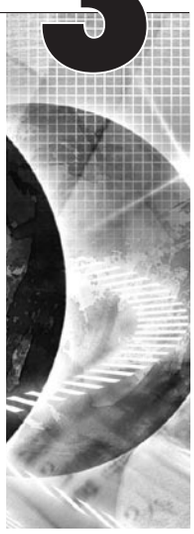
モデル検査技術によるUML設計検証



岸　知二
北陸先端科学技術大学院大学情報科学研究科
tkishi@jaist.ac.jp

設計品質の重要性

組み込みソフトウェアが社会のさまざまな部分で使われるようになり，その信頼性が大きな社会的問題となっている．またメーカにとってもいったん品質問題を起こした場合の損害は莫大となりかねず，組み込みソフトウェアの信頼性は重要な課題となっている．経済産業省の組み込みソフトウェア産業実態調査でも，プロジェクトにおいて最も重要なものは品質であり，プロジェクトに課せられている最重要の問題は設計品質の向上であるとの結果が出ており，メーカ自身が設計品質の向上に強い問題意識を持っていることが明らかになっている．

ソフトウェアのバグの多くが分析，設計といった上流工程で混入し，それがシステム統合などの下流工程で検出されていることはよく知られている．ソフトウェアの品質を向上させるためには，こうした上流工程の作業品質を改善して上流工程でのバグの混入を減らすとともに，上流工程で作りこまれたバグはできるだけその工程で検出することが重要となる．バグを下流工程まで見逃すことで引き起こされる手戻り作業はコストの増大，納期の遅れを引き起こすだけでなく，現実問題としてはリリース間際に場当たり的な対応を余儀なくされ，その後の改造や拡張を困難にするなど品質面にも悪影響を及ぼす．

従来組み込みソフトウェアはエンタープライズ系のソフトウェアと比較して規模も小さく，そのためより実装工程に重点を置いた開発形態がとられてきた．また技術的には，標準的なフレームワークやアーキテクチャの上で開発することの多いエンタープライズ系のソフトウェアと違い，新たに作られるハードウェアの上でリアルタイムOSのプリミティブなサービスコール等を用いて，定められた資源制約や時間制約の中で動作する適切な実行メカニズムを検討することが多いため，どうしてもこうした開発形態にならざるを得なかったともいえる．しかしながら，組み込みソフトウェアの規模や複雑さはますます肥大化しており，一方開発期間も短縮されていく中，もはや従来の実装中心の開発スタイルでは生産性や品質を維持することが次第に困難になってきている．そのため，こうした新たな状況に対応できるように，ソフトウェア工学やソフトウェア科学の成果を最大限活用した組み込みソフトウェアの開発スタイルが模索されている．

こうした背景の中，我々は文部科学省リーディングプロジェクト「e-Society基盤ソフトウェアの総合開発」の研究課題「高信頼組み込みソフトウェア構築技術」を実施しており，この中の活動の1つとして，形式検証技術を用いた組み込みソフトウェアの設計品質向上の問題に取り組んでいる．具体的にはUML（Unified Modeling Language）で記述された設計に対してモデル検査技術を適用する手法やツールを開発し，それを現実の組み込みソフトウェアにどのように適用すれば，設計品質の向上に有効となるかの検討を企業との連携の中で進めている．本稿では，形式検証技術，特にモデル検査技術を，組み込みソフトウェアの設計検証へ適用することについてソフトウェア工学的な観点より検討を行う．

まずUMLモデルに対してモデル検査技術を適用する方法，ならびにそれを支援するツールの紹介をする．次にソフトウェアの設計検証にモデル検査技術を適用する際のアプローチや手法について検討し，さらに実際に検証した例を紹介する．最後に，ソフトウェア開発の中でモデル検査技術をどのように位置づけて活用すべきか考察する．

UMLへのモデル検査技術の適用

◆UML検証の意義

ソフトウェアの分析・設計の品質を上げるための第一歩は，分析・設計した結果を正確にモデリングするこ

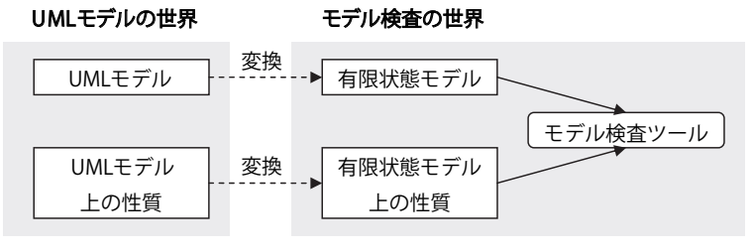


図-1　UMLモデルへのモデル検査技術適用の基本的な枠組み

とである．ここでモデリングとは，対象をある目的意識を持って抽象化し，それを読み方，書き方の定められた記述方法に則って表現することを意味する．従来，ソフトウェアの仕様書や設計書は自由な文章や図を中心として記述されることが多かったが，そうした記述では記述に漏れ，あいまいさ，矛盾等が入り込みやすく十分ではない．開発手法の世界では，そうした問題を改善するために，上流工程にモデリングを活用することが一般的であり，今までに多くのモデリング技法が提案されてきた．組み込みソフトウェアの分野でも状態遷移図・表などの活用が広く行われている．90年代になって，ソフトウェア開発でよく使われる各種モデリング記法がUMLという形で標準化され，分野によっては徐々に広く使われるようになってきた．組み込み分野においてはUMLそのものの活用はまだ広がってはいないが，ソフトウェア技術の進展を見ると，上流工程でのモデリングにUMLを活用することは大きな流れとなると考えられる．

組み込み分野に限らず実際のソフトウェア開発の現場でUMLがどのように使われているかを見てみると，自然言語や自由な図と比べて相対的に記述が正確であることや，記法が世界標準であることから，あくまで人間が読むドキュメントとしてコミュニケーションの円滑化をねらった活用となっている．もちろんそれでも効果はあるが，人間が見るだけでは記述の良し悪しの判断が難しく，またレビューでの検証が中心となるため，信頼性向上の目的のためには十分とはいえない．そこでUMLで記述されたモデルの妥当性を，人手ではなく，科学的手法で検証することにより，モデルの信頼性を向上させ，ひいては設計品質の向上へとつなげることを狙っている．

◆UML検証の枠組み

モデル検査技術は，有限状態モデルと論理的な性質が与えられると，そのモデル上で与えられた性質が成立するかどうかを自動的に検査する技術である[2]．したがって，UMLモデルをモデル検査技術で検証する1つの直接的な方法は，UMLモデルを有限状態モデルに変換し，UML上で検証したい性質を論理的な性質として表現し，それをモデル検査ツールにかけるという方法である．こうすることにより，UMLモデル上でその性質が成り立つかどうかを，厳密に検証することができる．たとえば組み込みソフトウェアでは，外界からさまざまな順序やタイミングでイベントがやってきても正しく動作する必要があるが，そうしたさまざまな順序やタイミングに関した網羅的な検証に，モデル検査技術が有効に適用できる．**図-1**にUML検証の基本的な枠組みを示す．

◆支援ツール

こうしたUML設計検証を支援するために，現在我々は支援ツールを開発している．このツールはEclipseプラットフォームの上に開発され，UMLの記述にはUMLプラグインを，またモデル検査ツールとしてはSPIN[3]を用いている．SPINでは検査対象モデルを記述するためにPromelaという言語を提供しており，また性質の記述にはLTLという時相論理を用いている．現在はUMLの図としてはクラス図とステートマシン図のみを使っている．またPromelaとの対応付けを行いやすくするためにステレオタイプを導入するなど記法上の工夫を行っている．なお検証の目的によって対応付けの方法を変更させたい場合があるため，いくつかのバリエーションを用意しており，対象システムの動作意味や検証目的に応じて変換方法を切り替えることができるようになっている．

UMLとPromelaの基本的な対応関係を**表-1**に示す．直感的には，ステレオタイプでprocessと指定されたクラスのインスタンスがタスクのように並行動作し，それらが同期もしくは非同期の通信を行うといった動作意味になるように対応付けがなされている．また検証したい性質を論理式で記述する作業を容易にするために，UML上の定義概念(クラス名や状態名など)を用いて記述されたLTLを，変換後のPromela上の定義概念（process名やラベル名）を用いたLTLに変換する機能を提供している．さらに，SPINの提示するPromela上での反例はUML上でのシーケンス図として表示される．本ツールの概要を**図-2**に示す．

UML	Promela
クラス<<process>>	・対応するステート図からprocessを生成 ・属性はグローバル変数 ・インスタンスはidで識別
クラス（非<<process>>）	・属性はグローバル変数
関連<<channel>>	・双方向のchannel ・両端のクラスに引数として渡される ・多重度はidで識別
関連<<shared_channel>>	・1つのchannel ・異なった関連でも同じ名称ならば同一のchannelになる.
初期状態	・init中で処理 ・各processは，初期状態から開始するように工夫

典型的なPromelaの構造

グローバル変数宣言等
processクラス名 (….)
（初期状態）
状態ラベル：
　エントリアクション
　if
　ガード条件 -> goto 次状態
　…..
　else goto 自状態
　fi
　….
}
init {
　atomic {
　　グローバル変数の初期化
　　属性の初期化
　　チャネルの初期化
　　インスタンスのrun
　}
}

表-1　UMLとPromelaとの対応関係

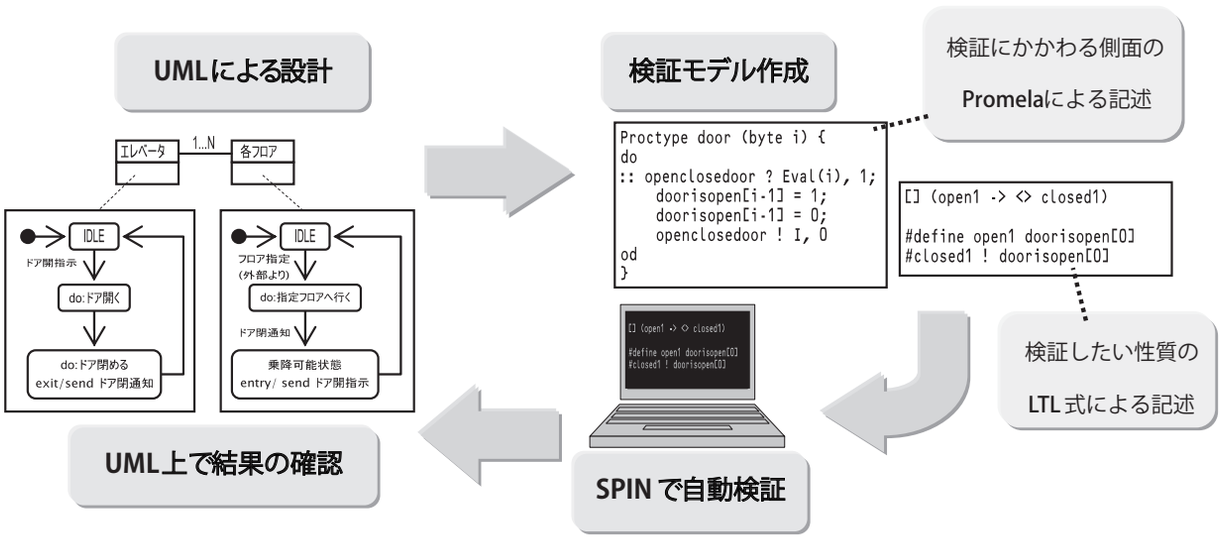


図-2　支援ツールの概要

UML設計検証手法

◆設計検証の課題

従来UMLモデルは人が読むドキュメントとして記述されていたため，記述が曖昧であったり，暗黙の了解事項に基づいて記述されたりすることが通常であった．しかしながら，モデル検査技術はコンピュータ上のツールが検証を実行するため，より厳密な記述が要求される．たとえば記述されたUMLモデルがどのような振る舞いをするのかという実行の意味をより明示的かつ厳密に定義する必要がある．しかしながら現実規模のソフトウェアの設計を，モデル検査可能な厳密なモデルとして定義することは，実際にはなかなか困難である．

またUML上で確認したい性質を，有限状態モデル上での性質として表現する必要があるが，ソフトウェアの性質を時相論理式として表現する作業はコストのかかる作業であり，またモデル検査技術も得手不得手があるため，検証可能な項目と検証困難な項目とがある．たとえば組み込みソフトウェアで重視されるリアルタイム性にかかわる性質の検証は，モデル検査技術の苦手な分野である．したがって，こうした特性を踏まえた上で，効果的な検証項目の選定をすることが必要となる．

このようにソフトウェア設計検証にモデル検査技術を適用するためには，単にツールを用意するだけでは不十分であり，どのように設計検証を行うか，検証の手法を整理する必要がある．

◆設計検証のアプローチ

我々はモデル検査技術を用いて設計検証を行う際には，以下のようなアプローチをとることが適切であると考えている[4]．

- **検証目的に対応した視点からの選別的なモデル化**：現実規模のソフトウェアの設計全体を，モデル検査可能な詳細さ，厳密さでモデル化することは困難であるため，検証目的に応じた視点(範囲と側面)だけを選別的にモデル化する．その際，具体的にどのような視点に注目して詳細なモデル化を行えばよいのかが問題になるが，我々は，その視点はアーキテクチャ設計の分野で言われているアーキテクチャ上の重要性という観点と一致すると考えている[1]．
- **重要なシナリオに即した検証項目の選定**：検証に際しては，やみくもに検証を行うのではなく，設計検証上重要なシナリオに照らした検証項目を選ぶことで，意味のある検証が可能となる．たとえば方式上の問題を確認する際には，類似した機能の確認をすることは大きな意味はなく，起動，終了，エラー発生時など，重要なシナリオを吟味し，それらに照らした検証を選別的に行うことが妥当である．
- **設計上の想定にかかわる網羅的な検証**：設計においてはその背後にある想定や文脈に関して広範囲かつ慎重な検討が必要となる．たとえばあるタスクが適切に動作するかどうかは，そのタスクの設計だけをみても判断できず，他のタスクとの優先度の関係，起こり得る割り込み，共有リソースに対するアクセスのポリシーなどがかかわってくる．モデル検査技術は網羅的な検証に向いており，こうした想定や文脈を踏まえたモデル化を行うことで，より確実な検証を行うことができる．

◆UML設計検証の手法

上述したアプローチを踏まえ，設計検証にモデル検査技術を適用する際の手法を検討した．**図-3**に検証の手順を示す．

1. 通常の設計モデルを作成する．従来の設計の流れの中で，UML等を活用した設計モデルを作成する．これは一般的には人間が読んで理解するためのモデルであるため，モデル検査技術などを適用することはできないが，全体を把握するためには有効かつ必須である．
2. アーキテクチャ上の重要性の中から，モデル検査技術による確認が有効である検証事項にフォーカスして，検証項目を選定する．典型的には，重要な処理の骨格が，さまざまな想定や文脈の中でも機能することを確認することなどに有効であると考えられる．
3. 上述した検証項目にかかわる重要な想定や文脈を整理する．一般的には外界の多様性，制御モデル（スケジューリング，優先度，割り込み），同時に動作する他のタスク等とのかかわり，共有リソースへのアクセスポリシーなどが考えられる．しかしながら潜在的に考えられる想定や文脈をすべて列挙することは無意味であり，現実的でもない．考慮すべき想定や文脈はアーキテクチャ上の重要性同様，経験に基づいて吟味されるべきものである．たとえばTrewらは過去の障害レポートを分析することでこれらを洗い出すなどしている．
4. 選定された検証項目ごとに，その検証を行うために必要な視点，抽象度，記述範囲を明確にし，検証目的の設計モデルを記述する．この際，記述の範囲はその検証にかかわる部分に限定し不要な部分を捨象する．一方で，かかわる想定や文脈は，どの範囲の想定や文脈を扱うかを明確にし，その範囲での妥当性確認ができるようにモデルに反映させる必要がある．たとえばより優先度の高いタスクが同時に動作していても良いという想定であれば，優先度の高いタスクがさまざまな状況で実行権をとるようなモデルを作成し，それでも確認すべき方式が機能するかどうかを確認する．
5. 検証を実行する．検証結果を検討し，必要なら設計モデルにフィードバックをかけ，検証のサイクルを繰り返す．

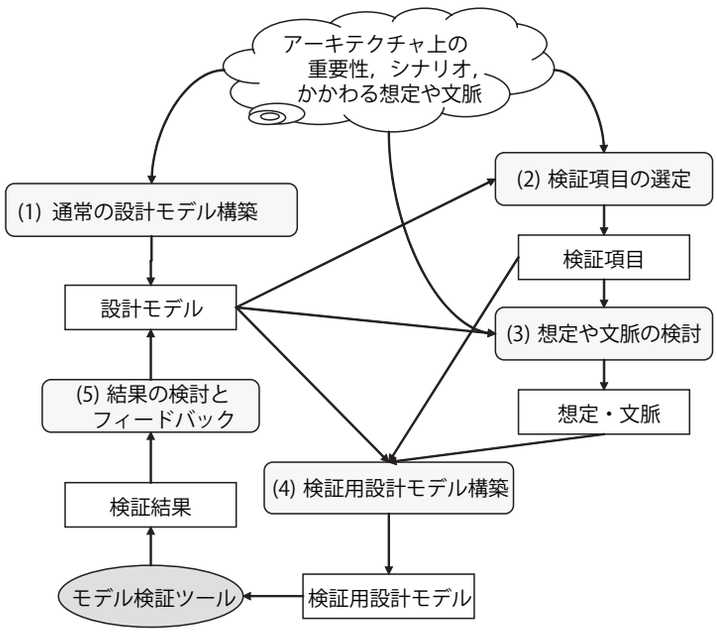
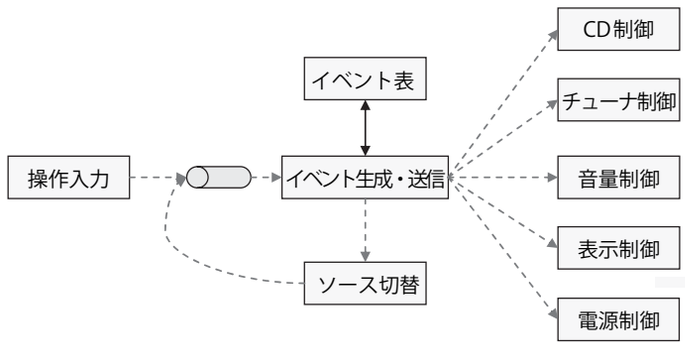
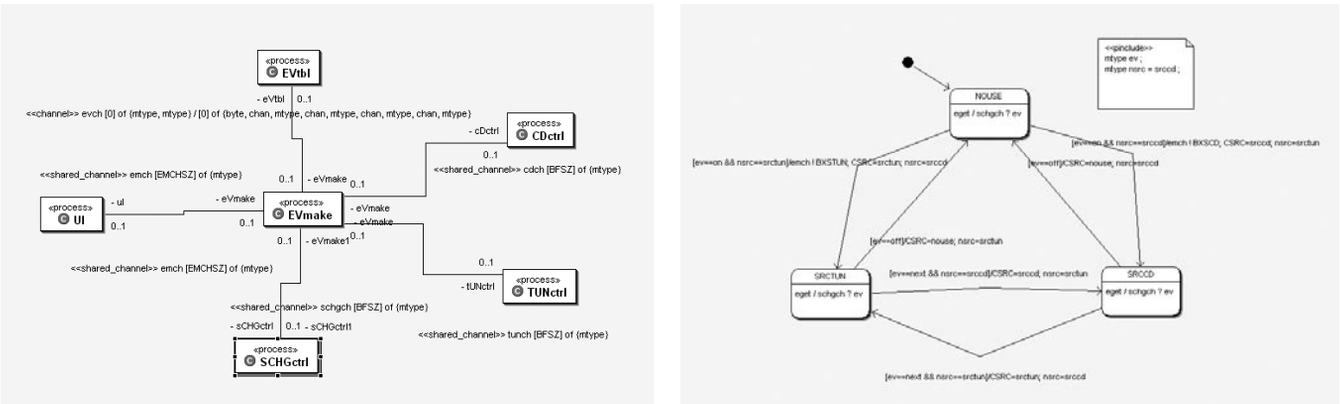


図-3　モデル検査技術適用のステップ



(a) 最初のアーキテクチャ案



(b) 検証のためのUMLモデル（クラス図とステートマシン図）

図-4 最初のアーキテクチャ案と検証モデルの一部

◆適用例

企業より提供されたカーオーディオシステムの設計を本手法に基づき，UML設計検証を行った事例を紹介する．本システムはリファレンスセットとして開発されており，実際の分析・設計にはUMLを，実装にはC言語を用い，32bitCPUの上でμITRONを用いて開発されている．適用にあたっては，仕様を限定し，実際の設計をベースにその仕様の範囲で再設計を行いながら，アーキテクチャレベルの設計の妥当性を確認した．

本システムでは，入力されたボタン操作がどういう処理に対応するかは，その時点でどのソース（CD，チューナ）が設定されているかに依存して決定される．**図-4**(a)は，この処理を行うための方式案の1つである．ここでは「イベント生成・送信」が「操作入力」からの操作指示を受け取るとソース状態を管理する「ソース切替」に現在ソースを確認した後，「イベント表」を参照して対応する処理（各種制御あるいは「ソース切替」に対する処理の指示群）を判断し，必要な制御や「ソース切替」に処理指示を送信するという方式になっている．この際，「ソース切替」は処理指示を受け取ると，その処理のために，さらに「イベント生成・送信」に対して指示を送ることがあり得る．この方式の妥当性を検討するにあたり，本手法に従い以下のように検証を行った．

1. 全体のUMLモデルを作成．
2. 検証項目として，本方式がどのような操作入力に対しても，正しくそれを処理できるかどうかを確認することを設定．具体的には，どのような操作入力に対しても処理が停止しないこと，「ソース切替」から「イベント生成・送信」への処理の依頼が行われる合間に「操作入力」から次の操作指示が来ても，処理の前後関係がくずれることがないこと，を調べる．
3. 考慮する想定としては，「操作入力」「イベント生成・送信」「ソース切替」各種制御が，並行して動作し，どのようなスケジューリングに対しても対応できることを考える．
4. 上記を踏まえ，以下の方針で検証用のモデルを作成する．**図-4**(b)にその一部を示す．
   - 方式にかかわるイベントの伝達という観点だけに注目し，他の処理は省略する．
   - 重要なシナリオとして起動，終了，ソース切替などを取り上げる．CD再生，停止，チューナ再生等の

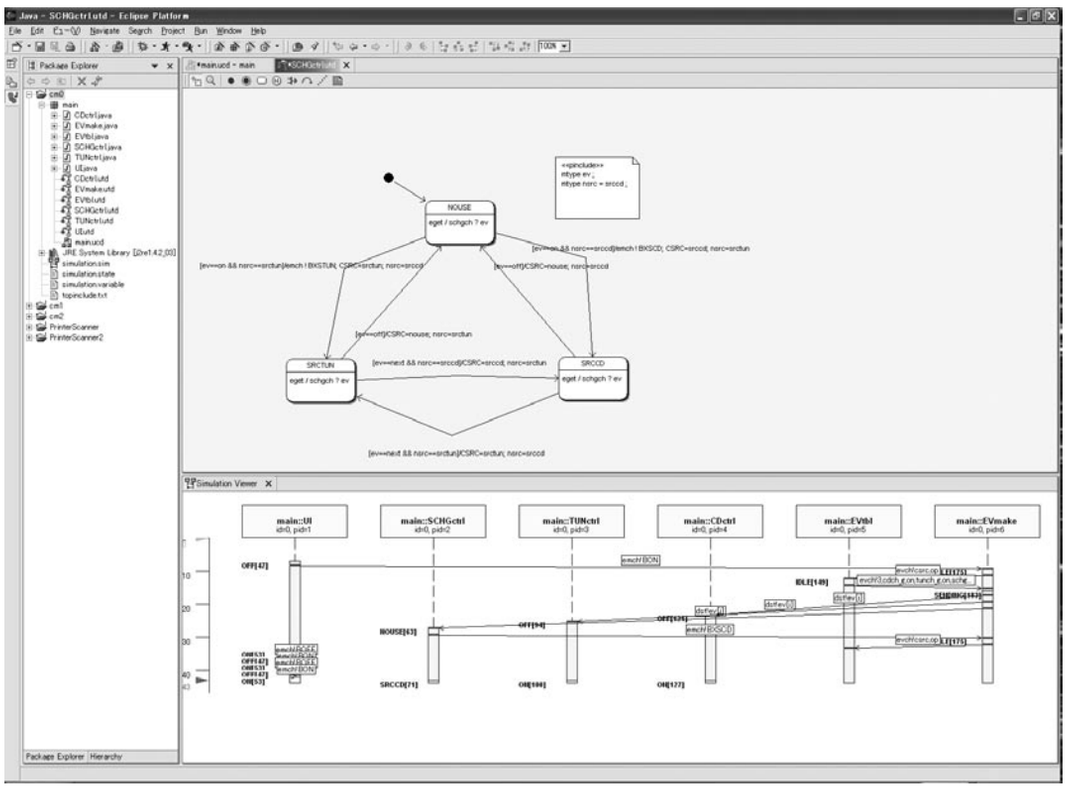


図-5　支援ツール上での反例の表示

処理は，本検証目的からすると類似した処理なので，それぞれを独立したシナリオとしては扱わない．

- ソースの制御にかかわらない「音量制御」「表示制御」「電源制御」も省略する．
- それぞれのモジュールは並行に動作し，スケジューリングは特段に行わない（スイッチできる個所にくればどのモジュールにもスケジュールされる可能性を考慮する）．
- イベントの送受信は非同期に行う．

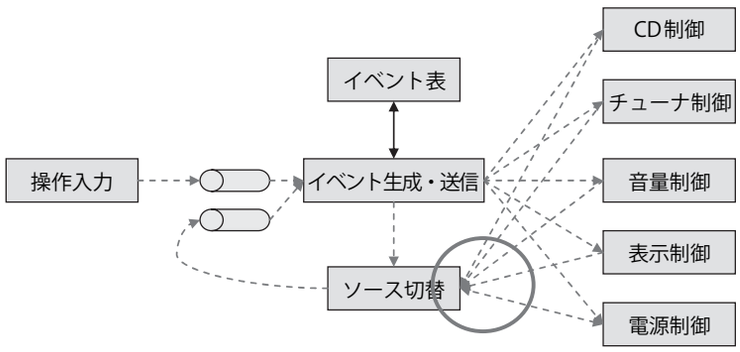
5. 実際に検証を行うと，ある状況で処理が停止してしまうことが確認された．具体的には，「操作入力」からの操作指示を処理する途中で「ソース切替」が「イベント生成・送信」に指示を送ろうとする前に，「操作入力」から次の操作指示が複数到着してバッファがフルとなり，「ソース切替」がブロックしてしまう状況があることが確認できた．**図-5**はこの状況を示す反例を支援ツールで表示したものである．

上記の検証のサイクルで最初の方式案に問題があることが確認されたため方式を見直し，「操作入力」からの操作入力のためのバッファと，「イベント生成・送信」からの指示のためのバッファを別に設ける方式を検討し，ほぼ上記と同様のステップで検証用モデルを作成し検証を行った．しかしながらこの方式も，イベント表を参照するタイミングによっては処理が停止してしまうことが検証により確認された．そこでさらに方式を見直し，各種制御の終了等をソース切替が確認し，実際の制御の状況とソース切替が把握している状況とのずれが生じないようにした．この方式は検証によって，さまざまな操作入力シーケンスを与えても停止することがないことが確認された．さらに操作入力のイベントに前後関係がくずれることがないことも検証され，アーキテクチャ設計の段階で本方式の妥当性について一定の確認を行うことができた．**図-6**に最終的なアーキテクチャと，検証のためのUMLモデルを示す．

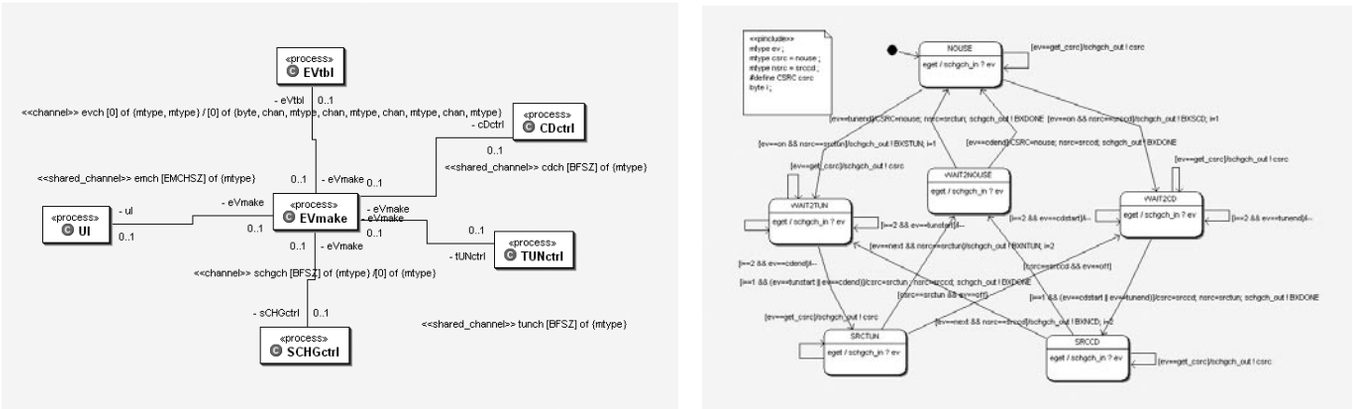
開発におけるモデル検査技術の位置づけ

モデル検査技術を実際にソフトウェア開発に適用しようとするためには，モデル検査技術を用いてソフトウェアのどのような問題が検出できるのかということだけではなく，モデル検査技術がソフトウェアの他の検証技術と比較してどのような特性を持ち，ソフトウェア開発の中でそれをどのように利用していくかの議論が必要となる．そこで，代表的なソフトウェア検証技術としてのレビューとテストをとりあげ，それとモデル検査技術の特性をいくつかの観点から定性的に順序付けしたのが**表-2**である．順序はより優れていると考えられるものを1，続いて2，3としている．

上記の順序付けの理由は以下のとおりである．



(a) 最終的なアーキテクチャ案



(b) 検証のためのUMLモデル（クラス図とステートマシン図）

図-6 最終的なアーキテクチャ案と検証モデルの一部

	レビュー	テスト	モデル検査技術
正確性・厳密性	3	2	1
検証項目の幅	1	2	3
未定義問題の検出	2	1	3

表-2 既存の検証技術との比較

- **正確性・厳密性**：これは検証がどのくらい正確・厳密に行われるかという観点である．モデル検査技術は検証したい性質を明示的に定義し，それを網羅的に確認するため最も正確性・厳密性が高いとした．テストは実際にマシン上で実行しながら確認するためその結果は正確だが網羅性などの観点では不十分である．レビューは人手で行われるため，正確性や厳密性は劣る．これは検証項目の定義の厳密さにも表れており，モデル検査技術では論理式，テストではテストデータやテスト仕様書，レビューではチェックリスト等となり，それぞれ正確性・厳密性が異なる．
- **検証項目の幅**：これは検証によってどれだけの広さの内容を確認できるかという観点である．レビューは人が行うためその過程でさまざまな問題に気づくなど，かかわる問題を幅広く見つけることが期待される．テストの場合は実際に動作させる過程で，テスト項目に類似した問題が検出されることもある．一方モデル検査技術では目的に応じて検証対象をモデル化し，検証したい性質を厳密に定義して検証をするため，ピンポイントな検証となる．この特性は前項の正確性・厳密性とは裏腹な観点であるといえる．
- **未定義項目の検出**：明示的に定義していない問題を検出できるかどうかという観点である．テストは実際に実行することで問題を検出するために予期しない問題でも実際に動作上の異常として観測されれば検出が可能である．長時間使い込みながら問題を出すというような検証はテストならではの方法である．レビューは人手で行うために検証目的とは別の問題に気づいたり

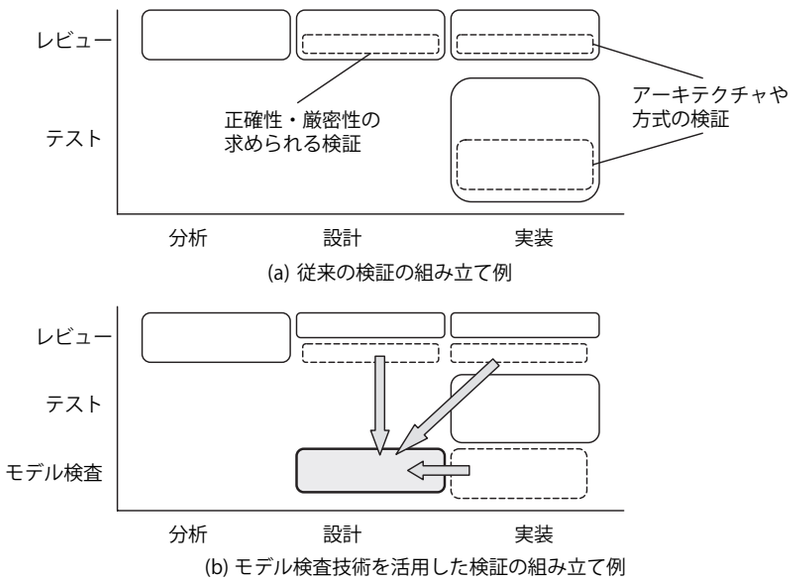


図-7　検証の組み立て方の変化イメージ

新たな問題を発見したりすることが期待できるが，人手であるためどうしてもアドホックなものとなる．モデル検査技術では未定義の問題を検出する可能性はかなり低くなる．

このように，それぞれの検証技術は異なった性格を持っており，ソフトウェア開発においては，それを踏まえた使いこなしが必要である．すなわち，気づきや発見的な視点が必要な検証にはレビューを，実際に動作させなければ分からない事項の確認にはテストを，そしてアーキテクチャや方式自体の論理的な妥当性等を厳密かつ網羅的に確認するにはモデル検査技術を活用することが望ましいと考えられる．従来はテストやコードレビューでアーキテクチャや方式の妥当性確認が行われることがままあったが，テストやコードレビューは実装上の問題にフォーカスすることが本質であり，またそれを改善しないと冒頭に述べた品質問題の本質的な改善にはつながらない．

図-7はこうした考察を直感的に示したものである．従来の方法（a）においては，方式やアーキテクチャ上の問題が実装段階のレビューやテストによる検証まで見逃されていたり，正確性や厳密性が必要な検証を設計レビューによって行っていたりしていたが，そうした検証のうち，モデル検査技術で行うことが有効な検証を設計段階で行うことにより，検証の組み立て方が変化することを示している．モデル検査技術を活用した方法例（b）では，検証の組み立ては以下のようになる．まず分析段階ではモデリングあるいは形式的仕様技術などを用いてより厳密な分析，仕様化を行い，レビューなどで検証を行う．検証内容によってはプロトタイピングなどの技術も活用できる．設計段階では発見的な設計レビューと，方式やアーキテクチャの厳密な評価のためのモデル検査技術による検証とを使い分ける．実装段階では実装レベルの間違いを発見するペアレビュー等と，実際に動作させなければ分からない項目を確認するためのテストとを組み合わせる．このようにそれぞれの検証技術の特性を踏まえ，それらを有効に活用した検証の組み立てを検討することが重要になると考える．

なお，ソフトウェアの種類や開発形態によってモデル検査技術の適用性が異なる．モデル検査技術は他の検証技術と比べて適用コストが高いため，そのコストに見合うだけの重要性を持ったソフトウェアに適用することが必要となる．特に再利用資産のように何度も繰り返し使われるソフトウェアへの検証は，単発のアプリケーションに比べて検証のコストをかけやすいため，適用がしやすい対象であろう．我々はプロダクトライン開発におけるモデル検査技術の活用が有効であると考え，検証モデルの再利用方法などの検討を行っている[5]．

今後の方向性

本稿では組み込みシステムの設計検証にモデル検査技術を適用することで品質向上を目指す取り組みについてそのねらい，設計検証の考え方，ソフトウェア開発の中での活用方法に関する展望などを紹介した．形式検証技術のソフトウェア検証への適用は現在さかんに検討されており実務でも使われ始めている．この技術がより広く使われるようになるためには，ソフトウェア工学面からの適用検討もいっそう重要になると考える．言うまでもなく，形式検証技術は銀の弾丸ではない．しかしながら従来の検証技術にはない優れた特性を持っていることは事実である．大切なことは，さまざまな検証技術の特性を理解して，それらを効果的に使い分けていくことである．我々のプロジェクトにおいても，今後さまざまな事例による適用評価を重ね，真に品質向上に有効な利用方法を明らかにしたい．

参考文献

1）Bass, L., Clements, P. and Kazman, R.: Software Architecture in Practice Second Edition, Addison-Wesley (2003).
2）Clarke, E., Grumberg, O. and Peled, D. : Model Checking, MIT (1999).
3）Holzmann, G. J.：The Model Checker SPIN, IEEE Trans. on Software Engineering, Vol. 23, No. 5, pp.279-295 (1997).
4）岸　知二，野田夏子：モデル検査によるアーキテクチャ設計検証，情報処理学会ソフトウェア工学研究会 SE150-2 (2005).
5）Kishi, T., Noda, N. and Katayama, T.: Design Verification for Product Line Development, Software Product Line Conference 2005（SPLC Europe 2005）.

（平成18年4月7日受付）