Vol.2010-SE-170 No.16
2010/11/12

設計検証のための UML プロファイル

野田夏子†

巨大化・複雑化するソフトウェアの信頼性向上のために，形式手法を活用することが注目されている．形式手法をソフトウェア開発の現場で実適用するためには，形式手法をソフトウェア開発の中でどのように位置づけていくかというソフトウェア工学面からの検討が不可欠である．我々は，設計検証に形式手法，特にモデル検査を活用することを検討しているが，モデル検査による設計検証をより幅広く行うためには設計モデルを構築するための手法や環境の整備が必要になると考えている．本稿では，こうした手法や環境の整備のために，UML プロファイルの機構を利用するアプローチを提案し，具体的な検証用プロファイルを提示する．

UML Profile for Design Verification

Natsuko Noda†

To increase the reliability of software, which is getting more gigantic and more complex, utilizing formal methods is widely attracting attentions. For application of formal methods to real software development, it is essential to consider from the viewpoint of software engineering how they are utilized and what role they play in the software developments. We are now studying application of model checking, which is one of the formal methods, to design verification, and recognize that building of design model development methods and environments is necessary. In this paper, we introduce an approach utilizing UML profile mechanism for the above mentioned purpose, and show some concrete profiles for design verification.

1. はじめに

あらゆるシステムにソフトウェアが使われるようになり，ソフトウェアの信頼性は大きな関心事となっている．その一方でソフトウェアの規模や複雑さはますます増大しており，信頼性の確保は困難さを増している．そうした中，従来のレビューやテストといった検証技術に加え，より科学的な形式手法，特にモデル検査技術の適用が注目されている．産業界でも実プロジェクトに適用され，その有効性に関する報告が複数なされている．しかし，その適用は単発的なものが多く，今後モデル検査技術をより広く組織内のソフトウェア開発に適用できるようにするためには，技術面からの可能性実証だけでなく，それをソフトウェア開発の中でどのように位置づけていくかというソフトウェア工学面からの検討が不可欠となる．

モデル検査技術によるソフトウェア検証は大きくコード検証と設計検証とに分けられる．我々は，上流工程での品質を向上させ手戻りを減少させることが重要と考え，上記のうち特に設計検証について検討を行っている．しかし，コード検証がコードを検証対象とし，検証する性質の多くがコードレベルの基本的な性質（例：NULL ポインタ参照等）であるために，ソフトウェア技術者にとって比較的容易に適用できるのに対して，設計検証は，設計や方式上の問題など設計上の意図に関わる検証を行うものであり，検証対象となる設計モデルを適切に構築する必要があり，相対的に適用のハードルが高くなる．今後モデル検査技術による設計検証をより幅広く行うためには，設計モデルを構築するための手法や環境の整備が必要になる．

本稿では，モデル検査技術による設計検証を，より広く体系だって適用できるようにするための検証用プロファイルの提案を行う．2 章で設計検証の課題を論じ，3 章では設計検証に検証用プロファイルを用いるアプローチの概略や，それにより設計検証の課題がどのように解決されるかについて説明する．4 章では，検証用プロファイルを具体的に提案し，5 章で本稿をまとめる．

2. 設計検証の課題

2.1 モデル検査を利用した設計検証の概略

ソフトウェアの不具合は，開発の上流工程において作りこまれるものも多く，それが手戻りを生じさせたり，製品の品質問題を生じさせたりする．いかに設計の品質を向上させるかは，ソフトウェア開発における重要な課題である．設計の検証は，従来は主にレビューによって行われてきた．しかしレビュー等の人手による検証には限界もあり，形式手法を活用した検証も併用することが有効と考えられる．

我々は，こうした設計検証のひとつとして，UML により記述された設計モデルをモ

† 日本電気株式会社 サービスプラットフォーム研究所
Service Platforms Research Laboratories, NEC Corp.

デル検査を用いて検証することを検討している 6) 7). 今日，ソフトウェア開発においては，UML が標準的に用いられる場合が多いため，UML により記述された設計を検証の対象としている．図 1 はこの設計検証の概略を示したものである．

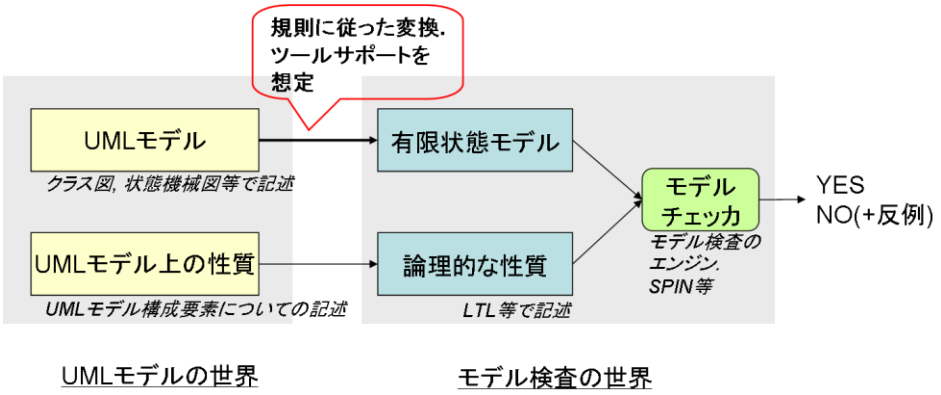

図 1　モデル検査による UML 設計検証

モデル検査による設計検証においては，UML モデルの世界をモデル検査の世界にマッピングする必要がある．すなわち，設計を示す UML モデル(クラス図や状態機械図等で記述される)をモデル検査の対象である有限状態モデルへ，UML モデル上の性質に関する記述をモデル検査において有限状態モデルがその性質を満たすかどうかを検査される論理的な性質(LTL 等で記述される)へ変換する．UML モデルの世界がモデル検査の世界にマッピングされれば，検証自体はモデル検査の世界においてモデルチェッカ(モデル検査のエンジン，SPIN 5) 等)を使って行われ，検証結果が返される．

2.2 設計検証における設計モデルの課題

2.1 で述べた設計検証を行うに際して，設計モデルの構築には以下のような課題があると考える．

- 設計モデルに実行意味論を与える必要がある：モデル検査委技術はシステムの動作に伴う状態変化に注目して検証を行うものであり，設計モデルの構築においてはその実行意味を明確にする必要がある．しかし，UML は Action Semantics で規定される基本的な操作の意味やひとつの状態機械図の実行意味などしか既定しておらず，モデル検査技術に必要な実行意味論を十分に提供していない．例えば複数の並行動作単位がどのような順序関係で動作するのか，どのような通信のモデルなのか，などは UML としては規定していない．したがって UML モデルに対してモデル検査技術による検証を行うためには，実行意味論を与える必要があるが，それに対する一定の方針やガイドラインが存在していないため，意味論の与え方が個人個人でアドホックになりがちで望ましくない．
- モデル検査の方法には多様性がある：モデル検査のエンジンには例えば SPIN　5) や UPPAAL　4) など様々なものがあり，それぞれが検証できる性質や効率などが異なる．したがって，ひとつの設計モデルに対してでも検証目的に応じて異なったエンジンを使う可能性がある．またひとつの検証エンジンだけを利用する場合でも，その利用法は多様である．例えばモデル検査を行う際に特定のスケジューリングや通信方式に関わる機能を疑似することがあるが，そうした疑似のテクニックは多様である．こうした多様性にアドホックに対応することは，設計モデルを構築するソフトウェア技術者にとって望ましくないだけでなく，設計モデルを検証エンジンの理解できる形式に変換するツールを開発するなど，検証の環境を整える立場の人にとっても不利である．したがって，何らかの一貫した考え方の下で体系だって多様性に対応できることが望ましい．
- モデル検査技術だけのために設計モデルを構築するわけではない：モデル検査技術による検証を行おうが行うまいが，設計は行う必要がある．また多くの場合，対象ドメインや組織ごとに，典型的な設計モデル構築の方法がある場合が多い．例えば組込みソフトウェアであれば，MARTE 3) などのプロファイルを使ってリアルタイム OS のメカニズムを使ったアプリケーションをモデル化し，それに基づいて開発を行ったり，性能解析やスケジュール可能性解析を行ったりする環境が整備されつつある．したがってモデル検査技術を適用するためだけに独自の設計モデルを別途構築することは現実的ではない．こうした対象ドメインや組織としての設計モデル構築の方法がある場合には，それと整合のとれる形で（望むらくはその構築方法どおりで）検証のための設計モデルを構築できることが望ましい．

3. プロファイルを利用した設計検証

3.1 アプローチ

我々は，モデル検査技術による設計検証を実際のソフトウェア開発においてより幅広く行うために，設計モデルを構築するための手法や環境の整備を検討している．この環境整備の一環として，2.2 で指摘した設計モデルの問題点の改善を行う必要がある．

具体的には以下のアプローチをとることを考える．

- 検証用プロファイルを構築する：実行意味論を与えるために，検証対象となる設

計モデルを記述する際に利用する検証用プロファイルを作成する．プロファイルは UML の提供するメカニズムのひとつで，特定の目的のために必要となるステレオタイプやタグ付き値などの定義をパッケージ化したものである 2)．その特定目的のためのモデリングを行う人は，そのプロファイルを利用してモデリングすることで，目的に合致したモデリングを行える．このプロファイルの機構を利用して，検証の目的のためのプロファイルを構築して利用する．検証用プロファイルをドメインや組織で共有することで，意味付けの方法を共通化することができ，ひいては設計モデルの構築方法をそろえることができ，設計モデルそのものの共有化，資産化に資することが期待される．

- 拡張のコアとなるプロファイルを定義する：いろいろな検証エンジンが存在するが，ソフトウェア検証への適用を意識した検証エンジンが想定する実行意味には類似性がみられる．具体的には，複数の並行動作単位が広い意味での通信を行いながら全体として協調して動作するという実行意味に基づくものが多い．検証エンジンの多様性に対応するために，そうした基本的な意味をあらわすコアとなるプロファイルを作成する．個々の検証エンジンが持つ特定の実行意味に対応した検証用プロファイルは，このコアとなるプロファイルをさらに拡張して定義する．同様に，個々のエンジンの利用方法の多様性に対してもコアとなるプロファイルをベースとして拡張を行い対応する．こうすることで，様々な実行意味への拡張を体系だって行うことが可能となり，検証用プロファイルの利用者にとって分かりやすくなるだけでなく，検証用プロファイルを構築する人にとっても一貫した考え方でプロファイル定義ができるようになる．
- 既存プロファイルとのマッピングを定義する：ドメインや組織で使われる既存のプロファイルがある場合，そのプロファイルと検証用プロファイルとの間のマッピングを定義することを考慮しつつ検証用プロファイルを構築する．そうすることによって，既存のプロファイルを用いて作られた設計モデルから，マッピングに基づき検証用プロファイルを用いた設計モデルへと変換し，それに対してモデル検査技術による検証を行うことができる．こうした検証用プロファイルの構築はそれなりに注意深い作業が必要とはなるが，こうすることによってモデル検査技術の適用のハードルを低くすることが期待できる．

以降の節では，このアプローチを踏まえて，具体的にどのような方針で検証用プロファイルを策定し，それにより 2.2 で述べた課題がどのように解決されるかを述べる．

3.2 実行意味の明確化

検証用プロファイルは，設計モデルに対して検証に必要となる動的な意味づけを与えるものである．

ソフトウェア検証への適用を想定したモデル検査エンジンの多くは，並行動作単位が通信をしながら動作する実行意味に基づいた対象のモデルのふるまいを網羅検査し，時相論理で表現される性質が成立するかどうかを検証する．したがって，設計検証を行う際には，構築した設計モデルをそうした動的意味に基づいて解釈する必要がある．しかし，前述したように UML それ自体は明確な実行意味を既定していないため，UML で記述しただけでは，モデル検査エンジンの想定する実行意味にマッピングすることができない．そこでモデルにそうした実行意味を与えるために検証用プロファイルを活用する．検証用プロファイルは，モデル検査エンジンが想定する実行意味に対応させたモデル要素を表わすためのステレオタイプ等をパッケージ化する．図 2 に，検証用プロファイルを用いることにより，設計モデルにどのように実行意味が与えられるかを示す．

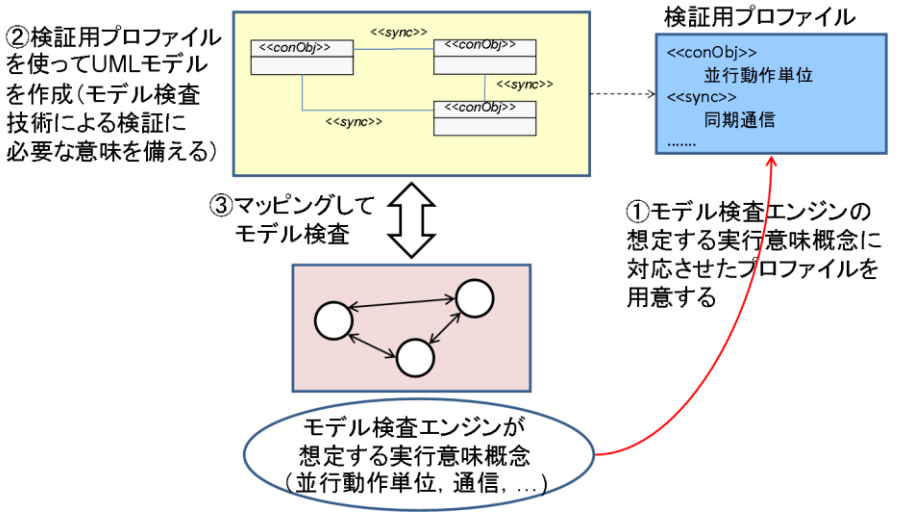

図 2　検証用プロファイルと実行意味

3.3 多様性への対応

モデル検査エンジンやその活用方法の多様性に対応するために，コアとなるプロファイルを定義し，それを元に多様性に対応した検証用プロファイルを拡張定義することを考える．

図 3 に多様性への対応のイメージを示す．前述したようにソフトウェア検証への応用を想定しているモデル検査エンジンは並行動作単位がなんらかの通信をするという

実行意味に基づくものが多い．そうした複数のモデル検査エンジンが想定する実行意味概念に対応したプロファイルをコアとなるプロファイルとして定義する．これを V-Core と名付け，以後 V-Core と呼ぶ（V-Core の詳細は 4.1 で述べる）．特定の検証エンジン特有の実行意味や，検証エンジンの利用方法に依存した実行意味に対応した検証用プロファイルは，V-Core を張することで定義する．図 3 はこの拡張のイメージを示す．ここでは V-Core を拡張して SPIN を使う際の検証用プロファイル(SPIN が初めから備えている同期通信，非同期通信などの実行意味概念に対応したプロファイル)や，特定ドメイン向けの検証用プロファイル(優先度付き並行動作単位などの実行意味概念に対応したプロファイル)などを拡張定義している．このように，V-Core をベースに，プロファイルの拡張という形で実行意味を拡張することで，各検証用プロファイルが一貫した考え方で定義されるため，利用者にとっても検証用プロファイルの定義者にとっても，分かりやすく検証用プロファイルを利用，定義できるようになることが期待される．

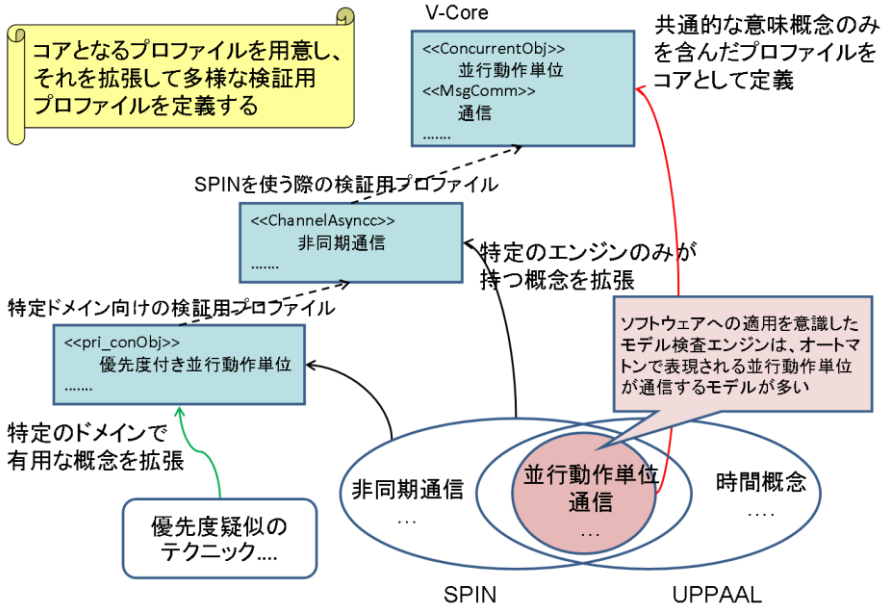

図 3　多様性への対応

4. 設計検証のための UML プロファイル

本章では，具体的なプロファイルを提案する．4.1 では，3.3 でそのコンセプトを示したコアとなるプロファイル V-Core の詳細を示し，4.2 ではその拡張例として，クラス図と状態遷移図に基づいた UML でのモデルの検証を SPIN で行うために必要な定義を与えるプロファイル UMLVerification を示す．なお，本稿ではプロファイルの各要素の定義については割愛し，プロファイルの全体像を理解するためのドメインモデルと，プロファイルに定義されたステレオタイプ等の全体を示すプロファイル図の提示を行う．

4.1 コアとなるプロファイル V-Core

V-Core は，複数のモデル検査エンジンが想定する実行意味概念に対応するように策定したプロファイルである．モデル検査エンジンには様々なものがあるので，V-Core はこれら広範囲のモデル検査エンジンの実行意味に対応することが望ましい．しかし，全てのモデル検査エンジンに完全に対応することは現実には困難であるし，また無意味に対応範囲を広げても実際の有効性は薄くなる．そこで，広くモデル検査エンジンを調査しつつ，近年実際の開発に用いられることが多いモデル検査エンジンとして，SPIN 5)，NuSMV 1)，UPPAAL 4) について特に調査を行い，これらのモデル検査エンジンとの対応に留意して V-Core を策定した．

V-Core を策定するにあたり，その構成については MARTE 3) を参考にした．MARTE は組込み設計で用いられるプロファイルであり検証と直接的な関係はない．しかし，モデル検査においては並行動作単位がお互いにどのようにインタリーブしながら振る舞うかをモデル化することが重要であり，これは MARTE におけるモデルと類似のものがある．したがって，このような点のモデル化及び全体の構成について MARTE を参考とすることとした．ただしこれは V-Core 策定時においてであり，策定された V-Core と MARTE の間に直接の関係はない．

(1) 全体構造

V-Core は，検証モデルの静的な構造に関わる要素を定義するプロファイル Structure と，動的な構造に関わる要素を定義するプロファイル Behavior から構成される(図 4 参照)．

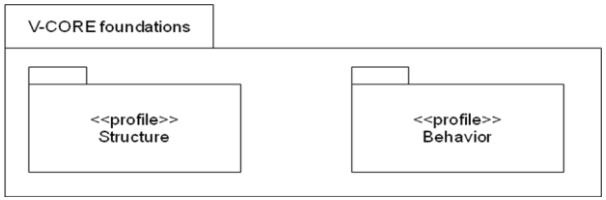

図 4　V-Core 全体構造

また，これらのプロファイルの構成要素の概念を定義するための CoreElements パッケージを定義する．ここに定義される要素は UML から拡張して新規に定義する要素ではないが，V-Core で全体をどのようにモデル化するかという観点から特に取り出して要素間の関係を示すものであり，UML と矛盾しないように再定義される．図 5，図 6 に CoreElements パッケージを示す．

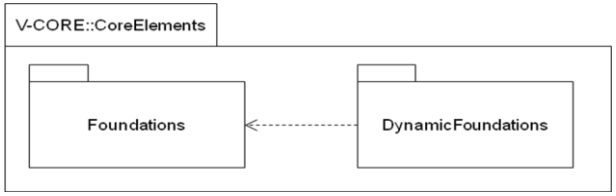

図 5　CoreElements パッケージ

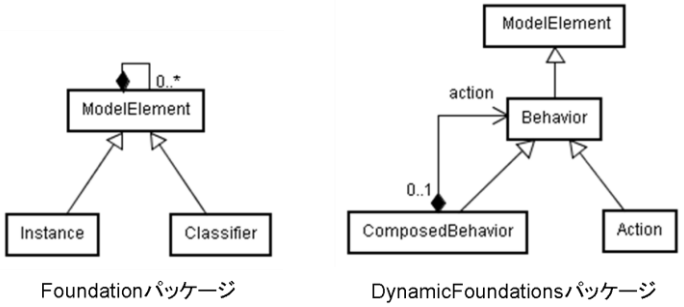

図 6　CoreElements パッケージ詳細

(2) ドメインモデルとプロファイル図

静的な構造に関わる要素を定義する Structure プロファイルのドメインモデルを図 7 に示す．Structure プロファイルには，並行動作単位（ConcurrentObj）と，それらが互いにやり取りする際のリソース（CommRsrc）という，並行動作モデルの基本的な要素が定義される．リソースはさらに共有データ（SharedData）とメッセージ通信（MsgComm）に分類され，メッセージ通信には同期（SyncMsgComm）と非同期（AsyncMsgComm）がある．これらの要素が，UML のメタクラスをどのように拡張しているかを示すプロファイル図が図 8 である．並行動作単位を示す ConcurrentObj は Class へのステレオタイプとなり，やり取りする際のリソース CommRsrc は Classifier へのステレオタイプとなる．

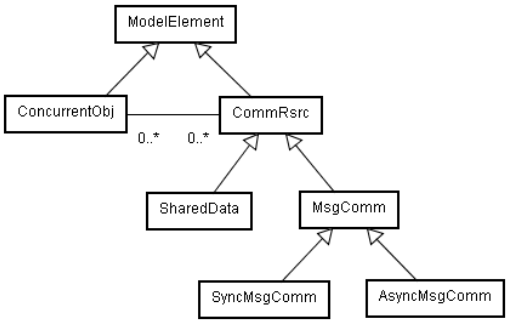

図 7　Structure プロファイルのドメインモデル

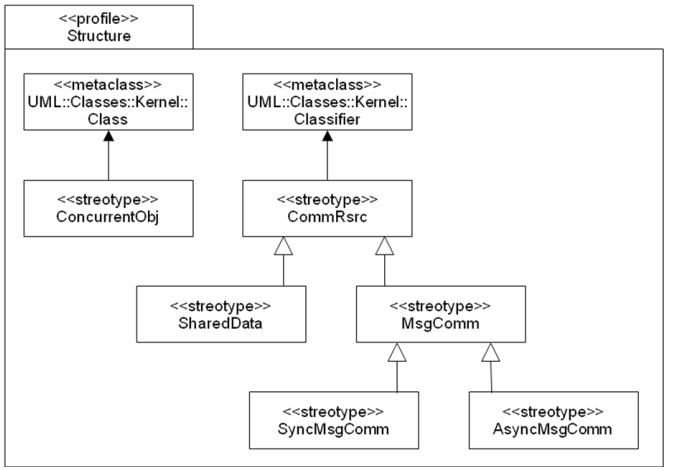

図 8　Structure プロファイルのプロファイル図

動的な構造に関わる Behavior プロファイルについてもドメインモデルを図 9 の(a)に，プロファイル図を(b)に示す．このプロファイルには，並行動作単位の実行の単位となるステップ（ExecStep）が定義される．ExseStep は，このステップが割り込まれることのない atomic なものであるかどうかを示す属性を持つ．ExecStep は Behavior へのステレオタイプとなる．

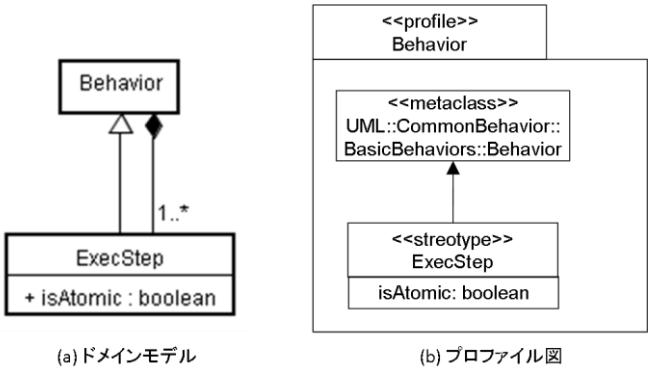

図 9　Behavior プロファイル

4.2 拡張例 UMLVerification

UMLVerification は，クラス図と状態機械図に基づいた UML でのモデルの検証に必要な定義を与えるプロファイルである．UMLVerification は V-Core の拡張であるが，直接 V-Core を拡張したものではない．SPIN を用いた検証を行う際に必要となる基本的な定義を与えるプロファイル V-SPIN を V-Core を拡張して定義し，さらに V-SPIN を拡張して UMLVerification を定義するという方法を取っている．このような多段の拡張をすることにより，各プロファイルの再利用性を高め，多様性に細かく対応している．V-SPIN の詳細については，ここでは割愛する．

UML の動作意味論，特に状態機械図の意味論は SPIN への入力言語 Promela の動作意味論とは，例えばイベントの種類や各種動作の実行順序などの観点で必ずしも一致しない．UMLVerification では，こうした動作意味論のギャップを埋めるように要素を定義した．具体的には，複数の状態遷移を行う並行動作単位が互いに協調しながら行う動作の検証をするために必要十分な要素を選択し，それらの要素が Promela にマッピングされた時，もとの動作意味論との差異が少ない形で動作するようにモデル要素を定義した．

(1) 全体構造

全体構造を図 10 に示す．UMLVerification が V-SPIN を通して V-Core を拡張していることが示されている．V-Core 同様，静的な構造に関わるプロファイル（StructureVerification）と，動的な構造に関わるプロファイル（BehaviorVerification）から構成される．

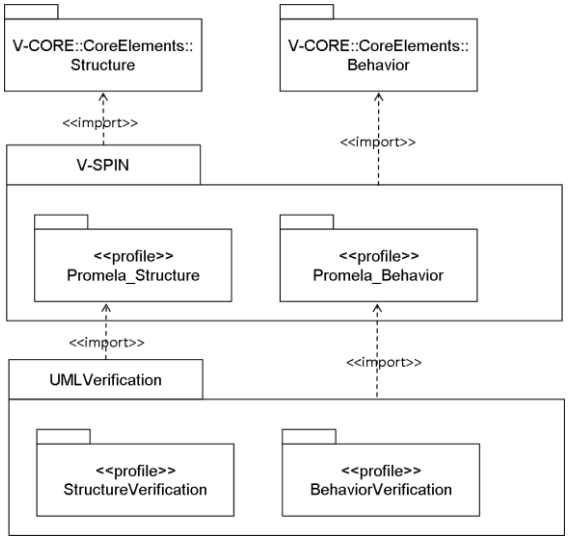

図 10　UMLVerification 全体構造

(2) ドメインモデルとプロファイル図

StructureVerification プロファイルのドメインモデルを図 11 に示す．このプロファイルには，SPIN におけるプロセスに対応する並行動作単位（Process）や，その間のチャネルを用いた通信（ChannelSync，ChannelAsync）が定義される．

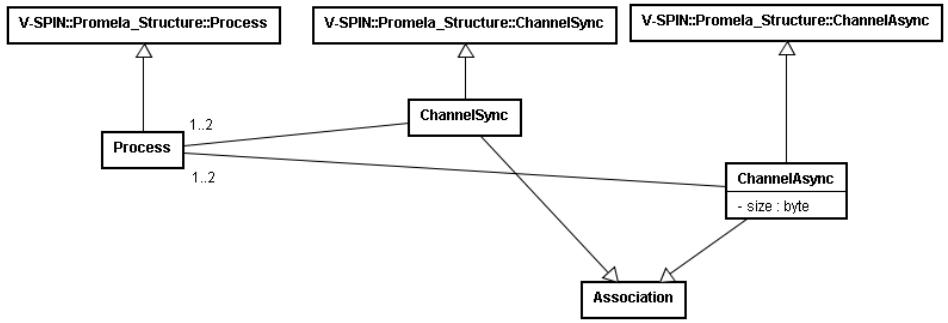

図 11　StructureVerification プロファイルのドメインモデル

また，プロファイル図を図 12 に示す．Process は Class へのステレオタイプ，ChannelSync，ChannelAsync は Association へのステレオタイプとなる．ChannelAsync には，チャネルのバッファサイズを示すタグ付き値 size が定義される．

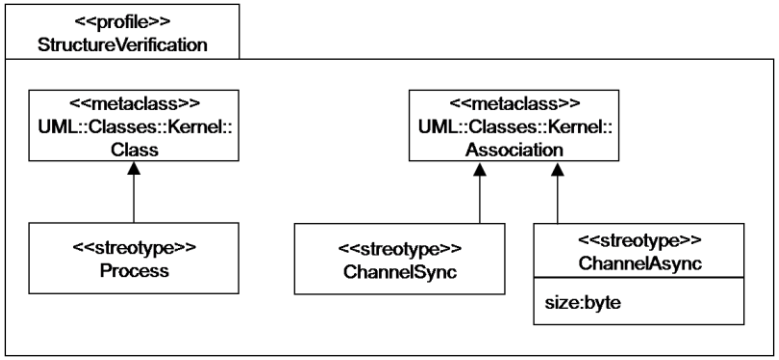

図 12　StructureVerification のプロファイル図

BehaviorVerification プロファイルのドメインモデルを図 13 に示す. このプロファイルは，UML 検証モデルの動的な構造に関わる要素を定義するものである．複数の並行動作単位であるプロセスがインタリーブされながら振る舞い，チャネルによるメッセージ通信によって，メッセージを受信したり送信したりすることにより協調動作が実現されるモデルである．また，実行のステップ（ExecStep，V-SPIN で V-Core を拡張して定義）の集合として状態（State）が定義され，これが振る舞いを定義する基本となる．状態は，その状態でプロセスが動作を止めることが正常な終了であり得ることや，繰り返しその状態に遷移することが正常であり得ることを示すために，ラベルを付与することができる．こうした状態は，状態の先頭となる実行ステップが対応するラベルを持つものとなる（LabeledState）．実行ステップは，メッセージ送信であっても良い．また，状態は，トリガ（Trigger）の発火により遷移（Transition）をする．トリガは，メッセージ受信（MsgReceiving）により実現される．

また，プロファイル図を図 14 に示す．メッセージ送受信（MsgSending，MsgReceiving）は，Behavior へのステレオタイプとなり，atomic に実行されるかどうかを示すタグ付き値 isAtomic を持つ．ラベル付きの状態（LabeledState）は，State へのステレオタイプとなり，付与するラベルを示す isPgoress，isEnd というタグ付き値を持つ．

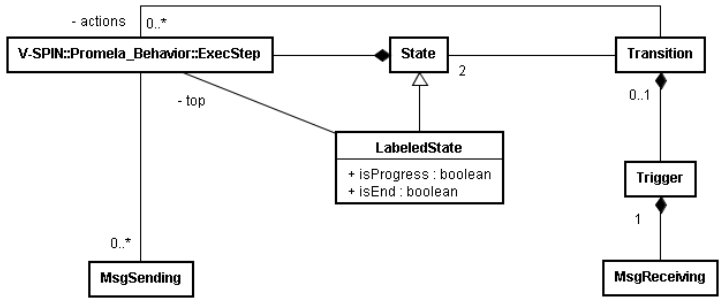

図 13　BehaviorVerification のドメインモデル

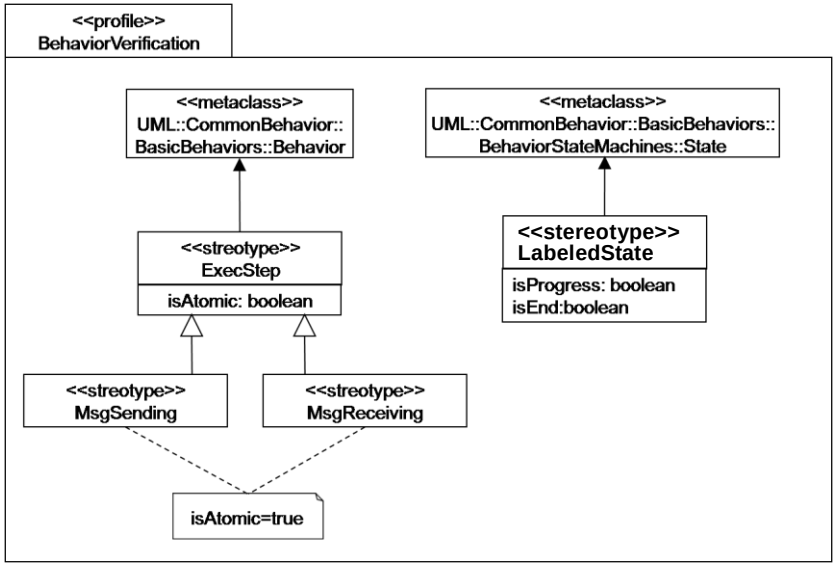

図 14　BehaviorVerification のプロファイル図

5. プロファイル活用方法

4 章では，検証用プロファイルを具体的に提案した．これらのプロファイルの有効性の評価はまだであり，今後実際のソフトウェア開発や開発手法・環境の構築に利用

することにより，有効性を示したい．ここでは，提案したプロファイルの活用方法を示すことにより，これらのプロファイルが具体的にどのように使われ得るのかを提示し，潜在的な有効性の示唆を行いたい．以下，検証用プロファイルの定義・利用に関する，典型的な枠組みについて説明する．

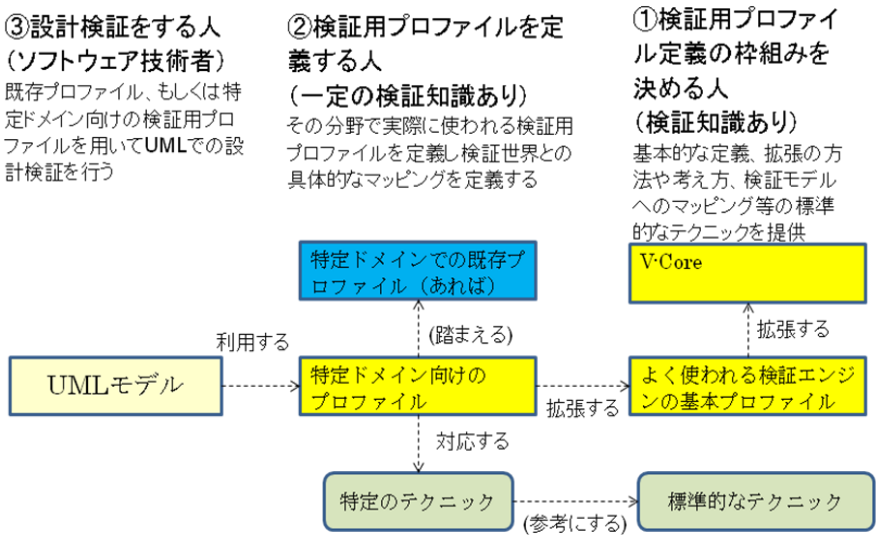

図 15　プロファイルの活用イメージ

図 15 は，プロファイルの定義と活用の典型的なイメージを示すものであり，ここでは大きく 3 つの役割が示されている．

① 検証用プロファイル定義の枠組みを決める人は，コアとなるプロファイルを定義し，またそれに基づいて具体的な検証用プロファイルへと拡張するための方法や考え方を提供する立場である．この役割は，検証に関する知識が必要であるが，基本的にはドメインや組織に依存せず全体で一か所存在すればよい．コアとなるプロファイルのみだけでなく，典型的な検証エンジンの基本機能に対応した検証用プロファイルまで提供する可能性もありうる．また，特定の検証エンジンに踏み込んで，典型的なマッピングの方法，あるいはよく使われる利用方法（テクニック）を提供することも考えられる．この立場は，全体として整合し一貫すべき部分を管理するとともに，次で述べる検証用プロファイルを定義する人にとって必要な定義や方法を提供する役割を持つ．

② 検証用プロファイルを定義する人は，特定のドメインや組織が必要とする具体的な検証用プロファイルを定義する立場であり，そうしたドメインや組織に対応して存在する．具体的には，検証用プロファイル定義の枠組みを決める人が提供するプロファイルや定義方法などを活用して，そのドメインや組織において実際に使われる検証用プロファイルを定義するとともに，それを利用して定義された設計モデルを検証エンジンが理解できる形式にマッピングし，実際に検証を行うために必要な環境（ツール等）を整備する役割を持つ．この立場の人は，検証用プロファイル定義の枠組みを決める人ほどではないが，一定の検証知識が必要となる．

③ 設計検証をする人は，実際のソフトウェア設計を行う立場である．一般にはドメインや組織に複数存在する．検証用プロファイルを定義する人が定義した検証用プロファイルを利用して設計モデルを構築し，それを設計検証する．この立場の人は，上述した二つの立場の人たちに比べ，より少ない検証知識でよい．

6. おわりに

形式手法をソフトウェア開発の現場で実適用するためには，形式手法をソフトウェア開発の中でどのように位置づけていくかというソフトウェア工学面からの検討が不可欠である．我々は，設計検証に形式手法，特にモデル検査を活用することを検討しているが，モデル検査による設計検証をより幅広く行うためには設計モデルを構築するための手法や環境の整備が必要になると考えている．

本稿では，こうした手法や環境の整備のために検証用プロファイルを用いるアプローチを提案し，コアとなるプロファイル V-Core とこの拡張例として UML で設計したモデルの検証に必要な定義を与えるプロファイル UMLVerification を提案した．今後は，これらのプロファイルを開発手法・環境の構築に実際に利用し，有効性を示したい．

参考文献

1) NuSMV, http://nusmv.fbk.eu/
2) OMG Unified Modeling Language™ (OMG UML), Superstructure Version 2.3, http://www.omg.org/spec/UML/2.3/Superstructure
3) UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems Version 1.0, http://www.omg.org/spec/MARTE/1.0
4) UPPAAL, http://www.uppaal.com/
5) Holzman, G.: The SPIN Model Checker: Primer and Reference Manual, Addison-Wesley Professional (2003)
6) 岸知二，野田夏子:組込みソフトウェアのための UML 設計検証支援環境，組込みシステムシンポジウム 2006 論文集，pp.50-57 (2006)
7) 八鍬豊，野田夏子: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，第 9 回情報科学技術フォーラム(FIT2010)