Vol.2010-SE-170 No.17
2010/11/12

状態遷移図の同期モデルへの詳細化および検証手法

大森洋一[†1]　園田貴大[†2]　荒木啓二郎[†1]

計算機システムの社会基盤への浸透とそれらにおけるソフトウェアの役割が増すに従い，設計段階での信頼性向上が重要な課題となっている．フォーマルメソッドは，ソフトウェアの信頼性向上手法として長い歴史をもち，さまざまな論理に基づくモデル化に関する研究がなされている．

本稿では，要求段階における抽象度の高い機能を対象とした制約を，フォーマルメソッドを用いた明示的な仕様として抽出し，主に時間に関する性質について，設計段階における状態機械のふるまいに関してそれらの仕様を検証する手法を提案する．本手法は，要件定義における記述，これら複数のモデル変換，および状態機械への時間制約記述からなり，計算機システムの開発段階において時間に関する扱いが異なることを示す．具体的な組み込み開発事例に対して本手法を適用し，評価した．

A Proposition of Refinement and Verification Method from Requirements to A Synchronous State Machine Model

Yoichi Omori,[†1] Takahiro Sonoda[†2]
and Keijiro Araki[†1]

Software reliability has became a major engineering concern, because of the prevalence of computer systems over the infrastructure of human society and the acceleration of its importance in such systems. Formal methods are studied for a variety of modeling techniques based on inherent mathematical logics. They have a long history as ways of improving software reliability. Verification of most systems is, however, not limited to a single issue but multiple properties, therefore researches about combination of several logics are the key to practical applications of formal methods.

In this paper, we propose a developing procedure converting from constraints on abstract functions in requirements into explicit formal specifications mainly related to the timing property, and verification method at the design phase for the behavior of multi layered state machines in a combined way. This method is applied to a concrete embedded system as an evaluation.

1. はじめに

計算機システムの社会基盤への浸透とそれらにおけるソフトウェアの役割が増すに従って，機能の提供だけでなく，信頼性を始めとした非機能要件の設計段階における確保が重要な課題となっている．特に，組み込みシステムにおいては，実装手段にすぎない計算機システムへのコスト低減要求がしばしば強くみられ，計算機資源の余裕のなさ，運用開始後の改修の難しさなどから，設計時の検証が重要である．

実際，IPA「2007 年版組込みソフトウェア産業実態調査」[21] p.29 によると，組込みソフトウェア開発の課題として「設計品質の向上」を事業責任者のおよそ半数が挙げており，最多となっている．その一方で，プロジェクト管理者はプロジェクトの 7 割以上が品質に関して計画通りもしくは計画以上であったが，およそ半数のプロジェクトは実施期間が予定を超過したと答えている．したがって，時間をかけて品質を確保する技術はあるが，そのための作業時間が事業責任者の期待する水準に達していない場合がままあると考えられる．

本研究では，ソフトウェア開発における上流工程を対象として，フォーマルメソッドを用いたモデリングによる設計品質の向上を目的とする．上流工程における設計品質として問題になるのは，主として「設計の抜け・漏れ」である．つまり，開発の初期段階からモデリングを実施することにより，要件定義の一貫性を向上させるとともにふるまいを確認し，従来，詳細設計や実装の段階で見つかっていたこれらの問題を早期に発見し，大幅な手戻りを防ぐことにより，前述の課題を解決する．

1.1 モデルベース開発

設計の初期段階からなんらかの観点に基づいたモデルを用いて，仕様記述と検証を行なうモデルベース開発を用いる利点として，早期の欠陥発見によるソフトウェア品質の向上が挙げられる[14)24)]．ここで欠陥とは仕様と設計の不一致を指し，モデル化，特に複数の観点からのモデル化により，要求の記述不足や要求仕様に対する設計の不備を検証可能とする．古典的な V 字モデルでは，要求に対する不備の検出は統合テスト時に行なわれるので，不具合の修正コストが非常に高くなる．モデルベース設計では，構成時の検証が可能であり，比較的短いサイクルで不具合を修正できる．

†1 九州大学 大学院システム情報科学研究院
Graduate School of Information Science and Electric Engineering, Kyushu University
†2 九州大学 大学院システム情報科学府
Graduate School of Information Science and Electric Engineering, Kyushu University

1.2 上流工程の品質改善

このための方法として，フレームワークの利用，シミュレーションに基づく検証，フォーマルメソッドによる検証などが挙げられる．

フレームワークの利用は，特定の問題領域に対してサーバなどの実行環境や[2)16)]，プロトコルスタックなどのライブラリといったプログラム・インターフェースを再利用するものであり，通信環境など，実装レベルでよく階層化された対象領域において有効性が高い．逆に，階層化されていない問題領域においては移植性が課題となり，また互換性を確保するために比較的大きな計算機資源を使用することがある．

シミュレーションに基づく方法は， Executable UML[12)] や実行可能仕様[17)]， Matlab/Simulink[11)] 等を利用し，仕様のふるまいを確認する手法である．この方法は，具体的なアルゴリズムや値付きのデータを用いたテストなどで妥当性を確認できるので，実行可能プログラムの自動生成との組合せが特に有用である．シミュレーションで確認できるのは，プログラムに対するテストケースと同様に，入力されたデータに関する部分だけである．このため，システムの性質を保障するためには，フォーマルメソッドを利用する場合と同様に，線形性や連続性といった数理的な性質を仮定する必要がある．

フォーマルメソッドは，数理的な概念に基づいて対象をモデル化する手法であり，ソフトウェアの信頼性向上手法として長い歴史をもつ．これまでにも組込みソフトウェア品質の向上を目的とした応用は数多い[4)5)8)]．よく用いられる手法として，集合論および述語論理に基づく機能記述あるいはプロセス代数に基づくふるまい記述がある．前者には VDM[10)], Z/B method[1)] など，後者には CSP[15)], CafeOBJ[13)] などがある．検証段階ではそれぞれの論理系において検証すべき表明を与え，記述されたモデルがその表明を満たすことを確認する．検証手法には，数学的な証明以外に，モデル検査[9)22)] 等総当たりによる方法がある．

これらの手法は互いに完全に独立しているわけではないが，本研究では上流工程への応用に適したフォーマルメソッドを中心に検討する．フォーマルメソッドは、アルゴリズムの外挿や補間による非機能要件の確認が容易であること，抽象的なモデルに対して数理的性質が検証可能という特長から，初期の中間生成物の保証へも応用できる．他の方式にはないこうした利点により，対象の本質を記述する仕様作成に適していると言える．

以下，第 2 章で差分開発を前提とした開発プロセスの再検討，第 3 章で本提案手法の構成要素とそれらの説明，第 4 章で組込みシステムの事例である電子ポットへの適用について，第 5 章でまとめを述べる．

2. 差分モデルベース開発

フォーマルメソッドは優位な点ばかりではない．例えば，本質的に対象システムを特定の数理モデルに関して抽象化するため，検証する性質毎にモデル化が必要となり，実用的なシステムについては検証コストは大きくなり易い．このため，フォーマルメソッドのシステム開発への応用は，ミッション・クリティカルなシステムの新規開発という，そもそもの開発コストが比較的大きく，フォーマルメソッドの利用による追加コストが問題になりにくいものがほとんどであった．また，こうした開発では古典的なウォーターフォールモデルが用いられており，要求の変更といった不安定な開発プロセスへの対応は十分検討されているとは言えない．

しかしながら，システム開発の多くは，既存のシステムに対する拡張であり，まったく新規の開発を行なう場合はあまりない．このような場合には，既存システムに対する追加要求を特定し，それらを詳細化し，既存システムと融合させた後に検証する必要がある．これは，フォーマルメソッドを用いるかどうかに依存しない．

本研究で想定する開発プロセスの大枠は,『共通フレーム 2007』[19)] にあるように，発注者からの「要求」に基づいて，受注者が「要件定義」を行ない，合意した「要件定義」すなわち，仕様に基づいて「設計」「実装」を行なうというものである．ここで，各段階の定義は

- 「要求」 ...発注者の関心事で，開発対象となる計算機システムが最低限満たさなければならない項目を抽出する．自然言語による記述が想定されており，システム開発の観点からは本質的に曖昧さを含み不完全な記述である．
- 「要件定義」 ...受注者が「要求」をとりこみ，曖昧さを排除した定義を行なうとともに，実現可能性を考慮して欠けている記述を補完し，対象システムの開発に必要な情報を明示する.「要求」のとりこみ結果だけでなく，場合によっては，欠けていた記述についても発注者に確認をとる．
- 「設計」...「要件定義」に基づいてシステム依存の詳細化や最適化を行なう．この段階の成果は後継システムや派生開発では保存されない可能性がある．
- 「実装」...「設計」に基づいて実行可能なプログラムを作成する．

である．本稿では，上流工程，特に要件定義部分に注目しているので，他の段階はおおまかな区分に留まる．

このとき，図 1 の日本情報システム･ユーザー協会による報告書「企業 IT 動向調査 2009」[20)] に示すように，要件定義が明確になっていることがプロジェクト成功の鍵となっている．

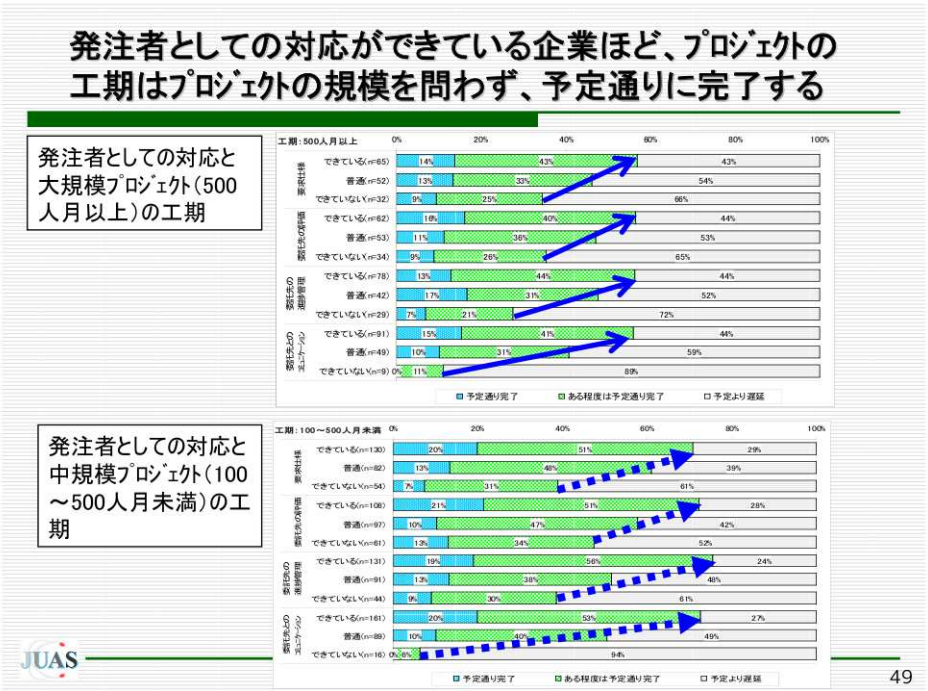

図 1 要件定義の重要性（「企業 IT 動向調査 2009」[20] より）

つまり，上流工程における設計の抜け・漏れを防ぐためには，要件定義の段階において統一性 (integrity) のあるモデル化を行ない，発注者の要求を要件定義に確実に取り込むと共に，要求に含まれない部分を補完することによって仕様の妥当性 (feasibility) を確認する方法が有効である．我々は，前者について自然言語からフォーマルメソッドによるモデルへの変換補助ツールを提案しており[25]，本稿では後者について検討する．

仕様の妥当性を確認するにあたっては，追加する機能は当然として，性能，安定性，拡張性といった点についてもモデル上での確認が望ましい．こうした非機能要件は，モデルのふるまい上で確認できる．したがって，フォーマルメソッドを用いた上流工程のモデルでは，ユーザ機能に関する要求をふるまいモデルへの変換およびふるまいモデル上での記述補完による要件定義が求められる．

2.1 差分開発への適用

仕様は機能単位で記述されることが多く，非機能要件は関連する機能に付随して記述されることが多い．ここで，システム内部で利用する機能をシステム機能，システム外部へ提供する機能をユーザ機能またはサービスと呼ぶ．追加される機能はユーザ機能であり，ユーザ機能を実現可能なシステム機能が用意されていれば，それらを組合せたふるまいにより妥当性が確認できる．システム機能が不足している場合，仕様が明示されなくとも，対応するシステム機能の追加が必要となる．これは，ソフトウェアによるシステム機能だけでなく，ハードウェアによるシステム機能や運用による制約なども含む．

モデルを用いたソフトウェア開発を差分開発へ応用するにあたっては，新規要求へ対応するとともに，既存のシステムについても含めた要件定義を行ない，全体を検証する必要がある．そうでない場合，不足しているシステム機能が発見できない．また，互換性を保つ部分と，修正に伴い変更可能な部分の切り分けもはっきりしない．図 2 に差分開発に関する概略を示す．

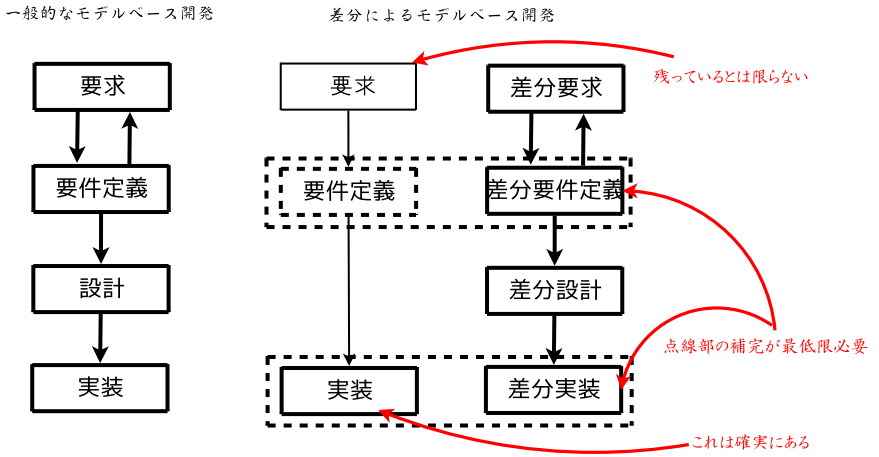

図 2 モデルを利用した差分開発

差分開発におけるもう一つの問題は，しばしば設計における中間生成物失われていることである．この結果，設計情報が属人的になり，問題となることがある．フォーマルメソッドを用いた明示的な要件定義は，開発の引継ぎや組織化にも有用な場合がある．

2.2 複数モデルの結合

フォーマルメソッドの組込みシステム開発への応用は既に数多くの適用例がある．要求は発注者と受注者の間で，ユーザ機能ごとにやりとりされることが多い．したがって，要件定義はユーザ機能ごとの記述に適した手法が望ましい．

その一方，システム開発にあたって，ユーザ機能を詳細化しシステム機能へ分割する部分は問題ないが，それらを組み合わせたふるまいに関する部分は，仕様記述者あるいは設計者がすべての場合を考慮できず，予想外のふるまいによる不具合を生じる場合がある．

そこで，本研究では，要件定義段階のモデル化を機能中心に行ない，それをふるまいモデルへ変換して検証する手法を検討する．これにより，両者の利点の組み合わせが期待できる．具体的には，機能モデルの記述に VDM を，ふるまいモデルの記述に Promela を用いる．

VDM は， Vienna definition language (VDL) と呼ばれる形式的かつ機械独立なプログラム言語の定義法を用いて，ソフトウェアの仕様を段階的に記述する方法である．VDL として 1996 年に ISO/IEC 13817-1 として標準化された VDM-SL と VDM++ がある．VDM++ はオブジェクト指向設計およびリアルタイムに関する性質の一部をサポートする記法が拡張されており，並列・並行処理の記述も可能である．VDL は集合論と一階述語論理を基礎とし，プログラム言語の意味する本質的な部分を集合や写像などを表す数式によって表現する．また検証ツール VDMTools[18] により文法検査，モデルの型検査，アニメーション機能が提供されており，計算機を用いたモデルの検証が可能である．

Protocol or Process Meta Language (Promela) は， 1980 年代後半に AT&T Bell Lab. UNIX group に属していた Gerard J. Holzmman によって開発された言語で，モデル検査のためのモデル記述言語である．Promela の開発当初の目的は通信プロトコルの検証であったが，現在は一般的な並列・分散システムのソフトウエアの検証に変化している．Promela は状態を保持するプロセスとプロセス間の通信を行なうチャネルによりシステムを記述し，並行プロセスとしてモデル化する．Promela により記述されたモデルは SPIN モデルチェッカにより，モデル検査を行なうことができる．

モデル検査とは、モデル化されたシステムの振る舞いが、ある特定の性質を満たすかどうかを調べる検証手法である。モデル検査は，モデル上の取り得る全ての状態について，その特定の性質を満たすかどうかを調べる手法であるため，通常は計算機上で検証が行われる．検証対象の性質は，時相論理 Linear Temporal Logic (LTL) を用いた式によって表され，その式をシステムのモデルが満たすかどうか検証を行って確認する．

3. モデル変換による検証

本研究の想定する開発手順では，第 2.2 節で検討したように，複数のフォーマルメソッドを併用する．ただし，一般的に用いられるフォーマルメソッドのほとんど，少なくとも本研究で対象とする手法は，開発対象システムを有限状態機械とみなして抽象化するという共通性があるので，これを手がかりにモデルの相互変換を行なう．

ただし，発注者の関心事による要求の違いによる記述の濃淡はもちろん，要求手法によって観点が変わる．よって，受注者は要件定義段階において，単にモデルを変換するだけでなく，観点の違いにより見つけ易くなる，対象システムの記述されていない状態または遷移を補完する．さらに発注者へ補完した情報を提示することでより抜け・漏れのない記述，すなわち品質の高い仕様を得ることができる．

さらに，システム自体の動作について，デッドロックやライブロックといった異常状態はあるが，一般にある状態が異常かどうかは外部環境との関係で決まる．例えば，斜めの机の上や高速で移動中の車内で電気ポット内の水が片寄り，水位センサーが水位を誤検出し，本来の最低水位を割った状態で加熱するという場合，システムとしては異常な動作ではないが，ポットを安全に使う環境も含めて考えれば異常とするべきである．このため，機能モデルからふるまいモデルへの変換では，有限状態機械のどの状態を異常状態として検証するかを選択するために，どの範囲までを系として検証するかという判断も要求される．また，遷移元および遷移先は異常でなくとも遷移自体は異常である場合もある．特にユーザが直接操作するような組込みシステムでは，さまざまな使用環境が想定されることから，こうした判断が重要となる．

3.1 状態遷移図への変換

Jeremy Dick らは， VDM からテストケースを生成する手法を提案している[3]．この中で状態遷移パステストケースを生成するためのに VDM による仕様記述から有限状態オートマトン (FSA) を生成する手順が述べられている．本節では，この手法について説明する．

Dick らはまず Partition analysis により, FSA の状態遷移に対応するサブ操作を抽出する．Partition analysis は $invariant \wedge pre - condition \wedge postcondition$ を加法標準形に変換することで得られる．図 3 に変換例を示す.

続いて, すべての操作と初期状態に対して Partition Analysis を行なうことで, 単純化したサブ操作を抽出する.

このとき，事前に, すべての参照やエイリアスは展開しなければならない．さらに，各サ

$$
\begin{array}{lll}
MAX(a:Z, b:Z) & & \{max = a \wedge max = b, \\
wrmax : Z & \Rightarrow & max = a \wedge max > b, \\
post(max = a \vee max = b) \wedge & & max = b \wedge max > a\} \\
max \geq a \wedge max \geq b & &
\end{array}
$$

図 3 不変・事前・事後条件の変換例

ブ操作の前状態と後状態の制約の集合を抽出する．図 4 に制約集合の抽出例を示す．

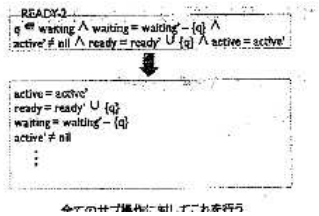

図 4 制約集合の抽出[3)]

続いて，図 5 に示すように，全サブ操作に関する制約の集合から状態を抽出する．

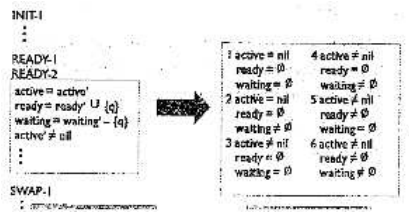

図 5 状態の抽出[3)]

最後に得られた FSA から path を網羅する path test のテストケースを生成する．

Dick らのアルゴリズムは特別な工夫はないが，数学的な正しさは保障されている．このため，発注者の要求を正しく VDM で記述すれば，それをまちがいなく要件定義におけるふるまいモデルへ取り込むことができる．

3.2 同期モデルへの変換

ふるまいモデルから実行可能なプログラムへ変換する際に，ユーザが直接操作するような組込みシステムでは，時間に関する性質を考慮する必要がある．このようなシステムでは，電子ポットで水を加熱したり，ハンドルを操作したりといった，外界へ影響を及ぼすような動きを伴うハードウェアの制御を行なう場合が多い．

遅延 物理的な動作は，計算機内部の演算に比べて無視できない遅延を必ず生じ，制御に関わる待ち状態や過渡状態を必要とする．このため，制御プログラムの設計はソフトウェア設計に用いている状態遷移図ではなく，制御ブロック図などの記法に変換される．

しかし，いったんブロック図に変換されてしまうと，例えば入出力データの意味は信号線の接続に，事前条件や事後条件は特定ビットのフラグに置き換えられてしまい，ここまで数理的な性質を保存しつつ詳細化してきた設計情報が失われる．

離散 時間の連続性に対する見方も外界における連続的な時間から，制御するための離散時間へ変わる．たとえばブロック図では制御ループによる処理が一般的であり，同じイテレーションにおける処理は同時に処理されるが，StateCharts[7)6)] や UML2 の状態遷移図では同時のイベントという概念はない．つまり，離散時間の世界になると，ある時区間における処理は同時とみなされるため，実用的な意味で同期処理が一般的になり，同期したイベントの処理に関して優先度を考慮する必要も生じる．

しかしながら，この変化に伴って外部からのイベントの意味が変わってしまい，ふるまいモデルの形も変わってしまうので，ここも意味的な乖離の原因となる．物理的な操作を伴う組込みシステムの設計段階における検証が難しい原因は，厳密なリアルタイム性の確保よりも，こうした連続時間と離散時間のギャップにある．

過渡状態 システムのふるまいモデルに，過渡状態を考慮しなくてはならない．ただし，ここで考慮するのはシステムとしての過渡状態一般ではない．ある遷移が過渡状態をもつか，そうでないかは外界と干渉する部分の遅延により決まり，制御を行なう FSA としては，そのような遷移中に対応する独立した状態として表現される．

シミュレーションによる検証は，計算機により離散的な処理を上手く扱えるので，現在のモデルを用いた仕様検証や設計検証で広く用いられている．たとえば VDM では VDMTools 上のアニメーション機能により，特定の入力データに関するふるまいを仕様のシミュレーションにより確認できる．

別の選択肢として，このようなギャップを上手く扱える論理体系のモデルへの変換がある．この場合，遅延や過渡状態を表現するために時区間論理の適用が有力である．

4. 事例研究

組込みソフトウェア開発の事例へ適用した．具体的には,「話題沸騰ポット（GOMA-1015型） 要求仕様書 第7版」[23] を利用し，フォーマルメソッドによるモデル化を行なった．

4.1 電子ポット

電子ポットは，電気により水温を調整する機能を備えたポットである．図 6 に機能に関する要件定義の一部であり，第 2.1 節で述べたユーザ機能に相当する．

```
class 『ポット』
types
『温度』 = real inv temp > -10 and  temp < 110;
『水位』 = int inv height >= 0 and height <= 100;

instance variables
ヒーター : <on> | <off> ;
コンセント : <差された> | <差さっている> | <抜かれている> ;
蓋 : <開> | <閉> | <ロック> ;
タイマー : 『タイマー』;
パネル : 『パネル』;
ブザー : <鳴っている> | <鳴っていない> ;

モード : <アイドル> | <沸騰行為> | <高温保温> | <節約保温> | <ミルク保温> | <エラー> ;
水位 : 『水位』;
水温 : 『温度』;
加熱可能水位 : 『水位』;
目標温度 : 『温度』;

operations
public 給湯する : () ==> ()
給湯する () ==
is not yet specified
pre 水位 > 0;

public 給水する : () ==> ()
給水する () ==
is not yet specified
post 水位 > 0;

public 加熱する : () ==> ()
加熱する () ==
is not yet specified
pre 目標温度 > 水温 and 水位 > 加熱可能水位
post 目標温度 <= 水温;

end 『ポット』
```

図 6 VDM によるユーザ機能記述 (一部)

この仕様書は，第 2 章における要求に相当し，かなり詳細に記述されているものの，例外処理を中心に受注者に任せられている部分がある．不足した記述を要件定義の中で補完して検証するとともに，差分開発を仮定したいくつかの機能追加について試行する．

(1) 保温モードに，水温を 60 度に保つ玉露モードを追加する．

(2) 加熱中に給湯し，加熱可能水位を下回る．

といったシナリオを事例として考える．関連したシステム機能に関わる変数を図 7 に示す．

```
ヒーター : <on> | <off> ;
蓋 : <開> | <閉> | <ロック> ;
水位センサー 1 : <filled> | <empty> ;
モード : <アイドル> | <沸騰行為> | <高温保温> | <節約保温> |
         <ミルク保温> | <玉露保温> | <エラー> ;
設定水温 : 『温度』;
水温 : 『温度』;
```

図 7 VDM によるシステム機能記述 (一部)

4.2 ふるまい仕様

第 4.1 節の要件定義から，ふるまいモデルを生成し，検証した．具体的には，システム機能に関する変数の全ての組合せにより状態機械を定義する．

4.2.1 玉露保温モードの追加

玉露モードは，既存のミルク保温モードや節約保温と類似したユーザ機能であり，設定温度が異なるだけである．実際，図 8 に示す既存のシステム機能の変数「設定水温」を変更するだけで実現できることが分かる．

4.2.2 加熱可能水位のエラー

ユーザが行なう外的な操作により，加熱中の水位が減少する場合がある．この場合，加熱可能水位を下回った状態でヒーターによる加熱が行われるという，禁止されている状態になる可能性がある．

ただし，すぐに分かるように，これは図 8 に示す連続時間における状態遷移では表現されない過渡状態であり，回復可能なエラーとして処理されるべき状態である．このように，連続時間における状態遷移と離散時間における状態遷移では，それらの間で本質的に対応する状態であっても異なる処理となる場合がある．

同様に，蓋が空いた状態ではヒーターは on にならないが，この操作はユーザが任意のタイミングで行なう可能性があり，ヒーターの on/off が任意のタイミングで生じ得る．

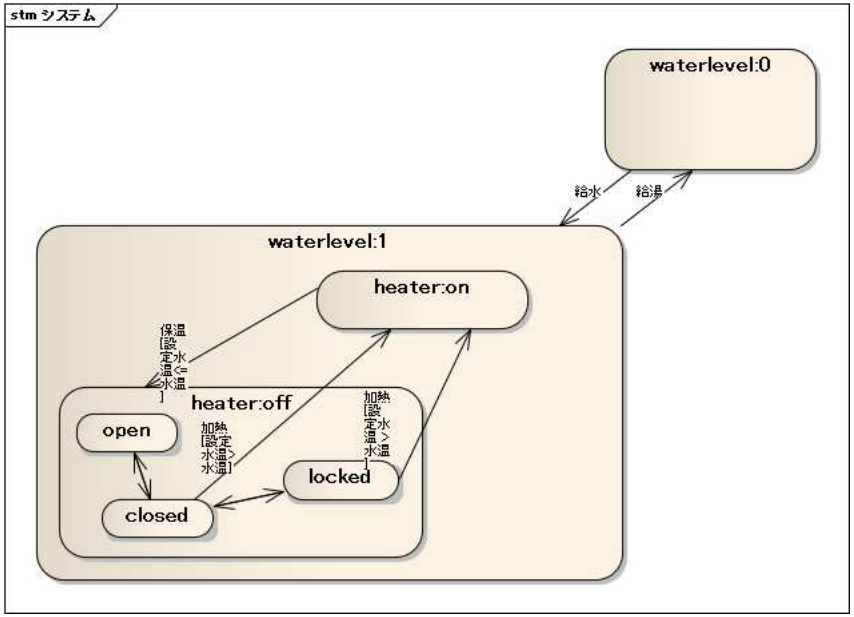

図 8 既存モードのふるまい

5. ま と め

本研究では，複数のフォーマルメソッドを用いる開発プロセスについて考察し，ユーザ機能の記述を行なう要求定義と，仕様の抜け・漏れを防ぐためのふるまい検証による要件定義の組み合わせを提案した．その際，前者から後者へ変換し一貫性の高いモデル上で，数理的な妥当性を検証できる．

また，モデリングそのものの難しさはあるが，上流工程においてフォーマルメソッドを適用し，数理的な性質を保存したまま実行可能なプログラムへ変換するための課題として，外界における遅延など時間に関する性質の検証が重要であることを述べ，それらを分類し，その分類に応じてモデルで検証すべき性質の切り分けを行えることを示した．本手法により，ソフトウェア分析と制御プログラムとのより深い結合が可能となる．さらに，電子ポットの例題により提案手法を評価し，簡単な検証を行なった．

今後の課題として，より洗練された論理体系の構築とモデル変換手法の確立が挙げられる．本手法における課題が一般的な時区間論理で検証可能なのは明らかであるが，記述能力が大きすぎて扱いが難しいからである．

謝辞 本研究の一部は科研費（21300009），基盤研究（B）「ソフトウェア開発の現場で使えるフォーマルメソッドに関する研究」の助成を受けたものである．

参 考 文 献

1) Abrial, J.-R.: *The B-Book: Assigning Programs to Meanings*, Cambridge University Press (1996).
2) Bächle, M. and Kirchberg, P.: Ruby on Rails, *IEEE Software*, Vol.24, No.6, pp. 105–108 (2007).
3) Dick, J. and Faivre, A.: Automating the generation and sequencing of test cases from mo del-based specifications, *Proceedings of First International Symposium of Formal Methods Eu rope*, pp.268–284 (1993).
4) Gerhart, S., Craigen, D. and Ralston, T.: Experience with formal methods in critical systems, *IEEE Software*, Vol.11, No.1, pp.21–28 (1994).
5) Hansen, K.M.: Validation of a Railway Interlocking Model, *Proceedings of Industrial Benefit of Formal Methods*, pp.582–601 (1994).
6) Harel, D.: Statecharts: A Visual Formalism for Complex Systems, *Science of Computer Programming*, Vol.8, No.3, pp.231–274 (1987).
7) Harel, D., Pnueli, A., Schmidt, J.P. and Sherman, R.: On the Formal Semantics of Statecharts (Extended Abstract), *Proceedings, Symposium on Logic in Computer Science*, pp.54–64 (1987).
8) Hinchey, M.G. and Bowen, J.P.(eds.): *Applications of Formal Methods*, Prentice Hall (1995).
9) Holzmann, G.J.: The Model Checker SPIN, *IEEE Transactions on Software Engineering*, Vol.23, No.5, pp.279–295 (1997).
10) Jones, C.B.: *Systematic Software Development using VDM*, Prentice-Hall (1990).
11) MathWorks: http://www.mathworks.com/products/simulink/requirements.html.
12) Mellor, S.J. and Balcer, M.: *Executable UML: A Foundation for Model-Driven Architectures*, Addison-Wesley (2002).
13) Ogata, K. and Futatsugi, K.: Modeling and Verification of Distributed Real-Time Systems Bas ed on CafeOBJ, *Proceedings of the 16th IEEE international conference on Autom ated software engineering*, IEEE Computer Society (2001).
14) SESSAME WG 2: 組込みソフトウェア開発のためのオブジェクト指向モデリング，翔泳社 (2006).
15) Schneider, S.: *Concurrent and Real Time Systems: The CSP Approach*, John Wiley & Sons, Inc. (1999).

16) Vanbrabant, R.: *Google Guice: Agile Lightweight Dependency Injection Framework*, Apress (2008).
17) Woo, H., Mok, A.K. and Browne, J.C.: Hybrid Framework for Resource Verification in Executable Model-based Embedded System Development, *ACM SIGBED Review*, Vol.5, No.1, Article No.5 (2008).
18) CSK システムズ：VDMTools, http://vdmtools.jp/.
19) 情報処理推進機構ソフトウェア・エンジニアリング・センター（編）：『共通フレーム 2007——経営者、業務部門が参画するシステム開発および取引のために』，オーム社 (2007).
20) 社団法人日本情報システム・ユーザー協会：企業 IT 動向調査 2009，技術報告，http://www.juas.or.jp/servey/it09/index.html (2009).
21) 情報処理推進機構：2007 年版組込みソフトウェア産業実態調査報告書，技術報告，独立行政法人 情報処理推進機構 (2007).
22) 中島　震：SPIN モデル検査 – 検証モデリング技法，近代科学社 (2008).
23) 組込みソフトウェア管理者･技術者育成研究会：話題沸騰ポット（GOMA-1015 型）要求仕様書，http://www.sessame.jp/workinggroup/WorkingGroup2/POT_Specification.htm.
24) 日経エレクトロニクス（編）：組み込みソフトウエア２００７モデルに基づく開発方法論のすべて，日経 BP (2006).
25) 大森洋一，荒木啓二郎：自然言語による仕様記述の形式モデルへの変換を利用した品質向上手法，情報処理学会研究報告, 2010-PRO-079，Vol.2010, No.79, pp.1–7 (online) (2010).