B-019

UML モデルの振舞いのモデル検査における表現方法について
Specifying Behavioral Aspect of UML Model for SPIN Model Checker

八鍬 豊† 野田 夏子†
Yutaka Yakuwa† Natsuko Noda†

1. はじめに

近年，ソフトウェアの複雑化・巨大化が進み，従来のテスト手法だけでは充分な検証が困難になってきている．この問題に対する一つの解決手段として，モデル検査による検証手法が知られている．特に並行ソフトウェアにおいては，各プロセスの動作や通信のタイミングに起因するバグを漏れなく検出する技術として注目されている．

一方で，多くのモデル検査ツールでは検証対象を専用の言語で記述する必要があり，一般的な開発者には敷居が高い．そのため，専用の言語での記述を支援するための研究がこれまで多くなされている．特に，設計モデルから専用の言語での記述へ変換する手法は，開発者が言語を習得するコストが軽減できるため，期待が大きい．

そこで我々は，開発現場で広く使われる UML による記述を，モデル検査ツール Spin[1]の入力言語 Promela での記述へ変換して検証する手法を検討している．通常の開発におけるモデル検査の利用の容易化という目的のためには，検証のために通常の設計モデルとは別のモデルを作成したり，Promela に依存した記述を設計モデルに書き加えたりすることは望ましくない．本手法では，極力 Promela 依存の記述を排した，通常の設計での成果物である UML モデルを Promela に変換する方法を検討している．

本稿では，UML で記述した並行ソフトウェアの振舞いを Promela で表現する方法を提案する．また，UML モデルの動作仕様[4]を適切に表現するために必要となった工夫点について述べる．

2. UML 図による並行ソフトウェアのモデル化

本稿で提案する UML モデルの Promela 表現は，並行ソフトウェアの上流設計において，処理の詳細な内容には依存せず，通信や処理の順序やタイミングに起因する問題の検証に利用することを想定している．そのため設計モデルは，それらの側面を表すのに必要最低限のモデル要素のみを用いて表す．静的側面をクラス図で，動的側面をステートマシン図で表現することとする．表 1 に，それらの図で使用するモデル要素一覧を示す．

2.1 クラス図の記述内容

クラス図では，対象ソフトウェアの静的な構造を表現する．並行ソフトウェアの振舞いの設計・検証においては，オブジェクト間の協調関係に注目することが重要となるため，構成要素であるオブジェクトとその接続関係をクラス図に表現することとした．つまり，現状ではクラス図をオブジェクト図のように使用し，1 オブジェクトを 1 クラスで表現する．さらに，オブジェクトの属性と通信接続関係を示す．

処理を行なうプロセスオブジェクトは，ステレオタイプ<<process>>をつけたクラスとして，データを保持するデータオブジェクトはステレオタイプを持たないクラスとして表現する．プロセスオブジェクト同士が通信を行なう場合，各オブジェクトに対応するクラス間に関連を引いて示す．プロセスオブジェクトがデータオブジェクトを参照する場合も，関連を引いて示す．

2.2 ステートマシン図の記述内容

ステートマシン図では，各プロセスオブジェクトの振舞いをそれぞれ表す．

本手法では，ステートマシン図において，トリガによって遷移が実行され状態が変化することを記述して振舞いを表現している．互いに通信を行うプロセスオブジェクトの協調動作に注目しているため，トリガとしてはメッセージ受信を想定し，遷移を引き起こす受信メッセージの名前をトリガに記述する．なお，遷移は，トリガにより状態が変化する通常のものに加えて，状態内に定義され状態の変化を起こさない内部遷移と，トリガが定義されていない場合に入場動作完了直後に起こる completion 遷移の 3 種類を記述可能とする．トリガには，受信するとその遷移を起こすメッセージの名前を一つ記述する．

振舞いの定義には，状態の遷移に加えて，必要ならば状態における各種動作(入場動作・退場動作，遷移の際に実行されるアクション)を記述する．これらの動作の基本的な要素として，変数の参照・更新，単一メッセージの送信を記述可能とし，これらの要素を組み合わせて動作を記述する．メッセージ送信は，送り先オブジェクトと送信するメッセージを指定する．なお，同期呼び出し等の様々な通信メカニズムは，より複雑な制御が必要であるので，設計者が基本要素を用いて明示的にモデル上に設計するものとする．

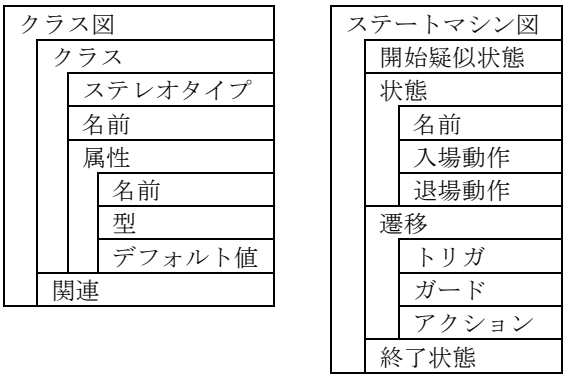

表 1 UML 図で使用するモデル要素一覧

3. Promela による UML モデルの表現

Promela ではモデル全体の挙動を，proctype で定義されるプロセス型のインスタンス(以下プロセス)の並行動作と

† NEC サービスプラットフォーム研究所
Service Platforms Research Laboratories, NEC Corporation

して表す．そのため，プロセスオブジェクトはプロセスとして表現することとする．ステートマシン図の各状態は，proctype 内のラベルからラベルまでのステップ列で表現し，それらを goto 文で行き来することで，状態間の遷移を表現する．メッセージ受信のために，プロセス毎に一つの受信用チャネルを用意すし，遷移のトリガであるメッセージ受信はそのチャネルからのメッセージ受信，オブジェクトへのメッセージ送信はそのチャネルへのメッセージ送信文で表現する．

プロセスは，各状態において受信用チャネルからメッセージを受信し，受信したのがトリガに指定されるメッセージであった場合に遷移する．このメッセージの評価においては，メッセージを変数に格納し，それを if 文や do 文の分岐に利用するという方法をとる．これは UML モデルの動作仕様を適切に表現するための工夫であり，詳細は 4 節で述べる．また，本稿で提案する表現方法の概観を図 1 に示す．

```
proctype プロセス名(...) {
...
状態名ラベル:
  入場動作;
  if  //  completion 遷移
  ::  ガード -> 退場動作; アクション;
      goto 遷移先状態名ラベル;
  ...
  ::  else -> skip;
  fi;
  do
  ::  チャネル ? トリガ用一時変数 ->
      if  //  通常遷移
      ::  (トリガ用一時変数 == トリガ) && ガード ->
          退場動作; アクション;
          goto 遷移先状態名ラベル;
          //  内部遷移
      ::  (トリガ用一時変数 == トリガ) && ガード ->
          アクション;
      ...
      ::  else -> skip;
      fi;
  od;
...
}
```

図 1　UML モデルの Promela 表現の概観

4. Promela 表現上の工夫

Promela における送受信文は副作用のある文であり，副作用のない文と一緒に使えないという問題があるため，受信するメッセージを変数に格納し，それを評価に利用するという工夫をしている．上記の問題がどのようなケースで発生するかを以下で述べる．

まず，ガードとトリガの成否を同時に評価するケースである．ガードとトリガを同時に評価しない場合，後に評価するものが成立するまで待ち状態に入ってしまう可能性があるので，これらは同時に評価すべきである．ここで，ガードは変数を用いた制約式で表現するため，副作用のない文になる．そのため，トリガの評価を受信文で直接表現してしまうと，論理積をとって同時に評価することができなくなる．受信するメッセージを変数に格納し，トリガの評価も変数を用いた制約式で表現することで，この問題を解決できる．

次に，トリガに記述されていないメッセージを受信した際の処理を表現するケースである．else を用いるのが自然であるが，else も副作用のない文であるため，トリガの評価の表現は副作用のない文に限られる．これも上記のケースと同様の方法で解決できる．

以上のように，トリガの評価は副作用のない文で表現するのが適切であると判断し，メッセージを変数に格納して評価に利用するという工夫を施している．

5. 関連研究

UML 図を Promela に変換して設計モデルのモデル検査を行なうというアプローチは，従来よりいくつか提案されている．その中で，例えば MOCES[2]のように，単一のステート(マシン)図で設計モデルを記述するものがあるが，この方法は複数のプロセスオブジェクトの通信や処理の様子を直接的に表すのが難しい．本研究では複数のステートマシン図を扱うため，それを容易に表せる．

上記のアプローチで並行ソフトウェアの検証を行なうものでは，HUGO[3]が我々の研究に近い．HUGO はステートマシン図の Promela 表現において，各状態を異なるプロセスで表現し，遷移等を制御するプロセスも生成するため，設計上のプロセスオブジェクトと Promela 上のプロセスが一対一に対応しない．そのため，反例等の実行パスの解析が複雑となり，設計の問題が見つけにくくなる可能性がある．我々は，設計上のプロセスオブジェクトと Promela 上のプロセスとの対応関係を分かりやすく保存することがこうした点から重要と考え，一対一に対応させる表現方法を取った．

また，近年日本の産業界においても，モデル検査技術を開発に利用する様々な試みがなされている([5],[6]等)．このうち，Early Bird[6]は，UML 図を Promela に変換する等，本研究と共通点が多い．しかし，Early Bird における検証対象は，ソースコードに近い詳細な動作を記述した UML であり，本研究とは対象とする設計モデルの詳細度が異なる．そのため，それぞれ異なる開発フェーズで活用できる．

6. まとめ

本稿では，UML で記述した並行ソフトウェアの振舞いを Promela で表現する方法を提案した．今後は，本手法により生成される Promela 記述の評価や，より適切な Promela 表現の検討を重ね，通常の開発でモデル検査を容易に利用できる手法の実現を目指したい．

参考文献

[1]Holzmann, G. J., “The Spin Model Checker: Primer and Reference Manual”, Addison-Wesley Professional(2004).
[2]Mikk, E. et al.,“Implementing statecharts in Promela/Spin”, Proc WIFT’98(1998).
[3]Schäfer, T. et al., “Model checking UML state machines and collaborations”, Electronic Notes in Theoretical Computer Science, Vol. 55, No. 3(2001).
[4]“OMG Unified Modeling Language (OMG UML), Superstructure”, Ver2.3(2010).
[5]篠崎孝一ほか, “支援ソフトウェアを活用した実践的モデル検査”, JaSST’08 Tokyo(2008).
[6]野村秀樹ほか, “ModelChecking 技術の専門性の排除とその効用－An Early Bird as the 2nd eye－”, JaSST’08 Tokyo(2008).