Vol.2014-SE-183 No.26
2014/3/20

ディペンダブルケースを中間表現としたフォーマルな仕様記述手法

大森 洋一[1]　日下部 茂[1]　林 信宏[1]　荒木 啓二郎[1]

概要：上流工程からの検証は，ソフトウェア開発における手戻りコストを抑えるために重要であることが知られている．フォーマルメソッドは，数理的なモデルにより対象を記述し開発の初期段階からの検証を可能とする．フォーマルなモデルを記述するためには，対象の仕様を理解しなければならないが，仕様書の構成は，必ずしもモデルにより抽象化された対象の構造と適合しない．したがって，モデル記述者は仕様書の全体を理解しなくてはならず，これがフォーマルメソッドの実問題への適用の敷居を高いものにしている．本研究では，モデル化の観点に基づくゴール分析により仕様書の構造を読み替えて，必要な情報だけを抽出したものに基づいてフォーマルなモデルを生成する手法を提案する．自然言語による仕様書に対して，ゴール分析の結果はディペンダブルケースとして記述し，これと共通の観点によりフォーマルなモデルを仕様として生成し，実際の仕様書に対して適用して評価する．提案手法を適用した結果，モデリングの目的を明示的に関係者が共有し，仕様書において検証に必要な情報のどの部分が不足しているのかを具体的に指摘することが可能となるといった成果があった．

キーワード：フォーマルメソッド, ディペンダブルケース, 要求理解, モデリング

Modeling a formal specification based on dependable case analysis

Abstract: It is well known that verification from the early stage is important to spare develping cost in software development. Formal methods enables description of the target as a mathematical model and verify it in early developing stage. It is necessary to understand the specifications of the target system to describe the model formal, the structure of the specification is not necessarily compatible with the structure of the abstraction model. Therefore, the model describer must not not understand the whole of the specification, this makes application to real-world problems of formal methods difficult. We propose a method to generate a formal model, extracting only the necessary information and reordering the structure of the specification document by the goal analysis of a modeling perspective. The results of the goal analysis of the specifications in natural language is described as D-case, and a formal model specification sharing the same abstractiion view is derived from it. Applying the proposed method achieves that stakeholders can share the modeling purposes which was explicitly expressed in the D-case, and concretly understand which part of the information is missed in the specification to verify it.

Keywords: Formal method, D-Case, Requirement understanding, Modeling

1. はじめに

ソフトウェア開発は，自然言語による要求記述が出発点になる場合がほとんどである．しかし，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，それらを解消するには入念なレビューと書き直しが必要であり，なおかつ完全な保証を与えることは難しい．これは，要求が顧客の関心事のみを記述してあるために十分でない場合が多いこと，情報処理の観点がなく実現のために必要な情報がかけている場合が多いことによる．

実際，ソフトウェア開発における不具合の多くは，曖昧すなわち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 [3].

[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

また，不具合の修正に要するコストは，開発の最後になるテスト段階では，開発の最初である要求段階に比べ，ソフトウェアのライフサイクル全体で 25 倍にもなる [2]．このような修正コストの傾向は，ユーザからの要求が高度かつ複雑化し，規模が増大している近年の開発においても続いており，要求段階と比較して，統合テスト段階では 90 倍，受け入れテスト段階では 440 倍，利用開始後では 470 – 880 倍にもなるという事例も報告されている [1]．したがって，以前の開発手法と比べ，ソフトウェア開発におけるリソースのより多くが，早期の開発段階へ注がれるようになってきている．具体的には，モデルベース開発や上流工程における厳密なレビュー手法の普及である．モデル記述およびそれを用いた検証や厳密なレビューを実現するためには対象となるシステムを検証者が理解しなければならない．対象システムに対する知識をもっている人間は限られるので，開発者が自ら検証を行う場合も多い．開発者が自ら検証を行う手法は多くの提案がなされており，その効果も上がっている．

しかしながら，開発者の検証では，どうしても自らの成果に対する評価が甘くなりがちであり，また本来の開発業務のオーバヘッドとみなされてしまい十分な検証ができない場合が多くみられ，開発者と検証者を分業する体制が望ましい．本研究ではこのような図 1 に示す，第二者あるいは第三者による検証体制を想定するが，この場合，検証者が新たに対象を理解する時間が必要となる．また，対象システムのあらゆる性質に保証を与えようとすれば，実装と同様のコストと時間を必要となり現実的ではない．なぜなら，上流工程における検証の間は開発作業を投機的にすすめることしかできないので，リスク管理の観点から工程を大幅に進捗させることはできず，それは直接的な開発期間の遅延に繋がるからである．こうした理由により，上流工程における検証では下流工程での修正が困難となる致命的な問題に対する性質に限定し，かつ短期間で検証を終えることが望まれる．

一方で，このような体制では，検証の基盤として仕様書に書かれた情報が重要となる．本研究では，自然言語とフォーマルメソッドを併用した仕様記述およびその検証を前提とする．

よい仕様には

a. 要求が正しく記述されていること

b. 記述の解釈に曖昧さを持たず，簡潔であること

c. 満たすべき要求が全て含まれており，それ以外が含まれないこと

d. 内部に矛盾を含まないこと

e. 要求の優先順位が明確であること

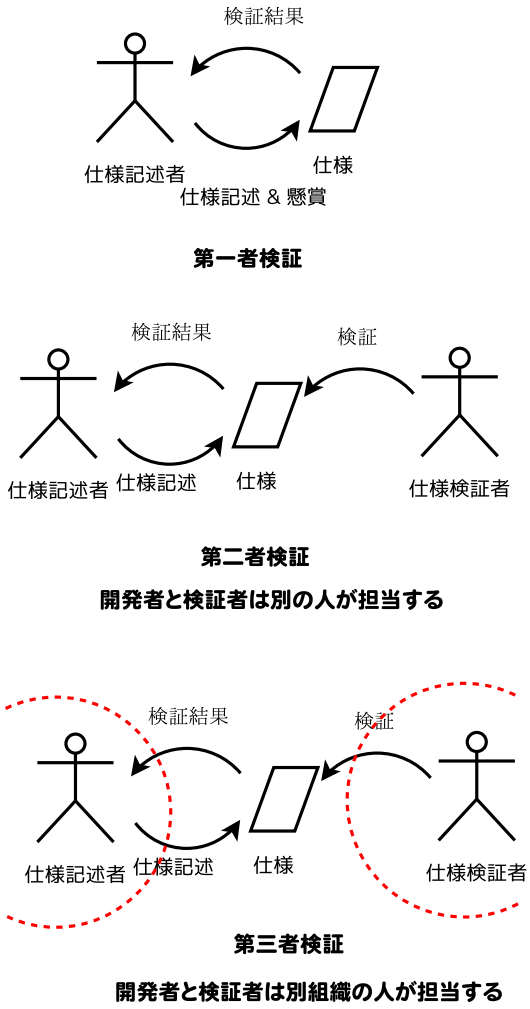

図 1　検証体制の違い
Fig. 1　Verification management

f. 仕様の検証が可能であること

g. 修正が容易であること

h. 仕様を満たしているかどうかの検証が容易であること

といった性質が求められる [4]．

フォーマルメソッドを用いるだけで，仕様の満たすべき性質はかなり達成できる．

A. 数学的な概念は誰でも同じ意味に解釈できる (上記 b)

B. 数学的な矛盾を検出できる (上記 d)

C. 修正の影響範囲が明確である (上記 g)

D. プログラムの意味論を定義可能である (上記 f)

E. 有限状態機械への変換，あるいは変換不能であることの証明が容易 (上記 h)

ただし，これらの言明は仕様記述に用いたフォーマルメソッドの背景となる数理論上の性質に限定される．これに対して，必ずしも数理的な知識があるとは限らない関係者

の合意も必要となる上記 a, c, e といった性質の確認については，自然言語による仕様を活用するのが適している．

ここでの問題は，仕様書が対象システムを実現することが前提となっているという点である．仕様書に限らず，文書は基本的に一次元の構造をしており，階層性はあるものの前から後ろへ読まれる．人間が物事を理解するときには，重要なものからそうでないもの，概要から個別的なもの，抽象的な説明から具体的な議論へとすすむのがよいとされている [10]．仕様書においても，このような原則にそってものごとが記述されているべきである．ただし，ものごとの重要性は注目している性質により変わる．すなわち，本研究で想定する状況では，仕様書は設計や実装に必要な情報を優先して書かれていると，個別の機能や特定の実装条件についての記述が優先され，検証に必要となる対象ドメインの前提知識や一般的な性質は軽視されるのが問題となる．また，ある程度の規模を超えたシステムでは，それらの情報に言及されている場合でも，実装のための論理構成や優先順位と検証に必要な論理構成や優先順位が異なるため，情報が分散し，非常に読みづらいものになってしまう．

この結果，仕様書および背景についての補完資料全体と検証のためのモデル全体が対応づけは可能でも，それ以上の細分化が困難となり，要求変更に伴う検証モデルの修正や，類似したシステムを対象としたソフトウェア開発であっても検証モデルの再利用が困難となっている．これは，以前から知られた問題であり，開発仕様書に対して，操作仕様書や保守仕様書といった目的ごとの仕様書を用意したり，複数の仕様書間での情報の統一や作業軽減を目的とした Hypertext の活用は一般的になっている．本研究で対象とするのは，そうした技法に関する部分ではなく，仕様書の目的に沿って情報を配置するにはどのような順序になっているのが望ましいかという点である．こうした点を確認するためには，出来上がったモデルを見るだけでなく，検証の方針や項目，システムの目的など，場合によっては営業戦略や経済情勢といった非技術的な情報も含めた，開発プロジェクト全体を見通した責任者による判断が必要になる．本研究は技術的な観点から，責任者が総合的な判断を行う上での戦略なども反映できるような手法を提案する．

本稿では，仕様書の理解を目的として，自然言語による記述とフォーマルメソッドによる記述の対応を明確化する手法としてディペダブルケース (以下， D-case) を用いる手法について検討する．以下，第 2 章でフォーマルメソッドと D-case によるソフトウェア検証ついて記述し，第 3 章ではこれらを組み合わせた仕様理解および理解した結果を明示する手法を提案する．さらに，第 4 章では事例研究およびその成果について，第 5 章ではまとめと今後の課題について述べる．

2. 上流工程における検証

2.1 上流工程におけるフォーマルメソッド

フォーマルメソッドは，計算機システムの仕様を数理的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 [7]．フォーマルメソッドを適用し，ソフトウェアの仕様書を数理的なモデルに変換することで仕様書中の矛盾や曖昧さを取り除くことが可能となる．また，フォーマルなモデルはツールにより数学的な検証が可能であり，ソフトウェアの正しさを証明することもできる [12]．このため，特に上流工程からのソフトウェア品質向上に有用であることが知られている．

一方，フォーマルメソッドによる仕様記述は，記述言語の習得が必要であったり，背景となる数理的な知識が必要とされたりといった理由により，多くの関係者にとって，日常使用している自然言語による記述と比較して，読み手・書き手とも限られている．我々は，自然言語による仕様記述とフォーマルメソッドによる仕様記述を併用し，それらの間で一貫性が保たれている状態を実現するための研究を行なっており，実開発への適用においても，そのほとんどで両者が併用されている [6]．このような複数の仕様記述の使い分けとそれらの間でのドメイン用語辞書による一貫性確保を達成するための基本的なアイデアは以下の手順になる [14]．

(1) 自然言語による仕様をフォーマルメソッドを用いたモデルへ変換し，その上で数理的な検証を行う．

(2) フォーマルなモデルの上で矛盾や記述不足が見つかった場合，それを自然言語記述にフィードバックする．

(3) 修正された自然言語による仕様に同じ手順を繰り返す．

(4) モデル上の問題がなくなったら，自然言語による仕様を関係者がレビューし，妥当性を確認する．

(5) レビューにより修正された自然言語による仕様を修正し，フォーマルメソッドを用いたモデルへ変換し同じ手順を繰り返す．

(6) 確立した自然言語による仕様に含まれる用語とそれに対応するフォーマルメソッドによる定義をドメイン用語辞書として保存する．

この過程で，上流工程での検証を行うためには，自然言語による仕様書を検証用のモデルへ変換する必要がある．本研究では，フォーマルなモデルの記述には VDM++を採用した．VDM++ はフォーマルメソッドの一種である VDM を構成する仕様記述言語であり，一階述語論理および集合論に基づく意味論をもつ [11]．VDM はモデル規範型と呼ばれ，モデルの状態を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できる．逆にいえば，どのようなレベルで仕様を記述するかはモデルの外部で決めておかなくてはならない．

第 1 章で述べたように，検証のための仕様書は，独立した仕様書を書き起こすのではなく，開発のための仕様書を読み替えるのが時間的な効率がよく，かつ開発対象との乖離を避けることができるという利点もある．このとき，仕様書からどのような情報を読み取って，どのようなモデルとして実現するかを整理する必要があり，これまではフォーマルメソッドも含めて，モデル記述者の能力に依存する部分が大きかった．フォーマルなモデルとして明示されれば，その妥当性を確認することはフォーマルメソッドに習熟していなくても可能であり，これまでのフォーマルメソッドを利用した開発では，モデルになってから確認作業を行なっていた．

しかし，結果としてモデル記述者のモデル化観点と，モデル検証者の記述されたモデルにおいて検証したい内容がずれてしまうと，モデルの大規模な修正が必要になってしまうという問題があった．極端な場合，なんらかのモデルを記述した後で，何を検証するかを議論することになる．つまり，モデルは対象の特定の観点からのモデル化であり，フォーマルメソッドも数理的なモデル化手法の一種であるので，開発対象のソフトウェア関係者の間で観点を一致させておかなければならないが，その選択過程は必ずしも明確ではなかったのである．

2.2 ゴール分析の適用

効率的な仕様記述には，ドメイン知識の活用が欠かせない．しかし，仕様書に初心者にとっても十分なドメイン知識の記述がなされることは稀である．対象ドメインにおいて経験を積んだ技術者にとって，そうした記述は冗長であり，むしろ既存のシステムとの差分や特徴的な点についての記述を重視する傾向にある．しかしながら，検証のためにはシステム全体の機能あるいはふるまいを知らなければならなず，そのためには対象システムの前提となるドメイン知識は必須である．

特に，本研究でも想定する第三者検証では短期間でのドメイン知識の獲得が求められる．このような状況では，検証のためにフォーマルなモデルを書き始める前に，関係者の間でモデル記述の目的を共有しておかなければ，大きな手戻りが生じたり，いつまでも検証モデルが完成しなかったりする恐れがある．こうした場合において，検証のためのゴール分析およびその結果の明示が有効である．検証のゴールが共有できて，必要なサブゴールおよび関連した仕様およびドメイン知識のみを理解すればよいのであれば，検証開始時点でドメイン知識を持たない第三者検証も実用的な期間で適用できるようになる．

D-case はディペンダビリティの分野に適用した Assurance Case の一種である．Assurance Case は，想定している環境下において，システムが正しく動作することを，構造化することによって，体系立てて保証する方法または，そこで表記された議論の構造である． Assurance Case はさまざまな規格において，具体的な記述法が定義されており，また記述のためのプロセスも定義されている場合がある．他の X-case と比較した D-case の特徴は，Goal Structured Network (以下， GSN) に基づく分析を行う点と，運用に関する記述が考慮されている点である．これらの特徴は，ゴール分析の適用および明快な記述，検証に必要な情報が開発工程に限定されない，といった点で，本研究で想定する仕様書の理解および理解した結果の関係者による共有に適している．

D-case 記述については，GSN Community Standard による方法が一般的である [15]．GSN を用いることで，検証目的を達するために必要な議論を系統立ててすすめることができる．GSN 構築には，トップレベルゴールからサブゴールを導出してゆくトップダウンによる方法と根拠から議論を関連づけてゆくボトムアップによる方法がある．いずれにせよ，トップレベルゴールから直接またはストラテジーを通じてサブゴールを導出しすることで，ゴール間の関係を明示的に与え，最終的に論拠またはモニタまたは未定義に帰着させることで，議論の構造を明示する [9]．それぞれの意味は

- goal . . . 達成すべき目標
- strategy . . . goal を subgoal へ導出する際の方針
- context . . . goal あるいは strategy の前提
- solution . . . 議論の論拠となる事実，分析結果，規定など
- underdevelopment goal . . . 論拠が見つかっていないまたは考慮しない goal
- monitor . . . D-case における GSN 拡張であり，運用中の実行時情報

である． GSN の主な表記を図 2 に示す．

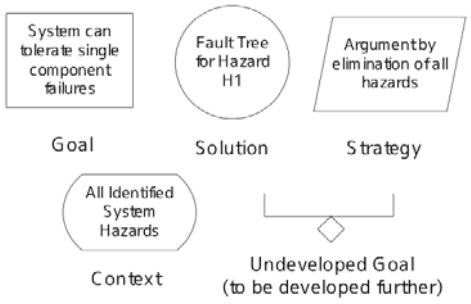

図 2 GSN の主なノード [9]
Fig. 2 Principal Elements of the GSN

3. D-case による中間表現の提案

本研究では，ソフトウェア開発の上流工程でのフォーマルメソッドを利用した，短期間でのモデル記述および検証を目的とする．このような前提条件は理論的なフォーマルメソッド適用からは外れるものであり，完全な検証は不可能である．しかし，検証対象となる問題の性質を適切に選択し，下流工程で修正できない問題を排除できればフォーマルメソッドを利用する価値がある．このような考え方は，計算機の能力向上に伴い，従来の数学的な証明の代わりに，モデル検査などの総当たりや自動定理証明といった検証ツールをより積極的に活用する軽量フォーマルメソッド (Light Weight Formal Methods) といったフォーマルメソッドの潮流とも一致する [8]．自然言語によるフリーフォーマットでツールを活用するのは難しいが，フォーマルなモデルでは文法だけでなく，規定された意味論に基づく検証も可能となる．

短期間での検証のポイントは

- 検証対象の明確化によるドメイン知識および仕様書理解範囲の限定
- 各種ツールの活用およびそれらの連携による検証作業の省力化

である．

本研究では次の手順により，検証方針を D-case として表現し，フォーマルなモデル化と検証を行う手法を提案する．

(1) D-case 記述者は，自然言語による仕様として記述された対象システムの主たる目的とドメイン知識を活用し，検証のための D-case を作成する．

(2) 関係者の間で検証用 D-case そのものおよび検証対象としての適切なゴールを選択して合意し， GSN の階層に基づいて対象システムの範囲を限定する．

(3) 仕様検証者は，限定されたシステムを，指定されたゴールの観点からフォーマルモデルとして記述し，モデル上の検証を行う．

(4) 仕様検証者は，D-case で表現された議論の構造に従って，達成されたゴール，達成されなかったゴールを識別する．

(5) 仕様記述者は，達成されなかったゴールについて，D-case で表現された議論の構造に従って，仕様を改善する．

(6) 目標のゴールが達成されるまで繰り返す．

この結果，従来のフォーマルメソッド適用では，図 3 のような作業フローが，図 4 のような作業フローへ変更される．

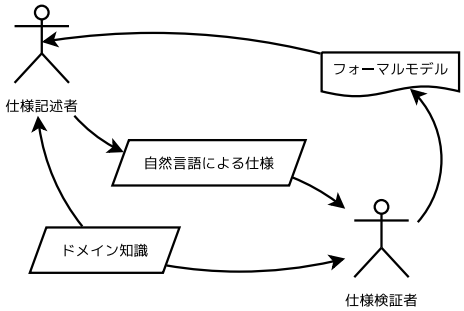

図 3 従来のフロー
Fig. 3 Existing verification process

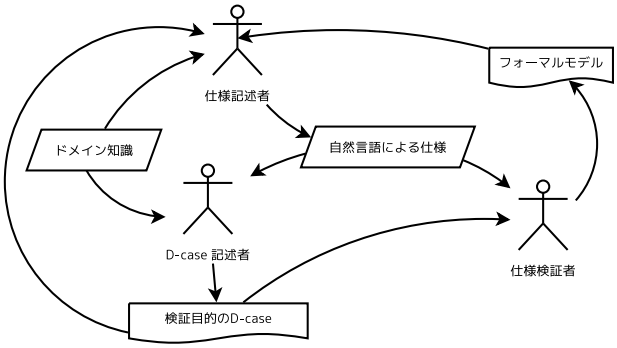

図 4 提案手法のフロー
Fig. 4 Proposing verification process

従来のフローでは，全ての関係者が全ての情報に関わっていたのに対し，仕様検証者はドメイン知識の獲得をしない，もしくは非常に小さいものにできる．一方で D-case 記述者はフォーマルメソッドに関する知識を持っている必要はなくなるので，お互いの役割が明確になる．検証のための D-case は，自然言語を用いるとともに図の要素の種類も少ないので，記述のための学習コストがフォーマルメソッドの習得と比べてかなり低い．したがって，ドメインに依存した議論の構造を表現するために仕様記述者が兼任することも容易である．

ツールの活用は，D-case の記述およびフォーマルメソッドの記述および検証で可能となる．ツールによる文法的な検査は，書き手に依存した記述のばらつきを少なくし，また不注意によるミスの防止にも役立つ．本研究で使用したフォーマルメソッドの一種である VDM++ は，一階述語論理に基づいて，静的な型検査，陰仕様記述における状態遷移確認，陽仕様記述による仕様アニメーションなどを可能とする．

具体的な方法は，第 4 章で事例研究を用いて説明する．

4. 事例研究

本章では，人工衛星の搭載ソフトウェアの一部について，提案手法を適用した事例について述べる．

4.1 人工衛星の姿勢制御

人工衛星のうち，現在地球を周回する衛星は，約 3,500 あり，近年はおよそ年間 90 機前後の打ち上げがある．日本でも年間 5 から 10 機の衛星が製造されており，通信・放送，地球観測，技術開発などさまざまな目的に利用されている．人工衛星の衛星系は，ソフトウェアの不具合があった場合にも修正が困難であることから，フォーマルメソッドの利用による信頼性向上は有効であると考えられる．また，長期に渡る衛星の運用中の故障や不具合に対しても，明確な仕様に基づいた改修が可能となる [19] ただし，人工衛星の打ち上げコストは非常に大きくかつ重量に依存すること，宇宙空間における放射線の影響を軽減するといった理由により，ハードウェアの制約が厳しく，十分な計算機資源が用意されない場合が多い．

本稿では，人工衛星の姿勢制御を対象とする．それぞれの人工衛星には固有のミッションがあり，それぞれのミッションの性格により適切な姿勢が異なる．しかし，ほとんどの人工衛星は太陽電池による発電を利用していることから，蓄電量が減少した時に電池パネルを太陽方向へ向ける，逆に電池パネルの温度が上がりすぎた時に太陽方向を外すといった共通の処理もある．加えて，物理的な制約によりその挙動が決まることから，周回軌道は初期値から計算可能であり，かつ姿勢制御ハードウェアや制御アルゴリズムも定番のものが決まっておりバリエーションは多くない．基本的な構成は図 5 のようになる．

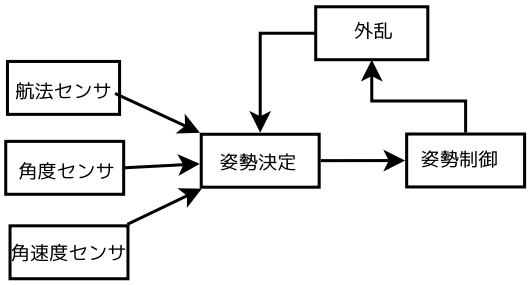

図 5 人工衛星の姿勢制御
Fig. 5 Attitude Contorol and Determinatnion

センサからの入力に対し，物理上の理論的な姿勢を決定し，それに応じた制御を行う．姿勢の制御は，モーメンタムを利用したホイール，磁気を利用したトルカ，噴射によるスラスターなどがあるが，すべての衛星がこれらすべてを備えているわけではない．ホイールやトルカによる姿勢変更はある程度の遅延があるので，それも考慮する必要がある．

4.2 D-case と仕様書の分析

D-case 作成にあたっては，次の手順によるトップダウンによる GSN 記述手法を用いた [15]．

(1) 保証すべき goal を同定する
(2) goal の議論の前提や環境に関する context を定義する
(3) goal を保証するための strategy を同定する
(4) strategy の前提となる context を定義する
(5) strategy を適切な粒度に設定し，ステップ 1 へ戻る，またはステップ 6 へ
(6) evidence を同定する

D-case 記述には，図 6 のように D-case Editor を用いた [17]．D-case Editor は，統合開発環境 Eclipse のプラグインであり，

- GSN (Goal Structuring Notation) をサポート
- GSN パターンライブラリ、基礎的な変数型チェック機能
- D-Case/Agda による整合性検査
- 対象システムのモニタリング機能

といった機能を提供する．

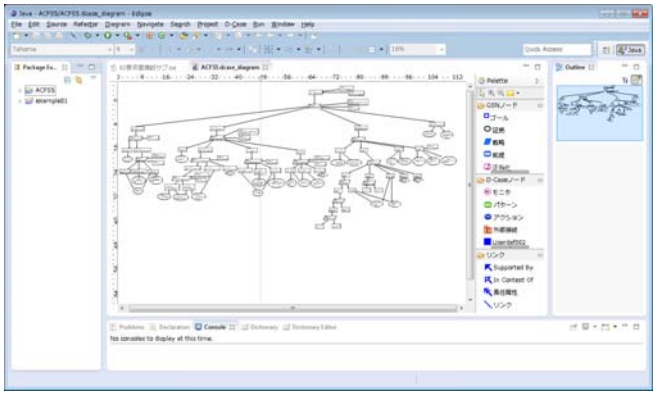

図 6 D-case エディタ
Fig. 6 User Interface of D-case Editor

我々は、姿勢制御系について，仕様書を分析するとともに，外部資料 [20] によりドメイン知識を獲得し， D-case を作成した．

姿勢制御ソフトウェアに関する仕様書は 5 章， 218 ページ + 付録という構成になっているが，仕様書中にはモデル化の前提となる衛星のミッションや，姿勢制御の目的は記述されたおらず，ドメイン知識を外部資料を利用して補完した．記述した D-case は 40 の goal, 24 の evidence をもつ最大 6 階層の構成となった．記述に要した時間は 5 人月程度である．

4.3 フォーマルモデルによる検証

作成した D-case の goal から，状態遷移についての検査を行うことを選択し，フォーマルなモデルを作成した．フォーマルなモデル記述においては，ドメイン用語辞書を作成する作業をサポートするツールを利用した [13]．このツールは 図 7 のような外観を持ち，

- 自然言語による仕様書に含まれる登録した見出し語をマークする「仕様書マーカー」
- 見出し語にフォーマルな意味定義を与える「辞書編集」
- フォーマルに検証可能なフォーマットを生成する「モデル出力」

の大きく 3 つの機能を提供する．

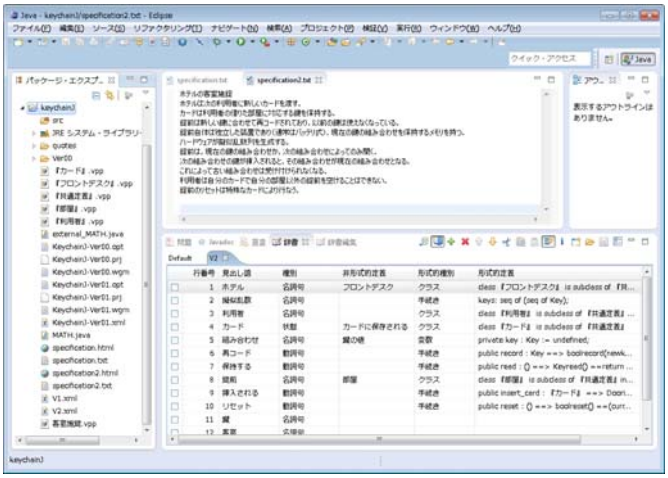

図 7 辞書ツール
Fig. 7 User Interface of Jack-o-Dict

対象の性質を検証するフォーマルなモデルは，表 2 のようになり，コメントも含めた全体で 278 行になった．

表 1 VDM++ 適用結果
表 2 The VDM++ model

項目	個数	備考
定数	6	
型	37	ユーザ定義のもの
状態変数	22	
操作	15	状態を変化させる機能
関数	2	状態を変化させない機能

検証目的を状態遷移の条件確認に限定したことから，対象となる仕様書のうち関係のある部分は 24 ページに限定できた．作業量は，ドメイン知識の獲得も含めて 2 月人程度であり，一般的なモデル化に必要となる 6 月人程度に比べて，大幅な短縮が達成できた．一方，この検証事例では，動作アルゴリズムの正しさや定量的な判定といった，重要ではあるが対象としていない性質については何も保証していない．こうした検証項目および条件の限定は，検証にかけられるコストとのトレードオフとなる．

5. まとめ

5.1 関連研究

D-case の人工衛星における姿勢制御ソフトウェアへの適用は，文献 [18] がある．この先行事例では D-case による分析とレビューを行ない，トレーサビリティの保証について成果があったことを報告している．

D-case とフォーマルメソッドの関係でもっとも疎なものは， evidence としてフォーマルモデルを利用する場合である．このような関係は，いつでも実現可能であるが，フォーマルメソッドを利用する意義は小さい．なぜなら，一般的な D-case はフォーマルな意味論をもたないので，証明された evidence に対して goal を保証するかどうかは人間による議論に依存するからであり，相互の関連が保証できないからである．

本格的なフォーマルメソッド適用については， D-case/Agda がある [5]．D-case/Agda は定理証明系であり，対象となる性質の保証に必要な定義を導出する．最終的な数学的証明は人間に依存するが，実際のシステムにおいては数学的にそれほど難しい問題になることは少ない．

本研究は，既存のフォーマルメソッドを用いたモデリングと D-case による分析を組み合わせた，いわばこれらの中間にあたる手法である．本研究の利点は，長年利用されて安定した既存の検証ツールを利用できること [16]，VDM++ 利用者のコミュニティがあること，ドメイン辞書の再利用など他のツールとの連携が考慮されていることである．

5.2 結果の考察

本稿では，ソフトウェア開発における仕様の自然言語記述とフォーマルなモデルの対応づけについて， D-case を利用した仕様書の分析手法を検討した．

事例により D-case は，人間による議論の整理を目的とし， goal を明確化することからモデル化および検証方針の整理に大きな役割を果たす可能性が確認できた．また，時間的あるいはリソース上の制約がある場合に，フォーマルなモデル記述の観点を限定するための指針としても役立つことが分かった．

提案手法では，フォーマルなモデルを記述する前に，観点の議論ができるので，複数の関係者がいるようなソフトウェア開発におけるフォーマルメソッド適用の効率を改善できる．

この結果，

- D-case による記述はフォーマルメソッドを適用する事前のセミフォーマルな整理に役立つ．
- 対象範囲や検証範囲の選択について，関係者の合意を取る部分は，検証を目的とした D-case に基づく議論がそのまま適用できる．

- フォーマルなモデルの保証できる範囲について，検証作業に関わっていない関係者に対して説明が容易になる．

といった利点があることがわかった．

具体的なモデル化範囲の見積りなどは他の事例などへも適用し，経験的な評価を蓄積する必要がある．

謝辞 本研究で使用した辞書ツール開発にご協力いただいた吉村康晴氏をはじめ，九州ビジネス株式会社のみなさま， D-Case Editor をご提供いただいた DEOS プロジェクト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] Baziuk, W.: BNR/NORTEL: path to improve product quality, reliability and customer satisfaction, *Proceedings of the 6th International Symposium on Software Reliability Engineering*, pp. 256–262 (1995).
[2] Boehm, B. W.: Software Engineering, *IEEE Transactions on Computers*, Vol. C-25, No. 12, pp. 1226–1241 (1976).
[3] Boehm, B. W. and Basili, V. R.: Software Defect Reduction Top 10 List, *IEEE Computer*, Vol. 34, No. 1, pp. 135–137 (2001).
[4] IEEE-830: Recommended Practice for Software Requirements Specifications, IEEE Std. 830-1998 (1998).
[5] in Agda, D.-C.: http://wiki.portal.chalmers.se/agda/pmwiki.php?n=Case-Agda.D-Case-Agda.
[6] IPA/SEC: 厳密な仕様記述における形式手法成功事例調査報告書，独立行政法人情報処理推進機構，http://www.ipa.go.jp/sec/reports/20130125.html (2013).
[7] 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).
[8] Jones, C. B., Jackson, D. and Wing, J.: Formal Methods Light, *IEEE Computer*, Vol. 29, pp. 20–22 (1996).
[9] Kelly, T. and Weaver, R.: The Goal Structuring Notation – A Safety Argument Notation, *Proceedings of Dependable Systems and Networks 2004 Workshop on Assurance Cases* (2004).
[10] 木下是雄：理科系の作文技術，中公新書 624, 中央公論社 (1981).
[11] 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).
[12] 中島 震：ソフトウェア工学の道具としての形式手法，NII Technical Report, http://www.nii.ac.jp/TechReports/07-007J.pdf (2010).
[13] 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実向上 に向けて，情報処理学会論文誌：プログラミング， Vol. 3, No. 5, pp. 18–28 (2010).
[14] 大森洋一，日下部茂，林 信宏，荒木啓二郎：ドメイン用語辞書の再利用に向けたグループ化，情報処理学会研究報告ソフトウェア工学研究会， Vol. 2013, No. 15, pp. 1–8 (2013).
[15] Standard, G. C.: .
[16] SCSK システムズ：http://vdmtools.jp/.
[17] DEOS プロジェクト：http://www.dependable-os.net/tech/D-CaseEditor/.
[18] 田中康平，松野 裕，中坊嘉宏，白坂成功，中須賀真：アシュアランスケースにおける品質到達性と トレーサビリティを考慮した記述ルール提案と超小型衛星開発への適用評価，第 10 回クリティカルソフトウェアワークショップ.
[19] 坂井真一郎：人工衛星姿勢制御ソフトウェアの状態遷移表ベース設計事例，第 17 回 ZIPC ユーザーズ・カンファレンス (2011).
[20] 木田 隆，小松敬治，川口淳一郎：人工衛星と宇宙探査機，コロナ社 (2001).