Vol.2012-SE-175 No.10
2012/3/15

制御ソフトウェアの仕様整合性検証技術の調査と分析

大貫智洋†

制御ソフトウェアの開発手法として，モデルベース開発の普及が進んでいる．モデルベース開発では実装コードを機械的に生成することが可能であるため，実装に誤りが混入する危険性は低い．その一方で，要求仕様や制御仕様の記述は人手により行っているため，誤りが混入しやすい．このような課題を解決するため，本論分では上流工程で仕様を正確に記述する手法について，調査・分析した．要求分析を正確に記述する手法，制御仕様を検証する手法についてまとめたのち，今後の課題を分析する．

A Research on How to Verify Consistency between Specifications of Control Systems

Tomohiro Onuki†

Recently, Model-Based Development (MBD) is commonly adopted as a method of developing software for control systems. While MBD enables automatic code generation and prevents errors in the implementation process, requirements specifications and control specifications are still written by hand. This paper describes the result of my research on how to describe requirements specifications precisely and verify consistency between specifications.

1. はじめに

1.1 背景

近年，制御ソフトウェアの複雑化・大規模化が進んでいる．例えば自動車に搭載される制御ソフトウェアの規模は，2000 年当時に 100 万行程度であったものが，2009 年時点で 500 万行～1000 万行にまで増加している**[1]**．このような複雑化・大規模化の背景には，制御系システムの電子制御化が進み，より広範な機能をソフトウェアによって実現するようになったことがある．例えば自動車では，エンジン制御やトランスミッション制御，パワーステアリング制御，ドアロック制御など，パワートレイン系，シャシ系，ボディ系の様々な制御に ECU（Electronic Control Unit：電子制御ユニット）を用いており，1 台あたりに搭載される ECU の数は年々増加する傾向にある**[2]**．このようにソフトウェアが広範かつ重要な役割を担うようになった結果，ソフトウェアの信頼性に対する要求は一層厳しいものとなっている．

大規模・複雑化したシステムの開発では，上流工程で誤りのない仕様を記述することがより大きな意味を持つ．システムの振舞いが多様になると誤りが潜在化しやすくなり，試験工程での誤りの検出に漏れが生じる可能性が高まるためである．また，試験工程で誤りを検出できた場合でも，仕様の作成にまでさかのぼって誤りを修正するためには多大なコストが必要となる．

一方，制御系システムの開発現場では，ソフトウェア開発手法としてモデルベース開発の普及が進んでいる．モデルベース開発は，システムの制御仕様を実装レベルのモデルで記述し，設計段階でのシミュレーション検証や実装コードの自動生成を可能とする手法である．早期の段階で実機レスの検証が可能であることや，実装コードを短期間で生成できること，開発者の間で設計情報を共有しやすいことの利点から，開発現場で普及が進んでいる．ソフトウェア品質の観点では，実装コードを機械的に生成することが可能であるため，従来の人手による作業に比べて実装に誤りが混入する危険性を抑えることができる点が特徴である．その一方で，要求仕様や制御仕様の記述は人手によって行っている．自然言語の曖昧さや記述漏れを原因とする誤りが混入しやすいため，いかにして上流工程で仕様を正しく記述するかが重要な課題となっている．

† 三菱電機株式会社
Mitsubishi Electric Corporation

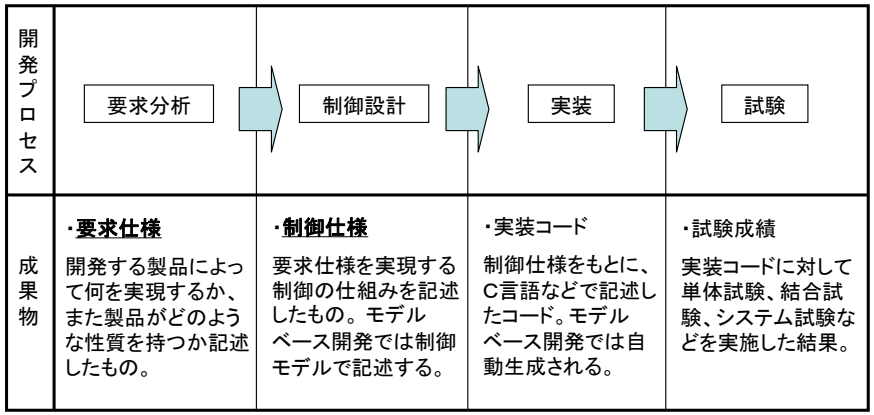

図 1　モデルベース開発のプロセス

1.2 本論分の目的

モデルベース開発のプロセスは，要求分析，制御設計，実装，試験からなる（図 1）．要求仕様や制御仕様といった用語の定義は文献によって様々であるが，本論分では次のように定義する．

・要求仕様： 開発する製品によって何を実現するか，また製品がどのような性質を持つか記述したもの．

・制御仕様： 要求仕様を実現する制御の仕組みを記述したもの．

本論分では，図 1 に示すプロセスのうち要求分析と制御設計に焦点をあて，各工程で仕様を正しく記述するための手法について関連研究の調査を行った（表 1）．

表 1　調査の体系

要求仕様の記述に関する手法	
	記述範囲の明確化に関する手法
	記述形式の明確化に関する手法
制御仕様の検証に関する手法	
	検査項目の作成に関する手法
	検査モデルの作成に関する手法
	複数のモデル記法で記述した仕様の検証に関する手法
	モデル検査の適用事例

2. 要求仕様の記述に関する手法

要求仕様の記述に漏れや曖昧さが生じる原因には，次の二つの問題が考えられる．

・要求仕様に何を書くべきか明確化されていないため，要求分析の段階で考慮すべき内容に見落としが生じてしまう．

・要求仕様を自然言語で記述する場合，様々な表現の仕方が可能であるため，設計者の技量によって曖昧な表現や矛盾した内容を含むことがある．

本章では，それぞれの問題を解決する方法として，「記述範囲の明確化」および「記述形式の明確化」を論じた研究について調査した結果をまとめる．

2.1 記述範囲の明確化

要求仕様に記述すべき内容を明確化する方法として，過去の成功事例を参照して要求仕様の枠組みを形式化する手法が研究されている．文献**[3][4]**では，優良プロジェクトの仕様書分析と有識者へのインタビューを通じて要求仕様に必要とされる項目を洗い出し，要求仕様のメタモデルを構築している．要求仕様に求められる記述項目や項目ごとの粒度がメタモデルで予め定められているため，要求仕様の記述漏れや曖昧さを防ぐことが可能となるとしている．この手法は，過去に実績のあるドメインでは有効であるが，新規に開拓するドメインではメタモデルの構築が困難である．また，過去の事例から要求分析に必要な項目は抽出できるが，要求仕様に記述されていない想定外のシナリオ，すなわち非正常系については抽出できない．

これに対し，文献**[5][6]**では非正常系の抽出手法を示している．まず当初の要求仕様からシステムの正常状態と正常イベント，一時的な非正常イベントを抽出し，マトリクスを作成する．次に，各状態とイベントの交点に注目して連鎖的に発生する二次的な非正常系を抽出する．新たな非正常系を抽出しなくなるまでこれを繰り返すことで，非正常系を漏れなく抽出する．この手法では，予め想定した正常状態やイベントから導出されるシナリオについて連鎖的に発生する非正常系を抽出できるが，当初に想定した状態，イベントに考慮不足があった場合には，そこで生じる非正常系を抽出することができない．

文献**[7]**では，数理的な考えに基づいて非正常系の抽出を試みている．まずユースケースを意味パターンによって構造化し，時相論理式に変換する．この時相論理式とユースケースの事前条件をもとに，事後条件が成立することの証明を行う．証明に不足する条件を，暗黙に想定された条件とみなし，証明の条件に追加する．列挙した条件の否定は，仕様に記述された条件，または暗黙に想定された条件の否定であり，非正常系に他ならないとしている．この手法では，当初に想定した事前条件の考慮不足を検出できるが，個々のシナリオ導出のために論理式を構築し証明を行うコストが必要となる．

各手法を比較した結果を表 2 に示す．要求仕様の記述範囲を明確化するためには，文献**[3][4]**の手法を用いて正常系の常態・イベントを抽出し，抽出結果に文献**[5][6]**の手法を適用して非正常系を扱うことが有効であると考える．文献**[7]**の手法は，実開発への適用を考えた場合，数学的な証明にかかるコストが問題になると考える．

表 2　記述範囲の明確化に関する手法の比較

文献	長所	短所
[3][4]	過去に実績のあるドメインで，要求分析に必要な項目を抽出できる．	新規に開拓するドメインでは項目抽出が困難である．また，非正常系については抽出できない．
[5][6]	予め想定した状態やイベントから連鎖的に発生する非正常系を抽出できる．	当初に想定した状態,イベントに不足があった場合,そこで生じる非正常系を抽出することができない．
[7]	当初に想定した事前条件の考慮不足を検出できる．	個々のシナリオ導出のために論理式を構築して証明を行うコストが必要となる．

2.2 記述形式の明確化

誤りのない仕様を記述する方法として，形式手法を用いた手法の研究が広く行われてきた．形式手法は数学に基づき論理的に仕様を記述･検証する手法である．形式手法には，形式的仕様記述と形式検証がある．形式的仕様記述は，仕様を論理式によって厳密に記述する手法で，自然言語による記述の曖昧さを排除することを可能とする．一方，形式検証は，仕様に誤りがないか論理的に検証する手法である．最も一般的な技術に，モデル検査技術がある．モデル検査技術については，3.1 節で説明する．

本節では，記述内容を明確化する方法として，形式的仕様記述を取り上げる．形式的仕様記述の言語には，VDM-SL，VDM++，RAISE/RSL，Z，B，LOTOS などがあり，このうち VDM-SL と Z は ISO にて標準化されている．

文献**[8]**では，VDM++によって実行可能かつ可読性の高い形式仕様を記述する手法が示されている．この手法では，VDM++の提供する拡張用関数定義の記法に従い，設計者に仕様を伝えるための”仕様伝達部”と仕様を実行するための”仕様動作部”に分離して可読性を高めているが，この分離には関数を区別するための命名規則が必要となる．同著者らの文献**[9][10]**では，形式的記述手法の有効性を示す好適な事例として，モバイル FeliCa IC チップのソフトウェア開発に VDM++を適用し，約 10 万行の形式仕様記述を行った事例が報告されている．

文献**[11]**では，VDM-SL によって仕様を記述する例を示している．この手法では，VDM-SL で記述した仕様から FSP（Finite State Process）モデルを抽出し，モデル検査ツール LTSA**[12][13]**に入力して矛盾した記述がないか検証することを可能としている．仕様記述の構造はプログラムに近く，LTSA による検証には適しているが，文献**[8]**に比べて可読性は低い．

文献**[14]**では，要求仕様を動作の実行列で記述し，FSP モデルによって形式化する手法を示している．当初の要求を動作の実行列に詳細化する方法としてマインドマップを利用し，詳細化の過程を認識しやすくしている．また，FSP モデルで記述した要求仕様を LTSA に入力し，矛盾した記述を抽出することが可能であるとしている．この手法では，動作の実行列を記述することはできるが，システムの構成要素間の相互作用を記述することが難しい．

各手法を比較した結果を表 3 に示す. 文献**[8]**の手法は命名規則を必要とするものの，その記述は容易であり，形式的仕様記述の可読性を高めるという点において他の手法よりも有用であると考える．

表 3　記述形式の明確化に関する手法の比較

文献	長所	短所
[8]	仕様伝達部の内容から仕様を把握できる．	仕様動作部と仕様伝達部の関数を区別するための命名規則が必要となる．
[11]	形式的仕様記述に矛盾した記述がないか網羅的に検証できる．	プログラムに近い記法で，文献**[8]**に比べて可読性が低い．
[14]	当初の要求と形式的記述の関係性が認識しやすい．	システムの構成要素間の相互作用を記述することが難しい．

3. 制御仕様の検証に関する手法

モデル記法を用いて記述した仕様の検証には，形式検証の一つであるモデル検査が広く利用されている．次節で説明するように，モデル検査は有限状態機械の検証に適している．モデルベース開発においても，制御仕様の記述に有限状態機械を用いており，その検証にはモデル検査の適用が効果的であると考えられる．

本章ではモデル検査の概要を説明したのち，モデル検査を用いた検証手法を論じた研究について調査した結果をまとめる．

3.1 モデル検査の概要

モデル検査は，有限状態機械で記述されたシステムの振舞いを網羅的に探索して，システムの満たすべき性質に反する状態遷移が行われるか否かを確認する検証技術である．モデル検査の入力は，有限状態機械で記述したシステムの仕様と，検証すべきシステムの性質を記述した検査項目の二つである．これらをモデル検査ツールに入力すると，システムの取り得る状態遷移パスを網羅的に探索し，検査項目を満たさない状態遷移パスを検出する．モデル検査ツールには SPIN**[15] [16]**, SMV**[17]**, NuSMV**[18]**, UPPAAL**[19]**，LTA などがある．

制御仕様の検証にモデル検査を適用する方法論については，文献**[20][21][34]**に詳しく解説されている．文献**[21]**によれば，モデル検査を適用するうえで重要な考え方は，①アーキテクチャ上の重要性という観点からソフトウェア構造を抽象化すること，②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，③アーキテクチャ上の想定や文脈の多様性に対して網羅的に検証すること，である．検証目的にフォーカスした検証用設計モデルを構築することで，現実的な規模の検証が可能であるとしている．

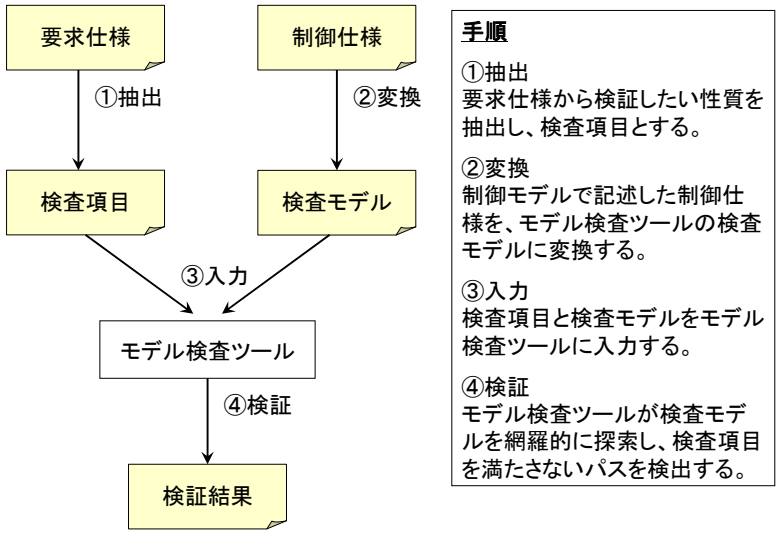

図 2　モデル検査の適用方法

また，文献**[22]**ではモデル検査適用のガイドラインを示しており，モデル検査を実施する目的と効果，モデル検査によって検証できること，モデルや検証式を作成する手順，モデルや検証式の記述テクニックが述べられている．文献**[22]**ではモデル検査ツールに UPPAAL を用いることを前提としているが，モデル検査の考え方や検証プロセスなどは他のツールを用いた場合にも共通の内容であり，モデル検査の導入を検討する上で有用な指針となる．

本論分では，制御モデルで記述した制御仕様が，要求仕様の定める性質を実現しているか否かについて検証するためにモデル検査を適用することを考える（図 2）．

3.2 検査項目の作成

前節で述べたとおり，モデル検査の入力は有限状態機械で記述したシステムの仕様と検査項目である．検査項目を記述する上で困難な点は，時相論理式を用いてシステムの性質を記述しなければならない点である．時相論理式に不慣れな開発者にとって，検査項目の記述は骨の折れる作業となる．

このような問題を解決する方法として，時相論理式のテンプレート化に関する手法が研究されている．文献**[23] [24]**では，有限状態機械の取り得る性質を列挙，分類し，それぞれを時相論理式で記述する方法を示している．システムの性質を Occurence（システム実行時の，特定のイベント／状態を表す性質）と Order（システム実行時の，複数のイベント／状態間の相対的な順序を表す性質）の二つのパターンに分類し，それぞれの項目で想定される性質を列挙し，時相論理式として記述するテンプレートを示している．本手法は，検証すべき性質を論理式として記述する手段を与えるが，要求仕様からどのような性質を抽出して検証するかは開発者が考えなくてはならない．

また，文献**[22]**ではこの手法を参考として UPPAAL の検査項目をパターン化し，時相論理式のテンプレートを示している．

検査項目の作成には，文献**[23] [24]**の手法を用いて検証すべき性質を時相論理式に変換することが有効であるが，検証すべき性質を抽出する方法は課題として残る．

表 4　検査項目の作成に関する手法の比較

文献	長所	短所
[23][24]	汎用的な時相論理式のテンプレートを示しており，検査項目を機械的に生成する手段を与える．	要求仕様からどのような性質を抽出して検証するかは開発者が考える必要がある．
[22]	UPPAAL における検査項目のパターンを示している．	同上．また，他のモデル検査ツールに向けたパターンは示していない．

3.3 検査モデルの作成

モデル検査ツールの多くは，固有の言語で記述した検査モデルを入力とする．例えば SPIN では，PROMELA（Process or Protocol Meta Language）と呼ばれる言語を用いて検査モデルを記述する．PROMELA はC言語に近い文法を持つ言語であり，制御設計の段階で記述する制御モデルとは大きな隔たりがある．この問題を解決する方法として，制御モデルから検査用モデルを生成する手法が研究されている．

文献**[25] [26]**では，UML クラス図，ステートマシン図で記述した仕様を PROMELA の検査モデルに変換する方法が示されている．また，この手法を取り入れた仕様の自動検証装置の開発が紹介されている．同ツールでは，UML モデルで記述した仕様と，UML モデル中のデータ構造と結び付けられた検査項目を入力すると，仕様を PROMELA モデルに変換して SPIN に入力・実行し，その結果を UML の概念に再変換して表示する．本手法では，階層構造を持つステートマシン図の変換方法については示されていない．

文献**[27]**では, UML ステートマシン図で記述した仕様から UPPAAL の検査モデルを自動生成する手法を示している．この手法では，ステートマシン図に記述された状態や状態間遷移の情報を UPPAAL のノード情報にマッピングする規則を設けている．また，階層構造を持つステートマシン図で記述した仕様に対しては，階層構造の変換を容易にするため一度 HUPPAAL モデルに変換してから UPPAAL モデルへ変換する手法を示している．この手法では，階層構造の有無によってことなる変換を行うため，個別の変換モジュールを実装する必要がある．

文献**[28]**では, UML シーケンス図で記述した時間制約を含む仕様を UPPAAL の検査モデルに変換する手法が示されている．時間制約を扱うために時間オートマトンの記述方法を論じているが，有限状態機械から変換する方法については示されていない．

文献**[29]**では UML アクティビティ図で記述した仕様を VDM-SL で形式的に記述し，これを PROMELA の検査モデルに変換する方法を提示している. UML 記法と VDM-SL 記述を組み合わせることでより厳密に仕様を記述する方法を示しているが，有限状態機械から変換する方法については示されていない．

各手法を比較した結果を表 5 に示す．制御モデルの記法を検査モデルに変換する方法としては，文献**[25] [26]**や文献**[27]**の手法を応用できると考える．とりわけ階層構造を持つ有限状態機械に関しては，文献**[27]**の手法が有効であると考える．

3.4 複数のモデル記法で記述した仕様の検証

制御仕様を記述する場合，システムの性質にあわせて複数のモデル記法を使い分けることがある．例えば，システムの個々の構成要素の振舞いを状態遷移図で記述し，構成要素間の通信やイベントの生起順序，システム全体の動作に関する仕様をシーケンス図やアクティビティ図で記述することがある．また，システム構成要素の静的な性質はクラス図で記述することがある．これらは一つの設計対象を異なる観点から捉えて記述したものであり，相互に矛盾してはならない．複数の異なる記述方法を用いてシステムの仕様を記述した場合，それら記述の間で整合性が保たれていることを確認する必要がある．

表 5　検査モデルの作成手法に関する手法の比較

文献	長所	短所
[25][26]	検証結果を UML 概念に再変換し，開発者に理解しやすく表現する．	階層構造を持つステートマシン図の変換方法については示されていない．
[27]	階層構造を持つUML ステートマシン図を UPPAAL 検査モデルに変換できる．	階層構造の有無によってことなる変換を行うため, 個別の変換モジュールを実装する必要がある．
[28]	時間制約を含む検査モデルを記述できる．	有限状態機械から変換する方法が示されていない．
[29]	UML 記法と VDM-SL 記述を組み合わせることで，より厳密に仕様を記述できる．	同上．

このような問題を解決する方法として，複数のモデル記法で記述した仕様の整合性を検証する手法が研究されている．文献**[30]**では，ステートチャートとシーケンスチャートで記述した仕様の整合性を検証する方法を示している．個々のプロセスの状態遷移を監視するオブザーバを利用した手法によって，シーケンスチャートに記述した処理の流れ，イベント発生，時間制約と，ステートチャートに記述した遷移条件の対応が取れているか確認することが出来る．この手法では，検査モデルに時間制約を含んだ場合に状態数が増加しやすい問題がある. また, 検査モデルの記述も煩雑になる．

文献**[31]**では，UML 状態マシン図とシーケンス図で記述した仕様の整合性を検証する手法を示している．整合性の観点として，シーケンス図で記述されたメッセージの順序性が状態マシン図で正しく表されていることを検証している．シーケンス図を基にメッセージの順序性を表す時相論理式を記述し, SMV を用いてモデル検査を行っている．

この他にも, 文献**[32]**では UML シーケンス図とユースケース図で記述した仕様の整合性を, 文献**[33]**では UML アクティビティ図とクラス図で記述した仕様の整合性を検

証する手法を示しているが，制御仕様の記述に用いる有限状態機械については論じていないため，ここでは参考にとどめる．

各手法を比較した結果を表 6 に示す．文献**[30]**で示す手法は時間制約を考慮して整合性を検証できるため，制御モデルの検証に適していると考える．時間制約を含まない検証では，文献**[30]**，文献**[31]**の手法に大きな違いはない．

表 6　複数のモデル記法で記述した仕様の検証に関する手法の比較

文献	長所	短所
[30]	ステートチャートとシーケンスチャートで記述した仕様に対し，時間制約を考慮した整合性検証ができる．	時間制約を含む検証を行う場合は，状態数が増加しやすい．また，検査モデルが煩雑になる．
[31]	比較的単純な検査モデルの記述で，UML 状態マシン図とシーケンス図で記述した仕様の整合性を検証できる．	時間制約を考慮した整合性検証はできない．

3.5 モデル検査の適用事例

これまでに，モデル検査を実プロジェクトに適用した例が報告されている．本節では，それらの適用例を取り上げ，その中で述べられている考察をまとめる．

文献**[35]**では，カーオーディオシステムの組込みソフトウェア開発にモデル検査を適用した事例を示している．UML 状態マシン図をベースとした実現構造モデル，アプリケーションモデル，方式モデルによって仕様を記述し，これを PROMELA モデルに変換し，SPIN を用いて検証している．状態爆発に対する考え方として，①検証の観点を分離して検証内容を絞り込むこと，②モデルの範囲を絞り込むこと，③操作シーケンスの長さを制約して検証の深さを絞り込むことが有効であると論じている．

文献**[36]**では，メイン処理および割り込み処理によって機能を実現する組込みソフトウェア開発にモデル検査を適用した事例を示している．この事例では，モジュールごとに詳細なフロー図が記述された仕様書を基に検査モデルを記述し，NuSMV で検証している．制御ソフトウェアの検証にかかる特徴として，①ソフトウェア仕様のモデル化はフロー図にプログラムカウンタを割り付ける手法によって比較的容易にモデル化可能である一方，②ハードウェア仕様のモデル化はメイン処理と割り込み処理の切り替えや，割り込み禁止区間における割り込み要求の保留などを NuSMV のコードでプログラミングする必要があること，③ハードウェア仕様のモデルについては再利用性が高いと思われること，④割り込み要因や外部要因変化の発生は非決定的動作モデルで記述するが，検査の精度，状態数の大きさ，反例解析のしやすさを考慮して適切にモデル化する必要があること，が挙げられている．

4. 課題の分析

モデルベース開発では，実装レベルの粒度で記述した制御モデルによって制御仕様を記述する．制御モデルの記法には有限状態機械が用いられ，その検証にはモデル検査の適用が効果的であると考えられるが，実際の開発現場でモデル検査を適用するためには次の二つの課題があると考える．

(1)検査項目の作成

検査すべき性質を要求仕様から抽出し，論理式に表すことが困難である．モデル検査によってどのような性質を検証すればよいか，また検査項目を表す論理式をどのように記述すればよいか検討するためには，開発システムのドメイン知識とモデル検査に関する知識を持っている必要がある．

(2)検査モデルの作成

実装レベルの粒度で記述された制御仕様から，モデル検査可能な規模の検査モデルを抽出することが困難である．モデルベース開発で記述する制御モデルは比較的大規模であり，そのままモデル検査を適用すると状態爆発を起こす可能性が高い．

まず，(1)検査項目の作成については「どのような性質を検証すればよいか」という問題と「論理式をどのように記述すればよいか」という問題に分けて考えることができる．

前者の問題は，「要求仕様から検証すべき性質を抽出する手法」と言い換えることができる．解決策の一例として，ドメインごとに検証すべき性質の枠組みを予め定めておき，メタデータを含む形式的仕様記述が与えられると，検証すべき性質の枠組みに従ってメタデータを解析して抽出する仕組みなどが考えられる．例えば文献**[31]**では，通信システムの仕様の整合性を検証するために，メッセージの順序性が複数のモデルで矛盾せずに成り立つことを確かめているが，これは通信システムというドメインにおいてメッセージの順序性が検証すべき性質の一つであることを示唆している．検証すべき性質の枠組みを考える上では，過去の開発事例や故障事例から検証すべき性質を洗い出すことが効果的であると考える．

後者の問題は 3.2 節で論じた問題に他ならない．検証すべき性質を抽出することができれば，文献**[23] [24]**の手法を用いてこれを時相論理式に変換することができる．

(2)の課題については，文献**[35]**で状態爆発の問題として論じられている．3.5 節で述べたとおり，モデルの観点を絞り込むこと，モデルの範囲を絞り込むこと，検査の深さを絞り込むことが重要であるとしている．仕様の抽象化に関しては，文献**[21]**でも論じられており，アーキテクチャ上の重要性という観点から仕様を抽象化することが必要であるとしている．文献**[36]**では，抽象化の具体的な方法が示されている．仕様レベルの抽象化として，変数の取捨選択，変数の定義域の有限化，変数の縮約，モジュールの取捨選択を行い,モデルレベルの抽象化として,プログラムカウンタの削減，モデルの単位時間の調整，非決定的動作の削減，割り込み可能箇所の削減を行っている．また，文献**[22]**では状態数を削減する方法として，一時的な値を持つ変数は使用しない間常に同じ値（0 など）にしておくこと，検証の意味を失わない範囲で簡単なシナリオを変更すること，モデルの範囲を絞り込むことを挙げている．このように，モデルの抽象化や検証範囲の絞り込みには，モデル検査の知識をベースとした細かな調整作業が必要である．文献**[35][21]**に示される抽象化・絞込みの方針に基づいてノウハウを蓄積し，体系化することで，モデル検査で検証可能な規模の検査モデルを容易に記述できるようになると考える．

それぞれの項目について分析した結果を表 7 にまとめる．

表 7　課題の分析結果

課題	従来手法の適用可能性	解決すべき課題
(1)検査項目の作成	検証すべき性質を抽出できれば，文献**[23] [24]**の手法を用いて時相論理式に変換することができる．	要求仕様から検証すべき性質を抽出するための枠組みが必要である．
(2)検査モデルの作成	文献**[36][22]**に示されるモデルの抽象化，検証範囲の絞り込みの手法によって，状態爆発しない規模の検査モデルを作成できる．	文献**[35][21]**に示される抽象化・絞込みの方針に基づいてノウハウを蓄積し，体系化することが必要である．

5. おわりに

本論文では，モデルベース開発における要求分析と制御設計のプロセスに焦点をあて,各プロセスで仕様を正しく記述するための手法について関連研究の調査を行った．

まず，要求仕様の記述から漏れや曖昧さを排除する方法として，要求仕様に何を記述すべきか，要求仕様をどのように表現すべきかの二点について調査した．前者に関しては，過去の成功事例を参照して要求仕様の枠組みを形式化する手法や，要求分析の段階で見落としがちである非正常系を抽出する手法によって，要求仕様の記述範囲を明確化できることが分かった．後者に関しては，VDM++や VDM-SL といった形式的仕様記述の記法に則ることで，曖昧さを排除した要求仕様を記述できることが分かった．

次に，制御仕様が要求仕様の性質を実現しているか検証する方法として，モデル検査を用いた検証方法を調査した．モデルベース開発では有限状態機械で制御仕様を記述するので，その検証にはモデル検査の適用が効果的であると考えるためである．モデル検査は検査項目と検査モデルの二つを入力とするが，それぞれ固有の記法を用いて記述するため，開発者にとって扱いづらいものとなっている．そこで，検査項目と検査モデルを効率的に作成する手法を調査した．検査項目については，パターン化されたシステムの性質に対して時相論理式のテンプレートを与える手法がある．この手法を用いれば，時相論理式に関する知識を必要とせずに検査項目を作成することが可能となる．検査モデルについては，UML などで記述された仕様を検査モデルに変換する手法がある．この手法を応用して適用することで，制御モデルから機械的に検査モデルを作成できると考える．

これらの調査結果をもとに，今後の課題を次のように分析した．

まず，検査項目の作成については，要求仕様から検証すべき性質を抽出する方法を明らかにすることが必要であり，このための枠組みを定めることが今後の課題であると考える．検証すべき性質を抽出することができれば，これを時相論理式のテンプレートに当てはめることで，モデル検査に用いる時相論理式が得られる．

次に，検査モデルの作成については，モデルの抽象化や検証範囲の絞り込みによって状態爆発の問題に対処する必要がある．抽象化・絞り込みは，検証の範囲においてモデルが持つ意味を損なわないように行う必要がある．今回の調査によって，これらの実施にはモデル検査の知識をベースとした細かな調整作業が必要であることが分かった．当面の課題として，抽象化・絞り込みに関するノウハウを蓄積し，体系化することが重要であると考える．

参考文献

[1] 経済産業省 商務情報政策局 情報処理振興課: 高度情報化社会を見据えた情報システム・ソフトウェアの信頼性向上に向けて, 2009.
[2] 佐藤弘明: トヨタ,日産,ホンダは ECU を何個使っているのか，日経エレクトロニクス 2010 年 12 月 27 日号, 2010, pp.175-186.

[3] 北川貴之, 橋本憲幸, 吉田和樹, 位野木万里: 要求定義における暗黙知の形式知化手法, コンピュータソフトウェア, Vol.27, No.3 (2010), pp.93-98.
[4] 位野木万里, 松尾尚典, 甲田修策: メタモデルに基づき仕様書作成と仕様検証を支援するツール SpecPrince, 東芝レビュー, Vol.63, No.12 (2008), pp.46-49.
[5] 三瀬敏朗, 新屋敷泰史, 橋本正明, 鵜林尚靖, 片峰恵一, 中谷多哉子: 組込みソフトウェア仕様抽出のための非正常系分析マトリクス, 情報処理学会 研究報告, 2004-SE-145 (16), 2004, 00.113-120.
[6] 三瀬敏朗, 新屋敷泰史, 橋本正明, 鵜林尚靖, 片峰恵一, 中谷多哉子, 組込みソフト非正常系における基礎モデル仕様分析手法の提案, 情報処理学会 研究報告, 2005-SE-147 (11), 2005, pp.81-88.
[7] 熊澤努, 玉井哲雄: ユースケースによる安全性要求分析のための想定外シナリオ抽出法の提案, 情報処理学会 研究報告, 2007-SE-155 (25), 2007, pp.191-198.
[8] 中津川泰正, 栗田太郎, 荒木啓二郎: 実行可能性と可読性を考慮した形式仕様記述スタイル, コンピュータソフトウェア, Vol.27, No.2 (2010), pp.130-135.
[9] 栗田太郎, 中津川泰正, 太田豊一: 携帯電話組み込み用“モバイル FeliCa IC チップ”開発における形式仕様記述手法の適用とその効果, ソフトウェア・シンポジウム 2006 論文集, 2006, pp.46-66.
[10] 栗田太郎: モバイル FeliCa のソフトウェア開発における品質確保のための構造と実践　抽象度の制御やコミュニケーションの活性化に向けて, 情報処理学会デジタルプラクティス, Vol.1, No.3 (2010), pp.148-157.
[11] 三好健吾, 日下部茂, 荒木啓二郎: データ型に着目した形式仕様記述からの状態遷移系の抽出, コンピュータソフトウェア, Vol.23, No.2 (2006), pp.211-224.
[12] Concurrency - State Models & Java Programs, http://www.doc.ic.ac.uk/~jnm/book/
[13] Jeff Magee, Jeff Kramer: Concurrency - State Models & Java Programs -, Wiley, 1999.
[14] 神田未来, 藤倉俊幸: 動作要求のモデル化と管理, 情報処理学会 研究報告, 2008-SE-160 (14), 2008, pp.101-106.
[15] Gerard J.Holzmann: The Model Checker SPIN, IEEE trans. SE, Vol.23, No.5, 1997, pp.279-295.
[16] Spin – Formal Verification, http://spinroot.com/spin/whatispin.html
[17] The SMV System, http://www.cs.cmu.edu/~modelcheck/smv.html
[18] NuSMV home page, http://nusmv.fbk.eu/
[19] UPAAL Home, http://www.it.uu.se/research/group/darts/uppaal/
[20] 中島震: モデル検査法のソフトウェアデザイン検証への適用, コンピュータソフトウェア, Vol.23, No.2 (2006), pp.72-86.
[21] 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 研究報告, 2005-SE-150 (2), 2005, pp.9-16.
[22] 青木翼, 長谷川哲夫, 宮元暢, 渡邊竜明: UPPAAL によるモデル検査適用ガイドラインの作成, 情報処理学会 研究報告, 2008-SE-159 (26), pp.203-210.
[23] Matthew B. Dwyer, George S. Avrunin, James C. Corbett: Property Specification Patterns for Finite-State Verification, Proceedings of the Second Workshop on Formal Methods in Software Practice (2008)
[24] About Specification Patterns, http://patterns.projects.cis.ksu.edu/
[25] 岸知二, 青木利晃, 中島震, 野田夏子, 片山卓也: プロジェクト紹介：高信頼組込み用オブジェクト指向設計技術, 情報処理学会 研究報告, 2004-SE-146 (7), pp.47-54.
[26] 岸知二: モデル検査技術による UML 設計検証, 情報処理, Vol.47, No.5 (2006), pp.498-505.
[27] 和田昭彦, 深海悟: 形式的検証用モデル自動生成機能を持つ上流工程支援システムの開発, 情報処理学会 研究報告, 2008-SE-159 (25), 2008, pp.195-201.
[28] 野村将人, 新川芳行: 時間制約が存在する UML の正当性検証, 信学技報, SWIM2010-7 (2010-6), pp.45-50.
[29] 井上貴至, 新川芳行: モデル検査による UML アクティビティ図の正当性検証, 信学技報, SWIM2008-19 (2008-11), pp.19-24.
[30] 長谷川哲夫, 深澤良彰: モデル検査によるステートチャートとシーケンスチャートの整合性検証, 信学技報, AI2006-19 (2006-11), pp.41-46.
[31] 宮崎仁, 瀬古剛一, 横川智教, 佐藤洋一郎, 早瀬道芳: 記号モデル検査を用いた状態マシン図とシーケンス図の無矛盾性の検証, 情報処理学会 研究報告, 2008-SE-161 (6), 2008, pp.41-47.
[32] 高谷彰俊, 新川芳行: ユースケースーシーケンスモデル間の整合性検証, 信学技報, SWIM2008-18 (2008-11), pp.13-18.
[33] 堂西祐一, 新川芳行: UML アクティビティ図ークラス図間の整合性検証, 信学技報, SWIM2010-6 (2010-6), pp.39-pp.44.
[34] 岸知二, 高橋弘, 徳田寛和: モデル検査技術を活用したソフトウェア設計・検証手法に関する考察, 情報処理学会 研究報告, 2008-SE-160 (13), pp.95-100.
[35] 岸知二, 金井勇人: 組込みソフトウェア設計検証へのモデル検査技術の適用と考察, SEC journal, No.12 (2008), pp.10-19.
[36] 水口大知, 渡邊宏: 組み込みソフトウェア開発におけるモデル検査の適用事例, コンピュータソフトウェア, Vol.22, No.1 (2005), pp.77-89.