ソフトウェア工学 102-24
（1995．1.27）

協調型ソフトウェア・アーキテクチャに基づく開放型システムの仕様記述モデル

来間 啓伸* 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）

ネットワークで結ばれた計算機が各々の計算資源を相互に利用するシステムでは，システムの変化を前提として，部分的なふるまいに分割して捉えるほうがシステムのふるまいを明確に把握できる可能性がある．本稿では，このような「開いた」システムを対象とする仕様記述モデルについて述べる．このモデルは，エージェントと，エージェントの直接の影響範囲を規定する場から構成される．エージェントが置かれた環境に適合するための記述を，それ以外の記述とは分離するしくみとして，メタレベルの記述を用いる．エージェント指向言語 Flage に基づいた仕様記述言語を設定し，簡単な例題について記述例を示す．

An Agent-Oriented Specification Model for Open Systems

Hironobu Kuruma*, Shinichi Honiden**

Laboratory for New Software Architectures
Information-technology Promotion Agency (IPA)
Shuwa-Shibakoen-3chome Bldg., 3-1-38 Shibakoen, Minatoku, Tokyo 105, Japan

As the communications between software components by networks become wide, it is difficult to understand the behaviour of the whole system at any time. In this paper, we propose a specification model for such a system, on which we intend to represent the behaviors of subsystems that consist of closely connected software components. This model based on the concepts of agent and field; an agent is an autonomous object and a field is a finite set of agents. We adopt a meta-level architecture to separate the descriptions of the communications between subsystems from the descriptions of the functions of components using the communications.

---

* （株）日立製作所より出向
** （株）東芝より出向

1．はじめに

ソフトウェアのネットワークへの依存度が増すにしたがって，計算機システムの全てのふるまいを一元化して捉えることは困難になりつつある．このようなシステムでは，むしろシステムの変化を前提として部分的なふるまいに分割して捉えるほうが，システムのふるまいを明確に把握できる可能性がある．ここでは，このようなシステムを開放型システム[1]と呼び，次のように特徴付ける．

・構成要素は他の構成要素とは独立に機能する．
・構成要素は相互にコミュニケートする．
・システムを大域的に管理する機構を持たない．

開放型システムでは，システムに関する情報が完全ではないため，構成要素の変化を予測することはできない．構成要素の変化は構成要素間のコミュニケーションを通じて他の構成要素に伝えられるため，構成要素の変化に対処するためのコミュニケーションや，不正な動作からの回復処理は，システムのふるまいを把握するうえで基本的な要因となる．

ここでは，各構成要素を互いに協調するエージェントとみなし，これらの問題をエージェントの協調問題と捉えることにより，構成要素の機能に関する仕様と変化への適応に関する仕様を統一的な枠組みのもとに扱う，仕様記述モデルの構成を考える．協調ソフトウェア・アーキテクチャのためのエージェント指向言語 Flage [6][7] に基づいた記法を設定する．

2．仕様記述モデル

2．1　エージェント十場 モデル

対象を，場とエージェントの集まりによって表現する．エージェントは１つ以上の場に属し，同じ場に属するエージェントとのコミュニケーションを通じて自己の内部状態を変える．他の場に属するエージェントとのコミュニケーションは，場を接続するエージェントがメッセージを中継することによって行なうことができる．このようすを図１に示す．ここでは，エージェントａは場Ｆ１に，エージェントｂは場Ｆ１とＦ２に，エージェントｃは場Ｆ２とＦ３に，エージェントｄは場Ｆ３に属する．エージェントｂ，ｃは場を接続するエージェントである．ａが直接メッセージを送ることができるのは同一場内のｂに限られるが，ｂがｃに，ｃがｄにメッセージを中継することにより，ａは場Ｆ１，Ｆ２，Ｆ３を経由してｄにメッセージを送ることができる．

エージェントは自律的にふるまい，さらに，属する場を変えることができる．場は，有限個のエージェントを含み，それらの間のメッセージ授受を保証する．同一場内のメッセージ授受は，成功するか失敗するかのいずれかである．

エージェントは，あるエージェントを基準として，そのエージェントに至るまでに経由する場と，それを中継するエージェントの列によって，相対的に識別することができる．しかし，場およびエージェントは消滅あるいは変化し得るので，同一場内のエージェントを除いて，経由する場や中継するエージェントはそれらとのコミュニケーションを通してのみ知ることができる．また，同じ理由により，このように識別されたエージェントと常にメッセージ授受できるとは限らない．一般には，メッセージ授受毎に中継エージェントとのコミュニケーションが必要となる．このようなコミュニケーションを表現するために，メタレベル記述を用いる．

2．2　メタレベル・アーキテクチャ

一つのエージェントのふるまいを，０レベルからｎレベルまでの記述によって規定する．ここで，上位レベルの記述は下位レベルの記述に解釈を与える．０レベルの記述は対象の記述であり，０レベルをオブジェクトレベルと呼ぶ．また，ｍ（０≦ｍ＜ｎ）に対して，ｍレベルをベースレベル，ｍ＋１レベルをメタレベルと呼ぶことにする．なお，本稿ではｎの値は記述上の必要に応じて増やせるものとし，特に規定はしない．

各レベルには，次の内容を記述することができる．

・属性および一時変数
・ベースレベルの実行状態
　（オブジェクトレベルを除く）
・自己および他のエージェントとの同一レベルでのメッセージ授受

各レベルでは，メッセージ授受を行なって内部状態を変更する操作を記述することができる．オブジェクトレベルの記述は対象を表現し，各メタレベルの記述は，ベースレベルの記述の解釈を表わす．

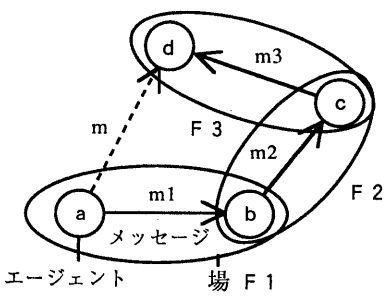

図１　エージェントの中継によるメッセージ伝達

メタレベルはベースレベルの実行状態を格納するメッセージ・キューを持っており，それを通じてベースレベルと結合する．mレベルの実行状態はメッセージ送信／受信を単位として，m＋1レベルのメッセージ・キューに格納される．m＋1レベルのメッセージ・キューの要素，つまり，m＋1レベルにおけるmレベルの実行状態の表現を，以下ではメッセージ閉包と呼ぶことにする．メッセージ閉包は，次の構造を持つ．

＜タグ　メッセージ・バケット　継続＞

・タグはメッセージの種類を表わす．

・メッセージ・バケットは，mレベルのメッセージの表現である．

・継続は，mレベルの実行を継続するのに必要な情報の集積であり，次の構造を持つ．

　＜局所環境　履歴＞

　・局所環境はmレベルの内部状態や処理の進行状態など．

　・履歴はメッセージに対する返信を評価する局所環境の退避スタック．

2．3　コミュニケーション

メタレベルでは，ベースレベルのふるまいを操作対象とすることができ，エージェント間の協調のように場や他のエージェントの状態に強く依存する処理を，そうでない部分から分離して記述するうえで有効である．したがって，エージェント＋場 モデルの下でメタレベル記述を用いることにより，構成要素の変化を前提としたシステムの仕様記述のための十分強力なモデルとなる．ここでは，エージェント間のコミュニケーション記述にメタレベルを用いた，仕様記述モデルについて述べる．

上述のメタレベル・アーキテクチャの下では，エージェントａがmレベルでエージェントｂにメッセージを送るふるまいは，次のようにモデル化される．

1．ａのm＋1レベルでメッセージ・キューからメッセージ閉包を取り出す．

2．メッセージ閉包をｂのm＋1レベルに伝達する．

3．ｂのm＋1レベルで，メッセージ閉包を受信メッセージとしてキューに格納する．

4．ｂのm＋1レベルでメッセージ閉包を取り出し，解釈する．

エージェントｂのmレベルからの返信もほぼ同様の順序で進行するが，エージェントａのm＋1レベルで返信のメッセージ閉包を取り出した後，メッセージ送信時のmレベルの局所環境をメッセージ閉包から回復して解釈する点が異なる．

この時，メッセージ閉包の伝達はm＋1レベルのメッセージ授受によって行われるので，mレベルのメッセージ授受に関する条件を埋め込むことが可能である．この結果，mレベルの1回のメッセージ送信は，一般にはm＋1レベルの複数回のメッセージ授受によって実行されることになる（図2）．このことは，エージェントの中継を通して目的エージェントに到達する処理を記述する際だけでなく，システム内に様々なコミュニケーション・プロトコルを持ったエージェントが混在する中で目的エージェントを探し出す処理を階層的に記述する際にも有効である．

また，mレベルのメッセージや実行状態をm＋1レベルで操作することができるので，mレベルの次の項目を操作する処理を，mレベルとは独立に記述することができる．

・メッセージの削除

・メッセージ送信先，返信先，内容の変更

・メッセージの処理順序の変更

・返信が到着するまでの処理の続行／停止

3．記法

一般に，エージェントの各レベルの記述は各々のメタレベルの記述によって解釈されるため，その記法も

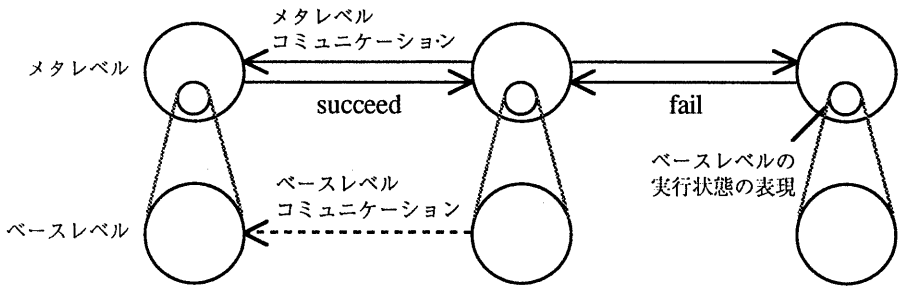

図2　動的なメッセージ授受におけるベースとメタの関係

メタレベルによって規定される．ここでは，各メタレベルには標準的な解釈メソッドが暗黙のうちに与えられていることを仮定して，一定の記法を与える．したがって，メタレベルによる解釈の変更とは，解釈メソッドの変更または追加を意味し，一般にはベースレベルの記法の変更をともなう．エージェントの記法を，図3に示す．エージェント記述子は記述上の識別子である．メタレベルでは，オブジェクトレベルのエージェント名に対してレベルの数だけmを付けたものをエージェント記述子とする．メッセージパターンには空メッセージの記述も許すものとし，アクティブなメソッドは，空メッセージによって起動されるパッシブなメソッドとして記述する．また，エージェントへの返信は，メッセージ識別子がreturnのメッセージ送信によって表わす．変数パターンへの代入は変数への代入の省略記法であり，左辺の変数パターンに現われる変数の各々に，右辺の式の対応する値が代入されるものとする．整数，文字などの基本データ型の値は各々エージェントとみなし，これらのエージェントは，各場で暗黙のうちに定義されているものとする．メソッドが属性値を書き換える場合，記述者は必要に応じて，そのことによる不整合を回避する記述を加えなければならない．特に，メソッドの実行が並列に行なわれる場合には，配慮する必要がある．

エージェント　::=
　"{" エージェント記述子 "|"
　［"< attribute >" 属性 "," … "," 属性 ";;" ］
　［"< method >" メソッド "," … "," メソッド ";;" ］ "}"
エージェント記述子　::=
　エージェント名　|　"m(" エージェント記述子 ")"
属性　::=　変数名 ":" 初期値
メソッド　::=　メッセージパターン ":" 手続き
メッセージパターン　::=
　"(" メッセージ識別子 ［項 … 項］ ")"
手続き　::=　文　|　文 ";" 手続き
文　::=　条件文　|　代入文　|　式
条件文　::=　"if" 式 "=" 式　"then" "(" 手続き ")"
　［"else" "(" 手続き ")" ］
　|　"case" 式 "of" パターン ":" "(" 手続き ")" …
代入文　::=　変数パターン ":=" 式
式　::=　項　|　メッセージ式
メッセージ式　::=　項 "<-" メッセージパターン
項　::=　エージェント名パターン　|　変数名
エージェント名パターン　::=
　エージェント名 "." 場の名前

図3　エージェントの記法

次の属性は，それぞれ各エージェント毎および各エージェントの各レベル（オブジェクトレベルを除く）毎に暗黙のうちに定義されているものとする．

・field　　場の集まり
・m_que　　メッセージ・キュー

メッセージ閉包は，次の構造を持つ．

＜タグ　メッセージ・バケット　継続＞

・タグの値は，メッセージ送信（send）またはメッセージ受信（receive）のいずれかである．
・メッセージ・バケットは，次の構造を持つ．
　＜受信先　メッセージ本体　送信元＞
　・受信先は，メッセージを受け取るエージェントのエージェント名パターン．
　・メッセージ本体は，送信／受信メッセージのメッセージパターン．
　・送信元は，メッセージを送るエージェントのエージェント名パターン．

空メッセージに対応するメッセージ閉包は，次の形式とする．

＜receive [ ] [ ]＞

エージェント内のmレベルの計算は，mレベルの環境とメッセージを引き数とするm＋1レベルのメソッドexecuteによって，実行される．

(execute　＜メッセージ・バケット＞　＜継続＞)

このメソッドは，以下の処理を行なう．

1．引き数の環境が空（新規メッセージ）の場合，mレベルの属性の値とメッセージパターンにマッチするメソッドから環境を作る．
2．メッセージ本体を環境の下で評価する．
3．評価の過程でメッセージ送信が現われたならば，そのときの環境からメッセージ閉包を作って処理を終わる．ここで，メッセージ閉包の環境にはその時の環境を，履歴には引き数として与えられた履歴の値をそのまま格納する．本稿ではexecuteの再定義は行わないものとし，環境の構造にも触れない．

4．記述例

例として，複数の研究会に所属している研究者aさんが，研究会Aで会った記憶のあるbさんに何かを依頼しようとする場合を考える．ここで，bさんは今は研究会Aのメンバではない，などの理由により依頼に失敗する場合があることを前提とする．この場合，aさんがbさんに何かを依頼するためには，bさんを捜すなどの，依頼内容とは直接には関連しない行動も行なわなければならない．一方，このような行動は，他の相手に対する依頼についても同様であり，「依頼する」という行動一般について共通性がある．

研究者をエージェント，その交流の場としての研究

会を場と考え，あるエージェントが同一場内のエージェントに対してメッセージを送るコミュニケーションとして，このような行動をモデル化する．この時，メッセージ授受は成功するとは限らないことを前提としてモデル化されていることが要求されるものとする．依頼する内容に関する記述とメッセージ授受に関する記述を分離することは前述の理由から自然であり，前者をオブジェクトレベルに，後者をメタレベルに記述する．すなわち，オブジェクトレベルでは，相手エージェントに対するコミュニケーションが正常に行われるものとして依頼する内容の記述を行う一方，メタレベルでは，このようなコミュニケーションを行なうための方法および失敗時の対処方法を記述する．オブジェクトレベル，メタレベルともに，相手エージェントを＜エージェント名＞.＜場の名前＞の形式で指定するが，オブジェクトレベルのコミュニケーションはメタレベルによって拡張されたものである．メタレベルの記述例を図4に示す．ここで，相手方エージェントも同様のメタレベル記述を持つものとする．

オブジェクトレベルのメッセージ送信は，メタレベルのメッセージ・キューからメッセージ閉包が取り出され，その伝達が成功するかあるいはエラー処理が行われることによって終了する．メッセージを送信したオブジェクトレベルのメソッドの実行はメッセージ閉包が作成された時点で停止し，メッセージ・キューから返信メッセージ閉包が取り出されて execute が呼び出された時点で再開される．相手エージェントからの返信がメソッドの実行に関係しないならば，メッセージ送信が終わった時点で execute を呼び出し，処理を続行させる記述を行なうこともできる．ベースレベルのメソッドの実行状態はメッセージ閉包に格納されてキューに並んでいるので，メタレベルでキューを操作することによって，メソッドの実行順序を操作することができる．例えば，特定のメソッドからの送信と，それに対する返信メッセージを優先して処理する，あるいはキュー中の送信／返信メッセージと新規メッセージを区別せず受け付けて複数のメソッドを並行して処理する，などの操作を記述することができる．

相手を捜す時に，名前だけが分かっている場合がある．この時には，例えば，自分が属する研究会全てについてその名前の研究者を捜し，見つからなければ，研究会内の友人をたどって別の研究会についても捜してもらう，という方法で相手を捜すことが考えられる．このような捜し方は，図4のメソッドに図5のメソッドを追加することによってモデル化することができる．この記述例では，オブジェクトレベルで相手エージェントを指定する形式を＜エージェント名＞だけにし，メソッド (forward_to) によって所属する全ての場を捜す．もしそこに＜エージェント名＞に適合するエージェントが見つからなければ，場内の全てのエー

```
{ m(agent) ;
  <method>
  ([]) :
    if not (m_que <- (is_empty))
    then (
      <Tag Packet Cont> := m_que <- (dequeue) ;
      case Tag of
      send: (
        NCont := self <- (get_continuation Packet Cont) ;
        Result :=
          self <- (send_closure <send Packet NCont>) ;
        if Result = fail
        then (self <- (error_handler <send Packet NCont>) ) )
      receive: (
        Result := self <- (execute Packet Cont) ;
        m_que <- (enqueue Result) ) ) ,

  (get_continuation Packet Cont) :
    <To MesBody From> := Packet ;
    if MesBody = (return  Args)
    then (<Env RetEnv:History> := Cont ;
      sender <- (return <RetEnv History>) )
    else (<Env History> := Cont ;
      sender <- (return   <[] Env:History>) ) ,

  (send_closure MesClosure) :
    <Tag Packet Cont> := MesClosure ;
    <To MesBody From> := Packet ;
    case To of
      AgentName.FieldName : (
        if field <- (is_in FieldName)
        then (Result := AgentName.FieldName
            <- (receive_closure MesClosure) ) ;
          sender <- (return Result))
        else (sender <- (return fail) ) )
      otherwise : (sender <- (return fail) ) ,

  (receive_closure <Tag Packet Cont>) :
    if self <- (executable Packet Cont)
    then (<To MesBody From> := Packet ;
      NewPacket := <To MesBody From:sender> ;
      m_que <- (enqueue <receive NewPacket Cont>) ;
      sender <- (return succeed) )
    else (sender <- (return fail) ) ;; }
```

図4　同一場内のコミュニケーション

ジェントに forward_to メッセージを送って転送を依頼する．forward_to メッセージを受け取ることのできるエージェントは，同様の手順で次々と転送の範囲を広げて行く．

5．おわりに

エージェント，場，メタから構成される仕様記述モデルについて述べた．ここでは，エージェントのメッセージ到達範囲を場によって制限することにより，エージェントの変化の影響が直接及ぶ範囲を限定した．場を越えたメッセージ授受は，複数の場に属するエージェントが連鎖的にメッセージを中継することによって行なうため，これらの中継エージェントはメッセージ授受を直接操作することができる．したがって，メッセージ授受に関して，エージェントの変化に対して一定の耐性を持つ記述を行なうことができる．

エージェントの変化に対応するための記述を，それ以外の部分の記述から分離するために，メタレベル記述を用いた．一般に，メタレベルの記述はベースレベルの記述に解釈を与えるが，本稿ではその範囲をメッセージ授受に限定し，ベースレベルのメッセージ送信／受信をメタレベルのメッセージ送信／受信によって操作的に解釈するものとした．したがって，ベースレベルでは抽象化したメッセージ送信／受信を用いた仕様記述，メタレベルではそのメッセージ送信／受信の実行手順の記述，と分離することになり，エージェントの変化に対応するためのメッセージ授受をメタレベルに階層的に記述できる．その反面，本稿のモデルでは，レベルに分けられた記述が操作的な解釈によって結合されるため，仕様を解析／検証するための枠組みとしては不十分である．

一般に，メタレベルではベースレベルのふるまいを操作対象とすることができるので，動作時の状況に応じて柔軟に変化するシステムや，変化しながら長期間動作し続けるシステムの記述に適していることが期待される．本稿の仕様記述モデルをこのような対象について評価すること，および，仕様を解析／検証するための枠組みの検討は，今後の課題である．

```
(send_closure  MesClosure) :
  <Tag Packet Cont> :=  MesClosure ;
  <To MesBody From> := Packet ;
  case To of
  AgentName : (
    Res := self <- (forward_to AgentName MesClosure {}) ;
    sender <- (return Res))
  otherwise : (sender <- (return fail) ) ;

(forward_to AgentName  MesClosure SearchedFields) :
  <Tag Packet Cont> :=  MesClosure ;
  <To MesBody From> := Packet ;
  NewPacket := <To MesBody From:sender> ;
  NewMesCont := <Tag NewPacket Cont> ;
  SetOfFields := (field <- (retrieve_all)) - SearchedFields ;
  if SetOfFields = [] then (sender <- (return fail));
    elseif (for all Field in SetOfFields (AgentName.Field
      <- (receive_closure NewMesCont) = fail))
    then (SearchedFields := SearchedFields + SetOfFields;
      if (for all Field in SetOfFields (
        SetOfAgents := Field <- (retrieve_all) ;
        for all Agent in SetOfAgents (Agent.Field
        <- (forward_to AgentName NewMesCont
        SearchedFields)))) = fail then (sender <- fail)
      else (sender <- succeed) else (sender <- succeed))
```

図5　場を越えたコミュニケーション

［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] Carl Hewitt and Peter de Jong, Open Systems, On Conceptual Modelling, Springer-Verlag, pp. 147-162 (1984)

[2] Pattie Maes, Issues in Computational Reflection, Meta-Level Architectures and Reflection, Elsevier Science Publishers B.V. (North-Holland), pp. 21-35 (1988)

[3] Takuo Watanabe and Akinori Yonezawa, Reflection in an Object-Oriented Concurrent Language, OOPSLA '88 Conference Proceedings, pp. 306-315 (1988)

[4] 渡辺卓雄, リフレクション, コンピュータソフトウェア, Vol. 11, No. 3, pp. 5 - 14 (1994)

[5] 岸本芳典, 小高信人, 本位田真一, 協調によるメソッド適合化について, マルチエージェントと協調計算ＩＩ 日本ソフトウェア科学会ＭＡＣＣ'９２, 近代科学社 (1993)

[6] 本位田真一ほか, エージェント指向言語Flage（１）－構想, 第４７回情報処理学会全国大会論文集 (1993)

[7] 大須賀昭彦ほか, エージェント指向言語Flage（２）－言語仕様, 第４７回情報処理学会全国大会論文集 (1993)

[8] Kinji Mori, Autonomous Decentralized Systems: Concept, Data Field Architecture and Future Trends, Proceedings of ISADS '93, pp. 28-34 (1993)