8. モデリングプロセスの構成と手順

モデル検査を用いた設計モデリングのプロセスを分類し、それぞれのプロセスの流れと手順を示す。本章の概要は以下の通りである。

対象読者	(1) 開発技術者 (2) 開発プロジェクト管理者
目的	モデル検査における設計モデリングにおいて、最初に利用できる情報に応じて、モデリングプロセスが分類されることを示し、その中で典型的なアーキテクチャ情報に基づくモデリングプロセスについて具体的に示す。
想定知識	ソフトウェア開発技術者で、SPIN に関する既存のテキストにより SPIN の基礎知識を身につけた者を対象とする。
得られる知見等	● モデリングの最初の時点で利用できる情報により分類できるモデリングプロセスのパターン ● アーキテクチャ情報に基づくモデリングプロセスのパターンの具体的な手順と留意点

8.1. モデリング・プロセスのパターン分類

モデル検査を利用したソフトウェアの設計モデリングにおいては、最初に利用できる入力情報と検証の目的に応じて、モデリングプロセスの流れは大きく異なることになる。本章では、モデル検査を用いたモデリングプロセスの考え方[142],[143],[144]に基づき、典型的なモデリングプロセスのパターンとその手順について整理する。

モデル検査のプロセス全体は、ソフトウェアの設計に関する人の知的作業が求められる部分と、検証や不具合検出などツールの利用が中心となる部分に分けられる。モデル検査手法を導入するにあたって、前者の知的作業のノウハウや留意点を把握することが重要となる。第 8 章、第 9 章では、そのような知的作業が必要となる設計モデリングとモデルの抽象化に関する方法や考え方の例を示す。

まず、モデリングプロセスにおいて、入力として利用される情報には主に以下のようなものが考えられる:

[142] モデル検査とパターン、SES2009 ワークショップ1、ポジションペーパ、中島震
[143] SS2009「形式手法適用」WG,「形式手法」の「適用」について, 中島震, 国立情報学研究所
[144] 第 12 回 ESEC 専門セミナー、形式手法の概要とモデル検査法の応用、2009 年 5 月 13 日、中島震、石黒正揮

- 検証性質の情報
  機能に対するユーザの要求（機能要求）や、安全性など機能に関する特性を表す非機能要求などから検証する性質を明確に記述したもの。
- アーキテクチャに関する情報
  実行環境（ミドルウェアやライブラリなど）や効率性の観点から生じる制約により、システムのコンポーネントやコンポーネント間の関係などのアーキテクチャについてあらかじめ制約がある場合、それらの構造を前提にモデリングしなければならない。
- アルゴリズム・機能設計
  開発するソフトウェアのアルゴリズムが既存のものあるいはあらかじめ設計により明確になっている場合、そのアルゴリズム記述に従い形式モデリングを行う。
- ソースコード
  開発済みのソースコードが入手可能であり、そのソースコードの不具合を解析する場合、ソースコードから、半自動あるいは自動で、モデルを変換生成する場合がある。

これらの情報のうち、どれを最初の入力情報として利用するかによって、モデリングプロセスを大まかに分類すると表 8-1 のような3つのパターンに分類される。3つのパターンは、入力情報と最初に形式記述する対象により分けられる。形式記述の対象は、ソフトウェアが何を実現するかを意味するWhatに相当するもの（検証基準）と、どのように実現するかを意味するHowに相当するもの（検証対象モデル）の 2 つに分けられる。モデル検査においては、前者は、検証性質を表す時相論理式に相当し、後者は、検証対象を表す状態遷移システム(Promelaコード）に相当する。

		先に形式記述する対象	
工程	入力情報 （日本語文書等）	何を実現するか（What） （検証基準） LTL論理式等	どのように実現するか（How） （検証対象モデル） Promela記述等
要求分析	検証性質 （要求仕様）	(1)検証性質駆動 モデリング	
基本設計	アーキテクチャ 構造制約		(2)コンポーネント駆動 モデリング
詳細設計	アルゴリズム ・機能仕様		(3)アルゴリズム駆動 モデリング
コーディング	ソースコード		（プログラム検証）

表 8-1：モデリングプロセスの入力情報によるパターン分類

このような区別に基づくと、(1)検証性質駆動モデリングは、検証性質に関する情報を入力として、検証基準を先に記述する場合 (2)コンポーネント駆動モデリングは、アーキテクチャ構造制約を入力として、検証対象モデルを先に記述する場合、(3)アルゴリズム駆動モデリングは、アルゴリズム情報を入力として、検証対象モデルを先に記述する場合、の3つのパターンに分けられる。

ソースコードを入力として、モデル検査を行う事例もあるが、この場合、ソースコードからモデルを抽出するための支援ツールを用いたり、モデルの自動抽出などが行われるため、人手によるモデリングとは異なるため、本章の中心的な対象とはしない。

実際のモデリングにおいてどのモデリングパターンとなるか、主な判断基準を以下にまとめる。

モデリングパターン	選択基準
検証性質駆動モデリング	● 要求が明確化されており、システムをどのように実現するか自由度が高い。 ● 要求からシステムの設計に対する条件を得て、分析したい。 ● アーキテクチャの制約が余り無い。 ● 要求仕様を積極的に使い、設計モデルを効率的に抽出したい。 ● モデリングの経験があり、時相論理式による性質の記述に慣れている。
コンポーネント駆動モデリング	● 詳細設計に入る前に、アーキテクチャ設計（コンポーネント間の関係）の正しさの検証や、不具合の検出を行いたい。 ● コンポーネントの振舞い（機能）に関する設計が与えられていない場合。 ● 分散システムの開発や既存システムをモジュールとして活用した開発では、アーキテクチャに対する制限が強い場合にアーキテクチャ駆動パターンが適している。 ● アーキテクチャレベルの設計書が明確であり、初期段階で、安全性などの要求仕様が明確でない場合。（実際の開発現場で多い状況） ● モデル検査における検証性質の形式記述の経験が少なく、対象システムのモデルの記述なしに検証性質を記述することに慣れていない場合。
アルゴリズム駆動モデリング	● 検証したいアルゴリズム（機能）の設計が与えられており、そのアルゴリズムの正しさを検証したい。 ● 規模の大きいソフトウェアのうち、特定のアルゴリズム切り出すことで、モデルのサイズを抑えられる。

	● アルゴリズムのデータ処理ではなく、制御部分を対象に検証したい。(データ処理の検証はモデル検査には適さない。) ● モデル検査における検証性質の形式記述の経験が少なく、対象システムのモデルの記述なしに検証性質を記述することに慣れていない場合。

表 8-2:モデリングパターンの選択基準

本章では、これらの3つのパターンについてプロセスの基本的な流れと考え方のみを示す。より具体的な手順等は、既存の書籍[145],[146]等に示されているため、それらを参照するとよい。ただし、そのプロセスの中で利用されるいくつかの具体的なテクニックの例を示し、実際の現場における参考例を示す。

以下の章では、モデリングの各パターンについて具体的なプロセスを示す。

8.2. コンポーネント駆動モデリング

8.2.1. プロセスの概要

ソフトウェアのアーキテクチャに関して、その構成要素と要素間の基本構造に関する設計の正しさ検証する場合に適用するパターンが、コンポーネント駆動モデリングである。ソフトウェアの実行環境に関わる制約で、アーキテクチャに対する制約条件が強い場合は、このパターンを参考にすると良い。このパターンにおいては、対象システムのモデルを形式記述し、その後にモデルの記述要素を用いて検証性質を記述する。検証性質駆動パターンのように、モデルの形式記述の前に、検証性質を形式記述することは、高いスキルが求められるため、本パターンの方が習得は容易である。また、アーキテクチャは、システムを抽象化した基本構造を記述するものであるため、システムの広い範囲を対象としていても、状態爆発の問題を回避しやすい。それらの点から、モデル検査の適用においては典型的なパターンとなる。本パターンは、10 章ケーススタディ1:ブルーレイディスクにおいて実践したものに相当する。

プロセスの概要は以下の通りである。

[145] 中島震: SPIN モデル検査 -検証モデリング技法, 近代科学社, 2008 .
[146] 萩谷, 吉岡, 青木, 田口, SPIN による設計モデル検証, 2008

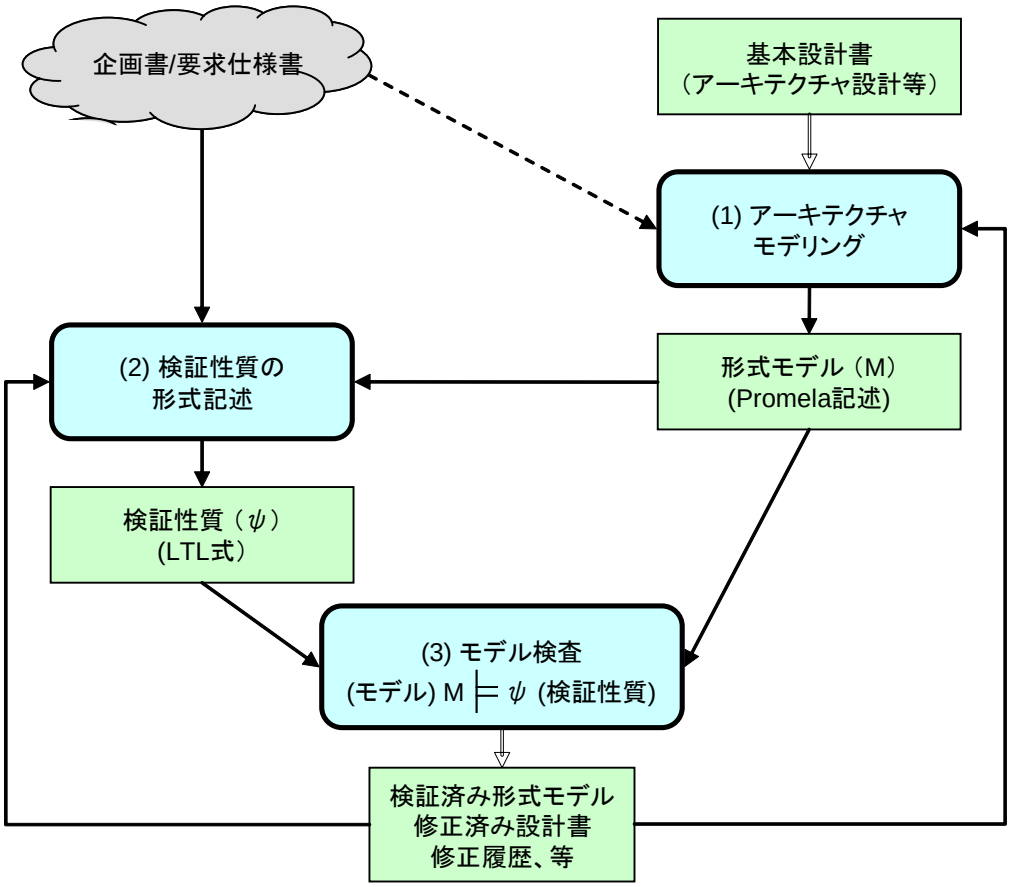

図 8-1:コンポーネント駆動パターンの概要

このパターンは、3つのサブプロセスから構成され、それぞれ以下のような作業を行う。

(1) アーキテクチャのモデリング

アーキテクチャ設計書と検証したい項目から、検証対象とするシステム構成要素と要素間の関係に関する情報を抽出し、処理の実行主体となる並行プロセス[147]および、要素間の関係を並行プロセス間の通信として形式モデルを作成する。

(2) 検証性質の形式記述

検証したい項目および形式モデルから検証したい性質を抽出し、形式モデルの記述に含まれる状態やイベントなどを用いて、検証性質を時相論理式として記述する。

(3) モデル検査

(1), (2)で作成した形式モデルと時相論理式に対してモデル検査を行い、形式モデルや検証性質に不具合が見つかった場合には、それらを修正し、モデル検査を繰り返す。

[147] モデル検査 SPIN では、並行処理の主体をプロセスと呼ぶ。ここでは、人の作業であるモデリング・プロセスとモデル検査の処理主体のプロセスを区別するために、後者を「並行プロセス」と呼ぶ。また、前者は、「作業プロセス」と呼ぶ。

8.2.2. プロセスの詳細

プロセスの概要で示したとおり、コンポーネント駆動モデリングは、(1)アーキテクチャのモデリングと(2)検証性質の形式記述、(3)モデル検査、の3つのサブプロセスから構成される。それぞれのサブプロセスにおける考え方と手順は以下の通りである。

8.2.2.1. アーキテクチャのモデリング

(1) プロセスの概要

アーキテクチャ設計書および企画書/要求仕様書をもとに、アーキテクチャに関して検証したい範囲を抽出し、形式仕様記述を行う。

(2) 入力

- 基本設計書(アーキテクチャ設計)
- 企画書/要求仕様等(自然言語など)

(3) プロセスの手順

① 検証要求の抽出

- 企画書等からアーキテクチャ設計に関する検証したい性質を抽出する。

② 並行プロセスの抽出

- 基本設計書から、検証したい性質に関連するアーキテクチャ設計を抽出し、実行主体となるコンポーネントをモデル検査のプロセスとして抽出する。並列分散実行されるコンポーネントやデバイスドライバなどがプロセスの候補となる。(例:図 8-2 では、アプリケーション・モジュール、モードモジュール、ミドルモジュール、デバイスドライバ、デバイス割込みハンドラー等がプロセスとして抽出される。)
- 対象システムへの入出力を行う外部環境は、別プロセスとして定義する。
- 検証したい性質を特定することで、関連性のないコンポーネントや相互作用は捨象し、モデルの規模の増大を抑える。
- モデル検査の状態爆発の可能性は、プロセス数が大きく影響されるため、モデルの抽象化の章を参考にプロセス数を抑える。

③ 並行プロセス間の関係の抽出

- 各プロセスの入出力を特定し、プロセス間の関係に関する情報を抽出する。
  (図 8-2 の例では、API 要求、ミドル関数呼出し、ドライバ呼出し、割込みなどが相互作用として抽出される。)
- プロセス間の関係は、プロセスへの入力となるイベントの種類、出力となるイベントの種類、入力と出力の関係を示す情報で特定される。

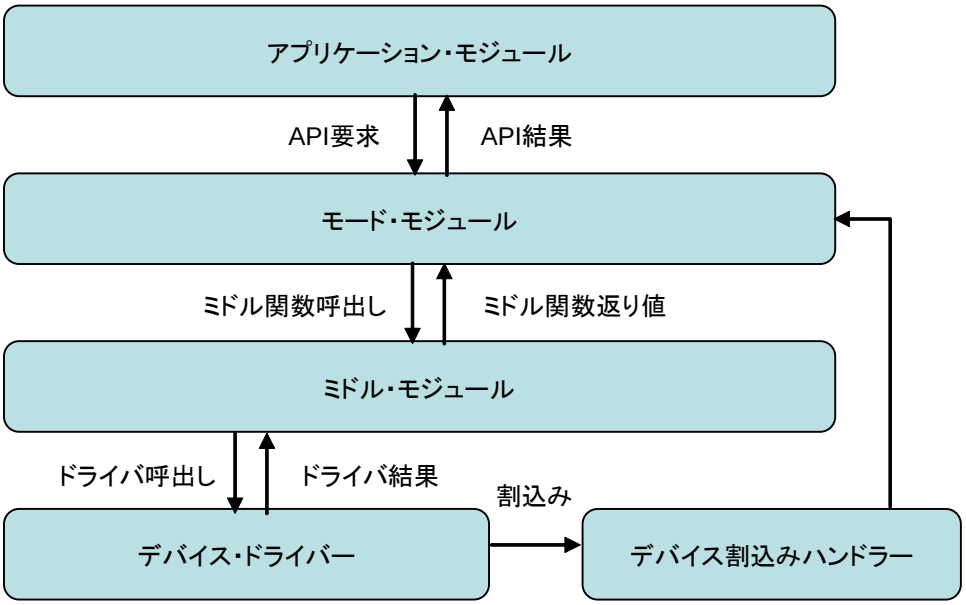

図 8-2:システムの構成要素と相互作用の例

④ プロセスごとの形式仕様記述。

- プロセスごとに、状態遷移システムの記述テンプレート(8.2.3 章)などを参考にして形式仕様記述を行う。
- 具体的には以下の通り:
  - 並行プロセスへの入力と出力のイベントの種類を定義する。
  - 入力イベントと出力イベントの関係を状態遷移システムとして記述するために必要な状態を定義する。
  - 基本設計書では、並行プロセスへの入出力の関係は記述されていても、入力に対してどのように出力を求めるか記述されていない場合があるため、新たな状態を定義する必要がある場合がある。
  - 並行プロセス間の入出力は、同期通信/非同期通信のいずれかで定義する。
- アーキテクチャ設計では、コンポーネント間の関係は記述されていても、入力から出力を得るためにどのような制御を行うか記述されない場合があるため、作業プロセスにおいて、状態遷移システムとしてモデリングする必要がある。
- 外部環境は、システムへの入力イベントの順序関係に制約が無い場合、非決定的に任意のイベントを送信するモデルとする。

(4) 出力

形式モデル(Promela による記述)

(5) 留意点

- モデル検査で扱える規模の制限を考慮して、平行プロセス数、プロセスの実行命令数を

一定以下に抑える。

8.2.2.2. 検証性質の形式記述

(1) プロセスの概要

要求仕様書等から、形式モデルに関する検証性質を抽出し、形式モデルで定義される状態、イベント等を用いて、検証性質を LTL 式で記述する。

(2) 入力

- 企画書/要求仕様書
- 形式モデル

(3) プロセスの手順

① 検証性質の抽出

- 記述した形式モデルに関連する検証性質を、要求仕様書の機能要求、非機能要求等の中から抽出する。
- 非機能要求は、要求仕様書に網羅されていない場合があるため、形式モデルにより明確化したアーキテクチャからも検証性質を洗い出す。
- 誘導語などをもとに、時相論理式で記述できる性質を抽出する。
- 対象とする検証性質は、LTL 式の典型的な記述パターン（8.2.4 章）などを参考に、記述可能な性質を抽出する。

② 検証性質の形式記述

- 抽出した検証性質について、形式モデルで定義された状態とイベント等を用いて検証性質の形式記述を作成する。

(4) 出力

- 検証性質（LTL 式）
- 修正された要求仕様

(5) 留意点

- 抽出した検証性質が、原始命題を記述するために必要な状態やイベントを形式モデルに追加して、形式モデルを修正する。

8.2.2.3. モデル検査

(1) プロセスの概要

8.2.2.1、8.2.2.2 節で示したサブプロセスで作成した形式モデルと検証性質を対象にモデル検査を行う。

(2) 入力

- 形式モデル（Promela 記述）
- 検証性質(LTL 式)

(3) プロセスの手順

① モデル検査の実行

モデル検査ツールに入力となる形式モデルと検証性質を与え、モデル検査を行う。

② 不具合解析

不具合が発見された場合、既存テキスト[156,158]を参考に不具合解析を行う。

③ 形式モデル、検証性質の修正

状態爆発が発生する場合には、別章「モデルの抽象化」を参考に形式モデルを修正する。

(4) 出力

- 検証された形式モデルと検証性質/修正された形式モデルと検証性質
- 修正された設計書と要求仕様書
- 修正箇所の履歴

(5) 留意点

- 並行プロセスを扱う対象では、状態爆発が発生しやすいため、次章に示す抽象化の方法を参考にするとよい。

8.2.3. 要素テクニック：テンプレート支援モデル記述法

8.2.2.1 節などで、並行プロセスが特定されると、並行プロセス単位で、下記に示すPromelaコードのテンプレートを参考に、並行プロセスを記述するとよい。なお、この方法は、文献 の方法をベースとしているため、その文献を参考にするとよい。

<u>モデル形式記述の手順</u>

(1) 目的に応じたテンプレートの選択

状態遷移表や簡易ステートマシン図等のアプローチに応じて、対応した Promela コードのテンプレートを選択する。

テンプレートの例:

- ■ 状態遷移表に基づくテンプレート（図 8-3）
- ■ 簡易ステートマシン図に基づくテンプレート（文献 ）

(2) テンプレートに応じたモデリング

テンプレートに基づき、具体的な記述を埋めていく。

例）状態遷移表に基づくテンプレート

- ■ インタフェース部の定義
  - (1) プロセスの特定（実行主体）
  - (2) チャネルの特定（同期/非同期）

(3) 入出力イベントの特定

■ 制御部の定義

(1) 状態の特定

(2) 状態遷移とイベント送信の定義

■ モデルの形式記述

```
mtype = {<イベントの列>}
mtype = {<状態の列>}

chan <入力チャネル> = [<バッファサイズ>] of <mtype>
            :
chan <出力チャネル 1> = [<バッファサイズ>] of <mtype>
chan <出力チャネル 2> = [<バッファサイズ>] of <mtype>
                                :

active proctype <状態遷移表 1>() {
  mtype state;
  mtype event;
  do ::<入力チャネル>  ? event ->
    if
      ::(event == Event1) ->
        if
          ::(state == State1) -> state == <遷移後の状態>;
                                          <出力チャネル>! <送信イベント>
          ::(state == State1) -> state == <遷移後の状態>
                                      <出力チャネル>! <送信イベント>
                                :
        （状態による場合分け）
                                :
          fi
       ::(event == Event1) ->・・・
          :
          (イベントによる場合分け)
              :
          fi
  od
```

```
}
```

図 8-3:状態遷移表に基づく Promela 記述テンプレート(例)

8.2.4. 要素テクニック：検証性質パターン支援抽出法

8.2.2.2 節に示す検証性質の記述においては、モデル検査で記述される検証性質の典型的なパターン(下記例参考)を参考にするとよい。具体的には、典型的なパターンを日本語で表現した「誘導語」を参考に、記述したい性質に対応するものを選択し、その形式記述を特定する。実際には、記述パターンは、これらの典型パターンを組み合わせた形式によることがあるため、検証性質を階層的に分割し、それぞれの性質について典型パターンの特定を繰返し、全体のパターンを類推する。

パターン名		説明等
応答パターン		□(request -> <>response)
	内容	ある要求(request)が成り立つとき、いずれは必ず応答(response)が成り立つ。最初の時相演算子□は重要である。□が無ければ、最初の request が成り立った時のみ、検査されるが、リアクティブシステムの場合、request が成立った時、常に response が期待されるためである。
	誘導語	～ の時はいつも、いずれは ～ が成り立つ。
排他制御パターン		□¬(critical1∧critical2)
	内容	条件 critical1 と critical2 は、同時に成り立つことは無い。
	誘導語	～ と ～ は、同時に成り立つことは無い。
先行パターン		([] ¬pre-condition) ∨(¬pre-condition U second)
	内容	特定の状態 second が成り立つ前に必ず事前条件 pre-condition が成り立つ。
	誘導語	～が成立つ時は、必ず～が先に成立つ。
進行性パターン		□<> condition
	内容	繰り返し必ず、条件 condition が成り立つようになる。
	誘導語	いつでも、必ず～の状況になる。
ラッチングパターン		<> □ condition
	内容	いずれは、条件 condition が成り立ち、その後継続的に条件が成り立つ。

	誘導語	いずれは、～の状態になり続ける。

表 8-3:検証性質パターン(モデル検査における時相論理式)

これらの典型パターンは、文献 ,[148]をベースに典型的な例を取り上げたものである。より広い範囲のパターンはそれらの文献を参照するとよい。

Dwyerの研究[149]では、要求仕様(自然言語)の調査事例のうち、40%程度が、応答パターン(時間スコープ 4 通り)で占められることが示されている。

8.3. アルゴリズム駆動モデリングの概要

アルゴリズム駆動モデリングについては概要のみ示す。

アルゴリズム駆動パターンは、機能を実現するアルゴリズムの正しさを検証するためのプロセスである。アルゴリズム駆動パターンは、検証対象が機能設計、ソースコード、プログラムなどによって場合に分けられるが、本章では、機能設計を対象とする。本パターンは、検証性質を形式記述する前に、対象システムの形式記述を行うため、アーキテクチャ駆動パターンを構造的に似ている。違いは、アルゴリズム駆動パターンでは、機能を実現するアルゴリズムが与えられていることを仮定し、そのアルゴリズムから制御部分を抽出することでモデリング行うが、アーキテクチャ駆動パターンでは、コンポーネント間の関係のみが与えられていることが多く、それらをどのように実現するか状態遷移システムを考える必要がある点である。また、機能設計の検証の場合、要求分析の段階で、検証性質が明確になっている場合が多いため、それらの情報を活用できる。

規模が大きいソフトウェアの場合、その中の特定の機能設計に限定することで、状態爆発の問題を回避する。

[148] Principles of the Spin Model Checker, Mordechai Ben-Ari
[149] Patterns in Property Specifications for Finite-State Verification, Dwyer

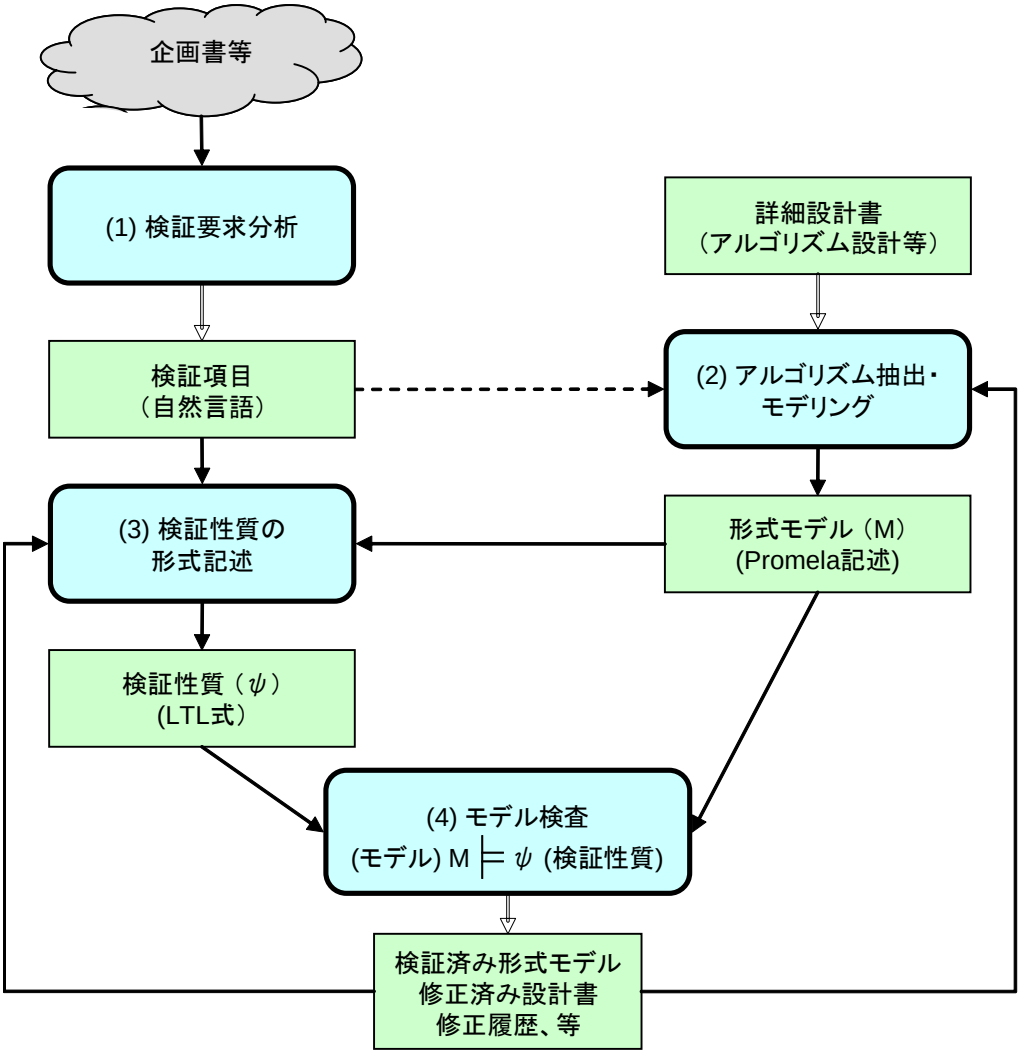

図 8−4:アルゴリズム駆動プロセスの概要

プロセスの概要は以下の通りである。

(1) 検証要求分析

企画書/要求仕様書から、検証したい機能アルゴリズムに関する検証項目を抽出する。

(2) アルゴリズム・モデリング

詳細設計書から、検証対象とする機能のアルゴリズム情報を抽出し、それらのうち制御に関わる部分を状態遷移システムとして捉えることで、形式モデルを作成する。

(3) 検証性質の形式記述

検証項目を、形式モデルの記述に含まれる状態やイベントなどを用いて、検証性質を形式記述する。

(4) モデル検査

(1), (2)で作成したモデルと検証性質に対してモデル検査を実施し、形式モデルや検証性

質に不具合が見つかった場合には、それらを修正し、モデル検査を繰り返す。

8.4. 検証性質駆動モデリングの概要

検証性質駆動モデリングについては概要のみ示す。

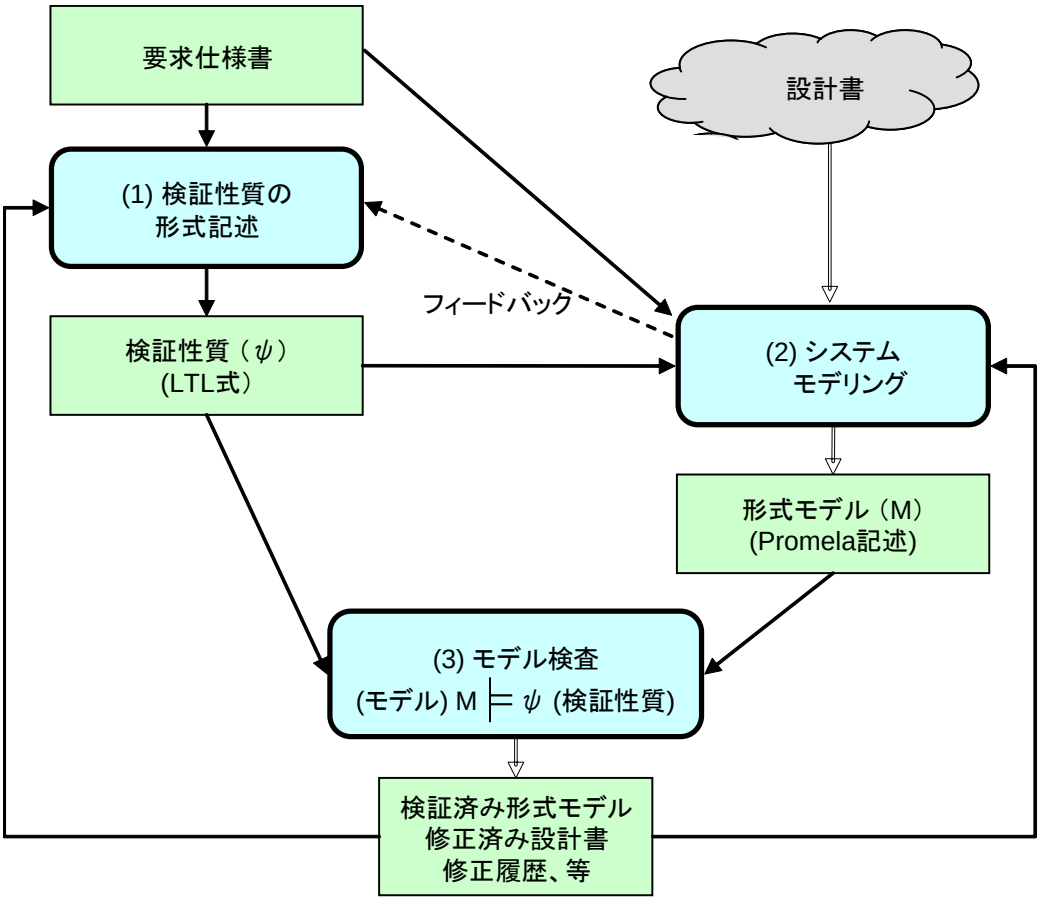

図 8-5:要求性質駆動プロセスの概要

検証性質駆動プロセスは、要求仕様に基づき、検証性質を形式記述し、それに基づきシステム設計の要件を分析するものである。要求性質の形式仕様記述と、対象システムの形式モデリングを比較すると、前者の方が抽象度は高いため、いきなり実践することは困難な場合が多い。そこで、本章で述べるプロセスは、最初は概念的に考え方を学び、アーキテクチャ駆動プロセスやアルゴリズム駆動プロセスを経験した後に、要求性質駆動プロセスを実践することが期待される。

プロセスの概要は以下の通りである:

(1) 検証性質の記述

要求仕様書から、着目する検証性質をいくつか抽出し、それらの検証性質を記述するために

必要な暫定的な命題を仮定し、それらの命題と時相論理式の演算子を用いて検証性質を記述する。具体的には、表 8-3 の検証性質パターンを参考に、必要となる命題を設定する。

(2) システムモデリング

記述した形式的な検証性質で使用する命題に対して、それらの命題を記述するために必要となる Promela コードにおける変数やイベントを暫定的に定義する。それらを用いて、図 8-3 などのテンプレートを参考に、システムのモデルを Promela で記述する。

(3) モデル検査

(1), (2)で記述した形式モデルと検証性質を入力としてモデル検査ツールによりモデル検査を実行し、不具合等が発見された場合は、(1)または(2)の作業プロセスに戻り、形式記述の修正を行う。

8.5. まとめ

本章では、モデル検査におけるモデリングプロセスに関して、モデリングの最初に注目する入力情報と検証の目的により、大きく3つの異なるモデリングプロセスに分かれることを示した。本章では、これらのモデリングプロセスの流れの違いを示すことを目的として、それぞれのプロセスの基本構造を示した。個々のサブプロセスの詳細については、既存の書籍を参照してそれらを組み合わせた利用が想定される。