Vol.2010-SE-168 No.9
Vol.2010-EMB-17 No.9
2010/6/1

モデル検査のための設計モデル構築手法に関する考察

岸 知二†

モデル検査技術によるソフトウェアの設計検証は有効な検証手段のひとつであるが、効果的に利用するためには注意深い適用が必要である。本稿ではモデル検査技術を行うためは設計モデル構築上どのような課題があるかを考察するとともに、それを踏まえた検証のための設計モデル構築手法について検討する。

On Modeling Method for Software Design Verification using Model Checking Techniques

Tomoji Kishi†

Model checking techn iques are consider ed to be promising approaches for softwar e design verification, but we need careful application of the techniques in order to verify software des ign effectively. In thi s p aper, we examine th e is sues in sof tware des ign modeling, and propose a verification oriented design modeling techniques.

1. はじめに

ソフトウェアに対する信頼性要求の高まりの中、モデル検査技術[4][10]などの形式手法の適用に対する期待が高まっており、様々な研究や実務での適用が進められている。我々も、組込みソフトウェアにおけるタスクの動作のタイミングに関わる性質など、通常のレビューやテストでは考慮漏れを起こしたり網羅的な確認が難しかったりする性質の確認に有効かつ強力な検証手段であると考えている。

しかしながら、モデル検査技術による検証はテストやレビューとは異なった特性を持っており、効果的な検証のためにはそうした特性を踏まえた注意深い適用が必要である。例えば検証対象となる設計やそれが満たすべき性質が、検証にとって必要十分な詳細度や厳密性を持って定義されていないと検証できない。あるいはプログラミング言語の記述を単純なマッピングでモデル検査ツールの理解できる言語記述に変換したとしても､実行意味や実行環境が異なるので意味のある検証にならないことも多い。

我々はモデル検査技術によるソフトウェアの設計検証を対象に、ソフトウェア技術者にとってより親和性が高く、かつ効果的な適用が行える手法や環境について検討してきた[1][3][6][7][8][9][11]。本稿では、それらの検討を踏まえ、検証を効果的に行うための設計モデル構築手法について考察する。ここで設計モデルとは、モデル検査技術による設計検証を行う際に、検証対象となる設計を表現するモデルを指す。従来的な設計書等を手掛かりに、いきなりモデル検査ツールの理解できる言語記述で設計を記述する方法もあり得るが、本稿で議論するように、我々はそうした適用は現実には多くの問題をはらんでいると考えており、そうした従来的な設計書等とは別に、検証を目的とした設計モデルを構築することが有効であると考えている。なお以降、特に断らない限り、設計検証とはモデル検査技術による設計検証を意味するものとする。

2 章では、設計検証のための設計モデルの構築に関わる課題を述べる。3 章では、設計検証のための設計モデルの構築手法について検討する。4 章では、関連する議論を行う。

2. 設計検証とモデリング

本章では、設計検証のための設計モデリングに関する課題について検討する。

2.1 設計モデルとは

設計検証を行うためには、検証対象となる設計と、その設計意図（仕様、上位設計、設計制約など）とをモデル検査技術の世界にマッピングする必要がある。図 1 の(a)は、ソフトウェアの世界での設計意図と設計とが、モデル検査技術の世界の性質記述(P)と検査対象モデル(M)に対応づけられることを模式的に示している。

† 早稲田大学
W aseda University

Vol.2010-SE-168 No.9
Vol.2010-EMB-17 No.9
2010/6/1

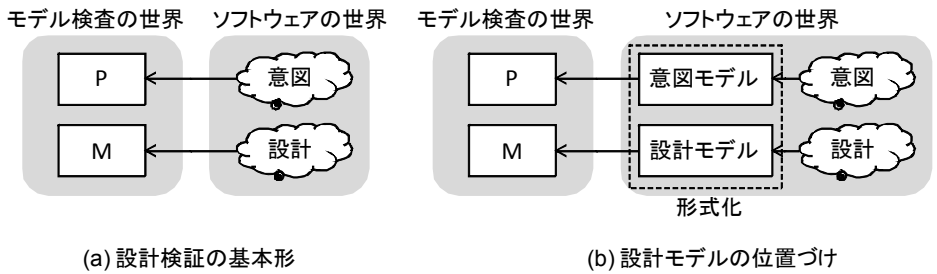

図 1　設計検証と設計モデル

しかしながら意図や設計をいきなりモデル検査技術の世界のPやMとして表現することは稀であり、通常はソフトウェアの世界で一旦それらを形式化（モデル化）する。本稿ではソフトウェアの世界で意図を形式的に記述したものを意図モデル、ソフトウェアの世界で設計を形式的に記述したものを設計モデルと呼ぶ。従来的な開発では前者は仕様書や上位の設計書に相当し、後者は対象となる設計書に相当する。

2.2 課題

我々は過去にいくつかの設計検証の適用を行ってきた経験[7][8]から、設計モデルの構築に関して、以下のような観点からの課題があると考えている。

2.2.1 モデルの位置づけ

検証を行うには、意図モデルと設計モデルの関係が明確になっている必要がある。

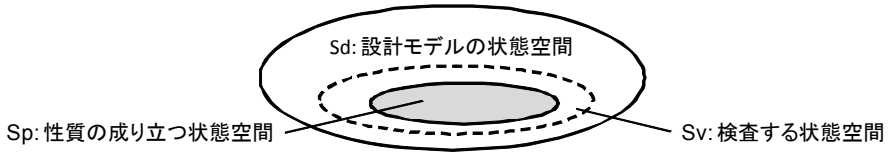

図 2　状態空間の構造

図 2 は説明のための模式図である。設計モデルによって規定される状態空間(Sd)があり、特定の性質はその中の一部(Sp)で成り立つことが意図され（Sp=Sd の場合や Sp が空の場合も含む）、それを検証するために Sd 中のある範囲(Sv)を検査して、上記の意図を確認する。一般に設計モデルは意図モデルよりも詳細な決定を含んでいるため、設計モデル上起こりうる状態を網羅検査すると、意図モデルでは意識していなかった状態が含まれうる。例えば意図モデルでは状態 S1 で性質が成立し、そこから状態 S2 に遷移すると性質が成立しないと定義されているとする。しかし設計上は S1 に対応する内部状態から S2 に対応する内部状態に 1 ステップで遷移するとは限らず、遷移中という状態が存在するかもしれない。この遷移中のときに性質が成り立てばよいのか、成り立ってはならないのか、気にしないのか、そういう立場を明確にしないと適切な検証ができない。こうした問題の具体例については、過去の報告[8]を参照されたい。

2.2.2 意図モデルの記述

モデル検査では意図モデルから導出された性質 P が成立すべき範囲 Sp を網羅検査することで検証を行う。したがって、意図モデル中でその性質がどういう条件下で成立すべきものなのかが明確に定義されていなければならない。

意図モデルは従来の仕様書や上位の設計書に対応するが、こうした上位の文書は一般に散文的あるいは断片的な記述であることが多く、必ずしもこうした条件が網羅的かつ明確に記述されないことが多い。また多くの暗黙理の制約を踏まえて書かれていることも多い。モデル検査エンジンは明示的に定義されていない条件を理解することは不可能であるため、意図の中ではどういう要因までを考慮するのか、その要因によって決定されるどのような条件下で性質が成立するのかを明示的に定義する必要がある。

2.2.3 設計モデルの記述

ソフトウェアの設計はあるサービスを実現するための構造であるが、設計検証はその構造上の性質を確認することが目的であるから、M は単にサービスの実現方法を示すだけでなく、その性質を確認しやすい構造になっていることが望まれる。例えば、性質 P の定義に用いられている概念（上位設計における状態など）が M 上において捕捉しやすいことが望まれる。あるいは外界からのイベントに応じた性質の違いを検証したい場合には外界に対応するモデルを作成することで検証が容易になる事もある。また検証目的とする性質に関わらない設計上の詳細を M から取り除くことで、モデルをシンプルにすることができ、モデルがより分かりやすくなり、検証上の状態を減らすことができることもある。すなわち M は設計そのものではなく、検証性質や検証方針に応じた適切な抽象であるべきである。したがって様々な性質の検証をしたい場合には、それぞれに応じて異なった M を用意する必要があるかもしれない。

3. 検証用設計モデル構築手法

前章で検討した課題を踏まえ、設計検証を指向した設計モデル構築手法について提案する。まず 3.1 でその全体像を説明した後、3.2 以降で、その詳細について述べる。ここでは設計モデルだけではなく、意図モデル等、検証に必要な情報などを含めた議

Vol.2010-SE-168 No.9
Vol.2010-EMB-17 No.9
2010/6/1

論を行う。

3.1 概要

設計検証のための意図モデルや設計モデルは、設計検証に適した形での抽象がなされ、かつモデル検査に必要十分な詳細度や厳密性を持ったものでなければならない。設計全体を詳細かつ厳密にモデル化することは一般には困難であるため、検証したい性質を明確にし、その性質の検証に必要な側面だけにフォーカスすることが重要であると考える。

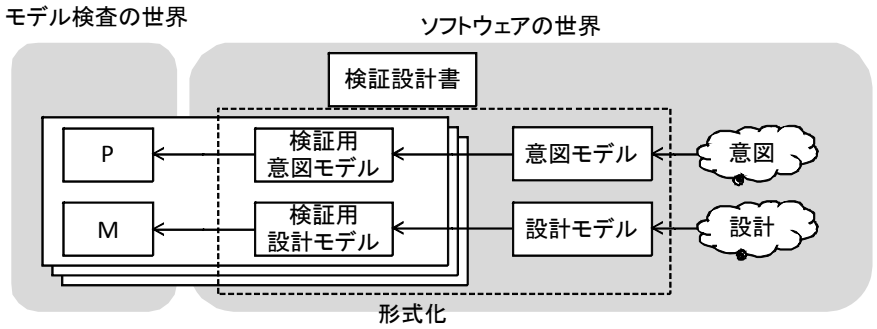

図 3　構築手法の概要

図 3 は、提案する構築手法の概要を示すものである。

- 意図モデルは通常の開発で構築される仕様や上位設計のモデル、設計モデルは通常の開発で構築される設計のモデルである。これらは実装へとつながる開発の流れの中で、従来的な使われ方をする。
- 検証設計書は、どのような目的でどのような性質に関わる設計検証を行うかを定義するものである。これ自体はモデルではないが、これを明示的に定義することで、次で説明する検証用意図モデルや検証用設計モデルの構築を容易にできると考えている。
- 検証用意図モデルや検証用設計モデルは、意図モデルや設計モデルに記述されている内容の中から、検証設計書に示される設計検証に必要な情報のみにフォーカスするとともに、設計検証に必要な詳細度と厳密性を持って記述されるモデルである。検証用意図モデル、検証用設計モデル、ならびに P や M は検証目的が異なれば違ったものになりうるが、現実には後述するように類似した検証性質ごとにグループ化できると考えている。なお検証用設計モデルは M と同等の構造を持たせ、M の作成をできるだけ平易にすることが望ましいと考える。

3.2 検証設計

検証計画書は、通常の意図モデルや設計モデルから、検証用意図モデルや検証用設計モデルを構築するための指針を与えるものである。具体的には、どのような性質を検証したいのか、その性質がどのような条件化で成立するのか、その性質が成り立つ際の想定は何なのかといった検証の目的や指針を示すものである。

- 検証性質：設計検証したい性質の記述を行う。この際、検証したい性質が以下のどちらであるかを明確にする。両方の概念の混在は検証の理解を困難にすると考えられる。
  - 仕様や上位設計など、検証対象となる設計よりも上位の概念に関わる性質。仕様通りのふるまいになる、上位設計の意図通りの動作になる等。
  - 検証対象となる設計上の概念に関わる性質。設計上導入したバッファに関しオーバーフローがないか、タスクにデッドロックがないか等。
- 性質が成り立つ条件：上記の検証性質が、どういう条件の時に成り立つことを検証したいのかを記述する。この際、条件が以下のどちらであるかを明確にする。上記同様、概念の混在は検証理解を困難にすると考える。
  - 仕様や上位設計など、検証対象となる設計よりも上位の概念に関わる条件。仕様に定義されるイベントや操作の系列、上位設計で規定されるモードで定義される条件等。
  - 検証対象となる設計上の概念に関わる性質。設計上導入した入力ポートへのイベントの系列、内部変数の値などによって定義される条件等。
- 性質が成り立つ際の想定：広義には条件に含まれるが、明示的に条件とは考えていないが、暗黙裡に想定されている事項。処理途中で電源スイッチを切ることは考えないとか、コンポーネントの故障は考えないといった、性質確認の範囲を規定するものである。検証上は条件と同様に扱われるため、仕様や上位設計の概念で定義される想定か、検証対象となる設計上の概念で説明される想定かが混在せず、かつ条件の定義と同じであることが望ましいと考える。こうした検討には想定モデリング[1][5]などの活用が有効と考える。
- 検証のスキーム：設計検証では、上記の検証性質が、定義した条件や想定下で成立するかどうかを確認するが、その条件や想定下以外で検証性質が成立するかどうかに関心がない場合と、成立してはならない場合とで、検証の内容が以下のように変わる（図 2 参照）。
  - 条件・想定下での性質成立のみを確認すればよい場合：条件の成立する範囲（Sp）のみを検査して性質が成り立つことを言えばよい。
  - 条件・想定下のみで性質が成立することを確認する場合：上記に加えて、条件・想定が成立しない範囲を検査範囲とし、そこで性質が成り立たないことも確認する必要がある。なお条件・想定が仕様や上位設計の概念で定

Vol.2010-SE-168 No.9
Vol.2010-EMB-17 No.9
2010/6/1

義されている場合には､ふたつの確認での検査範囲の和は Sd より小さいが、設計上の概念で定義されている場合には検査範囲の和が Sd と等しくなる。

3.3 検証用意図モデル

設計検証に必要な仕様や上位設計を定義する。ここでは通常の意図モデルから、検証設計書に示された概念に関わる部分のみを抜き出し、その部分に関しては正確な定義を行う。

検証用意図モデルの定義手段としては、例えば以下のような方法が考えられる。

- シーケンス図の活用。フラグメントを利用することで、並行性や正規表現的な記述が可能となるので、例えば外部からのイベント系列に関する条件などの記述に活用できる。
- ステートマシン図の活用。仕様や上位設計で規定されるふるまいを、ステートマシン図を活用して一般的に記述する。
- OCL の活用。不変条件などに関しては OCL で宣言的に記述する。

3.4 検証用設計モデル

設計検証に必要な検証用設計モデルを定義する。検証用設計モデルは、設計モデルから、検証側面に関わる部分にフォーカスして詳細化・厳密化したものである。

3.4.1 検証のアーキテクチャ

検証用設計モデルは、検証対象になる設計の構造を表すものであるが、さらに検証を容易にするための構造上の工夫を行うことも必要となる。図 4 はそうした構造の一例である。左は外部環境を表現するアクタを作り、特定の外部イベントを発生させ、その条件下での性質を確認する構造であり、右は設計上のふるまいを観測するモニタを作り仕様上でのふるまいを解釈する構造である[8]。またリアルタイム OS など実際の設計のプラットフォームを抽象化した機能を利用して検証を行うことも考えられる[1]。

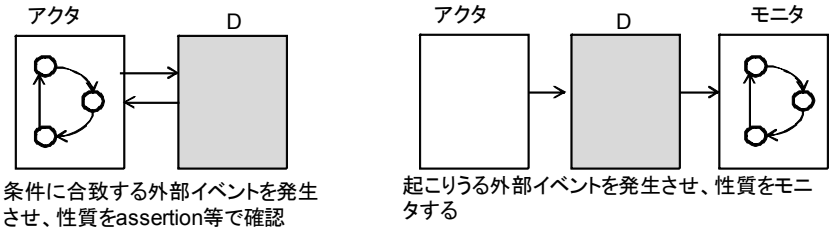

図 4 検証用設計モデルの構造例

図 5 はこうした検証用設計モデルの構造を考える際のリファレンスである。

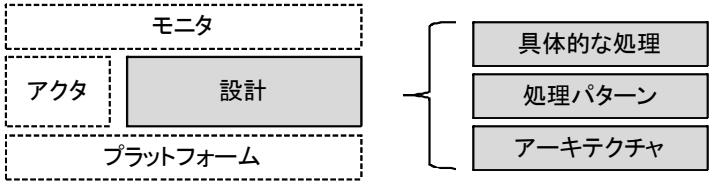

図 5　検証用設計モデルのリファレンス

図の左は、設計モデルの全体構造のリファレンスである（右については 3.4.2 で説明する）。

- 設計：検証対象となる設計に対応する部分。
- アクタ（オプショナル）：設計対象に対する外部環境であり、設計対象に対してイベントを与えたり、設計対象からのイベントを受け取ったりする部分。
- モニタ（オプショナル）：設計対象上のふるまいを、仕様や上位設計上のふるまいとして観測する仕組み。例えば設計上のふるまいを仕様や上位設計上のふるまいに解釈する機構を作ることも考えられるし、assertion などによって実現することも考えられる。
- プラットフォーム（オプショナル）：実際の設計が利用しているプラットフォームに対応する構造。設計対象を実際の設計に近い形で実現するために利用される。前述したリアルタイム OS のふるまいを抽象化した機能等に相当する。

検証性質が仕様や上位設計の概念で定義される場合、モニタ部分が必要となる。条件や想定が外部環境とのやりとりで規定される場合には、アクタ部分を作成すると便利にモデル化や検証ができる場合がある。条件や想定がプラットフォームに関わるときは、プラットフォーム部分を作成すると便利にモデル化できる場合がある。このように、検証計画書を踏まえ、適切な検証用設計モデルの構造を決めることが重要である。

3.4.2 パターン化

図 5 左の設計の部分は、検証対象となる設計を表すものであるが、通常の設計モデルには検証性質と無関係な情報が含まれているため、検証性質にとって関係のある部分にフォーカスする必要がある。上述のプラットフォームを利用したとしても、現実のソフトウェアの実行環境を忠実に再現することはあり得ないため、通常の設計をそのままモデル化しても、意図したふるまいの検証にならない危険性が高い。特に詳細設計に近いレベルまでモデル化するとモデルの規模が大きくなり、そもそもその設計モデルが妥当なものかどうかの確認が困難になる。したがって、通常の設計をリアルになぞることよりも、重要な方式やアーキテクチャの性質を確認するという目的で、

Vol.2010-SE-168 No.9
Vol.2010-EMB-17 No.9
2010/6/1

通常のソフトウェア開発者が理解できる適切な大きさのモデルを作ることが重要と考える。

そうしたモデル化の一つの方法として、個々の機能を詳細に記述するのではなく、処理のパターンに注目し、そうしたパターンにあてはまる処理から代表元を取り出す形で確認をすることが有効と考える。例えば複数のイベント処理があり、個々のイベントごとに具体的な処理の内容は違っていても、ふるまいやタイミングの面からは類似した処理パターンを持つのであれば､すべてのイベント処理を確認するのではなく、その代表元に関するモデル化を行うことが現実的である[7]。もちろん、複数の処理の競合などを確認したいのであれば、方式に応じて例えば 2 つあるいは 3 つの処理までをモデル化するなど、目的に応じた対応が必要である。

図 5 の右は上記をリファレンス的に表したものである。

- アーキテクチャ：設計の骨格構造であり、主要なソフトウェア構成要素と、その間のメッセージの処理手段など全体を支配する基本的メカニズムを実現する部分である。
- 処理パターン：ある処理を行う際の方式や処理構造を意味する。一般には複数の処理が同じパターンを持つことが多いため、そうした処理パターンに注目し、それを実現する部分である。これは上記のアーキテクチャ上に構築される。
- 具体的な処理 : 処理パターンを代表する特定の処理の詳細を含んだ部分である。上記のアーキテクチャならびに処理パターン上で実現される。

検証用設計モデル構築においては、設計上のどういうレベルの構造を、どの範囲まで検証するのかという目的の明確化が重要と考える。こうしたリファレンスは、設計において何を決定し何を検証したいのかを明確に意識することの支援になると考える。

3.4.3 アスペクト指向技術の活用

検証用設計モデルの構築には、アスペクト指向技術の適用が有効である場合があると考えており、今までに、ジョインポイントモデルに基づくアスペクト指向でのモデル検査用の言語や UML モデリングを活用した検証の試行を行ってきた[3][11]。アスペクト指向技術の適用が有効と考える局面として、例えば以下のような状況があると考えている。

- 検証性質に応じたモデルの変更：同一の設計構造に対して複数の性質を検証する際には、対象となる設計構造は同一でも、検証を行うための構造部分に変更が必要となることがある。例えばアクタやモニタを変更したり、異なった assertion を挿入したりする状況が考えられる。
- 類似した構造をもった処理の検証：設計の一部を変更してみて性質の確認を行うことがある。例えば通信の方式、バッファサイズ、優先度の変更などを変更して性質を確認する状況が考えられる。
- 処理パターンに属する特定のふるまいの検証：処理パターンとして定義された共通的な構造に対して、特定のふるまい依存の詳細（コマンドや条件判断の真偽など）を付加して性質を確認する状況が考えられる。

こうした状況では、ベースとなる構造を用意し、それに対して目的に応じてアスペクトとして定義した変更部分をウィーブすることによって､モデルを変更することで、繰り返し様々な性質や構造に対する検証を行う作業が容易になるとともに、不用意な作業ミスを防ぐことが期待される。検証用設計モデルは検証性質に関わる部分のみを含んだコンパクトなモデルとすることが妥当性確認や状態数の面から有利であり、必要な部分を必要に応じてウィーブして検証を行うアスペクト指向技術の適用は有効性が高いと考えている。

3.5 構築の手順

以上の構築手順をまとめたのが図 6 である。ここでは典型的な構築手順と、そこで定義される情報を示している。図中の波線矢印は、情報が意味的に影響を及ぼす先を示している。なお意図モデル（仕様書や上位設計書）や設計モデル（設計書）は従来的な作業の中で作られるものと考える。なお、図中 1, 2, 3 で定義される情報が検証計画書に含まれる。

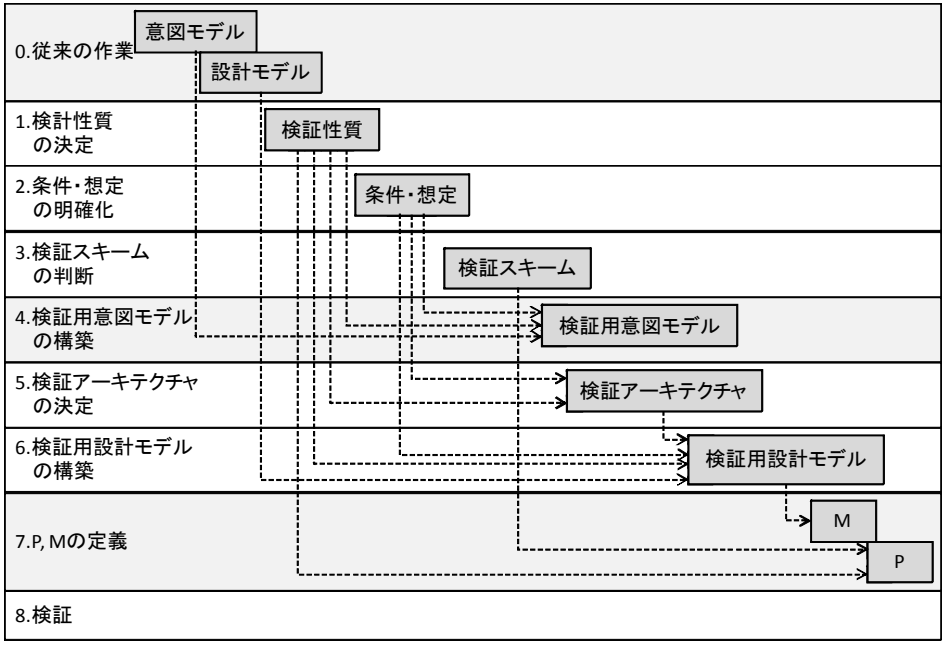

図 6　構築手順

1. 検証性質の決定：検証したい性質を定義する。検証性質に出てくる概念は、検証用意図モデルや検証用設計モデルを記述する概念粒度を決定する。
2. 条件・想定の明確化：検証性質が成り立つ条件や想定を明確化し定義する。条件・想定に出てくる概念は、検証用意図モデルや検証用設計モデルを記述する概念粒度を定義する。またどのような条件・想定下での検証性質の確認をするかということが、検証のアーキテクチャの決定に影響を持つ。
3. 検証スキームの判断：条件・想定下で検証性質が成立することのみを確認するのか、条件・想定下でのみで検証性質が成立することを確認するのか、といったことで性質の定義や検証の方法が変わる。
4. 検証用意図モデルの構築：検証用意図モデルは、意図モデルをベースに、検証性質、条件・想定に関わる部分にフォーカスし、そこで必要十分な概念粒度を用いて定義される。
5. 検証アーキテクチャの決定：検証性質、条件・想定に応じた検証アーキテクチャを決定する。
6. 検証用設計モデルの構築：検証用設計モデルの設計部分は設計モデルをベースに、検証性質、条件・想定に関わる部分にフォーカスし、そこで必要十分な概念粒度を用い、検証アーキテクチャに沿って定義される。
7. P, M の定義：検証性質に基づき P を、検証用設計モデルに基づき M を定義する。また検証スキームによっては、条件・想定下以外では検証性質が成り立たないことを確認するための P を定義する。
8. 検証：P, M に基づき実際の設計検証を行う。検証に当たっては、まず自明な性質を M 上で検証するなどして M の妥当性を確認することが望ましいと考える。

4. 議論

設計検証とひとくちに言っても、様々なコンテキストで様々な目的で行われうる。例えば現状のソフトウェアの構造をリバースしてその検証を行うような状況もあるが、本稿では設計された方式やアーキテクチャが意図した性質を持っているかどうかを確認するというコンテキストを想定して議論を進めた。こうした状況では、その設計が持つ重要なポイントを明示的に意識することが不可欠である。定義と検証はペアとなる行為であり、検証対象や目的の明確化は、よい設計へとつながると考えている。

モデル検査技術による設計検証は強力な検証手段であると考えているが、検証用設計モデルの妥当性の確認は困難な作業である。したがって検証用設計モデルは、必要十分な複雑さに抑えられることが重要である。そうした際に、処理パターンに注目し、類似した処理についてはいくつかの代表元のみを検証用設計モデルに反映させる方法が状態数の面からも有効と考える[7]。

我々はアスペクト指向技術が設計検証に有効に適用できると考えており、検証のためのアスペクト指向言語やモデルの研究を進めてきた[3][11]。検証用設計モデルは。検証性質に必要十分な内容のみを含むことが有利ではあるが、一般に様々な検証性質を検証する必要があるため、それらに対応した検証用設計モデルをアドホックに構築することは非効率かつ間違いが入りやすい。そうした局面ではアスペクト指向技術の適用が有用であると考えている。

5. おわりに

本稿では、モデル検査技術による設計検証を対象に、検証のための設計モデル構築手法について、検討を行った。この検討は、我々の過去の設計検証の実験、試行、適用経験からの観測に基づいたものであるが、全体としてはまだ検討過程のものである。今後さらに多面的な検討や評価を行いリファインしていきたい。

参考文献

[1] 青木利晃, 片山卓也: RTOS に基づいたソフトウェアのためのモデル検査ライブラリ, 組込みソフトウェアシンポジウム 2005, pp55-63, 2005.
[2] 朝倉功太, 岸知二: 想定モデリングに基づくソフトウェアプロダクトラインのコア資産検証手法, 情報処理学会 組込みシステムシンポジウム 2009, pp169-177, 2009.
[3] 大野真一朗, 岸知二: モデル検査のためのアスペクト指向でのモデル記述支援環境, 情報処理学会ソフトウェア工学研究会, Vol2008, No.29, 2008.
[4] Clarke, E., Grumberg, O., Peled, D.: Model Checking: MIT (1999).
[5] 岸知二, 野田夏子: プロダクトライン開発のための想定モデリング, 情報処理学会組込みシステム研究グループ合同研究会, pp39-46, 2006.
[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, 情報処理学会 組込みソフトウェアシンポジウム 2006, pp50-57, 2006.
[7] 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, IPA/SEC, SEC Journal 12 号, (2007).
[8] 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, 情報処理学会ソフトウェア工学研究会, Vol2008, No.55.
[9] 岸知二: ソフトウェア設計・検証手法に関する考察～モデリングの観点より～, 情報処理学会ソフトウェア工学研究会, Vol2009-SE-165, No.5.
[10] Holzmann, G.J.: The SPIN Model Checker - Primer and Reference Manual, Addison-Wesley (2004).
[11] 金井勇人, 岸知二: モデル検査のためのアスペクト指向メカニズムの切り替え手法の提案, 情報処理学会ソフトウェア工学研究会, Vol2008, No.93, pp49-56.