Vol.2015-SE-190 No.15
2015/12/16

ポットの例題による「手軽さ」を考慮したフォーマルメソッド適用の検討

大森 洋一[1]　林 信宏[1]　日下部 茂[1]　荒木 啓二郎[1]

概要：これまでのフォーマルメソッドを利用したソフトウェア開発は，数理的証明やモデル検査などリスクを 0 にする検証が強調されてきた．しかし，最近のソフトウェア開発は従来の手法では達成できなかった開発期間の短縮や複雑な機能を達成するためにリスクを容認するようになっている．本研究では，リスクに応じてフォーマルメソッドの適用強度を変えることにより効率的なソフトウェア開発を行う手法を提案する．特に安全性について，自然言語による要求記述に対するゴール分析に基づくリスク評価を行ない，エンドユーザ向け製品の仕様としてしばしば表れる「手軽さ」をフォーマルな仕様としてどのように扱うかを検討する．さらに，このリスク評価に基づいたフォーマルメソッドの適用強度の変更について評価する．具体的には，話題沸騰ポットの例題に対して，本手法を適用する．

キーワード：フォーマルメソッド, 要求分析, 仕様記述, プリフォーマル

Study about Application of Formal Methods in Consideration of ”Convenience” by the Example of the Electric Pot

Abstract: It had been emphasized that formal methods in software development are verification techniques reducing risk to 0, such as mathematical proof and model checking. However, modern software development accepts risks in order to achieve more complex functions and shorter development period, which could not be achieved by conventional techniques. In this paper, we propose a method for efficient software development by changing the application intention of formal methods, depending on the risks. We analyzed risks on the requirements in a natural language for safety by the goal analysis, and considered how to deal ”convenience” in formal specification which often appears in the specification of end-user product.
in particular it was subjected to risk assessment based on
Specifically, for the example of buzz pot, consider the procedure for applying this method.

Keywords: Formal method, Requirement Analysis, Specification Description, Pre-formal process

1. はじめに

現在の高度化したコンピュータシステムは，その代償としてさまざまなリスクを内包している．また製品の市場投入開始時期 (Time To Market， TTM) が重視され，従来の手法で必要とされていた開発期間をさらに短縮するためにリスクを許容した商品開発が行われている．特に先端的な分野においては，高機能化や早期 TTM はユーザに高く評価され，それに伴うリスクがある程度容認されている．これはネットワーク接続の普及やソフトウェアによる機能実現によって，機能的欠陥を稼働後に改修できる技術的環境が整ってきたからという側面もある．しかし，これはコンピュータシステムのリスクが全般的に容認されたということではない．むしろ，ネットワーク接続と機能のソフトウェア化に伴い，ソフトウェアの不具合に伴う負の影響も大きくなっており，製品が広く使われるためには，ソフトウェア・アップデートなどを通じて潜在するリスクを問題が顕在化しないように下げる必要がある．すなわち，従来よりもソフトウェア開発，運用，保守におけるリスク管理の重要性が増している．

フォーマルメソッドは，計算機システムの仕様を数理的

[1] 九州大学大学院システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu Univ ersity

に表記することおよび，その仕様を設計の検証基盤として利用する手法である [3]．フォーマルメソッドは，その基盤とする数理体系上で開発対象をモデル化することによって，開発の上流工程からの数理的な仕様検証を可能とし，不具合を早期発見して手戻りを防ぐために効果的であることが知られている．その一方，証明により仕様記述段階におけるリスクを 0 にできることを強調されすぎたために，フォーマルメソッドはリスクを許容する現代的なソフトウェア開発に適用が困難であるとみなされる場合も多い．実際には，ソフトウェア開発を詳細に検討すれば，リスクを 0 にしなければならない部分，低いリスクであれば許容される部分，機能と引き換えに高いリスクが許容される部分などさまざまな部分がある．また，リスクの種類も品質に影響するもの，コストに影響するもの，発生頻度は低いが影響が大きいもの，発生頻度は高いが影響が小さいものなどに分類される．本稿では，こうしたソフトウェア開発における機能モジュールや開発フェーズ毎に分類されたリスクに応じてフォーマルメソッドの適用強度を変えることにより，リスクと開発コストの両方を最適化するソフトウェア開発を検討する．

フォーマルメソッドの適用強度とは，どれほどの手間をかけて対象の数理モデルとして記述し，検証するかという基準である．フォーマルメソッドの適用は，開発対象が満たすべき性質を，それぞれの手法が基盤とする数理上の性質へ帰着させる作業である．この過程で，対象の性質は抽象化され，数理モデルとして記述される．この記述過程をフォーマルなモデル記述，モデルに対する数理的な性質の検証をフォーマルなモデル検証と呼ぶ．フォーマルなモデル記述にあたっては，対象とする機能の詳細度，数理モデルの抽象度，環境の作り込みなどに関する選択肢があり，フォーマルなモデル検証にあたっては，完全な証明，限定した範囲に関する証明，直観的な証明などの選択肢がある．すなわち，適用強度が低い場合，例えば特定の入力値についてのみ証明すればモデル記述およびモデル検証にかかる適用コストは小さくできるが，異常入力値に対する振る舞いや，保守工程における移植において障害が発生する潜在的なリスクは高いままとなる．逆にモデル検査などを用いてフォーマルメソッドの適用強度を高めれば，想定されるあらゆる入力値に対して仕様を満たすことを証明可能となり入力値に対するリスクを極小化できるが，適用コストも高価となる．

2. ソフトウェア開発におけるリスクの定量化

許容すべきリスクを判断するには，まず出発点となる要求分析の段階からリスク分析を行う．ソフトウェア開発は自然言語による要求記述から始める場合がほとんどであるが，自然言語による要求記述には曖昧さや矛盾が含まれることが多く，実際ソフトウェア開発における不具合の多くは，不完全な要求や矛盾した要求に起因することが知られている [1]．こうした仕様記述段階での不具合を解消するにあたり，まず不具合を解消する優先順位を設定しなければならない．リスク評価の分析は，まず安全性 (safety) を対象とする．安全性は非機能要求の中でも重要かつ運用後の修正が困難であること，先行研究も豊富であることがその理由である．

2.1 Dependable Case の適用

要求分析の工程は，

(1) 要求抽出 ...顧客からシステム化に必要な要求を漏れなく引き出す．
(2) 要求整理 ...顧客の要求を整理し，優先順位をつける．
(3) 仕様記述 ...要求を仕様として記述し，顧客と合意する．
(4) 仕様検証 ...仕様の無矛盾性を検証し，妥当性，実装可能性を確認する．

に分けられる [10]．現在のフォーマルメソッド適用は主に工程 3 と 4 を対象としており，工程 1, 2 の検討が少ない．

本研究では，ゴール指向分析の中でも安全性を考慮した分析手法である Dependable Case (D-Case) を適用する [8]．安全性ケースは，テスト結果や検証結果を根拠にシステムの安全性を議論し，システム認証者や利用者などに保証する，あるいは確信させるための構造化されたドキュメントであり， D-Case はデペンダビリティに関するケースを表現することを目的とした安全性ケースの一種である．D-Case は以下に挙げるノードの関係を Goal Structuring Notation (GSN) を用いて表現する [5]．

- ゴール (Goal)
  対象システムに対して議論すべき命題
- 戦略 (Strategy)
  ゴールが満たされることを、サブゴールに分割して詳細化するときの議論の方針や視点
- 前提 (Context)
  ゴールや戦略を議論するとき、その前提となる環境や条件
- エビデンス (Evidence)
  ゴールを最終的に保証する情報
- 未達成 (Undeveloped)
  ゴールを保証する先の議論やエビデンスの不在
- モニタ (Monitor)
  運用中のシステムより取得可能なエビデンス
- 外部接続 (External)
  他のシステムの D-Case へのリンク

リスク定量評価の基本的なアイデアは，安全性を考慮したゴール分析手法である D-Case により開発対象の要求を

木構造にを整理し，

(1) Goal Rank をトップゴールからのグラフのノード間距離として定義する．

(2) 想定されるハザードごとに機能的な Root Cause Analysis (RCA) を適用する．

(3) RCA により，リスクごとの根本原因をゴールグラフのノードへ帰着させる．

(4) 帰着したノードのリスクが高い場合，強度のフォーマルメソッド適用により高い保証を与える．

(5) 帰着したノードの Rank が相対的に高いリスクは検証の優先度を上げる．

(6) 帰着したノードのリスクが低い場合や rank が相対的に低いリスクは，実行可能な仕様を特定の値についてのみテスト実行したりといった，限定的な保証を与える．

という手順によりフォーマルメソッドの適用強度を選択するというものである．

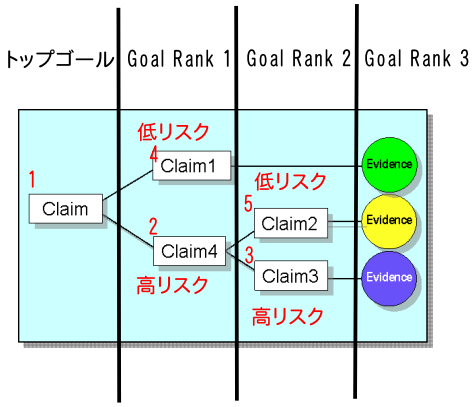

図 1 Goal Rank の概念図 (http://www.sei.cmu.edu/ dependability/tools/assurancecase/ の図を一部改変)

図 1 に概念図を示す．この場合の検証優先順位は，図に示すようにリスクが高いゴールを優先し， Goal Rank が高いゴールを定量的に決定する．Goal Rank はトップゴールを Rank 0 とし，トップゴールからの距離によって定義する．Goal Rank が高いゴールは開発対象システムにおいてより重要な要素であるので，実現および検証を優先する．したがって，このゴール指向分析によるゴール木作成の段階で要求の選別と優先順位づけを行なっておく必要がある．個々のゴールに対するハザード分析は問題領域に対する専門知識が必要だが，このゴールモデルは従来のレビューでは暗黙の前提とされていた部分も含めて問題領域の知識体系を一部分であっても明示した表現となり，再利用が容易となるという効果も期待できる．現実のソフトウェア開発プロジェクトの多くは，時間的あるいは資源的な制約により，フォーマルメソッドの適用が限定的にならざるを得ない．このような状況で，開発対象のどの部分を優先的に記述し，検証するべきかという指標が求められおり，対象システムに対する早期のハザード分析およびそれらのリスク分析が重要となってきている．本提案手順により，より重要性の高いゴールを脅かすハザードを特定し，そのリスクに関連する仕様を優先的に検証することで，限られた開発リソースを効果的に活用することを期待する．

2.2 Goal Oriented Analysis

最初に GSN を用いた要求のゴール指向分析を行う．GSN は，トップゴールを根とする木構造であり，ある戦略に基づいてゴールをサブゴールへ階層的に分割する．GSN における分割の戦略は任意であるが，実際はいくつかのパターンに整理される [4][12]．しかしながら，あるゴール G とその直接のサブゴール群 $G_1, G_2, \ldots G_n$ に対して， G が達成される十分条件は， G が前提条件 G_{Pre} を満たしかつ $G_1, G_2, \ldots G_n$ がすべて満たされるという関係になっていなければ，個々のゴールについてフォーマルな仕様記述および検証を行う意味がない．つまり，これらの要素が

$$G_{Pre} \implies (G_1 \wedge G_2 \wedge \ldots G_n \vDash G) \qquad (1)$$

を満たしていることが前提となる．同じゴール指向分析手法の KAOS では，それぞれのゴールは Liner Temporal Logic による命題として記述される [11]．また，ゴールとサブゴールの関係は完全であることが要求されるので，式 1 を満たす．GSN にもネットワーク構造に関する研究はあるが，各ゴールの意味論と関係づけられていない [2]．その代わり KAOS と比べて複数の視点による分析が可能となっており，複数の戦略による多様なゴールおよびサブゴールへの分割戦略を表現できる．D-Case では一般的な GSN によるゴール木よりも厳密な定義が試みられており，ゴールは Propositional Logic の範囲における意味論の検討が行われている [7].

2.3 Functional Hazard Analysis

次に，開発対象のリスクを分析する．リスク分析手法として一般的な FTA や FMEA は，リスクの原因を明らかにすることを目的としている．このような手法は Root Cause Analysis と呼ばれ，ここではまず機能的な要求を対象とする．しかし，ハードウェアに対する RCA は大きな成果を挙げているが，ソフトウェアに対する RCA は原因と結果の因果関係が明確でないことから，プログラムレベルの特定機能に対する解析に限定されている [9]．この問題を解決するために，開発対象に対して想定される機能的なハザードについて要求のゴール木に還元するように RCA を適用することで，要求分析および仕様記述段階でのリスク評価を可能とする手法を提案する．構造化した要求であるゴール木は，開発対象システムの実現すべき性質をその

優先順位とともに明示的に表現し，開発関係者の合意を得ることを目的としている．それぞれの機能ハザードおよびその根源的原因を要求分析結果としてのゴール木と結びつけることにより，実現すべき要求の優先順位による重み付けが可能となる．

開発にかけられる時間が限られる現代のソフトウェア開発では，要求分析のような早期の開発段階でこうした要求およびそれらの関係が整理され，優先順位をつけられることは，以降の開発工程においても有用である．

2.4 VDM と D-Case の組み合わせ

要求をフォーマルな仕様として書くためは，仕様記述者の開発対象に対する高い理解と数学的なセンスや経験が必要となる．数学的なセンスは，対象のシステムの数理的な性質を見抜き，フォーマルな仕様として数理モデルを記述する能力である．これに対して 2.1 章の手順は，フォーマルメソッドあるいはソフトウェアの専門家でなくとも，開発対象に関する知識があれば，ある程度機械的な手順で要求を分析できる．その成果であるゴール木は，関係者の要求に関する合意としての仕様の基礎となりうる．このように整理された要求をフォーマルなモデルとして記述することで，さらに自然言語を用いることによる解釈の曖昧さを排除することができる．このようなフォーマルメソッドを適用する予備的な工程をプリフォーマルな処理と呼ぶ．プリフォーマルな処理は，最終的にフォーマルな仕様を書くかどうかに関わらず，重要な工程である．

本研究では，フォーマルな仕様の記述および検証には VDM++ を採用した．VDM++ は，一階述語論理と集合論に基づく意味論をもつフォーマルメソッド VDM の仕様記述言語である [6]．VDM は対象の仕様を有限状態機械によるモデルとして記述していくことに重点をおいており，数学的な仕様から実装に近い仕様までさまざまな抽象度で記述可能である．したがって，2.2 節で示したようなある種の形式性を備えた D-Case の一部を等価な VDM による表現で置換することが可能であり，逆も可能である．ただし D-Case の記述は元来の定義よりも制限されているものの，独自の意味論をもっており， VDM すなわち仕様記述および検証に用いるフォーマルメソッドも独自の意味論をもつ可能性がある．つまり，ここで検討する互換性は仕様の特定部分に関するものである．言い換えれば，ある前提であるゴールが成り立つことを，特定の述語とその事前条件・事後条件で表現できる．

3. 話題沸騰ポットの事例への適用

本研究では，「組込みソフトウェア管理者・技術者育成研究会」(SESSAME: Society of Embedded Software Skill Acquisition for Managers and Engineers) が教育教材として公開している「話題沸騰ポット（GOMA-1015 型） 要求仕様書」第 7 版 (以下，要求仕様書) を用いて， D-Case およびそれに基づいた VDM++ モデルの作成および検証を行った [13].

3.1 D-Case による分析

電気ポットのトップゴールとしてまず思いつくのは「湯を沸かすこと」である．しかし，湯を沸かすだけでは電気ポットのトップゴールとしては十分でない．湯を沸かす手段として，家庭あるいは職場において多くの場合電気ポットが使われる理由を，他の湯を沸かす手段と比較しながら考えると「手軽さ」が重要な概念であることが分かる．図 2 に話題沸騰ポットの D-Case のトップゴールを示す．実際には「湯を沸かす」ことより「手軽さ」に関するサブゴールが圧倒的に多くなった．要求仕様書は教育目的であることから「湯を沸かす」機能に関する記述だけであることも理由であるが，「手軽さ」は非機能要求であり，ポットの様々な機能およびハードウェアにも関係することからこの結果自体は驚くことではない．安全性を考慮したより現実的な仕様記述でも，この要求仕様書同様に記述量がかなり大きくなることが予想される．

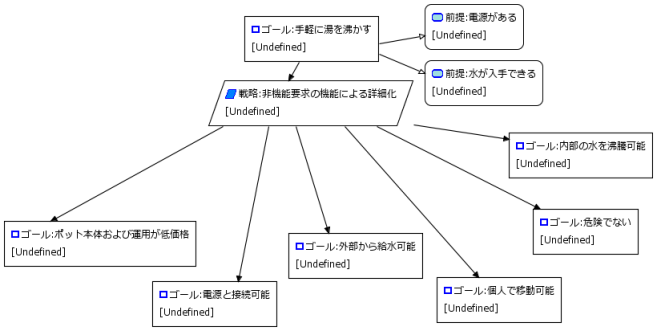

図 2 話題沸騰ポットのトップゴール

さらに，図 2 からサブゴールへの分割を進めた．ここでは，ハードウェアに関するゴールは don't care とみなし，ソフトウェアによる機能実現の事例として，図 3 に過熱防止に関する部分を示す．

過熱防止は「湯を沸かす」ゴールと「危険でない」ゴールに関連する．ただし，要求仕様書に過熱防止に関するゴールは明示されておらず，関連する記述は，

> POT-500: 想定外の高温状態になった場合はエラーを検知し、ヒータ機能を停止する。

のみである．これをそのまま仕様としてみると，話題沸騰ポットには単独の「過熱防止機能」があるのではなく，ヒータ機能と温度センサーの相互作用における制約条件となっている．

「湯を沸かす」ゴールについてのハザードを考えると，「想定外の高温状態」は，「湯を沸かす」ゴールのサブゴール「指定した湯温で PID 制御する」を原因とする指定し

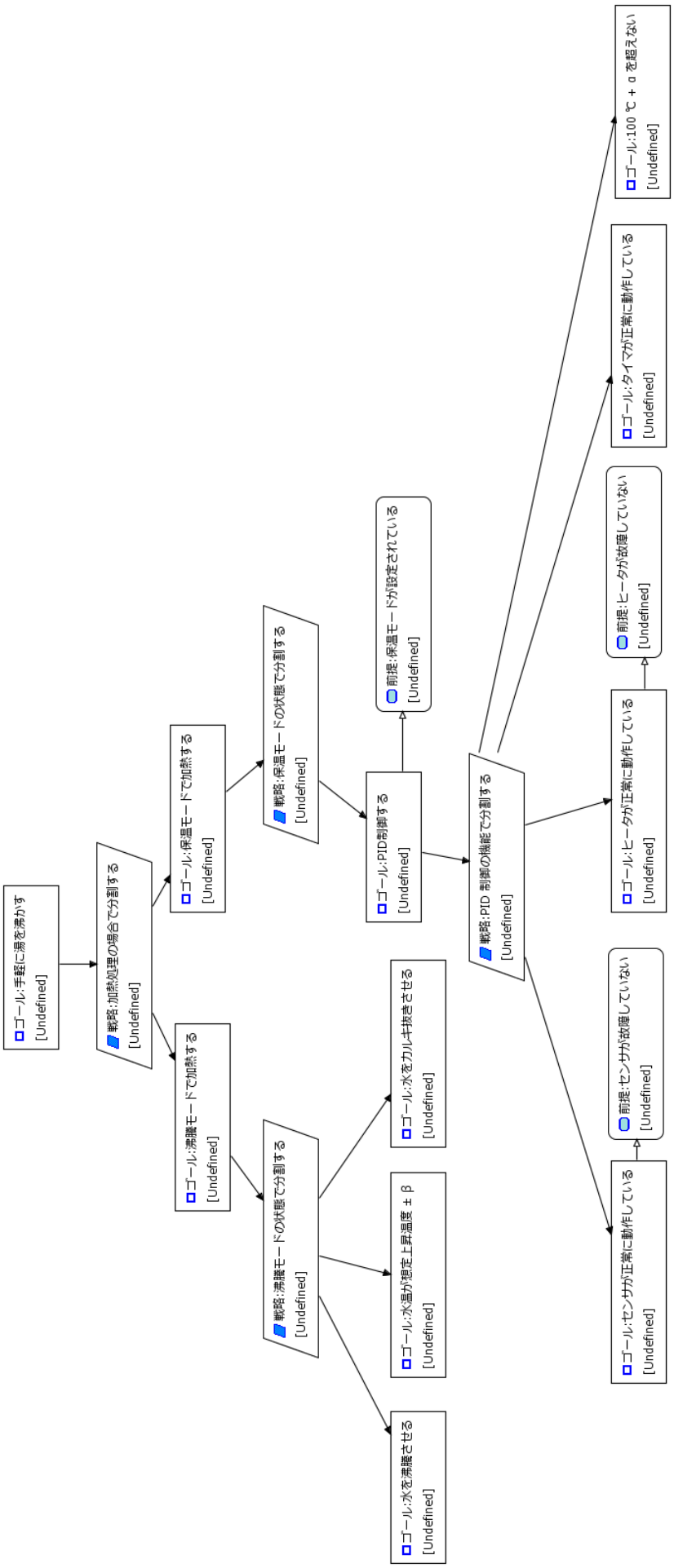

図 3 過熱防止に関するゴール木

た湯温よりも高くなりすぎているというハザードであり，Goal Rank 2 が与えられる．一方の「危険でない」ゴールについては，「物理的危険」「電気的危険」「化学的危険」などのサブゴールに，さらに「物理的危険」から導出される「やけどしない」「発火しない」などの複数のサブゴールに対するハザードに帰着される．この場合の Goal Rank は 3 となる．

つまり，「想定外の高温状態」ハザードは要求仕様書では比較的簡単に触れられているだけにも関わらず，話題沸騰ポット本来のゴールに大きな影響を当たる可能性が高いハザードであることが分かる．このようなハザードの解消には，限られた開発リソースの中でも優先的に取り組むのが望ましいといえる．

3.2 VDM モデルと検証強度

「想定外の高温状態」ハザードのリスクを低減させるため仕様記述言語 VDM++ によるフォーマルな仕様記述を行ない，このハザードを中心に検証を行った．VDM++ は，VDM のオリジナル仕様記述言語 VDM-SL を OOA/OOD 記述や非同期並行処理の記述が可能なように拡張した仕様記述言語である．クラスによるカプセル化とアクセス制御，継承など OOA/OOD 記述や非同期並行処理オブジェクト指向設計の記法を一部サポートしている．

関連するクラスおよびそれらの相互作用は，図 4 のようになっている．

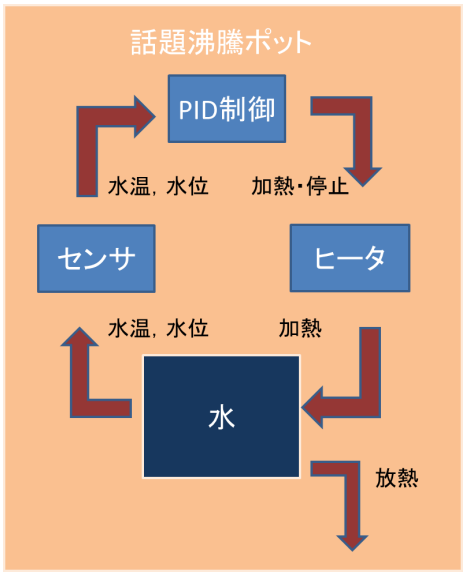

図 4 話題沸騰ポットの制御構造

- 「話題沸騰ポット」クラス
  ポット全体を表現するインスタンス
- 「水」クラス
  ポットで加熱される物質としての水
- センサクラス
  水の水位と水温を検知するセンサ
- ヒータクラス
  水を加熱するアクチュエータ
- PID 制御クラス
  指定温度を目標として閉ループ PID 制御を行うコントローラ

VDM++ 仕様の適用強度は，

(1) 文法および型に関する静的検査

(2) 仕様アニメーション

(3) 数学的証明

の順に高くなる．それぞれの検証に必要となる仕様記述の強度も準ずる．図 5 に記述した VDM++ 仕様の一部を示す．さらにこの仕様は実行可能な形式で記述されており，ツールを利用して入力値について評価する仕様アニメーションが可能である．より優先度の低いハザードについては，ほとんど時間のかからない型検査だけですませ，「高温エラー」と呼ばれる「想定外の高温状態」ハザードについて，仕様アニメーションにより，具体的な値に対して対象となる制約が満たされることを確認できる．

4. 「手軽さ」要求はなぜ難しいか

3 章でみたように，「手軽さ」というゴールの達成はその言葉に反して難しい．その理由として次の 2 点が挙げられる．

- 相対性
  そもそも「手軽さ」という言葉は相対性を含意しており，評価に他の基準を必要とする．電気ポットの場合は，ガス台，コンロ，蝋燭などと比較して，相対的な安全性の高さ，運用費の安さなどを「手軽」と表現している．つまり，他の手段に対する優位性を前提としており，このような暗黙の前提は環境の変化や関係者の入れ替わりなどにより容易に解釈が変わりうる，意味的に脆弱な言葉である．したがって，誤解なく理解可能な仕様として定義するためには，他の手法との比較や絶対的な評価への置き換えが必要となる．
  しかしながら，非機能要求の例に違わず「手軽さ」の誤解のない定義を行うには相当な記述が必要となる．
- 創発性
  もうひとつの理由は，「手軽さ」がポットのシステムとしての全体に関わる性質であることである．「手軽さ」ゴールを構成するサブゴールは「低価格」「扱いやすさ」「個人で給水」「危険でない」など多岐にわたり，それぞれのサブゴールが関連を持っている．3 章の事例でも，「手軽さ」を阻害する「想定外の高温状態」ハザードは 3 つのゴールに関連しており，それぞれの観点からそれぞれのゴールに対するリスクとなっ

ていた．これは「手軽さ」というゴールがさまざまな観点に関連する非機能要求であることによる．
非機能要求の中でも，このような創発的な性質を検証するためには，抽象度の高いシステムの仕様を把握する必要があり，フォーマルメソッド適用における数理的センスと同様に，要求整理や仕様記述についての訓練や対象への深い理解が必要になる．

5. おわりに

本稿では，ソフトウェア開発におけるフォーマルメソッド適用を前提とした，効率的な検証手法を提案した．

ソフトウェア開発にはリスクを許容できない部分やできる部分などさまざまな要素がある．また，リスクの種類もハザードにより発生頻度や発生時の影響などさまざまである．本稿では，こうしたソフトウェア開発におけるリスクに応じてフォーマルメソッドの適用強度を変えることにより，リスクと開発コストの両方を最適化するソフトウェア開発を提案した．リスクの定量的な評価のために，安全性を議論し，システム認証者や利用者などに保証するための構造化されたドキュメントである D-Case を利用する．D-Case は木ゴールの関係を構造となる GSN により表現するので，トップゴールからの距離により相対的な優先順位を定義した．本手法を話題沸騰ポットの事例に適用した結果，次のような知見が得られた．

- Goal Rank の定義
  想定されるハザードごとに RCA を適用し，リスクごとの根本原因をゴールグラフのノードへ帰着させる．帰着させたノードの Goal Rank の高い順に強度のフォーマルメソッド適用により高い保証を与える．
  Goal Rank はトップゴールを Rank 0 とし，トップゴールからの距離によって定義する．Goal Rank が高いゴールは開発対象システムにおいてより重要な要素であるので，実現および検証を優先する．
  これにより，十分なリソースが確保できない場合でも検証の優先順位をつけることができる．
- フォーマルメソッド適用強度の使い分け
  フォーマルメソッドの適用コストは静的検査，動的検査，数学的証明などにより大きく異なる．事例により，それぞれの検証に必要となるコストに応じ，あるいはフォーマルメソッド以外の適用も含めて検証の優先順位に応じて選択できる．
  この項に関しては，引き続き定量的な評価が課題である．
- 「手軽さ」の難しさ「手軽さ」という用語に対してゴール指向要求分析を適用し，非機能要求の中でも相対性や創発性に起因する難しさが明らかになった．
  特に創発性については，ゴール木上でさまざまなゴールが関連をもつ複雑な構造という形で視覚化された．
  具体的には，あるゴールが多数かつ多様なサブゴールへ分割されるような場合，そのゴールは創発的な性質である可能性が高い．また，同じようなゴール-サブゴールの構造が離れたゴールでみられるような場合には，一貫性をもつ要求整理を実現するのが困難になる．

謝辞 本研究で使用した「話題沸騰ポット要求仕様書 (GOMA-1015 型) 第 7 版」は， NPO 法人 組込みソフトウェア管理者・技術者育成研究会 (SESSAME) の公開された成果であり，ご関係のみなさまに感謝する．本研究の一部は， JSPS 科研費 24220001 ，基盤研究 (S) 「アーキテクチャ指向形式手法に基づく高品質ソフトウェア開発法の提案と実用化」の成果による．

参考文献

[1] Boehm, B. W. and Basili, V. R.: Software Defect Reduction Top 10 List, *IEEE Computer*, Vol. 34, No. 1, pp. 135–137 (2001).
[2] Denney, E., Pai, G. and Whiteside, I.: Formal Foundations for Hierarchical Safety Cases, *the 16th IEEE International Symposium on High Assurance Systems Engineering*, pp. 52–59 (2015).
[3] Jones, C. B.: Software Development based on Formal Methods, *Proceedings of the CRAI Workshop on Software Factories and Ada*, LNCS, Vol. 275, Springer-Verlag, pp. 153–172 (1987).
[4] Kelly, T. and McDermid, J.: Safety case patterns-reusing successful arguments, *Proceedings of IEE Colloquium on Understanding Patterns and Their Application to Systems Engineering*, pp. 1–9 (1998).
[5] Kelly, T. and Weaver, R.: The Goal Structuring Notation – A Safety Argument Notation, *Proceedings of Dependable Systems and Networks 2004 Workshop on A ssurance Cases* (2004).
[6] Larsen, P. G., Mukherjee, P., Plat, N., Verhoef, M., Fitzgerald, J., 酒匂寛 (訳)：VDM++によるオブジェクト指向システムの高品質設計と検証，翔泳社 (2010).
[7] Matsuno, Y.: Design and Implementation of GSN Patterns: A Step toward Assurance Case Language, *Information and Media Technologies*, Vol. 9, No. 3, pp. 262–271 (2014).
[8] Matsuno, Y., Nakazawa, J., Takeyama, M., Sugaya, M. and Ishikawa, Y.: Toward a language for communication among stakeholders, *Proceedings of the 16th IEEE Pacific Rim Inernational Symposium on Dependable Computing*, pp. 93–100 (2010).
[9] Reifer, D.: Software Failure Modes and Effects Analysis, *IEEE Transactions on Reliability*, Vol. R-28, No. 3, pp. 247–249 (1979).
[10] Sommerville, I.: *Software Engineering*, John Wiley & Sons (2010).
[11] van Lamsweerde, A.: *Requirements Engineering - From System Goals to UML Models to Software Specifications*, Wiley (2009).
[12] Yamamoto, S. and Matsuno, Y.: An evaluation of argument patterns to reduce pitfalls of applying assurance case, *The 1st International Workshop on Assurance Cases for Software-Intensive Systems*, pp. 12–17 (2013).
[13] 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.

```
class 「話題沸騰ポット」

instance variables
--ポット内部の状態を示す変数
--nil はコンセントが刺さっていないときの状態
private 状態 : [<アイドル>|<加熱中>|<カルキ抜き中>|<保温中>] := nil;

--仕様書で明記されているエラー
--上記のエラーが発生していない場合は nil
private エラー : [<高温エラー>|<温度下がらずエラー>|<温度上がらずエラー>] := nil;

--保温のモード
-- nil の場合はコンセントが抜かれている状態
private モード : [<高温モード>|<ミルクモード>|<節約モード>];

--水を加熱するためのヒータ
--目標温度 ON/OFF 動作は沸騰行為、PID 制御動作は保温行為中のヒータの動作
--電力遮断状態はヒータ用電源が off の場合の状態
private ヒータ : <電力遮断状態>|<停止中>|<目標温度 ONOFF 動作>|<PID 制御動作>;

--ここからは操作パネル上のボタン
public 沸騰ボタン : 「ボタン」 := new 「ボタン」(<無効>);
public 解除ボタン : 「ボタン」 := new 「ボタン」(<無効>);
public 給湯ボタン : 「ボタン」 := new 「ボタン」(<無効>);

--蓋の開閉の状態を示すセンサ
--ON の時蓋は閉じている、OFF の時は蓋は開いている状態
--デフォルトでは off となっている
private 蓋センサ : 「センサ」:= new 「センサ」(<OFF>);

--ポットの水量の許容値を超えている場合 ON
--デフォルトでは OFF となっている
private 満水センサ :「センサ」:= new 「センサ」(<OFF>);

--水位を検知するセンサ
private 第 1 水位センサ : 「センサ」 := new 「センサ」(<OFF>);
private 第 2 水位センサ : 「センサ」 := new 「センサ」(<OFF>);
private 第 3 水位センサ : 「センサ」 := new 「センサ」(<OFF>);
private 第 4 水位センサ : 「センサ」 := new 「センサ」(<OFF>);

private 水位センサ : seq1 of 「センサ」
:= [第 1 水位センサ, 第 2 水位センサ, 第 3 水位センサ, 第 4 水位センサ];

private サーミスタ : [<初期温度>|<ミルクモード時の温度>|<節約モード時の温度>|<高温モード時の温度>|<
沸点>|<温度が上がらない>] := <初期温度>;

--ポットの状態をユーザに知らせるためのブザー
private ブザー : <鳴っていない>|<鳴っている> := <鳴っていない>;

--操作パネルに付いているタイマ
public タイマ : 「タイマ」 := new 「タイマ」();

--システム内で使用する時間
--規定時間とはシステムで明記されている時間を満たしている状態
--nil の場合は明記されている時間を満たしていない状態
private 時間 : [<規定時間>] := <規定時間>;

--ポット内の水位、モデル内では空と許容上限を超えている場合のみに着目する
--nil の時は水位は正常とする
private 水位 : <空>|<正常>|<許容上限超過> := <正常>;


inv if (蓋センサ. 参照 () = <OFF>) then (ロック = <ロック解除> and 水位メータ. 状態 = <消灯>)

--ここからはランプとセンサに関する条件
else
( /*このポットのモデルでは水位が問題となるのは空の場合と許容上限を超えている場合
なのでその 2 つ場合について着目する*/
/*水位と水位センサ間の制約*/
(水位 = <空> => forall s in set elems 水位センサ & s. 参照 () = <OFF>)
and
(水位 = <許容上限超過> => 満水センサ. 参照 () = <ON>)
and

/*ここからはセンサと水位メータ間の制約*/
/*満水の場合のインジケータの表示*/
if(満水センサ. 参照 () = <ON> and 状態 <> <アイドル>)
then(水位メータ. 状態 = <満水時点滅>)
/*空の場合のインジケータの表示*/
elseif(forall s in set elems 水位センサ & s. 参照 () = <OFF> and 状態 <> <アイドル>)
then(水位メータ. 状態 = <空時点滅>)
/*水位が正常な場合のインジケータの表示*/
else(水位メータ. 状態 = <正常時表示>) )
and

/*沸騰ボタンは保温中のみ有効*/
(状態 <> <保温中> => 沸騰ボタン. 状態 = <無効>)
and

(( ロック = <ロック> or 水量異常の判断 () = <異常> or 蓋センサ. 参照 () = <OFF> or
ヒータ = <電力遮断状態> or エラー <> nil or 状態 <> <保温中>) => 給湯ボタン. 状態 = <無効>)
and

/*給湯は保温中のみ可能*/
((給湯状態 = <給湯中>) => (状態 = <保温中>))

--保温モード設定を行う
public モード設定 : () ==> ()
モード設定 () ==
( --保温のモードは高温→節約→ミルク→高温... と遷移する
cases モード:
<高温モード> ->( モード := <節約モード>;
ブザー操作 (<鳴らす>) ),
<節約モード> ->( モード := <ミルクモード>;
ブザー操作 (<鳴らす>) ),
<ミルクモード> ->( モード := <高温モード>;
ブザー操作 (<鳴らす>) ),
others -> モード := nil
end;
パネル. 表示 (サーミスタ, モード) )

pre 蓋センサ. 参照 () = <ON>;
```

図 5 VDM++ によるモデル記述