社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－SE－148 (1)
2005／5／31

知識ベースを用いたシステム設計における仕様の検証と直接実行

金子 誉万† 間野 暢興†

実時間・分散システムを対象とした、知識ベースを用いたシステム設計方式を本論文で提案する。本研究のアプローチは論理的なアプローチに属し、本方式のモデルは状態記述を基としている。知識ベースにおけるモデルは、本方式独自の仕様記述言語の記述から組み立てられる。これらの知識ベースにおけるモデルは、ソフトウェア開発の上流工程におけるシステム設計の（半）自動化（本研究では、検証によるデッドロックの検出、仕様の直接実行システムによる仕様の確認を扱った）に大いに貢献する。本論文では ATM システムを例に本方式の利用形態について説明する。

Verification and Direct Execution of System Design Using a Knowledge-Based Modeling Approach

Takakazu Kaneko† and Nobuoki Mano†

We propose an approach using models in the knowledge-base for the system design of real-time distributed systems. Our approach belongs to a logical one and uses models based on state descriptions. Models in the knowledge-base are constructed from the specification descriptions in our specification description language. These models greatly contribute to the semi-automation of various activities in the course of system design, such as design verification with deadlock detection and specification validation by direct execution of specifications. We explain these processes, using ATM system design as an example.

1 はじめに

近年、実時間・分散システムの効果的な開発方式の必要性が高まっている。これらのシステムは事象、メッセージの交信、時間情報の表現を扱う必要がある。

並行システムの動作列記述を中心としたアプローチとしては、シナリオに基づく方法[1]や代数的仕様法などがある。これに対して、論理表現を仕様記述に用いるアプローチには、Eiffel[2]、VDM-SL[3]やシナリオからの状態図生成[4]がある。最近注目を浴びているモデル駆動型開発方式(MDA)[5]は、UML[6]とOCL[7]を併用することにより、設計者により自然な方式でソフトウェア開発を進行させるもので、OCLは論理表現をベースにしている。

本方式では、並行システムの活動全般をプロジェクトと呼び、インタラクションフェイズの集まりとそれらの間の遷移に分割して表す。一つのインタラクションフェイズは幾つかのコンポーネントが交信を含むアクション列により相互に作用しながら進行する。アクションの仕様は、述語とメッセージ送受信に関する記述を含む論理式で表した事前事後条件で定義される。仕様記述言語により記述されたアクション、インタラクションフェイズ、プロジェクトの事前事後条件と内部記述は、モデル表現されて知識として知識ベースに蓄積される。これらの知識は、次の２つの設計フェイズで活用される：①Finite State Process(FSP[1])により指定されたインタラクションフェイズの並列有限状態プロセスの順方向検証、②①により作り出された制御構造モデルを基にした仕様の直接実行（エージェントシステム[8]の考え方にヒントを得た）。これらは、設計の半自動化を目指す場合に、欠くことのできない基本的機能であると考えられる。制御構造モデルは、さらに段階的精製を経てJavaなどのプログラミング言語のコードに変換される。その過程においては、アクション知識を用いたプラン合成[9]が必要となるものと予想される。

明星大学大学院 情報学研究科†
Graduate School of Informatics, Meisei University†

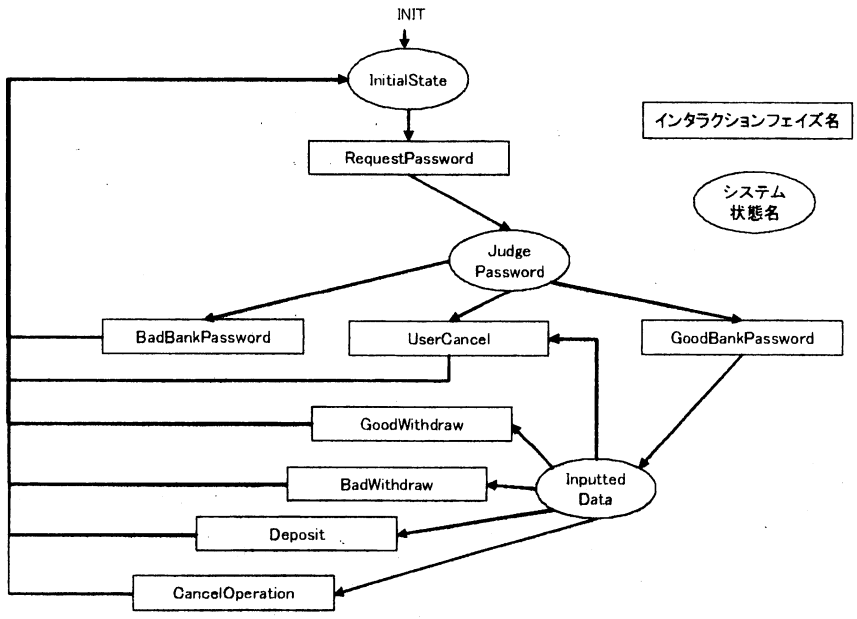

図1　ATM システムのプロジェクト表現

2　本方式の主要概念とシステム構成

2.1　主要概念

以下に、対象のシステムを記述するための、本方式で用いる主な用語を説明する。

- プロジェクト
  システム全体を表す。インタラクションフェイズとその間の遷移から成る（文献[1]の high-level Message Sequence Chart:hMSC と似た表現となっている）。図1は ATM システムのプロジェクト表現である。システムは矢印に従って遷移する。プロジェクト表現はシステムの正規の処理、失敗した場合の処理、キャンセルすべてを含んだものになる。状態名は次に遷移するインタラクションフェイズの操作の内容を総括した名前になっている。
- コンポーネント
  エージェントのように自律的に動作するオブジェクトである。Use、ATM、Consortium、Bank は ATM システムのコンポーネントである。
- インタラクションフェイズ
  コンポーネント間の相互作用を表したモデルである（文献[1]の basic Message Chart:bMSC と似た表現となっている）。図2は ATM システムのプロジェクトにおける BadBankPassword インタラクションフェイズである。このインタラクションフェイズはユーザのアカウントとパスワード（誤り）を ATM、Consortium を経由して銀行で認証を行うものである。
- システム状態
  インタラクションフェイズ間のグローバル状態を表す。
- 受動エレメント
  Bank における口座データベースのような受動的オブジェクトを表す。
- アクション
  コンポーネントと受動エレメントの動作を言う。図2における bMSC のメッセージを送信するノードとメッセージを受信するノードはコンポーネントのアクションのインスタンスを表す。
- ローカル状態
  コンポーネント内の隣接した2つのアクション間のローカル状態を表す。

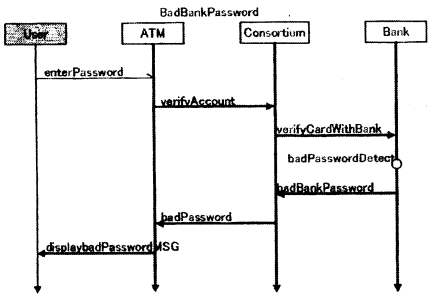

図2　BadBankPassword
インタラクションフェイズのbMSC表現

2.2　システム構成

システムの全体の構成を図3に示す。

本方式では述語論理式を仕様記述言語で知識ベースに登録するために、テキスト表現を用いた。これらの知識は、モデル表現に変換され、知識ベースに蓄えられる。また、FSPを基にしたシステム構造記述言語を使用することでプロジェクトとインタラクションフェイズの内部の構造記述を与える。

これらの情報を基に、システムは以下の設計活動を行う。

(1)　対象システムの設計モデルの前向き検証、及び、デッドロック検出。

(1-1)　プロジェクトの仕様におけるインタラクションフェイズの事前条件・事後条件、及び、システム状態の表明を基にプロジェクトの検証を行う。

(1-2)　インタラクションフェイズの仕様におけるアクションの事前条件・事後条件を基にインタラクションフェイズの検証を行う。

(2)　仕様の直接実行

bMSCにおけるアクションの仕様記述はテストデータを用いてインタプリティブに実行することができ、それによって対象のシステムの動作を確認できる。

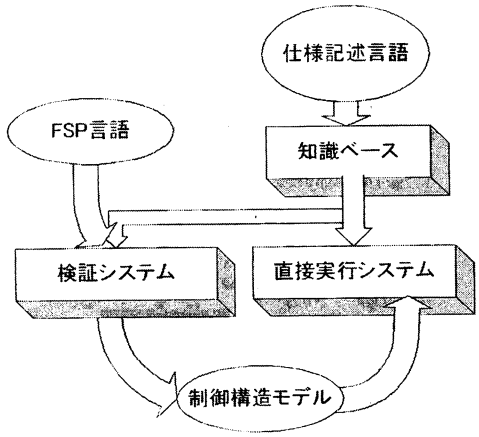

図3　システム全体構成

3　仕様記述言語と知識ベースにおけるモデル表現

3.1　入力言語

3.1.1　知識記述言語

1つは、知識ベースに登録される、アクション、インタラクションフェイズ、プロジェクトを記述するための言語である。その中のアクションとインタラクションフェイズは、事前条件、事後条件の記述を含む。これらの条件記述は、メッセージを除き、全て述語論理式で記述する。述語引数は、現在のところ、文字定数、変数のみを扱い、述語引数での関数の記述は含まないものとしている。変数としてコンポーネント名を記述する際、コンポーネント名の前に「*」を付け、それ以外の場合は「?」を付ける。論理演算子は、And(カンマ記号)、Or（事後条件内のみの使用）、Not (!)、Imply (=>)、を用いる。

アクションの記述の例を図4に示す。

この2つのアクション定義は、図2のUser、ATM間のenterPasswordの矢印の両端におけるアクションを、送信側では「sent_」、受信側では「receive_」を前に付けて別々に表したものである。送信側アクションの事後条件では「sent」を、受信側アクションの事前条件では「message」を用いて、送信主、受信主、送受信内容、の順に記述する。

```
actionDef User::send_enterPassword(){
  pre:ATM(*ATM),User(*User),
      passwordRequesting(*ATM,*User);
  post-0:inputted(*User,?acc),  inputted(*User,?pass),
       account(?acc),password(?pass),
       verifying(*User,*ATM),!passwordRequesting(*ATM,*User),
       sent(?User,*ATM,accountNum(?acc),password(?pass),
         verifyAccount(?acc,?pass));
}
actionDef ATM::receive_enterPassword(){
  pre:ATM(*ATM),User(*User),passwordRequesting(*ATM,*User);
     message(*User,*ATM,accountNum(?acc),password(?pass),
         verifyAccount(?acc,?pass));
  post-0:verifyAccount(?acc,?pass),verifyingfor(*ATM,*User),
       !passwordRequesting(*ATM,*User);
}
```

図４　仕様記述例（パスワード入力部分）

インタラクションフェイズの仕様記述もアクションと同様に事前条件、事後条件と、それに加えて対象のインタラクションフェイズに登場するコンポーネントも記述する。インタラクションフェイズの記述例を図５に示す。

BadBankPassword インタラクションフェイズには、User、ATM、Consortium、Bank のコンポーネントが存在し、誤ったパスワードを送信するインタラクションフェイズであることから、badPassword(?pass)が事後条件に記述される。

```
interactionPhaseDef BadBankPassword {
  pre:ATM(*ATM),User(*User),
       Consortium(*consort), Bank(*bank),
      AccountBase(?accBase), in(?accBase, *bank),
      passwordRequesting(*ATM,*User);
  post-0:badPassword(?pass),
           !passwordRequesting(*ATM,*User);
}
```

図５　インタラクションフェイズ
仕様記述例(BadBankPassword)

このパスワードの認証は、Bank 内にある受動エレメント AccountBase の操作、badPasswordDetect を用いて行われる。badPasswordDetect の仕様記述例を図６に示す。

```
actionDef AccountBase::badPasswordDetect
        (?acc,?pass,?accBase) {
   pre: accountNum(?acc),password(?pass),
       AccountBase(?accBase);
   post: badPassword(?pass);
}
```

図６　アカウントの認証の仕様記述

3.1.2　FSP 言語

もう１つの入力言語は、プロジェクト、インタラクションフェイズ等のモジュールの構造を指定するための言語である。プロジェクトの内部構造はインタラクションフェイズと状態がネットワーク状につながったものにより表される。各々のインタラクションフェイズの内部構造は、その内部に複数のコンポーネントの記述を持つ。インタラクションフェイズは分岐や反復を含まない。

それぞれの入力言語について以下に示す。

(1) インタラクションフェイズ間の遷移指定

インタラクションフェイズ間の遷移指定の主な記述は以下の通りである。

- 状態名宣言
  インタラクションフェイズ遷移定義内で登場する状態を、ここで列挙する。
- インタラクションフェイズ宣言
  インタラクションフェイズ遷移定義内で登場するインタラクションフェイズ名を、ここで列挙する。
- 初期状態指定

```
projectVerify ATM_system {
  COMPONENT: User,ATM,Consortium, Bank;
  PELEMENT: AccountBase;
  STATE: InitialState, JudgePassword, (略);
  INIT = InitialState;
  InitialState = (RequestPassword -> JudgePassword);
  JudgePassword = (BadBankPassword -> InitialState | GoodBankPassword -> (略));
  ASSERTION:  InitialState@(User(user1),ATM(ATM2),  Consortium(consort1), Bank(Bank1),
                  AccountBase(AccBase1), in(AccBase1, Bank1)),
                  JudgePassword@(passwordRequesting(*ATM,*User));
                      (省略)
}
```

図７　インタラクションフェイズ間の遷移指定の記述例

このプロジェクトにおいて、最初に実行するインタラクションフェイズを指定する。

- インタラクションフェイズ遷移定義
  従来のFSPの記述とほぼ同じである。ただし、本システムではFSPの一部の記述のみを使用する。従って、when等の記述はできない。
  記述の左辺は状態名を記述し、右辺はアクションの実行順序を記述する。アクション間の順序制約は「->」によって表す。アクションの実行順序の最後は必ず状態名宣言で宣言した状態名が記述されなければならない。
- 状態表明
  システムが、指定した状態に遷移したときの状態表明を記述する。

インタラクションフェイズ間の遷移指定の記述例を図7に示す。

この記述は、図1で示したプロジェクトの一部を記述したものである。対象のシステムに、状態InitialState、JudgePassword、他、インタラクションフェイズRequestPassword、BadBankPassword、GoodBankPassword、他などがある。また、システムの初期状態はInitialStateである。システムは状態InitialStateからそれぞれのインタラクションフェイズを処理を進めるごとに状態が「->」記号に従って遷移する。また、各々の状態はASSERTIONにおいて状態表明をする。

(2) 並列プロセス記述

アクション間の遷移順序はプロジェクトの記述におけるインタラクションフェイズ間の遷移と同様に、「->」記号で表現する。指定したアクション、コンポーネント、インタラクションフェイズは必ず知識ベースに登録されているものでなければならない。

並列プロセス記述の例を図8に示す。

この記述はBadBankPasswordインタラクションフェイズを記述したものである。BadBankPasswordインタラクションフェイズにはUser、ATM、Consortium、Bankコンポーネントがあり、各々のコンポーネントのアクションの実行順序を「->」に従って記述する。

3.2 知識ベース

仕様記述言語によって記述された仕様は、解析され、モデル表現に変換され、知識ベースに格納される。本システムではJAVAのクラス階層構造を利用することによって知識ベースを構築した。この知識ベースは、Project、InteractionPhase、Action、SystemState、Component、LocalState、PElementクラスからなる。

4 仕様の検証と直接実行

4.1 対象システムの検証

順方向検証システムは3.1.2で示した入力言語を用いた記述により、(1) プロジェクトにおけるインタラクションフェイズ間の遷移指定の検証と、(2) インタラクションフェイズにおける各コンポーネントごとのアクションの実行順序を指定した並列プロセス記述とを基にしてシステムの検証を行う。検証時に、記述内に登場するアクション、インタラクションフェイズ、コンポーネントは知識ベースから詳細な情報を取得する。検証の過程において生成された検証木からデッドロックの有無を判定する。正当な仕様であることが確認された場合、制御構造モデルを知識ベースに登録する。

順方向検証システムの構成を図9に示す。

```
interactionPhaseVerify BadBankPassword {
  User = (send_enterPassword() -> receive_requestPassword());
  ATM = (receive_enterPassword() -> send_verifyAccount() ->
         receive_badPassword() -> send_requestPassword());
  Consortium = (receive_verifyAccount() -> send_verifyCardWithBank() ->
         receive_badBankPassword() -> send_badPassword());
  Bank = (receive_verifyCardWithBank() -> badPasswordDetect() -> send_badBankPassword());
}
```

図8 並列プロセス記述例(BadBankPassword)

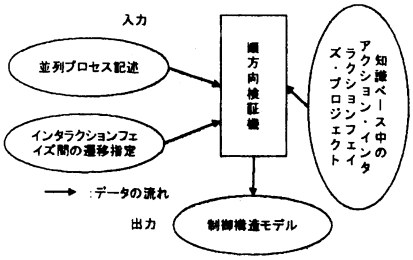

図９　検証システムの構成

（1）プロジェクトの検証

検証が正しく行われたかどうかは、検証システムによって生成される検証木を基に判定する。検証木において終端ノードが無い場合はシステムがデッドロックに陥ることを意味するため、仕様からデッドロックを検出したことになる。図１０は図１のプロジェクトを検証した際に生成される検証木の一部である。

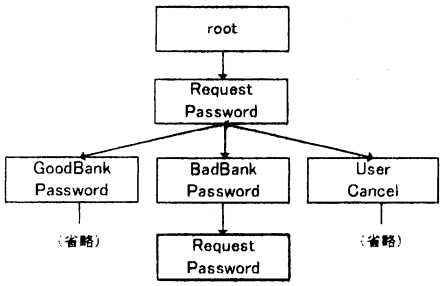

図１０　ATM システムのプロジェクト検証

図中の太い線は終端ノードを表す。RequestPassword ノードから GoodBankPassword、BadBankPassword、UserCancel のいずれかに遷移することを示している。

（2）インタラクションフェイズの検証

インタラクションフェイズの検証の評価も検証木を基に判定する。アクションが半順序で実行される場合は検証木に枝分かれが生じる。検証木におけるあらゆるパスが対象のインタラクションフェイズの事後条件に到達している場合は仕様にデッドロックが無いことを意味する。

図１１は BadBankPassword インタラクションフェイズの検証木の一部である。

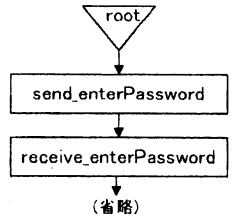

図１１　BadBankPassword
インタラクションフェイズの検証木

BadBankPassword インタラクションフェイズは最初に send_enterPassword アクションが実行されるため、root ノードの子ノードとして登録される。次に receive_enterPassword アクションが実行されるので

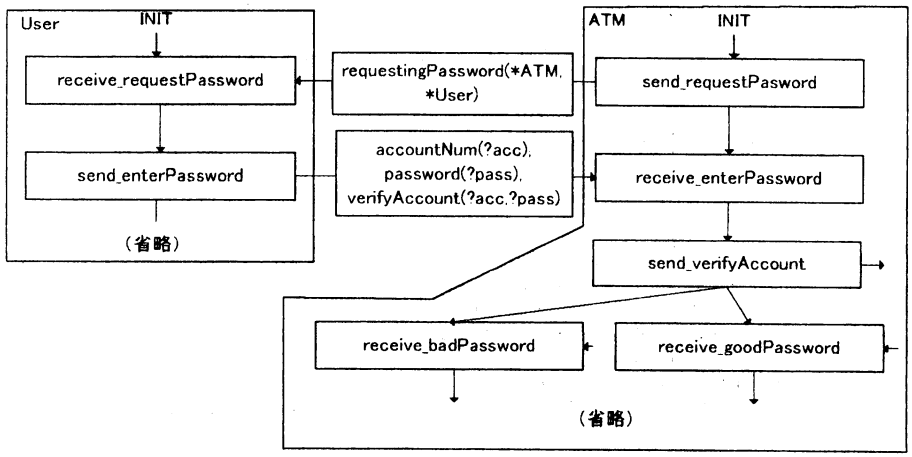

図１２　制御構造モデル（始めの部分）

send_enterPassword アクションの子ノードとして登録される。このように最後のアクションまで木が構築される。この例ではアクションが全順序で実行されるため、一切枝分かれが生じず、単純な木になる。

プロジェクトの検証を正常に完了する場合は、プログラムへの精製、仕様の直接実行を行うために不可欠である制御構造モデルを出力する。制御構造モデルはプロジェクト全体について、グラフ形式による、各コンポーネントの制御の移行を表したものでコンポーネント間のメッセージ受信も表現されている。UMLのアクティビティ図に類似しているが、判定の表現はなく、その代わりに枝分かれを持つ。制御構造モデルの一部を図１２に示す。

5　仕様の直接実行

検証において、知識ベースに登録した制御構造モデルとその制御構造モデルに関係する知識ベースに登録されたアクションを基に、仕様の直接実行を行う。

直接実行システムの構成を図１３に示す。

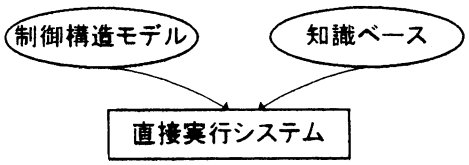

図１３　直接実行システム構成

コンポーネントはそれぞれ独立に動作することからスレッドによって実装する。それぞれのコンポーネントはローカル状態を持つ。また、システム全体の状態を保持するためのグローバル状態もある。それぞれの状態はアクションを１つずつ逐次的に実行するごとに変化する。グローバル状態は共有データ構造として実装する。

メッセージを送信する際は、制御構造モデルにおけるメッセージの情報をバッファする部分に情報を記録する。記録した情報は、情報を取得するコンポーネントから情報をバッファする部分にアクセスし、情報を取得する。ただし、情報がまだ記録されていない場合は、記録されるまで待つ。そのようにすることによって

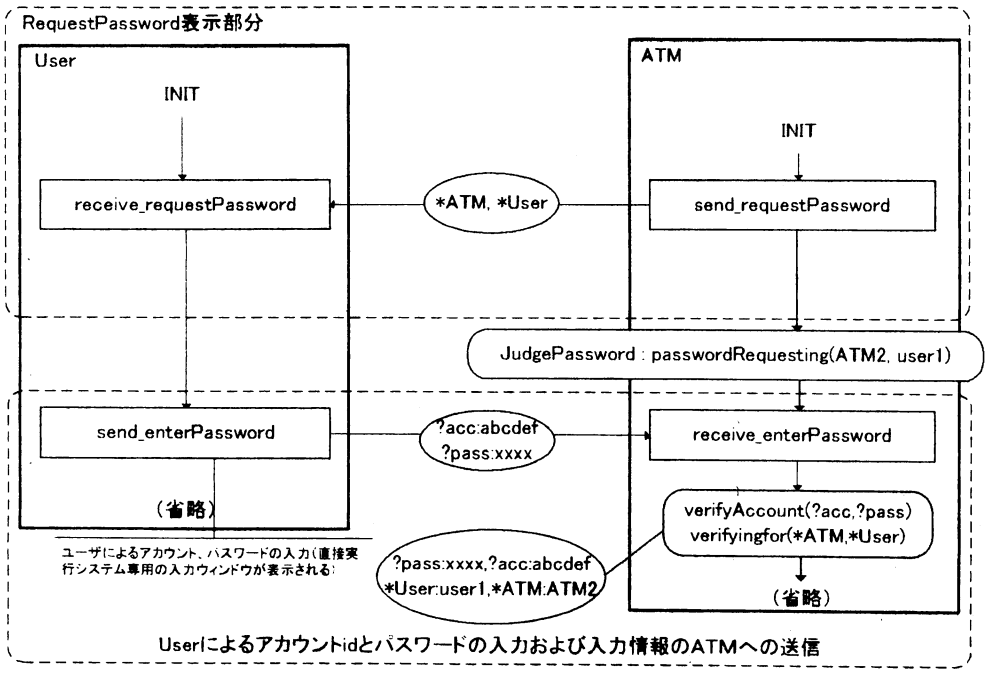

図１４　直接実行例

コンポーネント間の同期をとる。

実際に仕様を直接実行した際のシステムの状態などを図１４に示す。これはBadBankPassword インタラクションフェイズの序盤の部分（図１２を含む）を直接実行したものである（図４）。図中の長方形はアクションを表し、角が丸い長方形はローカル状態を表す。楕円形は状態における変数の値を表す（:の左側は変数名、右側は値を表す）。太線で描かれた長方形はコンポーネントを表す。破線は対応するインタラクションフェイズを表す。

以下に図１４を直接実行した際のシステムの動作について順を追って説明する。図４、図５、図８を参照のこと。

User コンポーネント：

① システムの画面を見ることにより、アカウント(?acc)とパスワード(?pass)の２つのデータの入力を促されていることを知る。(receive_requestPassword アクションの実行)

② アカウント(abcdefg)とパスワードデータ(xxxx)をキーボードによって入力し、ATMに送信する。(send_enterPassword アクションの実行)

ATM コンポーネント：

① User に対してアカウントとパスワードの入力を促す。(send_requestPassword アクションの実行)

② User が入力したアカウントとパスワードのデータを受け取る(receive_enterPassword アクションの実行)

③ アカウントとパスワードのデータを、Consortium を経由して銀行に送り認証を求める。

（以下略）

6 まとめ

本研究では、ソフトウェア開発の上流工程における知識ベースのモデルを使ったソフトウェアの開発方式を提案した。そして、これらのモデルがどのように、検証、直接実行で使われるかをATM システムの例題を用いて示した。

本方式は、実時間システムを対象としたが、時間の表現、非同期交信の対応がまだ無い。これらを追加すること、および、最終的にはプログラムまでの導出を考慮に入れること、デザイン・パターンの知識[10]を設計に利用できるようにすることなどが今後の課題である。

参考文献

[1] S.Uchitel, J.Kramer, and J.Magee, Synthesis of Behavioral Models from Scenarios, IEEE Trans. on Software Engineering, Vol.29, No.2, pp.99-115 (2003).

[2] B.Meyer, Object-Oriented Software Costruction [Second Edition], Prentice-Hall (1997).

[3] J.フィッツジェランド（著），P.G.ラーセン（著），荒木啓二郎（訳），張感明（訳），荻野隆彦（訳），佐原伸（訳），染谷誠（訳），ソフトウェア開発のモデル化技法，岩波書店 (2003).

[4] J.Whittle and J.Schumann. Generating Statechart Designs from Scenarios. Proc.22nd IEEE Int'l Conf. Software Eng. (ICSE '00). pp.314-323 (2000).

[5] A.Kleppe.J.Warmer. and W.Bast. MDA Explained - The Model Driven Architecture. Practice and Promise. Addison-Wesley (2003).

[6] G.Booch. J.Rumbaugh. and I.Jacobson. Unified Modeling Language User Guide. Addison-Wesley (1999).

[7] ヨシュ・ヴァルメル（著），アーネク・クレッペ（著），竹村 司（訳）：UML/MDA のためのオブジェクト制約言語 OCL，星雲社，(2004).

[8] 木下哲男(著)：エージェントシステムの作り方，電子情報通信学会，2001.

[9] S.Russel and P.Norvig, Artificial Intelligence [Second Edition], Prentice-Hall (2003).

[10] F.Buschmann, et al., A System of Patterns: Pattern-Oriented Software Architecture, Wiley (1996).