Vol.2009-SE-165 No.5
2009/7/2

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～

岸 知二†

ソフトウェアの設計検証にモデル検査技術を適用する検討が多くなされている。モデル検査技術は有効な検証手段のひとつであるが、正確な検証モデルを作る必要があるなど、注意深く適用する必要がある。本稿では設計検証へのモデル検査技術の適用を取り上げ、特にモデリングの観点から留意点を整理する。さらに検証のためのモデリング手法について考察する。

On Software Design/Verification Method - from modeling viewpoint

Tomoji Kishi†

There reported various approaches of software design verification utilizing model checking techniques. Though the technique is one of promising approaches to software verification, we have to carefully apply the techniques, e.g. we have to develop correct verification model. In this paper, we examine issues that have to be kept in mind in design verification, especially focusing on modeling techniques. Also, we discuss a modeling method for design verification.

1. はじめに

ソフトウェア検証に形式検証を適用する研究や実践が多く報告されているが、われわれは特にモデル検査技術[1]による設計検証に注目し、UML設計の検証を支援するツールの開発や、企業での組込みソフトウェアの事例への適用などをしながら効果的な設計検証のあり方について検討している[2][3]。

モデル検査技術による検証は万能ではないが、組込みソフトウェアで重要となるタイミングに関わる検証など、有効性が高い適用局面があると考えられる。一方、モデル検査技術による設計検証を正しく行うためには、対象を正しくモデル化し、また適切な性質を定義しなければならないが、その作業は十分に注意深く行う必要がある。少しでもモデルや性質に間違いや不正確さがあると、検証を行ってもその結果は意味のないものになる。

われわれは機器制御を行うプログラムを例題にとりあげ、モデル検査技術よる設計検証の難しさについて報告した[4]。本稿では、その報告に基づいて、モデル検査技術を行う際に注意しなければならない点を特にモデリングの観点からさらに検討する。またその検討に基づき、検証のためのモデリング手法についての考察を行う。

2章では以前報告した例題とその検証上の課題について概要を紹介する。3章では、モデル検査技術による設計検証を説明するための概念的なリファレンスを示し、そのリファレンスの上で例題での課題を位置づける。4章ではリファレンスとモデリング技術とを対応付けることで、検証のためのモデリング上での留意点について検討する。以上の検討に基づき、5章では検証のためのモデリング手法について考察する。

2. 例題による考察

われわれが行った例題による検証実験[4]について、以下に概要を示す。

2.1 対象ソフトウェア

機器の状況を複数のセンサで捉え、状況に応じて定義された処理を行うソフトウェアである。ソフトウェア内部では機器の状態の変化を状態遷移モデルとして捉え、その状態変化に応じた処理を行う。ソフトウェア内部では、状態を複数の変数値の組み合わせで表現する。センサ入力はイベントとして捉えられる。例えば 2 値のセンサ値は一方の値を取ることがイベント発生として捉えられたり、アナログ値のセンサ値は特定の閾値を超えることがイベント発生として捉えられたりする。

2.2 検証内容

設計意図は状態遷移図で記述された文書で与えられた。設計検証においては、上記

† 早稲田大学
Wased University

のセンサ値や変数値に基づく設計を、状態遷移図として与えられた状態モデルに照らして２種類の検証（検証１、検証２）を行った。検証１では意図した状態遷移モデルが設計に反映されているかどうかを、検証２では意図していない状態遷移が設計上で起こり得ないかどうかを検証した。

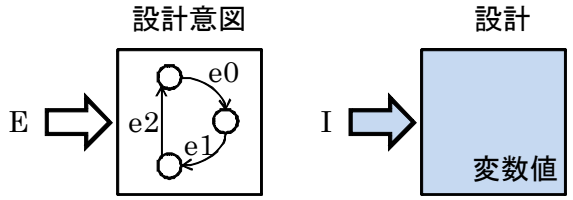

図 1　例題の概要

図 1は例題の概要である。設計意図はイベント(E)に対する状態遷移図として与えられ、設計はセンサからの入力(I)から判断されるイベントと内部変数値によって表された状態によって状態モデルを実現している。検証１では、設計意図で定義された状態遷移を引き起こすイベント(E)系列に対して、設計上でそれに対応するセンサ値(I)変化を引き起こし、その結果としての変数値が設計意図どおりの状態に対応しているかどうかを検証した。検証２では、起こりうるセンサ値(I)の変化を引き起こし、変数値をモニタしながら設計意図として定義されていない遷移が起こらないかどうかを検証した。なお検証にはSPIN[4]を用いた。

2.3 検証結果

検証１は成功したが検証２はうまくいかなかった。その理由、あるいは検証を行う際に作業を困難にする要因などとして、以下があることがわかった。

1. 網羅範囲に対するプロパティが明確でないと検証ができない。起こりうるセンサ値の組み合わせ、あるいはその変化系列のすべてが設計意図に対応づくわけではない。例えば論理的には起こり得ないが設計上は機器の異常などによって起こりうるセンサ値などが存在する。そうした場合に対応した設計意図が未定義だったため、検証２はうまく行えなかった。
2. 設計意図の設計へのマッピングのわかりやすさが重要。設計上の変数値の組み合わせと設計意図中の状態との対応関係の理解に時間を要した。
3. 設計意図と設計との関係を明確に設定しないと検証ができない。設計は設計意図のみを実現するのか、設計意図を実現しているが他のふるまいも認められるのか、といった設定の置き方によって検証２が意味のある検証かどうかが決まる。

3. 問題の整理

前章で紹介した例題による検証結果を整理するための、概念的なリファレンスを図 2に示す。

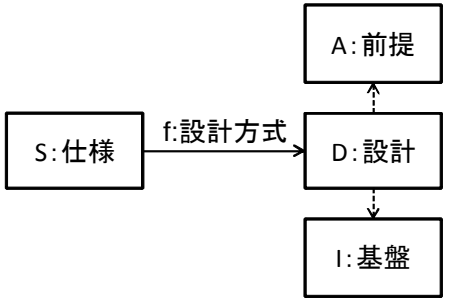

図 2　設計検証のリファレンスモデル

このリファレンスは以下の要素から構成されている。

- 仕様(S)：ソフトウェアに対する仕様。操作やふるまいの外部仕様や特定の条件において成り立つべき性質などが含まれる。例題では設計意図（どのようなイベント系列に対してどのような処理を行うかという定義）に相当する。
- 設計(D)：検証対象となる設計。例題ではセンサ値系列を受け取り内部変数値で状態を表現するソフトウェアの設計が相当する。
- 設計方式(f)：S を満たす D を構築するにあたっての方針。例題ではセンサ値からのイベント判断方法や、変数値での状態を表現方法などが相当する。
- 前提(A)：設計にあたっての前提事項。設計上設定した制約や条件、具体的な外部とのインタフェース、次項の基盤(I)に依存した仕様や制約など。例えばバッファサイズの具体的な制限、考慮すべき機器の異常などの範囲など。
- 基盤(I)：設計が依存するプラットフォーム。例えばリアルタイム OS やミドルウェアなど。モデル検査にとって重要な実行意味はアプリケーションだけではなく、多くの場合基盤によって決定される。

このリファレンスに照らして、例題での検証結果を再度考察する。

1. S、A、D、I が何なのかが明確に定義されていないと検証はできない。例題では S 中の状態を D では変数値の組み合わせで表現したが、すべての変数値の組み合わせが状態に対応付けられているわけではなかったため、対応づかない変数値の

組み合わせが起きたときのふるまいなどは未定義なものもあった。また A については、対象ソフトウェアが考慮すべきエラーの範囲などは、開発者にとっては多くの場合自明であるなどして暗黙理に決まっていることも多いが、検証にあたってはそれらを明示化する必要がある。

2. f が明確でないと、S に照らした D の検証は困難である。例題の適用においては、複数の変数値の組み合わせで状態が表現されていたが、状態が異なると参照する変数群が異なったりするなど、その対応理解が単純ではなかった。
3. S と D の関係をどう理解するかという基本的な位置づけが重要である。つまり S は D が満たすべき性質の一部を示しているのか、D が必ず満たさなければならない性質をきっちりと示しているのか、という位置づけの問題である。ただし A や I を考えると、現実にはほとんどの場合前者であると考えられる。前述したように例題実験では、A の存在のために検証２がうまくいかなかったといえる（A が明確化されないまま検証２を行うことに意味がなかった）。

4. モデリング技術との関わり

前章で示したリファレンスをモデリング技術と対応づけ、検証におけるモデリング上での留意点を議論する。

4.1 モデリング技術との対応

われわれは設計検証と設計を親和性よく行うために、設計検証に必要な情報を通常ソフトウェア技術者が用いているモデリング技法をできるだけ用いて行うことが望ましいと考えている。そうした問題意識より、上記のリファレンス上での各種情報を表現するためのモデリング技術として典型的にどのようなものがあるかを考えた。もちろんモデリング手法は多様であるが、過去の UML 設計検証ツール開発・適用経験に即したひとつの提示であると理解されたい。なおモデル検査技術として扱いづらい側面（例えば品質特性に関わるものなど）はここでは除外する。

- 仕様(S)： 一般にフィーチャなどの単位で捉えられる要求単位は、シナリオ化され、それに基づき抽象マシンとして整理されることが多い。またそれらに照らした一定の性質記述を宣言的にすることもある。典型的にはこれらは以下のようにモデル化されると考えられる。なおいずれの場合も、仕様議論に必要な概念はクラス図で表現されると考える。
  - S1： シナリオはシーケンス図などで表現される。特に UML2.0 以降ではフラグメントなどでの記法の強化がなされ、ひとつのトレースだけでなく複数のトレースを包括的に表現することができており、有用である。
  - S2：抽象マシンはステート図などで表現される。ここで抽象マシンとは、仕様を議論する際の概念的な対象の動作モデルをさす。例えば交換機であれば、受話器をあげるとダイアル待ちになり、ダイアルをすると発信状態になり、といった仕様議論上の概念的なモデルが存在するが、本稿ではこれを抽象マシンと呼んでいる。
  - S3：宣言的な性質は上記モデルに対する OCL 表現などで与えられる。例えばバッファサイズが一定値以下である、ふたつの属性値の間の関係など。
- 設計(D)： 設計はクラス図とステート図で表現される。
- 設計方式(f)： S 上の概念と D 上の概念のマッピング。マッピングは必ずしもモデルとして表現されず、例えば変換ルールのような形で表現されうるが、それが S や D のモデル上の概念で疑義なく定義されることが望ましい。
- 前提(A)： 設計の範囲やインタフェース、考慮する異常の範囲など、モデル化の前提事項である。範囲やインタフェースの表現にはクラス図を使うなど、それぞれ何らかのモデル化手法がありうるが内容依存であり、またその定義量も場合によっては膨大となり、暗黙となっていることも多い。しかしながら暗黙的なことは検証を阻害するため、検証目的に応じてその明確化が必要となる。
- 基盤(I)：基盤は実行意味などに影響するため、特に組込み分野などでは、その明示的なモデル化の必要性が議論されている。典型的には以下の方法がとられることが多い。
  - I1：ステレオタイプによる表現。D 中のモデル要素が基盤のどのような概念と対応づいているかをステレオタイプで表現する。例えばタスクやメッセージキューなどを意味するステレオタイプを利用するなど。
  - I2：基盤そのものをモデル化する。クラス図などを用いて基盤のモデルを作り、Dとの間の対応をモデル化する。MARTEのentry pointなど[6]。

なお設計はその基盤との間に横断的な関係を持つ場合があるので、そうした場合にはアスペクト指向技術の適用も有効であると考えられる[7]。

4.2 検証上の留意点

上記のモデリング技術を利用すると仮定して3章で指摘した問題を再考すると、モデリングに関わる以下のような留意点が考えられる。

1. S、A、D、I の明確化。
   ① S に関しては仕様化段階では設計段階ほど厳密なモデル化がなされない傾向があるが、それでは検証はできない。抽象度が高くても概念の定義は明確かつ厳密に行う必要がある。S1 のようにシナリオを用いる際は、そのシナリオがどのような状況におけるシナリオなのか、またシナリオ間のインタラクションや干渉はないのかなどを十分に注意して検討する必要がある。S2 の場合は概念の明確化に加えて、実行意味の明確化を行う必要がある。S と D は実行意味が同一でない状況が普通なので、その明確化は重要である。また S3 については、その性質がどのような状況で成立するのかを吟味する

必要がある。実際のソフトウェアで意味のある不変条件というものはそれほど多くはなく、なんらかの条件がつくことが多い。

② D については S との対応議論がやりやすい必要がある。次項で f とあわせて議論する。

③ A や I は従来暗黙、未定義になりがちな部分が多い。またすべてを明確にすることは困難なことも多い。検証項目に影響する要因を理解して、その部分はモデルに反映する必要がある。例えばリアルタイム OS のスケジューリングの詳細を精密にモデル化することが必要なのか、より広いあるいは狭いスケジューリングポリシーで検証すればよいのか、といった判断によってモデル化の内容は異なる。

2. f の明確化。S の表現によって D との対応関係の議論が変わってくるが、いずれの場合にも検証に関わる仕様上の概念と設計上の概念の対応関係の明確化が基本である。さらにシナリオ(S1)の場合には、ステート図で表現された D から導出されるふるまいのどの部分がそのシナリオに相当するのかを厳密に理解できる必要がある。ステート図(S2)の場合には仕様上と設計上のイベント概念、状態概念の対応の明確さが重要となる。宣言的な制約(S3)の場合にはその制約が成り立つ条件などが明確でないといけない。また A や I に依存して、D の設計範囲が決まったり、設計判断や基盤 I に依存した設計上の仕様が増えたりするため、そうした部分との対応関係を明確にする必要がある。

3. S と D の関係。前述したように、S が D に対してどのような制約を課すものであるかを明確化することが本質である。その上で、モデル検査技術で検証を行うという利点をどう活かすか、より即物的にいえばどの部分をどのように網羅検査することを狙うのかを明確にする必要がある。例題の場合、E を網羅するのか、I を網羅するのかで検証の意味も、S、D、A としてモデル上で明確にしなければならない範囲も異なってくる。そうした基本的な検証の枠組みを明確にすることで、目的指向のモデリングができると考えられる。

5. 検証モデリング手法

上記までの検討に基づいた検証のためのモデリング手法について以下に示す。

1. 検証項目の明確化：何に照らして、どういう範囲について、どういう性質を検証したいのかを決める。例えば仕様 S に照らして、S で定義されたイベント系列の範囲で、D が定義されたとおりの状態遷移を実現することを検証する、といったことを決める。当然ながらその性質が設計上の重要性を持ち、かつモデル検査のもつ網羅性が活かせる検証であることが本質である。

2. 検証アーキテクチャの決定：検証を行うためのモデルの全体構造を明らかにする。これは検証を行う際にモデルがどのような役割を持っているか、その関わりを明確にすることである。例えば例題実験では、検証１や検証２を図 3で示すような構造で検証した（詳細は[4]参照）。こうした検証構造は例題特有のものではなく、類似した検証に応用できるものであり、一種の検証アーキテクチャパターンであるともいえる。こうした検証アーキテクチャを決めることで、S、D、Aなどの位置づけが明確になり、何をどこまでモデル化するかの方針が決まる。また何を網羅するのか、網羅される範囲はモデル上でどのように表現されるのか、などが明確となる。以降の作業はこうした検証アーキテクチャに基づいてなされる。

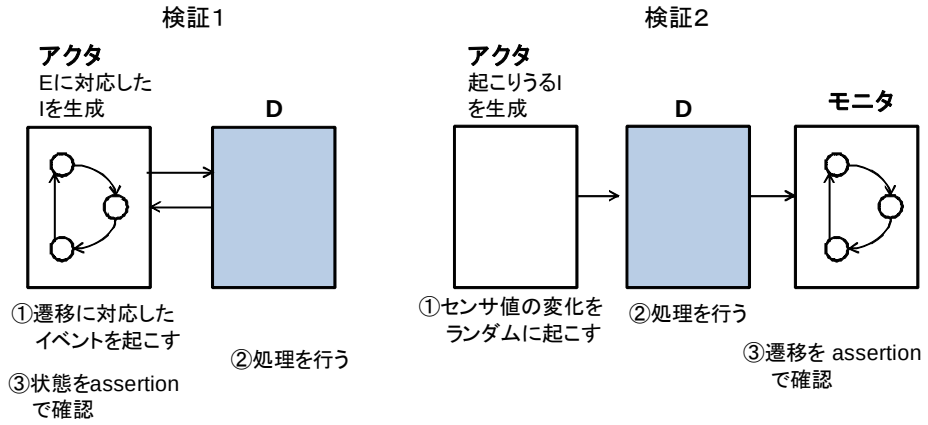

図 3　例題での検証アーキテクチャ

3. 仕様Sのモデル化：検証に必要な仕様側面をモデル化する。図 3の検証１を例にとると、Sがシナリオ S1 で与えられるのか、状態モデルS2 で与えられるのかといったことによってアクタの実現も、その検証の意味も変わってくるので、留意が必要である。

4. 前提 A、基盤 I の明確化：検証にとって必要な前提 A や基盤 I を明確にする。前述したように、A や I は量的にも多く、暗黙化されていることも多い。したがっていたずらなモデル化ではなく、検証目的によって考慮すべき A や I の範囲を吟味した必要十分なモデル化が重要である。

5. 設計方針 f の明確化：仕様 S 上の概念と設計 D 上の概念の基本的な対応方針を決め（例えば 1:1 対応なのか、1：多なのか、対応づかない概念があるのか等）、その方針にそって具体的な対応関係を定義する。この作業は次項の D のモデル化戦略と強く関連する。

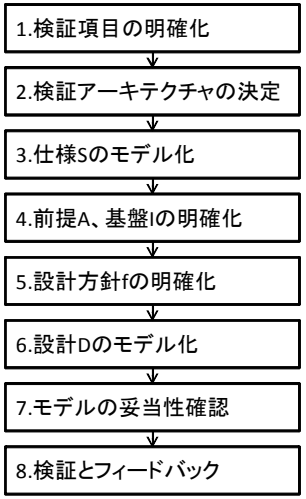

図 4　検証モデリング手法の概要

6. 設計 D のモデル化 : 一般に設計 D は A や I への考慮があるために S よりもより多くの情報を含んでいる。そのため、S と D を対応付ける際には構造化の方針が重要となる。例えば仮想マシンのレイヤを作り、その上に S との対応付けが容易な部分のみを構築するというのもひとつの戦略である。設計 D の構造によって検証が容易になったり難しくなったりするので、重要な検証事項に関わるモデルはできるだけ S との対応が容易に構築すること(つまり明確な f に基づいて設計すること)が重要である。次に、何を網羅するかによって網羅の基本単位が異なる。検証１では S 上のイベント列の網羅であったが、検証２では D 上で起こりうるセンサ値系列の網羅であった。検証したい網羅単位が何なのか、その網羅範囲における S は明確に定義されているのか、などを注意してモデル化する必要がある。さらに実行意味に関する考慮も重要である。前述したように、一般に S の想定する実行意味と D の想定する実行意味は完全に一致することは少ない。それらを踏まえて検証の実行意味をどう設定することが適切かを十分に考慮する必要がある。
7. モデルの妥当性確認：検証に用いる S や D のモデルは、正確でなければならない。その正しさを確認するためには、明白な性質について検証を行って妥当性を探ったり、検証モデルをインクリメンタルに構築しながらリグレッション検証しつつより大きな検証モデルへと構築したりするなどの考慮も必要となる。いずれにせよ、検証モデルそのものはできるだけ理解容易にしなければ、そもそも検証することの意義自体が失われてしまう。
8. 検証とフィードバック：以上に基づき構築されたモデルを活用して検証、その結果に基づく設計へのフィードバックを行う。

図 4に本手法の全体像を示す。

6. おわりに

本稿では、モデル検査技術による設計検証を行う際のモデリング技術に注目し、その留意点や手法について概略的な検討を行った。SPIN などを利用して検証する際には、Promela という言語でモデルを作るとシミュレーションができるため、ともすれば簡単なプログラムを作って動作確認をするような形で検証モデルを作ってしまう危険もあるが、モデル検査の厳密性に照らすと極めて危ういことである。今後本稿の検討をベースに、より運用しやすくかつ有用性のあるソフトウェア技術者のための手法について検討を深めたい。

参考文献

[1] Clarke, E., Grumberg, O., Peled, D.: Model Checking: MIT (1999).
[2] 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, pp50-57, 2006.
[3] 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).
[4] 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.
[5] Holzmann, G.J.: The SPIN Model Checker - Primer and Reference Manual, Addison-Wesley (2004).
[6] OMG: A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Beta 2, 2008.
[7] 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.