献

[1] 本位田真一ほか: エージェント指向言語 Flage(1) ～ (5), 第 47 回 情報処理学会 全国大会 (1993).

[2] Meseguer, J.: A Logical Theory of Concurrent Objects, in *Proc. ECOOP/OOPSLA '90* (1990), pp. 101–115.

[3] 大須賀昭彦ほか: 並行オブジェクト系のための代数的記述法, 第 45 回 情報処理学会 全国大会 (1992), pp.4-243 ～ 244.