3N-2

知識化されたモデルを用いるシステム設計方式

金子　誉万†　　間野　暢興†

明星大学大学院　情報学研究科†

1.　はじめに

近年ソフトウェアの開発方式としてモデル駆動開発方式(MDA)が注目を浴びている。本研究もその方式に属するものである。MDA と我々のアプローチとの関連性は MDA は論理的手法をベースとした検証を目標にしている点が似ていることである[1]。これに対して、シナリオに基づく仕様記述法を用いるアプローチは、仕様の記述が極めて簡単な記述でできる[2]。しかし、本研究では仕様記述の読解性、設計の精製や合成などには不可欠なことから状態仕様記述を含む仕様記述法を採用する。

本研究では、対象モデルを計算機内モデルとして生成し、知識ベースに蓄え、それらを基に、ソフトウェア設計に必要不可欠と考えられる利用形態への半自動化された適用方式を提案する。この利用形態としては、(1)順方向検証、(2)仕様の直接実行、(3)プラン合成、の３つを当面の目標としている。

2.　本方式の対象とする領域およびシステム構成

2-1.　対象とする領域

本システムは実時間システムを対象として扱う。実時間システムの特徴は次の点である。

- 複数のコンポーネントが存在し、コンポーネント間でメッセージを通して交信する。
- それぞれのアクションが実行されるタイミングが重要である。
- 時間が経過するごとにシステムの状態も変わる。

これらの点から実時間システムを設計する上でアクションが実行される条件記述が重要になってくる。

2-2.　システム構成

システムを利用するには最初に知識を知識ベースに登録しなければならない。知識を登録する際のシステム構成は図 1 の①の領域に示すとおりである。

知識を登録する際は定められた仕様入力言語に従って仕様を記述する。その仕様は知識登録システムによって解析され知識ベースへと登録される。入力する知識としては、プロジェクト、インタラクションフェイズ、アクションがあるが、いっぺんに登録する必要はなく、仕様には常に変更が生じるように随時知識を登録することが可能である。インタラクションフェイズは文献[2]の bMSC(basic Message Sequence Charts)に対応し、コンポーネント間の逐次的な相互作用を表す単位である。プロジェクトは hMSC(high-level Message Sequence Charts)に対応し、システム全体の動作をインタラクションフェイズ間の処理の遷移で表すものである。なお、本研究ではプロジェクトのインタラクションフェイズへの分割は、ユーザが行うものとしている。

An Approach to System Design Using Knowledge-Based Modeling
Takakazu Kaneko, Nobuoki Mano, Graduate School of Informatics, Meisei University†

知識登録後は登録した知識を用いて検証、プラン合成、仕様の直接実行を行う。それぞれのシステムの構成を図 1 の②、③、④の破線で示す。

順方向検証システムは、インタラクションフェイズ間遷移指定で与えられたプロジェクトにおける各インタラクションフェイズ間の遷移と、並列プロセス記述で与えられたインタラクションフェイズにおける各アクションの動作を FSP(Finite Sequential Processes) [2]風の記述に倣い、仕様のデッドロックの有無を検証する。

プラン合成システムはインタラクションフェイズを問題と見なし、半順序プラナーによって知識ベースにある知識を用いて、その仕様を満たすアクション列を合成するものである。

仕様の直接実行システムは、TAF エージェントシステム[3]を利用している。順方向検証システムによって生成された制御構造モデルあるいはプラン合成システムによって生成された制御構造モデルを TAF 表現変換機により、TAF のルールへと変換し、TAF システムにより直接実行する。

3.　知識とそれを用いたシステム

3-1.　仕様記述言語

知識を入力する際は述語論理式で記述した入力言語を用いて、知識ベースに登録する。登録する知識として、アクション、インタラクションフェイズ、プロジェクトの３つがある。アクションとインタラクションフェイズは事前条件と事後条件、プロジェクトは所属インタラクションフェイズの記述をもつ。また、受信アクションは事前条件に受信メッセージ、送信アクションは事後条件に送信メッセージに関する記述を含む(図 2)。

3-2.　知識ベース

本システムは、クラスの階層構造が本システムを実装する上でもっとも適していることから、JAVA 言語を用いて実装している。個々の知識及び問題の解は、クラスのインスタンスを用いて表現される。Project、InteractionPhase 、Component 、Action から、Predicate、Variable までの様々な粒度のクラスを用いている。知識ベースには仕様情報を所持するだけでなく、他のシステムから情報を扱いやすくするために索引の情報も持っている。索引は述語をキーとしてその述語を事前条件あるいは事後条件に含むアクションを検索する。

3-3.　知識を利用した順方向検証

検証は順方向と逆方向の二種類ある。双方の検証を行うことによってより確実な検証を行うことができる。

順方向検証システムは次の二つの事柄について利用可能である。

- インタラクションフェイズにおける各コンポーネント内のアクションの実行順序を記述した並列プロセス記述
- インタラクションフェイズの実行順序を記述したイ

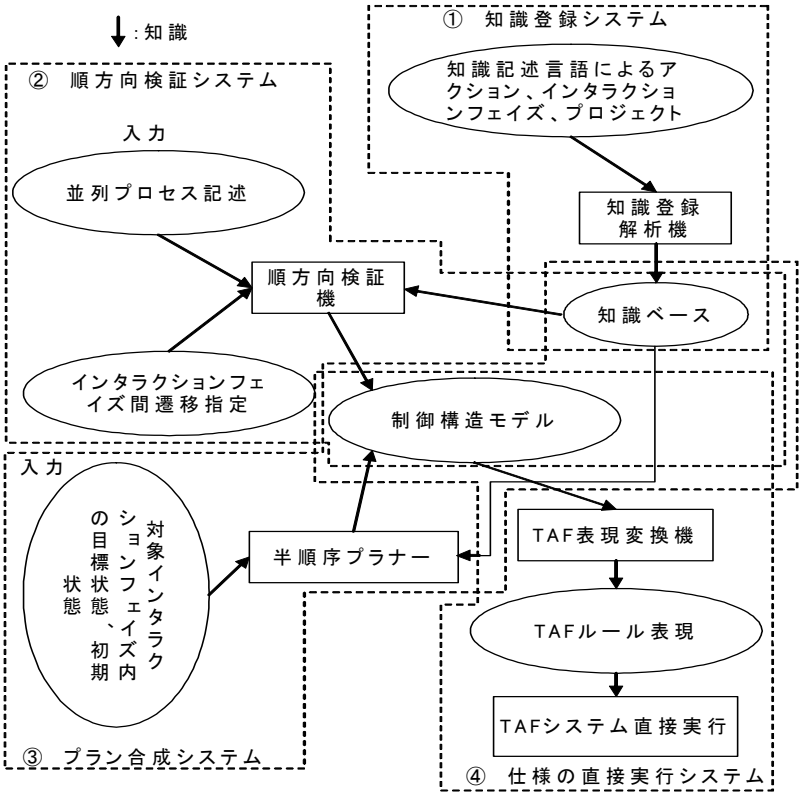

図 1 システムの構成

```
actionDef ATM::receive_insertCard() {
  pre      : card(?card), user(*User), ATM(*ATM),
   EMPTY(*ATM),
   message(*User, *ATM, inserted(*User, ?card));
  post     : !EMPTY(*ATM), inserted(*ATM, ?card));
  post-1   : (bank(?card, MS) => goodCard(?card));
  post-2   : (bank(?card, MT) => badCard(?card));
}
```

図 2　知識登録例

```
projectVerify ATM_System {
  STATE : INIT, InitialState, JudgeCard;
  INTERACTIONPHASE : CustomerArrives,
          RequestPassword, EjectCard, (略);
  INIT = InitialState;
  InitialState = (CustomerArrives -> JudgeCard);
  JudgeCard = (RequestPassword -> (略) |
                  EjectCard -> InitialState);
  ASSERTION :   INIT@(EMPTY(*ATM), (略)),
                  InitialState@(EMPTY(*ATM)),
                  JudgeCard@(insertedCard(?card));
}
```

図 3 インタラクションフェイズ指定例

```
interactionPhaseVerify VerifyExample {
  componentA = (send_msgA());
  componentB = (receive_msgA() -> receive_msgC());
  componentC = (send_msgB() -> send_msgC());
  componentD = (receive_msgB());
}
```

図 4 並列プロセス記述例

ンタラクションフェイズ間遷移指定

インタラクションフェイズ間遷移指定はプロジェクト名を指定し、そのプロジェクトに登場してくるインタラクションフェイズとシステム状態を宣言する。その後にインタラクションフェイズの実行順序を指定する(図 3)。この記述を解析し、検証木を構築する。

検証法は木の状態に関する矛盾をチェックする。各インタラクションフェイズ実行後ごとに状態が変化していくので、その状態と各インタラクションフェイズの事前条件が整合をとる。また、各状態表明と状態とが無矛盾であることも確認する。いずれも矛盾があった場合は木が矛盾していることになる。

並列プロセス記述はインタラクションフェイズ名を指定し、そのインタラクションフェイズにおける各コンポーネントの実行するアクション列を指定する(図 4)。この記述を解析し、検証木を構築する。

検証法は完成した木のループの有無、アクション間の事前条件と状態との不一致などで仕様の矛盾を検証する。

3-4.　知識を利用したプラン合成

半順序プランニング[4]は開始と終了ステップを表現する初期プランから始まり、一つ以上のステップを繰り返し追加していく。プラン合成は実時間システムにおいて、メッセージの送受信の順序の確認に大きな役割を果たす。これは半順序プランニングにおける脅威ステップの検出を利用することによって実現している。脅威ステップとは保護された事前条件を打ち消すステップのことを言う。プランが完成することによって、正しい順序でメッセージが送信可能であることが保証される。

3-5.　仕様の直接実行

仕様の直接実行は、設計の段階での仕様の不備の検出に大きく貢献する。

直接実行を実現するには、生成した制御構造モデルを TAF の表現に変換する必要がある。本システムの事前条件は TAF における条件部に、事後条件は実行部に、コンポーネントはエージェントに相当する。条件記述に関しては述語表現から OAV 型データ記述への変換を行う。その際、オブジェクト(O)の属性(A)と値(V)のペアをリストにまとめる。

4.　まとめ

本研究では知識ベースを利用した方式を提案し、実際に知識ベースを用いて仕様の検証、プラン合成、仕様の直接実行が行えることを示した。本方式を実現するシステムの開発は現在進行中である。今後の課題として、時間に関する記述の導入などの仕様記述言語と処理システムの拡張、および、デザインパターンの本方式への取り入れなどがある。

参考文献

[1] Jos Warmer, Anneke Kleppe : The Object Constraint Language [Second Edition] : Getting Your Models Ready For MDA, Addison-Wesley, 2003.

[2] Sebastian Uchitel, Jeff Kramer, Jeff Magee : Synthesis of Behavioral Models from Scenarios, IEEE Transaction on software engineering, vol.29, February, 2003, pp.99-115.

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

[4] Stuart Russell & Peter Norvig(著), 古川康一(訳), エージェントアプローチ人工知能, 共立出版, 1997.