

フォーマルメソッド導入ガイダンス

～ソフトウェアの安全性・信頼性向上のための技術導入に向けて～

Version 1.0

2011年6月

 株式会社 三菱総合研究所

はじめに

本導入ガイダンスは、ソフトウェアの信頼性・安全性等を向上させるための技術であるフォーマルメソッド(=形式手法)を開発現場に普及促進させることを目的として、経済産業省「新世代情報セキュリティ研究開発事業」における委託プロジェクト「モデル検査による組み込みソフトウェア検証とモデリングパターン化の研究開発」において作成したものである。導入ガイダンスは、フォーマルメソッドやソフトウェア開発に関わる国内の有識者から構成されるフォーマルメソッド導入ガイダンス検討委員会(18章に記載)においてレビューしたものである。

本書の目的

ソフトウェア等の安全性、信頼性、セキュリティ等の向上に関わるフォーマルメソッドをソフトウェア関連産業の開発現場に普及させるために、普及の障害となっている問題点に対して、マネジメント面および技術面の具体的な解決策、手順、考え方、ノウハウ等を示す。本ガイダンスにより、フォーマルメソッドの有効性や限界を把握し、その導入効果が期待できる対象や方法を判断しつつ、具体的な導入を行う際の手引きとして利用されることが期待される。

本書の構成と対象読者

本書は、ソフトウェアの開発や利用に係わる関係者や組織で読まれることを想定している。4つのパートから構成され、それぞれの想定読者は大まかに章構成の右に示したとおりである。

第1部 意識啓発編 (1) フォーマルメソッド応用に関する背景 (2) フォーマルメソッドの概念と特徴 (3) フォーマルメソッド導入の意義と効果の概要	<table border="1"><tr><td>読者</td><td>発注者(CIO, CTO)、 上級管理者</td></tr><tr><td>目的</td><td>意識啓発</td></tr><tr><td>用途</td><td>理解の共有</td></tr></table>	読者	発注者(CIO, CTO)、 上級管理者	目的	意識啓発	用途	理解の共有
読者	発注者(CIO, CTO)、 上級管理者						
目的	意識啓発						
用途	理解の共有						
第2部 マネジメント編 (4) 組織へのフォーマルメソッドの導入方法 (5) フォーマルメソッド導入のコストと効果の考え方 (6) フォーマルメソッドと関連技術の位置づけ (7) 手法の選択方法(主な手法の概観および特徴比較)	<table border="1"><tr><td>読者</td><td>プロジェクト管理者等</td></tr><tr><td>目的</td><td>管理面の手引き</td></tr><tr><td>用途</td><td>プロジェクト管理・計画</td></tr></table>	読者	プロジェクト管理者等	目的	管理面の手引き	用途	プロジェクト管理・計画
読者	プロジェクト管理者等						
目的	管理面の手引き						
用途	プロジェクト管理・計画						
第3部 技術編 (8) モデリング・プロセスの構成と手順 (9) モデルの抽象化と状態爆発の対策法	<table border="1"><tr><td>読者</td><td>開発技術者等</td></tr><tr><td>目的</td><td>技術面の障害解決</td></tr><tr><td>用途</td><td>適用ノウハウの学習</td></tr></table>	読者	開発技術者等	目的	技術面の障害解決	用途	適用ノウハウの学習
読者	開発技術者等						
目的	技術面の障害解決						
用途	適用ノウハウの学習						
付録 参考情報 (10) ケーススタディ (11) 応用事例集 (12) モデル検査ツール(SPIN)の使い方ヒント (13) 関連文献、リンク集 他	<table border="1"><tr><td>読者</td><td>発注者、管理者 開発技術者</td></tr><tr><td>目的</td><td>具体情報の提供</td></tr><tr><td>用途</td><td>詳細情報へのポインタ</td></tr></table>	読者	発注者、管理者 開発技術者	目的	具体情報の提供	用途	詳細情報へのポインタ
読者	発注者、管理者 開発技術者						
目的	具体情報の提供						
用途	詳細情報へのポインタ						

第 1 部は、フォーマルメソッドに初めて接する人のための意識啓発を目的にフォーマルメソッドの概念や効果について示す。主に、ベンダー上級管理者や情報システムユーザ企業・事業者の CIO¹、CTO²に向けた内容をまとめる。第 2 部は、フォーマルメソッド導入に関するプロジェクト管理における留意点や考え方をまとめる。主にベンダー上級管理者、開発プロジェクト管理者、開発技術者などに向けた内容をまとめる。第 3 部は、モデル検査を対象として、具体的な適用方法や技術的困難を解決する方法を示す。主に、開発技術者に向けた内容をまとめる。最後に、付録では、実システムに対するケーススタディ、フォーマルメソッドの実践応用に関する概要情報の事例集、手法選択に関する情報源等をまとめる。導入ガイダンス本編から参照される。

本書が対象とする情報システムは幅広く、重要インフラシステム、セーフティクリティカル・システム、家電など特定のハードウェアに組み込まれて利用される組み込みソフトウェアから汎用システムによるエンタプライズ系への適用が想定される。

本書の対象範囲と読み方

フォーマルメソッドを開発現場に導入するための方法やノウハウは、マネジメントから技術に至るまで幅広い範囲に渡って必要となる。プロジェクトマネジメントを行う管理者とシステムやソフトウェアの設計開発等を主に行う技術者のそれぞれに求められる手法、ノウハウを、マネジメント領域と技術領域に分類すると大まかに下表のようにまとめることができる。

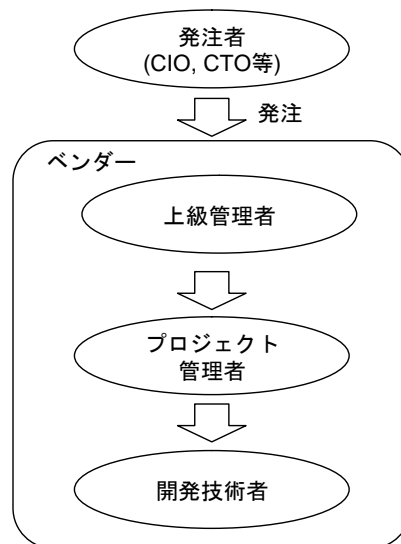
領域		対象者	
		管理者向け	技術者向け
意識啓発	意識啓発	<ul style="list-style-type: none"> ●形式手法を取り巻く環境と背景 ●形式手法の概要と特徴 ●導入効果と意義 	<ul style="list-style-type: none"> ●形式手法の概要と特徴 ●手法の比較情報 ●ケーススタディ
	組織管理・プロジェクト管理	<ul style="list-style-type: none"> ●費用対効果と具体的情報 ●組織への新技術の導入方法 ●形式手法導入の留意点 ●導入プロセスの手順 	
要求分析、設計	要求分析、設計		<ul style="list-style-type: none"> ●設計検証プロセス・アプローチ ●モデルの抽象化・状態爆発への対処 ●検証性質の記述パターン ●SPINオプションの使い方
	実装		(モデル検査ガイドブック(メルコパワー)等)
	テスト		(テストカバレッジの計測(VDMtools))

¹ CIO:最高情報責任者(Chief Information Officer)。企業において情報に関する資源を統括する最高責任者。

² CTO:最高技術責任者(Chief Information Officer)。企業において技術面を統括する最高責任者。

本ガイドンスは、管理者向けの意識啓発、組織管理、プロジェクト管理と、技術者向けの要求分析や設計などの上流工程に重点を置く。また、フォーマルメソッドの導入に関して、既存の文献ではあまり扱われていない領域のうち優先度の高い部分に重点を置くもので、フォーマルメソッドの導入方法を幅広く網羅するものではない。フォーマルメソッド導入の費用と効果の評価は困難な課題であるが、困難であることを理由に放置することなく、情報セキュリティ経済学などの関連する専門分野における知見に基づき、現在分かっていることや、何が困難であるかについて可能な限り示すことに努めた。技術編は、要求分析と基本設計工程を対象とし、実装やテスト工程は、既存の文献に委ねるなど、本ガイドンスの対象範囲を選別した。第 17.1 章などに示す既存の文献で具体的な方法が示されている領域については、本文でそれらの文献を参照することにより、本ガイドンスと合わせて活用して頂くことを想定する。

本書は、章ごとに想定される読者層が異なる。ここでは、フォーマルメソッドに関わるステークホルダーを大まかに下図のように考える³。



それぞれのステークホルダーの位置づけと本ガイドンスの読み方は以下の通りである。

(1) 発注者(CIO、CTO等)

発注者は、ベンダーに対して情報システムの開発を委託する組織である金融機関、重要インフラ事業者、情報システムのユーザ企業や政府などの組織において情報システムに関する技術的な責任を持つチームや CIO、CTO を指す。これらの人は、フォーマルメソッドを直接利用することはあまり想定されないが、厳格な受入れテストのためにフォーマルメソッドを理解することが求めら

³ ステークホルダーには、ここで挙げる主体以外に、製品・サービスの利用者、事業会社の株主、部品サプライヤー、開発コンサルティング会社などが存在するが、ここでは開発現場へのフォーマルメソッド導入という観点で関連の深い主体を対象とする。

れる場合も考えられる。

これらの読者は、第 1 部の意識啓発および、第 2 部の費用対効果の章を読むことで、フォーマルメソッドの導入に関する考え方や参考情報が得られる。また、発注者であっても、納品物の検査などにおいて、フォーマルメソッドの適用結果をレビューする場合には、第 2 部および第 3 部、付録のケーススタディなどを読むことを勧める。

フォーマルメソッドのような新しい技術をベンダーが導入するためには、その費用対効果について発注者からの理解が得られることが重要である。ソフトウェアの不具合に起因する事故リスクなど、発注者の視点からシステム納入や出荷後の影響も含む費用対効果を考慮して、意思決定することが重要である。

(2) 上級管理者、プロジェクト管理者

ベンダーにおいて開発プロジェクトの責任者となるプロジェクト管理者と、その上位の立場にある上級管理者は、フォーマルメソッドの導入について意思決定を行う立場にある。フォーマルメソッドに初めて接する人で、直接フォーマルメソッドを使う立場には無い場合が多いと想定されるが、導入意思決定において、フォーマルメソッドの効果や制約を理解することが重要である。フォーマルメソッドは、単一のプロジェクトで最初から採算性を確保することは困難な場合が多く、複数のプロジェクトを想定して組織として導入意思決定が行える上級管理者の理解が重要である。委託開発ではなく、製品開発の場合、ベンダーが発注者と同様の視点をもつことも求められる。

これらの読者は、第 1 部の意識啓発の章をよみ、フォーマルメソッドの概要とその意義について知るとともに、第 2 部のフォーマルメソッド導入における留意点と費用対効果の章を読むことで、プロジェクト管理者等の現場の責任者と協力して、フォーマルメソッドの導入判断を行う。

(3) 開発技術者

開発技術者は、開発現場において実際にフォーマルメソッドを利用する人を想定する。読者として、はじめてフォーマルメソッドに触れる人とすでに利用経験のある人を想定する。初めてフォーマルメソッドに接する読者は、第 1 部を読むことでフォーマルメソッドの概要と意義を理解する。第 2 部はマネジメント編であるため、詳細に理解する必要はないが、マネジメント層とのコミュニケーションを円滑に行うために役立つため、目を通し、フォーマルメソッド導入プロセスの全体像を把握し、プロジェクト管理者をサポートする知見を得ることが望ましい。第 3 部の技術編は、既存のフォーマルメソッドに関する入門書を読んでいることを想定しており、既存のテキストを参照しつつ、それらを補完するノウハウを提供する。

フォーマルメソッドをすでに実践している人にとっても、第 1 部、第 2 部は有用である。また、第 3 部を読むにあたって、フォーマルメソッドに関する基礎的な知見があれば、そのまま第 3 部に読み進むことができる。さらに、付録のケーススタディなども参考に読むことを勧める。

第2章以降の本編では、各章の最初に想定読者と前提知識等を記載することで、読者に指針

を与えるようにした。

目次

はじめに.....	i
本書の目的.....	i
本書の構成と対象読者.....	i
本書の対象範囲と読み方.....	ii
第1部 意識啓発編.....	1
1. フォーマルメソッド応用に関する背景.....	2
2. フォーマルメソッドの概要と特徴.....	5
3. フォーマルメソッド導入の意義と効果の概要.....	9
第2部 マネジメント編.....	12
4. 組織へのフォーマルメソッドの導入方法.....	13
4.1. フォーマルメソッドの導入プロセスパターン.....	13
4.2. フォーマルメソッド導入のポイント.....	21
5. フォーマルメソッド導入のコストと効果の考え方.....	24
5.1. コストと効果に関する検討手順の概要.....	24
5.2. フォーマルメソッドの効果とコストの全体像.....	25
5.3. フォーマルメソッドの効果について.....	28
5.4. フォーマルメソッドの投資コストについて.....	41
5.5. 導入判断における留意点のまとめ.....	44
6. フォーマルメソッドと関連技術の位置づけ.....	45
6.1. ソフトウェアに対する要求の把握.....	46
6.2. ソフトウェアの品質に関する対策技術とフォーマルメソッドの位置づけ.....	53
6.3. システム全体の一部としてのソフトウェアの位置づけ.....	55
6.4. まとめ.....	58
7. 手法の選択方法.....	60
7.1. 形式手法の比較選択の手順.....	60
7.2. 主な形式手法の概観.....	61
7.3. 主な形式手法の参考情報.....	64
第3部 技術編.....	81

8.	モデリングプロセスの構成と手順	82
8.1.	モデリング・プロセスのパターン分類	82
8.2.	コンポーネント駆動モデリング	85
8.3.	アルゴリズム駆動モデリングの概要	93
8.4.	検証性質駆動モデリングの概要	95
8.5.	まとめ	96
9.	モデルの抽象化と状態爆発の対策法	97
9.1.	状態爆発の問題点	97
9.2.	抽象化に関する留意点	98
9.3.	抽象化および状態爆発の対策の全体像	102
9.4.	Promela記述に関する状態爆発の対策	104
9.5.	本章のまとめ	114
第4部	付録	115
10.	ケーススタディ1:ブルーレイディスク	116
10.1.	検証の概要	116
10.2.	検証対象システムの仕様	117
10.3.	検証対象システムのモデリング	124
10.4.	検証性質の形式記述	131
10.5.	モデル検査と結果	133
11.	ケーススタディ2:入退室管理システム	135
11.1.	検証の概要	135
11.2.	検証対象システムの仕様	135
11.3.	検証対象システムのモデリング	142
11.4.	検証性質の形式記述	158
11.5.	モデル検査と結果	163
12.	応用事例情報	166
12.1.	事例の整理項目	166
12.2.	事例の調査結果	167
13.	手法の概要および選択法に関する情報	210
13.1.	その他の代表的な形式手法	210
13.2.	LTLとCTL	212
14.	ソフトウェア再利用とフォーマルメソッド	216
14.1.	ソフトウェア再利用	216
14.2.	再利用における検証の課題	217

14.3.	再利用における検証手法の例	219
14.4.	フォーマルメソッドの適用	220
14.5.	再利用資産の検証とフォーマルメソッド	221
14.6.	プロダクトの検証とフォーマルメソッド	223
14.7.	まとめ	225
15.	ソフトウェアの品質に関する計測の重要性	227
15.1.	計測の重要性	227
15.2.	計測データの信頼性	228
15.3.	計測情報と品質の関係について	228
15.4.	プロセス管理での利用上の効果	233
15.5.	簡便な計測方法	233
16.	モデル検査ツール(SPIN)の使い方等のヒント	235
16.1.	SPINモデル検査の基本的な実施手順例	235
16.2.	SPINモデル検査のオプション組合せヒント	235
16.3.	Windows版インストールの注意点	238
17.	関連文献、リンク集	240
17.1.	形式手法に関する解説文献のガイド	240
17.2.	フォーマルメソッドの研修や導入支援サービス	244
17.3.	フォーマルメソッドの普及活動等	245
17.4.	海外におけるフォーマルメソッド適用に関する情報	246
18.	フォーマルメソッド導入ガイダンス検討委員会	249
	表目次	251
	図目次	253
	索引	256

第1部 意識啓発編

概要

第1部は、主に上級管理者、開発プロジェクト管理者、発注者を想定し、フォーマルメソッドの概要や特徴を示すとともに、導入の効果や意義を示すことで、フォーマルメソッド導入の検討を促進することを目的とする。フォーマルメソッドをすでに導入済みあるいは検討中のベンダーを含めて幅広い層にとっても、フォーマルメソッドの背景について知る上で参考となる。

1. フォーマルメソッド応用に関する背景

電力、通信、金融分野等の重要インフラシステム、自動車、鉄道、航空機、医療機器等のセーフティクリティカル・システム⁴、情報家電等の様々な分野においてソフトウェアの大規模化、複雑化が日々進んでいる。このような中、ソフトウェアの不具合に起因する大規模な事故がしばしば発生していることから、ソフトウェアの安全性・信頼性⁵等を確保するための技法や工夫に対する要求が高まっている。

米国商務省技術標準局(NIST)は、ソフトウェアの品質等に関わる問題提起をするレポートにおいて、ソフトウェアの不十分な品質と、テスト方法の不適切さのために発生する損失が米国経済全体で年間 595 億ドル(約 5 兆 4000 億円)⁶に達すると報告している。

このような問題に対する対策アプローチの一つとしてフォーマルメソッド(=形式手法)に注目が集まっている。欧米では、鉄道、航空機、金融、セキュリティ分野などにおいて、ソフトウェアの安全性・信頼性を高めるためにフォーマルメソッドを適用した実践的な事例が増えている。フォーマルメソッドは、システムやソフトウェアの要求や設計仕様を、数学的に意味付けられた言語を用いて記述し、ツールを用いた論理的な検証⁷を行うことにより、従来のソフトウェア・テストでは困難であった性質の保証や不具合の検出などに利用される技術である。

フォーマルメソッドの導入効果として代表的なものには以下のようなものがある：

- 品質向上による事故損害リスクの低減
従来のテストでは検出できない不具合の検出や、厳密な仕様記述を通じて設計者とプログラマーの間で生じる誤解の防止などにより、ソフトウェアの品質を向上し、事故損害のリスクを低減する。従来のテストでは、並行動作するシステムのタイミングに関する検証は困難である。
- 品質に関する客観的な保証や説明責任の向上
論理に基づく検証により、検証結果に対して一定の保証を与えることが可能になる。これにより、人命や重大なセキュリティに関わるシステムの動作や性質の正しさについて一定の保証を与えることができる。重要インフラ事業者やセーフティクリティカル・システムの提供者は、利用者に対してシステムの安全性について十分な対策をとっていることを示すことが求められており、そのような要求を客観的に示すことに適している。
- 開発効率やプロセスの改善
開発プロセスの要求や設計等の上流に重点を置くことにより、不具合を早期に取り除き

⁴ システムの障害が、人命や身体を危険に陥れるような可能性のあるもの。

⁵ ソフトウェアの信頼性、システムの信頼性、安全性の違いや関係については、第 6.1 章で詳しく記述する。

⁶ NIST Study, Planning Report 02-3: "The Economic Impacts of Inadequate Infrastructure for Software Testing", 2002

⁷ ソフトウェアに対してテストデータを与えて実行することにより検査するのではなく、ソフトウェアの仕様や前提などから、論理的な推論に基づき満たされる性質を検証する。

見通しの良い開発プロセスとすることで、開発の生産性を向上する。

従来のソフトウェア開発のように作ってしまってから下流工程でテストを行う場合、見つかった不具合の修正に大幅な手戻り工数が発生していた。フォーマルメソッドの利用により、要求、設計等の開発の上流工程においてある程度の検証や解析が可能となり、プログラムを実装する前に早期に問題を発見し、下流工程からの手戻り等のコスト発生を防止する。

フォーマルメソッドによる品質向上の効果を考える上で、事故リスクの損害規模の把握が重要である。通常、ソフトウェアの利用者や開発者は、ソフトウェアに関わる事故リスクとして、サービスや業務の停止による収益の逸失、復旧コスト、リコール・回収コストなど直接被害を想定することが多いが、企業に対する信用失墜により、将来に渡るビジネスへのマイナス影響を含む企業価値への影響を十分に認識されていることは少ない。情報セキュリティ経済学の分野では、情報システムに関するセキュリティ事故によるこのような企業価値への影響を評価する手法について検証が進んでおり、直接被害以上に信用失墜のマイナス影響が大きいことが示されている^{8,9}。例えば、情報システムの不具合等による情報セキュリティ事故(不正アクセス、機密情報漏洩等)に係わる日本の上場企業が抱える潜在リスクは 29 兆円と見積られ、一社平均 79 億円の潜在リスクを抱えている。このようなリスクが十分に認識されていないことは深刻な問題である。

我が国の情報セキュリティ政策を検討する情報セキュリティ政策会議(議長:内閣官房長官)において決定された国の中期戦略「国民を守る情報セキュリティ戦略」(2010年5月)においては、情報家電、モバイル端末、電子タグ、センサ等あらゆるものがネットワークに繋がる環境において情報セキュリティを確保する方策として、開発者に対する検証ツールや安全性評価体制の整備等の環境整備・技術課題の解決を図ることを掲げている。また、「国民を守る情報セキュリティ戦略」に基づき策定された年次計画「情報セキュリティ2010」においては、情報技術の社会基盤化に伴い、情報システムに起因する事故が、経済活動全体の停滞や国民生活の生命・財産そのものにかかわるリスクをもたらしかねない状況が生まれつつあるため、対症療法的ではなく根本的な問題解決を目指した技術への取り組みを掲げている。このように、政府としても、情報システムやソフトウェアの品質を高めるための根本的な技術の確立に力を注いでいる。

一方、新興国を中心とした鉄道や電力など社会インフラ投資の増加、組込みシステム等の市場のグローバル化、ソフトウェア開発のオフショア・アウトソーシングの増加など、グローバル化が進む経済活動において、我が国のソフトウェア産業が国際競争力を向上させるために、品質の高さ追求することのみならず、品質の高さを客観的に説明する努力が求められている。また、経済・社会生活に浸透したソフトウェアを含むシステムの品質に対して、利用者である一般国民に対する説明責任も求められている。フォーマルメソッドは、鉄道、航空、プラント、セキュリティなどさまざまな分野における安全性・信頼性を確保するための手法として国際標準の中で採用され、国際的

⁸ 経済産業省、「グローバル情報セキュリティ戦略」、2006

⁹ Masaki Ishiguro, Hideyuki Tanaka, Kanta Matsuura, The Effect of Information Security Incidents on Corporate Values in the Japanese Stock Market, WESII 2006

に受け入れられる技術として活用することができる。さらに、オフショア・アウトソーシング¹⁰が国際的に進展する中で、これまで日本が得意としてきたものづくりの方式が変化してきている。今後、国内のソフトウェア関連企業は下流工程で付加価値を確保することが難しくなっており、付加価値の高い要求や設計などの上流工程で収益を確保するものづくりの方式を実現する上でフォーマルメソッドの導入は有効と考えられる。

以上のように、ソフトウェア開発に関わる環境は大きな転換期を迎えている。ソフトウェアに関わる事業者、発注者、開発者は、ソフトウェアの安全性・信頼性・セキュリティを確保するための有望な技術としてフォーマルメソッドに注目し、組織としての導入とその方法について具体的に検討する時期に来ている。

フォーマルメソッドには、様々な有効性があるにも関わらず、欧米と比較して国内の開発現場にはあまり導入が進んでいない。その理由として以下のようなことが挙げられる：

- 従来のソフトウェア開発手法を利用する開発者にとって、馴染みの無い概念やスキルが必要となり、新たな取組みのきっかけが得にくい。
- 実績に関する情報が得にくく、また、コストと効果に対する評価が難しいため、導入の意思決定が難しい。
- 導入初期の障壁を乗り越えるための組織としての導入ノウハウや留意点を示したガイダンスが少ない。

本導入ガイダンスは、このような障害を緩和し、フォーマルメソッドを開発現場へ普及促進させることを目指すものである。特に、フォーマルメソッドの費用対効果については、定量評価が難しい部分もあるが、第 5 章において費用と効果の全体像を示し、具体的なデータを示すことで、どの程度の評価が可能か示すようにした。

¹⁰ 開発等を含む自社内の業務の一部までや全てを物価の安い海外(オフショア)の外部企業に委託する形態。

2. フォーマルメソッドの概要と特徴

本章では、フォーマルメソッドの概要と特徴についてまとめる。本章の概要は以下の通りである。

対象読者	(1) 発注者(CIO, CTO 等) (2) ベンダー上級管理者 (3) 開発技術者等
目的	フォーマルメソッドを導入する組織の管理者やフォーマルメソッドを初めて学ぶ人に、フォーマルメソッドの概要や特徴について分かりやすく説明する。
想定知識	ソフトウェア開発の概要
得られる知見等	<ul style="list-style-type: none">● フォーマルメソッドの概要● フォーマルメソッドで何が得られるか● フォーマルメソッドの特徴

フォーマルメソッド(=形式手法)は、数学的に厳密に意味付けられた言語(「形式仕様記述言語」と呼ぶ)を用いて情報システム(ソフトウェア、ハードウェア等)の要求¹¹や設計¹²等の仕様を記述し、情報システムがユーザの要求等を満たしているかなど論理的に推論するための仕組みを提供する手法である。

フォーマルメソッドは、基礎とする数学理論などによって異なる多数の手法の総称であり、100以上の異なる手法¹³が提案されており、それぞれ記述する対象範囲や検証目的等の適性に違いがある。比較的、実践でよく利用されるフォーマルメソッドには、Bメソッド、Event-B、VDM++、SPIN、NuSMV、Zなどの例がある。また、ProVerifのように、プロトコル検証など特定の目的に対応した専門家向けのフォーマルメソッドなどにも有用なものがある。フォーマルメソッドは、鉄道分野、航空分野、金融・セキュリティ分野など、セーフティクリティカル・システムやミッションクリティカル・システムなどにおいて、欧米を中心に実システムに対する適用事例が増えている¹⁴。

フォーマルメソッドと対比される従来のソフトウェア・テストは、ソフトウェアの品質¹⁵(安全性、信頼性、セキュリティ等)を高めるため、不具合(バグ)を発見する目的で利用されるが、不具合が存在しないことを示すことができないという問題がある。特に、並列動作するソフトウェアや組込みシステム等において、並列動作のタイミングに関わる処理は再現性がないため、いくらテスト件数を増

¹¹ ソフトウェアに求められる機能などを明確に規定したもの。

¹² ソフトウェアに求められる機能をどのように実現するかその方法を規定するもの。

¹³ フォーマルメソッドの個々の手法を包括的に挙げたWEBサイトとして、Jonathan Bowenのサイト http://formalmethods.wikia.com/wiki/Formal_Methods_Wiki が代表的である。

¹⁴ IPA, 「形式手法適用調査」, 2010

¹⁵ 第6.1章に国際標準に基づくソフトウェアの品質についてまとめている。

やしても、すべての可能性について保証することができない。また、従来のテストでは、実行されなければいけない処理をテスト・ケースとして与えて動作確認をすることには向くが、テストケースだけを用いて、安全性やセキュリティなどのような望ましくない事が起きないことを示すことは困難である。

フォーマルメソッドでは、ソフトウェアがある性質を満たしていることを論理的に検証するため、一定の不具合が存在しないことを保証できる。そのような点で、完全な網羅性を持たない従来のテストの問題点を解決することができる。ただし、フォーマルメソッドを大規模なシステムに適用する場合などにおいて、コストが増大することもあるため、現実には、ソフトウェア・テストと部分的に使い分けるなど補完的な利用が想定される。

下図は、従来の開発プロセスにおける成果物とフォーマルメソッドの関係を示したものである。

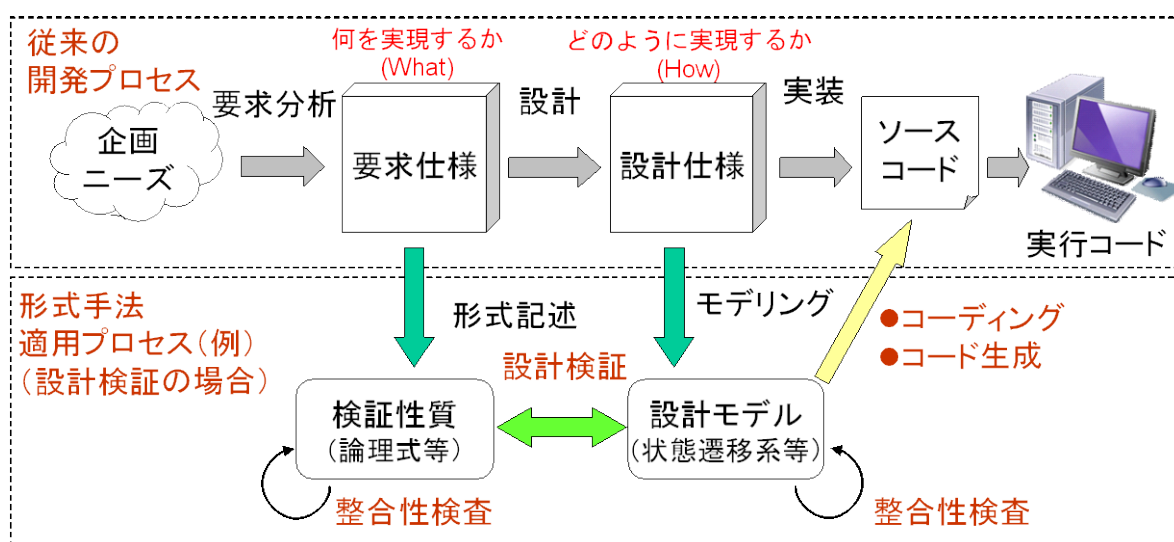


図 2-1:フォーマルメソッドの位置付け(設計検証の場合)

形式検証は、検証の基準(検証性質)と検証の対象(設計仕様)の両方について形式仕様言語を用いて記述し、それらをツールに入力することにより検証対象が、検証基準を満たしていることを検証する。

より具体的には、下図のとおり、日本語やダイアグラム等で書かれた要求仕様と設計仕様から、それぞれ形式仕様言語を用いて、検証性質と設計モデルをテキストベースで記述する。その両方をモデル検査ツールの入力とし、ツールの自動検査機能を用いて、検証性質が満たされるか、反例が存在するか確認することができる。

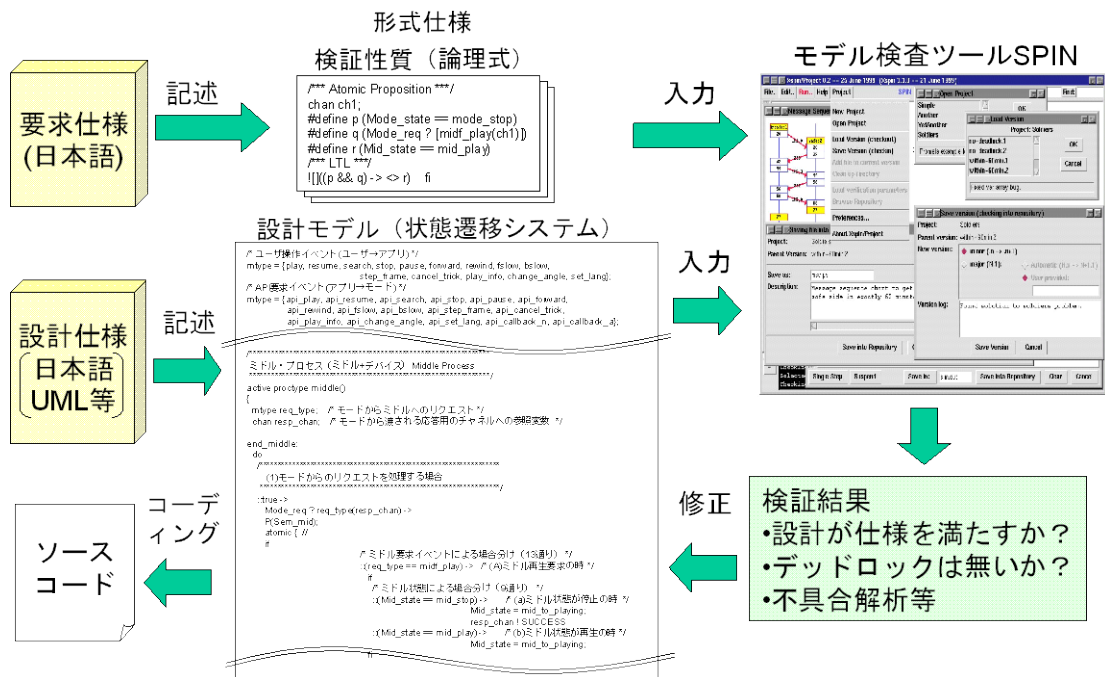


図 2-2: フォーマルメソッドの具体的なイメージ(モデル検査 SPIN の場合)

開発プロセスにおいて、形式仕様記述をどの範囲、どの程度(詳細度)まで行い、形式検証をどの程度(検証項目のうち検証する比率)まで実施するか検討すること(「フォーマルメソッドの適用レベル」と呼ぶ)は、非常に重要である。それは、適用レベルによって、開発コストや得られる効果は大きく異なるためである。それは適用対象に応じて判断することが重要である。適用レベルは、大まかに以下のように分けられる。

表 2-1: フォーマルメソッドの適用レベル

適用レベル	説明
レベル0	形式仕様記述 数学的な記述が可能である形式仕様記述言語を用いて仕様を記述する。形式仕様記述言語には VDM++ や Z 記法などがよく用いられる。記述だけでも、要求仕様の厳密な定義ができ、曖昧な定義による誤解の発生の防止などバグの低減に有効な場合がある。
レベル 1	形式的開発および検証 形式仕様を詳細化することでプログラムを開発、またはプログラムの性質を証明する。B メソッドなどが用いられる。
レベル 2	自動検証 プログラムの性質を自動的に検証する。

一方、フォーマルメソッド・ツールが提供する形式検証機能は、大まかに以下の2つのアプローチに分けられる。

表 2-2: 形式検証のアプローチ

検証法	説明
モデル検査	対象システムを状態遷移系によりモデル化し、検証したい性質を、時相論理式と呼ばれる論理的な式で記述し、状態遷移系の状態を網羅的に探索し、時相論理式が満たされるか検査する。検証は自動で実行されるため一般向けに導入しやすい面はあるが、状態爆発と呼ばれる問題により検査処理が終了しない場合があるなど、技術的な制約がある。性質が満たされないことが判明した場合、反例としてエラー状態に至る実行系列を表示させることができ、これにより不具合の原因特定に役立てることができる。代表的なツールに、 SPIN , NuSMV などがある。
定理証明	システムを論理式の集合などにより記述し、公理と推論規則に基づく証明を行う。証明においては人の支援が必要な場合が多く、証明をおこなうために専門知識を必要とする場合がある。モデル検査では困難な数値計算などを扱うことにも適している。代表的なツールとして、 B メソッド、 PVS などがある。

フォーマルメソッドは、文書(自然言語)で書かれた要求や設計の記述と比較して、厳密に記述するため記述コスト自体は大きくなるケースが多いが、ソフトウェア開発プロセスの上流に適用することにより、設計や要求の不具合を早期に発見し、テスト工程からの手戻りコストを大幅に抑える効果が期待できる。また、厳密な記述言語を用いてモデリングすることにより、ソフトウェアの安全性や信頼性などの品質が向上するという効果が期待できる。

適用対象に応じて、適用するフォーマルメソッドを適切に選択し、ソフトウェア全体のうち特定の部分に適用することで高い効果が得られる場合が多くみられるため、そのようなノウハウを身に付けることが重要である。本ガイダンスでは、そのような点についても手引きを示すことを目的としている。

3. フォーマルメソッド導入の意義と効果の概要

本章では、ソフトウェア開発における従来のソフトウェア・テストとフォーマルメソッドを比較して、フォーマルメソッド導入の効果と意義について概要を示す。

対象読者	(1) 発注者(CIO, CTO 等) (2) ベンダー上級管理者等
目的	フォーマルメソッドの効果と導入意義について、フォーマルメソッドの前提知識無しに分かりやすく説明する。
想定知識	ソフトウェア開発の概要
得られる知見等	● 現状のソフトウェア開発における問題点 ● 現状問題点に対するフォーマルメソッド導入の効果と意義の概要

開発現場にフォーマルメソッドの導入を図るためには、フォーマルメソッドの効果について、それを直接利用するベンダーから理解されるだけでなく、システム開発の発注者からも理解が得られることが期待される。システム納入後にソフトウェアの不具合に起因する事故が発生した場合、その損害は、発注者であるシステム利用者にも影響をおよぼし、企業価値の毀損などにつながるため、フォーマルメソッドの適用検討は、生産性向上のみならず、事故リスクの低減などシステム利用者による長期的な視点でコストと効果を判断することが必要なためである。

効果とコストに関する詳しい説明は第5章にまとめるが、本章では、その中でも重要な点を技術的な知識を用いず説明する。

フォーマルメソッドの効果と意義を示すために、従来のソフトウェア開発の問題点を挙げる。関連する問題点は以下のような事が挙げられる。

- ソフトウェア・テストに関する問題点

従来のソフトウェア・テストでは、どんなにテスト件数を増やしても、不具合が無いことを保証することが難しい。特に、処理タイミングに関わる再現性の無い動作については、テスト件数を増やすだけでは不具合が存在しないことを保証できない。

人命に関わるセーフティクリティカル・システムの制御、電力等の重要インフラ、決済、基幹システム、セキュリティ、膨大な普及数になる家電などの場合、ソフトウェアの不具合に起因する障害は極めて大きい損害をもたらす場合がある。このような場合に、明確に定義した性質が満たされることを保証することは大きな意味がある。

- 開発プロセスに関する問題点

ソフトウェア・テストは、設計、コーディングを経てプログラムが実行できるようになった段階で初めて行うことができる。そのためソフトウェア・テストで発見された不具合を修正するためには、開発の上流工程の要求仕様や設計仕様に遡って作業をやり直すことが必要になるこ

ともありトータルコストが増大するリスクがある。これにより納期遅延などの深刻な問題につながる。

以上のような問題に対してフォーマルメソッドは、以下のような効果が期待できる。

- ソフトウェア・テストの弱点の補強

フォーマルメソッドによりソフトウェアの設計仕様と検証性質を明確に記述し、ツールを用いて検証が成功すれば、設計仕様が検証性質を満たすことを保証することができる。セーフテイクリティカル・システムなど人命に関わるシステムにおいては、ソフトウェアが特定の性質を満たすことを保証することは極めて重要な意味を持つ。

一方で、従来のソフトウェア・テストでは、不具合の存在が発見されても、その原因を特定することが困難な場合がある。フォーマルメソッドの一部の手法では、不具合に至るソフトウェアの実行履歴を再現することで、不具合の原因の特定に有用な情報を得ることができる。

フォーマルメソッドは、ソフトウェアの信頼性や安全性等について論理的な検証により、客観的・科学的な保証を与えるためのアプローチとして利用することも有用である。

- 開発プロセスの改善

フォーマルメソッドの利用により、プログラムを作成する前に、要求仕様や設計仕様を記述した段階で、設計仕様が要求仕様を満たすことの検証や、矛盾の発見などの解析を行うことができる。これにより開発の上流工程で早期に不具合を発見することができるようになり、開発の手戻りによる工数増大と納期遅延リスクを抑えることが可能になる。

以上のようなフォーマルメソッドの効果のうち、定量的な評価が難しいものもあるが、大まかな規模を捉える上で以下のように考えることができる。

- 事故損害リスクの低減

フォーマルメソッドの特徴的な効果である検証性質が満たされることを保証したり、不具合の原因を特定することによるソフトウェアの品質向上は、最終的にはソフトウェアの不具合に起因するIT事故による損害リスクの低減として具現化される。

IT事故の損害リスクは、ソフトウェアを利用する事業者やベンダーが十分認識していると思われがちであるが、現実には、IT事故によってもたらされる事業者やベンダーに対する信用失墜を考慮すると、一般的な認識に比べて想像以上に損害リスクの規模が大きいことが分かっている。具体的には、情報セキュリティ経済学などの進展により、IT事故における復旧コスト、業務中断による営業機会損失、リコール、損害賠償等の直接的コスト以上に、信用失墜により将来に渡るビジネス機会の損失などの潜在的な損害が大きいことが示されている。詳しくは 5 章に示すが、具体的には、直接コストに対して信用失墜による損失を含む企業価値全体のコストは3倍程度になる場合がある結果が示されている¹⁶。

- 開発プロセスの改善

¹⁶ 第 5 章参照。

ソフトウェア開発におけるテスト工程のコストはバラつきはあるが 5 割程度と言われている^{17, 18}。また、NIST報告書では、開発コストの 80%は、欠陥を特定して修正することに費やされると見積もっている。フォーマルメソッドの適用により開発の上流工程に重点を置くフロントローディング¹⁹により、開発工数の大幅な削減が達成される実例が示されている。分野による違いは大きいですが、航空分野で適用されている実例では、フォーマルメソッドに基づく言語を用いて設計し、自動コード生成を活用することにより、コーディング、レビュー、テスト工程の 7～9割の工数が削減されている。一方で、鉄道分野においては、設計から実装に至るまでの工程を、フォーマルメソッドを用いた検証を徹底することにより、自動生成されたコードの単体テストを全く行わない事例などもある²⁰。

以上に示したようなフォーマルメソッド導入効果に関しては、適用対象や適用の仕方に応じて、メリットの規模に違いがみられるため、一概に大きなメリットが得られるとはいえないが、実践的な応用事例として、フォーマルメソッドの導入判断の参考とできる重要な情報である。より具体的な情報と考え方については、5章を参照すると良い。

¹⁷ ROBERT M. HIERONS, et. al., Using Formal Specifications to Support Testing, ACM Computing Surveys, Vol. 41, No. 2, pp. 9-76, 2009

¹⁸ W.S. Humphery, "Winning with Software. An Executive Strategy", Addison-Wesley, 2001

¹⁹ 開発の上流工程の要求分析や設計に重点を置いて集中的に負荷・資源を投入することにより、下流工程で発生する負荷を減らし、トータルの工数を下げる活動を指す。システム開発や製品製造で用いられてきた方法であるが、ソフトウェアの分野でも取り込もうとする動きがある。

²⁰ Meteor, A Successful Application of B in a Large Project, FM' 99, Vol. I, LNCS 1708, pp. 369-387, 1999.

第2部 マネジメント編

概要

第2部は、主に開発プロジェクト管理者、上級管理者を対象として、フォーマルメソッド導入におけるマネジメント面に関して留意点、必要な情報、ノウハウなどをまとめる。第5章のコストと効果については、ベンダーだけでなく、発注者や情報システムユーザにも参考となる情報を提供する。第2部で扱う内容は、フォーマルメソッドを導入する際の手順を網羅的に示すものではないが、開発プロセス全体において、フォーマルメソッド特有の困難や障害となる点を乗り越えるために必要な考え方を中心にまとめる。第4章は、フォーマルメソッドを開発現場に組織として導入する際の留意点や注意事項を示す。第5章は、フォーマルメソッド導入のコストと効果の全体像を示し、特に評価することが困難な部分について、具体的な参考情報を提示することで、評価の考え方をまとめる。第6章は、ソフトウェア開発プロセス全体におけるフォーマルメソッドの位置づけと他の関連技術との関係を示すことで、フォーマルメソッドの役割と限界についても記述する。第7章は、フォーマルメソッドのコストと効果に大きな影響を与える手法の選択について、考え方と俯瞰的な整理を示す。

4. 組織へのフォーマルメソッドの導入方法

本章では、フォーマルメソッドを組織に導入する際の障害を解決するための手順やポイントを示す。本章の概要は以下の通りである。

対象読者	(1)ベンダー上級管理者 (2)開発プロジェクト管理者 (3)開発技術者、等
目的	フォーマルメソッドを組織に導入する際の作業プロセス(導入プロセス)と導入のポイントについて整理する。さらに導入検討の際に有用と思われる情報源(研修サービス、コミュニティ等の情報)についても現状をまとめる。
想定知識	ソフトウェア開発プロセスの概略。
得られる知見等	<ul style="list-style-type: none"> ● フォーマルメソッドの導入プロセス ● 導入のポイント ● 導入検討に有用な情報源(研修サービス、コミュニティ等の情報)

4.1. フォーマルメソッドの導入プロセスパターン

フォーマルメソッドを組織に導入する際の手本となるプロセスを導入パターンとして示す。これらのプロセスパターンは、国内の企業・団体(約10組織)におけるフォーマルメソッド適用に関する代表的な事例(表 4-1)およびフォーマルメソッド導入に関する留意点等をまとめた文献^{21, 22}等に基づき、典型例をパターンとしてまとめたものである。

表 4-1: 導入プロセスの参考としたフォーマルメソッド適用事例

適用対象	適用の目的	手法・ツール	導入組織
IC カードファームウェア	仕様記述、 検証	形式仕様記述言語 VDM++、 VDM Tools、モデル検査 SPIN	開発組織
鉄道信号システム	仕様記述	形式仕様記述言語 VDM	開発組織
運賃計算システム	仕様記述	形式仕様記述言語 VDM++、 VDM Tools	開発組織
複写機制御ソフトウェア	検証	モデル検査 SPIN	開発組織
電力関連システム	検証	モデル検査 SMV	開発支援組織

²¹ Jonathan Bowen, Ten Commandments of Formal Methods, Ten Years Later, 2006, Computer, Vol. 28, No. 4, pp. 56–63.

²² Jonathan P. Bowen and Michael G Hinchey, Seven More Myths of Formal Methods, IEEE Software, 1995, Vol. 12, pp. 34–41

宇宙制御システム	検証	モデル検査 SPIN,UPPAAL,他	開発支援組織
----------	----	---------------------	--------

パターンは、適用の目的と導入組織の役割によって表 4-2 の通り、仕様記述導入パターン(開発)、検証導入パターン(開発)、検証導入パターン(開発支援)の 3 つのパターンにまとめている。

表 4-2: 導入プロセスのパターン分類

		適用の目的	
		仕様記述	検証
導入組織	開発組織	仕様記述導入パターン(開発)	検証導入パターン(開発)
	開発支援組織	—	検証導入パターン(開発支援)

各パターンとも導入の工程として、おおよその手順は共通している(各パターンにおいては導入工程における各フェーズの実施内容が異なっている)。その手順と本ガイダンスの参考箇所との対応は以下の通りである。

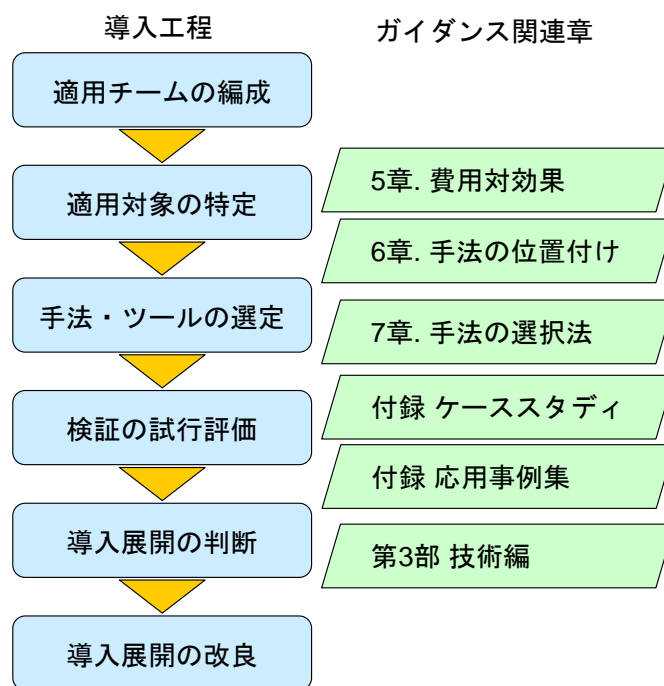


図 4-1: 導入プロセスの概略(共通部分)

上記の3つのパターンは導入を成功に導く典型例を示すもので、広く網羅しているわけではない。たとえば、仕様記述を適用の目的とし、開発支援組織が導入する際のパターンは成功事例の情報が得られなかったため、提示していない。ただし、上記の導入パターンの仕様記述導入パタ

ーン(開発)と検証導入パターン(開発支援)を参考に導入手順を検討することは可能である。実際に導入する際には、ここで提示しているパターンを参考に、各自の組織環境、適用対象などの事情に合わせて、導入計画を検討することが重要である。

各パターンの記述項目は以下の通りである。

項目	内容
導入組織	フォーマルメソッドを適用する組織。たとえば、システムの設計やソフトウェア開発を行う開発部門、システムの検証を担当する組織や品質管理部門、ツールや方法論等の整備普及を行う組織等。
適用対象	フォーマルメソッドを開発工程のどのフェーズでどこに適用するか。たとえば、新規開発システムの仕様記述や設計書に対する検証、既存システムの仕様書に対する記述、既存システムのプログラムコードに対する検証等。
導入によって解決したい課題 (適用の目的)	フォーマルメソッドによって解決したい課題。たとえば、仕様書の品質向上(曖昧な記述、記述漏れやあやまりの除去)、設計段階でのシステム検証、既存システムの不具合除去やテストではカバーできない検証の実施など。
利用する手法・ツール	形式仕様記述言語、モデル検査等のフォーマルメソッドの手法と具体的なツール例。
導入手順	フォーマルメソッドの導入計画、準備から実施にいたる作業項目。たとえば、適用対象の特定、情報収集とツールの選定、有効性評価、ツール・ドキュメント整備、教育、本格導入と評価等。
参考事例	当該パターンのベースとなった具体的事例に関する情報。

以下に各パターンの具体的な手順を示す。それぞれベースとなる事例は存在するが、パターンとして抽象化・整理を行っているため、実際の事例における導入の経緯とは異なる点もあることに留意されたい。

4.1.1. 仕様記述導入パターン (開発)

- ・ 導入組織
組込みシステム・制御システム、企業情報システム等の開発部門におけるソフトウェア開発チーム。
- ・ 適用対象
フォーマルメソッドを開発工程の中に組織的に組み込む。具体的には、システム開発の上流工程(外部仕様書あるいは機能仕様書の作成)にフォーマルメソッドを適用する。

- 導入によって解決したい課題
外部仕様書の品質確保(記載した内容の曖昧性、記述漏れ、間違いの排除)。開発従事者間における仕様・設計に関するコミュニケーションの改善。
- 利用する手法・ツール
形式仕様記述言語およびツール(VDM++およびVDM Tools、B method および Atelier B 等)。ただしモデル検査ツールの活用でも課題解決に繋がるケースもある。
- 導入手順
 - (1) チーム編成
形式仕様言語の導入検討、準備、推進を行うチームを編成する。推進チームは、ソフトウェアエンジニアリングに関する知見を持つ者や、形式仕様言語の有効性・限界を把握している者で構成する。推進者の負担軽減とフォーマルメソッドの適用における困難な問題を解決するために、外部有識者からの協力を得られる体制を構築する。
推進チームは以降の活動を行う。
 - (2) 適用対象範囲の特定
開発対象となるソフトウェアにおいて、ソフトウェア全体の開発に形式仕様言語を適用するのか、一部のモジュールに適用するのか、その範囲と適用する詳細度を特定する。
詳細度は、要求仕様、基本設計、詳細設計等の適用するレベルの選択を意味する。
 - (3) 手法・ツールの選定
適用対象とするソフトウェアの外部仕様書の作成に対し、手法の特徴、適用実績、ドキュメント・書籍やツールのサポート状況等の観点で候補となる手法・ツールを選定する。
ドキュメント・書籍等で得られる情報には限界があることから、手法を熟知している外部有識者から技術指導を受けることが効果的である。《手法・ツールの選定にあたっては、本ガイドの「7. 手法の選択方法」も参照のこと》
 - (4) 適用の試行・評価
開発対象となるソフトウェアの一部あるいは類似の小規模なソフトウェア等を対象に選定した手法の適用を試行し、手法の有効性を評価する。手法やツールの適切な利用方法、限界を踏まえて評価を行う必要があるため、手法・ツールを熟知している外部有識者と共同で試行評価を行う。また、実際の記述では、対象システムのドメイン知識が不可欠となるため、当該ドメインに詳しい者の協力が必要である。
 - (5) 導入可否の判断
上記の試行評価の結果から、導入可否の判断を行う。導入を決めた場合は、導入計画を策定する。導入準備、プロジェクトへの適用および評価等で以降に記すような活動を計画し、そのための体制を確保する。《導入可否の判断に関しては、費用対効果の面での検討も必要である。本ガイドの「5. フォーマルメソッド導入に関するコストと効果の考え方」も参照のこと》
 - (6) 周辺環境の整備

導入の準備として、開発チームのメンバーに手法・ツールを使ってもらうための周辺ツールの整備(対象ソフトウェアを前提とした仕様テンプレートや開発フレームワークの整備など)やドキュメント整備を実施する。

(7) 研修の設計と実施

さらに、手法・ツールと周辺ツール(仕様テンプレートや開発フレームワーク等)の使い方に関する研修を設計し、実施する。研修の設計にあたっては、手法を理解するために必要な基礎知識(特に論理、集合、写像などプログラミング以外の知識)についても整理し、研修内容に組み込む。

(8) 実践適用

実際の開発プロジェクトに適用する。適用の際には、手法およびツールに関する質問や適用に際しての問題点を開発チーム内で常に共有し、対応できる体制を整えておくことが必要である。

(9) 有効性評価と適用方法の改良

開発工程の各フェーズでかかった工数や不具合検出数などの定量データに基づき、コストおよび品質の両面から適用の有効性を評価する。また、開発チームのメンバーから、手法・ツールの利用に関する質問(良く出た質問)や手法・ツールの良い点・課題などの声を集める。これらの結果に基づき、2回目以降の開発に向けて、ツールやドキュメント、開発プロセスの改善を行う。チームメンバーの習熟により、2回目以降の開発では初回の開発より工数の削減が期待できる。

- 参考事例

栗田:「仕様書の記述力を鍛えるーモバイル FeliCa 開発における形式仕様記述手法の導入事例」、日経エレクトロニクス, pp133-152, 2007

高橋他:「鉄道信号システムへのフォーマルメソッドの適用」、鉄道と電気技術、Vol.20、No.2、2009

幡山他:「フォーマルメソッドによる仕様品質向上への取り組み～運賃計算仕様書の記述の改善～」、鉄道サイバネ・シンポジウム、2010

4.1.2. 検証導入パターン (開発)

- 導入組織

組み込みシステム・制御システム等の製造部門におけるソフトウェア開発チーム。

- 適用対象

開発中あるいは開発済みのシステムの設計書(UML 等で記述した設計モデル)やプログラムコードに対する検証を行う。

- 導入によって解決したい課題

設計フェーズや製造フェーズで作りこんでしまった不具合がテストフェーズでも検出できない場合がある。複数プロセスが並行動作する非同期制御システムで再現性の低い不具合

では不具合除去が難しい。こうした不具合を可能なかぎり検出・除去したい。

- 利用する手法・ツール

モデル検査 SPIN、SMV、UPPAAL 等

- 導入手順

(1) 適用チームの編成

モデル検査の概要を理解できる技術者をメンバーとした適用チームを構成する。適用にかかる負担を減らすため、チームでの実施が望ましい。また、モデル検査手法・ツールのより深い知見の習得のため、手法・ツールを熟知している外部有識者の協力が望ましい。適用チームは以下の活動を行う。

(2) 適用対象範囲の特定

適用対象となる設計モデルやプログラムコードを特定する。また、デッドロックなど検証したい内容を明らかにする。適用対象がプログラムコードで不具合が明確には、その不具合の現象に基づいて検証内容を明確化する。

(3) 手法・ツールの選定

適用対象となるシステムの特長(並行性やリアルタイム性など)と検証項目の内容にもとづき、モデル検査手法の特徴、ドキュメント・書籍やツールのサポート状況等から利用する手法・ツールを選定する。ドキュメント・書籍等で得られる情報には限界があるため、手法・ツールを熟知している有識者から技術指導を受けることが効果的である。《手法・ツールの選定にあたっては、本ガイドの「7. 手法の選択方法」も参照のこと》

(4) 適用の試行・評価

適用対象となるシステム的设计モデルあるいはプログラムコードから、検証対象となる部分をモデル検査の記述言語で記述し、検証を試行評価し、有効性を検証する。適切な記述や検証の方法、検証の限界、検証結果の分析方法を理解して、評価を行う必要があるため、手法・ツールを熟知している有識者と共同で試行評価を行うことが望ましい。また、選択した形式手法によっては検証には高い技術レベルが要求されるものがあるため、検証部分は外部にアウトソーシングすることが効果的な場合もある²³。また、実際の記述では、対象システム的设计および実装に係る詳細情報とドメイン知識が不可欠となるため、開発当事者等の協力が必要となる。

(5) 適用可否の判断

上記の試行評価の結果から、他のシステム・モジュールの検証にも導入するかどうかの判断を行う。導入する場合は、他のシステム・モジュールにも展開するためのツール整備を行う。たとえば、設計モデル(xUML や状態遷移表等で記載されたもの)やプログラムコードからモデル検査の記述言語(SPIN の場合は Promela)への変換ツール、検証結果(不具合に関する情報)の分析ツール等を整備する。《導入可否の判断に関しては、

²³ 検証サービスを提供する企業は、フランス ClearSy などある。国内では、形式手法を専門とするベンチャー企業が設立される例も見られるが、企業数はまだ少ない。

費用対効果の面での検討も必要である。本ガイドの「5.フォーマルメソッド導入に関するコストと効果の考え方」も参照のこと》

(6) 有効性の評価と適用法の改良

他のシステムやモジュールにも適用し、適用可能なシステムの規模や検出できる不具合の条件(検証可能な項目)等について評価を行う。また、適用チーム外のメンバーが利用できるようにするためのドキュメント整備や評価結果に基づくツールの改良整備を行う。

- 参考事例

村石他:「Model Checking を適用した実践的非同期制御検証」、ソフトウェアテストシンポジウム JaSST'07、2007

篠崎他:「モデル検査のデバックへの適用」ソフトウェアテストシンポジウム JaSST'06、2006

篠崎他:「支援ソフトウェアを活用した実践的モデル検査」ソフトウェアテストシンポジウム JaSST'08、2008

只野他:「モバイル FeliCa IC チップ開発における SPIN を用いたモデル検査による品質確保」、信学技法 SS2010-21、2010

4.1.3. 検証導入パターン（開発支援）

- 導入組織

開発現場に対する支援部門・品質保証部門(開発現場へのツール展開や成果物に対するレビューやテスト支援などを担当するチーム)

- 適用対象

開発中あるいは開発済みのシステムの設計書(UML 等で記述した設計モデル)やプログラムコードに対する検証を行う。

- 導入によって解決したい課題

開発部門が抱える以下の不具合を検出・除去する支援をしたい。または開発部門に以下の不具合を検出・除去する手法やツールを展開したい。

設計フェーズや製造フェーズで作りこんでしまった不具合がテストフェーズでも検出できない場合がある。複数プロセスが並行動作する非同期制御システムで再現性の低い不具合では不具合除去が難しい。

- 利用する手法・ツール

モデル検査 SPIN、SMV、UPPAAL 等

- 導入手順

(1) 適用チーム編成

モデル検査の概要を理解できる技術者をメンバーとした適用チームを構成する。適用

にかかる負担を減らすため、チームでの実施が望ましい。また、モデル検査手法・ツールのより深い知見の習得のため、手法・ツールを熟知している外部有識者の協力が望ましい。適用チームは以下の活動を行う。《手法・ツールの選定にあたっては、本ガイドの「7. 手法の選択方法」も参照のこと》

(2) 適用対象範囲の特定

適用対象となる設計モデルやプログラムコードを特定する。また、デッドロックなど検証したい内容を明らかにする。適用対象がプログラムコードで不具合の存在が分かっている場合には、その不具合の現象に基づいて検証範囲を絞り込む。

(3) 手法・ツールの選定

適用対象となるシステムの特長(並行性やリアルタイム性など)と検証項目の内容にもとづき、モデル検査手法の特徴、ドキュメント・書籍やツールのサポート状況等から利用する手法・ツールを選定する。ドキュメント・書籍等で得られる情報には限界があるため、手法・ツールを熟知している有識者から技術指導を受けることが効果的である。

(4) 適用の試行・評価

適用対象となるシステムの設計モデルあるいはプログラムコードから、検証対象となる部分をモデル検査の記述言語で記述し、検証を試行評価し、有効性を検証する。適切な記述や検証の方法、検証の限界、検証結果の分析方法を理解して、評価を行う必要があるため、手法・ツールを熟知している有識者と共同で試行評価を行うことが望ましい。また、実際の記述では、対象システムの設計および実装に係る詳細情報とドメイン知識が不可欠となるため、開発当事者等の協力が必要となる。

(5) 適用可否の判断

上記の試行評価の結果から、他のシステム・モジュールの検証にも導入するかどうかの判断を行う。導入には、当該部門が、フォーマルメソッドを活用し、開発部門を支援する形態と、開発部門に対し、フォーマルメソッドの導入を支援・促進する形態がある。前者の場合、フォーマルメソッドの想定ユーザは開発支援部門であり、後者の場合は、開発部門の現場の技術者である。導入する際はどちらの導入形態とするかを定めた上で、導入計画を策定する。いずれの場合も、導入準備、プロジェクトへの適用および評価等で以降に記すような活動を計画し、そのための体制を確保する。《導入可否の判断に関しては、費用対効果の面での検討も必要である。本ガイドの「5. フォーマルメソッド導入に関するコストと効果の考え方」も参照のこと》

(6) 周辺環境の整備

導入準備として、他のシステム・モジュールに展開するためのツール整備を行う。たとえば、設計モデル(xUML や状態遷移表等で記載されたもの)やプログラムコードからモデル検査の記述言語(SPIN の場合は Promela)への変換ツール、検証結果(不具合に関する情報)の分析ツール等を整備する。

(7) 実践適用

実際に他のシステムやモジュールにも適用し、適用可能なシステムの規模や検出できる不具合の条件(検証可能な項目)等について評価を行う。また、今後の利用に必要なドキュメント整備や評価結果に基づくツールの改良整備を行う。

(8) 適用方法の改良

開発部門に導入を支援する場合は、そのためのツールの改良整備、ドキュメント整備を行う。たとえば、ツールの使い方に関する研修を設計し、実施する。研修の設計にあたっては、手法を理解するために必要な基礎知識(特に時相論理などプログラミング以外の知識)についても整理し、研修内容に組み込む。開発現場から、ツールの利用に関する質問を受け付ける体制を作る。また、手法・ツールの良い点・課題などの声を集め、さらなる整備・改善につなげる。

・ 参考事例

篠崎、早水:「企業におけるフォーマルメソッドの実践」、組込みシステム技術に関するサマールワークショップ SWEST10 2008

篠崎他:「支援ソフトウェアを活用した実践的モデル検査」ソフトウェアテストシンポジウム JaSST'08、2008

氏原:「JAXAにおける高信頼性ソフトウェア開発のためのモデル検査の実用化」、システム設計検証技術研究会、2010

氏原他:「宇宙機搭載ソフトウェアの高信頼化を目的とした第三者評価活動(Independent Verification and Validation: IV&V)の成果と今後の課題」、第5回システム検証の科学技術シンポジウム、2008

4.2. フォーマルメソッド導入のポイント

前節で触れた導入事例の多くに共通する成功要因や導入を検討・試行したものの本格導入に至らなかったケースの障害要因からフォーマルメソッド導入のポイントを以下にまとめる。

4.2.1. 適用チームの編成

前節であげた事例のいずれの組織においても導入推進役の存在が重要である。導入推進は単独の担当者によるものではなく、複数人で構成されるチームによって実施されている。導入プロセスパターンにあるように、導入の検討から本格導入まで多くの作業が考えられる。さらに技術的な検討においては単独の担当者では負担が大きく、組織で理解を得る途中で挫折するリスクも大きい。フォーマルメソッド導入の必要性に関して同じ問題意識を持つメンバーを集め、適用チームを編成することが重要である。社内コミュニティや勉強会等のテーマとしてフォーマルメソッドを提案することや、社内研究制度などを活用し、組織的に認められたチームを編成することも1つの方策である。適用チームに組織管理者を巻き込めれば、より効果的である。また、試行評価において、対象システム的设计および実装に係る詳細情報やドメイン知識が不可欠となるため、開発当

事者や当該ドメインに精通した者の協力が必要となる。

4.2.2. 適用対象・課題の明確化

フォーマルメソッドの導入目的として、仕様書の品質向上、設計段階での検証、既存システムの不具合除去等があげられている。フォーマルメソッドの有効性を検証するためには、実際のシステム開発案件において、こうした観点で現場が直面している具体的な課題を切り出し、適用を試みることが重要である。適用システムや課題が架空のものでは、その効果に関する訴求力は限定的である。しかしながら、現在進行中のプロジェクトに適用することはリスクが高い。開発済みのシステム案件で、今後、類似の開発プロジェクトが見込める案件の成果物への適用や、従来のテストとは異なるアプローチからの再検査が望ましいシステム(一部モジュール)への適用など、現開発プロジェクトに対する影響が少なく、フォーマルメソッド導入が今後の品質向上に寄与することがイメージしやすい案件を対象にすることが現実的である。

4.2.3. 外部有識者との連携

フォーマルメソッドに関する書籍やドキュメントも徐々に増えており、その概要を理解することは容易になりつつある。一方、実際の導入にあたっては、各手法・ツールの特性と限界を正確に理解し、正しい使い方による試行と評価を行う必要がある。しかしながら、このような実際の適用に必要な情報が充分に開示されているとはいえないのが現状である。外部有識者の支援なしに導入を進めることは、困難な問題に直面した際に適切な評価なしに導入を断念するケースに陥るリスクもある。フォーマルメソッドの導入に成功している多くの企業では外部有識者の支援が重要な役割を果たしている。

4.2.4. 試行評価を通じた成功体験

フォーマルメソッドは数学的知識を必要とし、一般のソフトウェア開発者・技術者にとって理解が難しいとの指摘がある。また、フォーマルメソッドに限らず、新規の開発手法の導入は手法の理解や習得などのために現場に追加の負担を強いることとなる。したがって、フォーマルメソッドの導入・展開にあたっては、導入現場のメンバーから導入の意義や有用性の理解を得ることが非常に重要である。こうした理解を得るために成功事例(公開情報)は必要であるがそれだけでは十分ではない。公表される成功事例はあくまでも他組織の事例であり、当事者の環境や開発自体は多様であり、自らの課題に適用できるかどうか不明確であること、目に見えない技術的課題やデメリットが存在する可能性がその理由である。有用性の理解には、その有用性を自らが体験すること、すなわち成功体験が重要である。たとえば、「適用対象・課題の明確化」でも述べたように、現場が抱えている課題に対して適用を試み、従来のレビューやテストで検出できなかった仕様書のミスやシステム不具合が検出できたなど、分かりやすい形で成果を得られることが重要である。また、同時にフォーマルメソッド適用の限界や他のアプローチ併用の必要性を、体験を通じて理解することも必要である。

4.2.5. 管理者・導入現場の理解

フォーマルメソッドを組織的に導入するには、導入現場の理解以前に組織管理者あるいはプロジェクト管理者の理解が必要であることはいうまでもない。組織管理者あるいはプロジェクト管理者に対しては、「5. フォーマルメソッド導入に関するコストと効果の考え方」にあるようなフォーマルメソッドの意義とコストに関する理解を得ることが必要である。さらに、「適用対象・課題の明確化」、「評価試行を通じた成功体験」で指摘したように、従来のレビューやテストで検出できなかった仕様書のミスやシステム不具合が検出できたなど分かりやすい形で具体的な効果が得られ、品質、コスト、リスクの観点で見える化できれば、理解の促進が期待できる。

5. フォーマルメソッド導入のコストと効果の考え方

本章では、フォーマルメソッドの導入について検討する際のコストと効果に関する考え方と考慮すべき点について示す。概要は以下の通りである。

対象読者	(1)ベンダー上級管理者 (2)開発プロジェクト管理者 (3)発注者(CIO, CTO 等)
目的	フォーマルメソッドを組織として導入するか検討するための判断材料として、フォーマルメソッドの効果とコストの全体像を示し、それぞれの要素がどのぐらいの規模になるか実績や具体例を示す。これにより、事故リスク等の規模などまだ十分には認識されていないものについて、考慮すべき点を示す。
想定知識	ソフトウェア開発プロセスの概略。
得られる知見等	<ul style="list-style-type: none">● フォーマルメソッド導入の際に検討すべきコストと効果の全体像● 損害リスクの規模感の認識とリスク評価の重要性● 組織として導入を検討する際の注意点● 実践的な応用事例に基づく現状の把握● 実践的な応用事例(付録)の利用法

5.1. コストと効果に関する検討手順の概要

組織へのフォーマルメソッドの導入の可否を判断する上で、コストと効果に関する検討のための参考手順の概要を以下に示す。

- (1) 本ガイダンスに示すフォーマルメソッドの効果とコストに関する全体像(5.2 章)を把握し、それを参考に実際の対象事例・分野について主要要素を洗い出す。
- (2) 効果とコストの主要要素について、ガイダンスに示す具体例や考え方(5.3 章、5.4 章)に基づき、対象事例・分野について規模を大まかに見積もる。
- (3) 適用事例によっては、品質向上、コスト削減、リスク低減等の効果を相互に比較できるほどの定量化を行うことが難しい場合も多いため、適用事例の状況に応じて、優先すべき効果の要素を中心に考える。
- (4) コスト(学習コスト、設計コスト、ツール導入コスト等)の考え方、事例を参考にして、対象事例のコストの規模を大まかに見積もり、効果の全体あるいは優先すべき特定の効果の規模と比較・検討することにより、導入を判断する。

フォーマルメソッド導入のコストと効果の要素には定量的に比較することは難しいものが多い。

しかし、主要な要素を見落とすなど偏った判断を避けるために、主な要素と全体像を把握して、各要素の大まかな規模を見積もることは非常に重要である。本章で示す内容は、コストと効果の考え方やそれらの規模に関する参考情報であり、費用対効果に関する厳密な定量評価手法を示すものではない。また、そのような定量化手法を示すことは、現実の場では困難な場合が多いと言える²⁴。

5.2. フォーマルメソッドの効果とコストの全体像

フォーマルメソッドの効果には、従来のソフトウェア・テストでは発見が困難な不具合の検出やソフトウェアが要求を満たすことを論理的に保証することなどによりソフトウェア等の安全性・信頼性等の品質²⁵の向上や、上流の設計工程で不具合を除去することによりテスト工程の工数を削減することなどが挙げられる。これらの効果は、適用対象、適用手法、適用度合いによって異なるため、それぞれの事例に応じて効果を評価することが必要である。品質の向上に関する効果は、不具合による情報システムの障害等が発生して始めて顕在化するものであり、最終的には、情報システムに関わる組織（ユーザ、ベンダー等）の事業や企業価値への影響として捉えられるものである。これらを適切に捉えるためにはリスク評価の考え方が必要になる。多くの企業等では、情報システムに要求される品質に対して十分な対策をとっていると考えているが、実際に発生している情報システムの不具合に起因する事故の件数やその被害の影響規模の大きさをみれば、対策が十分とは言えない^{26, 27, 28}。対策が十分であるかどうかは、利用される情報システムの重要度に応じて異なる。PCなど個人用途や基幹系ではない事務向けなどの一般的なソフトウェアであれば、ソフトウェアの信頼性の高さよりも、機能の新規性や有用性を重視し、市場にいち早くリリースすることが市場競争力を決める決定的要因となっている^{29, 30, 31}。一方で、重要インフラ、企業の基幹系、決済系、セキュリティなどのソフトウェアにおいては、その影響度は極めて大きいため、ソフトウェア品質に対する要求は高い。

フォーマルメソッドの効果の全体像に把握するためには、IT投資対効果の評価手法であるVMM(Value Measuring Methodology)^{32, 33}などを参考にすることができる。VMMやプロジェクト

²⁴ 機能安全標準(IEC61508)では、障害を確率的ハードウェア障害(Random hardware failure)と系統的障害(Systematic failure)に分類し、前者についてはハードウェアの物理的劣化による障害発生時の定量的取り扱いを行うが、後者については設計や実装の不具合など確定的にきまるものであり、その不具合の混入確率を定量的に評価することは困難であるため、開発の手順や手法などの定性的な取り扱いによる対策を定めている。

²⁵ ソフトウェアの品質や機能要求、非機能要求等の関係については、第 6.1 章を参照。

²⁶ 障害を発生させない、被害を拡大させないためのシステム対策ガイド、JUAS, IPA, 2009

²⁷ 情報システムの信頼性向上ガイド, JUAS, 2010

²⁸ 高信頼化のための開発手法ガイドブックー予防と検証の事例を中心にー, IPA, 2010

²⁹ Michael Cusumano, Critical Decisions in Software Development:Updating the State of the Practice, IEEE Software, 2009.

³⁰ Michael Cusumano, Software Development Worldwide - The State of the Practice, IEEE Software, 2003.

³¹ Michael A. Cusuman, ソフトウェア企業の競争戦略, サイコムインターナショナル

³² 米国連邦調達庁(GSA)、ハーバード大学、ブーズ・アレン・ハミルトンコンサルティングなどにより開発された IT 投資対効果の評価手法で、米国政府内のさまざまなプロジェクトに活用されている。

³³ CIO council, Best Practice Committee, Value Measuring Methodology, 2002.

管理等の手法では、通常、このような技術投資の効果は、(1)価値の向上、(2)コストの削減(効率化)、(3)リスクの低減の3つの要素に大きく分類される。フォーマルメソッドの場合については、表 5-1 のように分類される。価値の向上は、フォーマルメソッドのそもそもの導入目的であるソフトウェアの品質向上であり、コストの削減は、生産性の向上による工数の削減、リスクの低減は、特定の工程における負荷の集中による開発要員の変動や、開発手戻り等による工数見積りの大きな変動等による納期リスクの低減である。

表 5-1:フォーマルメソッド導入の主な効果

分類	主な例	説明
(B1)品質向上	不具合の除去。 要求仕様の動作保証。 説明責任の向上。	バグに起因するシステム障害により、サービス停止、人命・経済損失等の被害をもたらす可能性を減らす効果がある。また、不具合が発見されることにより、企業に対する信用失墜の損害も大きい。
(B2)コスト削減	開発工数の削減	開発上流の要求・設計工程に重点を置くことで、コーディング、テストの工数を削減できる場合がある。また、開発の効率化により、現場の負担を軽減する。
(B3)リスク低減	工数、予算見積もり、品質のバラつきを抑える。	要求・設計工程の重点化により、手戻り等による工数の変動リスクや、品質のバラつきリスクを抑える。

品質向上の効果は、具体的には、不具合による製品の修理回収やシステム障害による損害の回避や、さらには、事故やシステムに不具合の顕在化により企業に対する信用失墜、企業価値の毀損の回避などが挙げられる。前者については、人命に関わるセーフティクリティカル・システムや企業等の基幹業務に関わるミッションクリティカル・システムのみならず、情報家電においてさえ、不具合が見つかった場合の回収修理コストなどで 100 億円程度の大規模な損害が発生している例もある³⁴。後者の規模については、一般に、把握している人は少ないが、欧米を中心とする情報セキュリティ経済学分野では企業の市場価値に基づく評価手法が示されており、その影響はきわめて大きいことが示されている^{35, 36}。あるベンダーが開発したシステムに不具合が見つかり企業の信用が失墜すれば、その製品の売上げのみならず、同社のビジネス全体に大きな影響を与える場合がある。フォーマルメソッドは、ソフトウェアの新しい機能を生み出すものではなく、機能などの品質を高めることが目的の一つであるため、具体的な効果は障害の発生どうい不確定な事象

³⁴ http://www.sony.co.jp/SonyInfo/News/Press_Archive/200107/01-0706/

³⁵ Lawrence A. Gordon, Martin P. Loeb, Managing Cybersecurity Resources A Cost-Benefit Analysis, McGraw-Hill, 2006.

³⁶ Masaki Ishiguro, Hideyuki Tanaka, Kanta Matsuura, The Effect of Information Security Incidents on Corporate Values in the Japanese Stock Market, WESII 2006.

に依存して決まる³⁷。

コスト削減の効果については、フォーマルメソッドの導入により要求分析や設計などの上流工程で、仕様の検証やシミュレーション実行などの解析が可能となり、開発工程の早期に不具合を取り除き、テスト工程の工数を削減するとともに、テスト工程からの手戻りによる工数増大を防止する効果が大きい³⁸。また、従来の開発プロセスのように、テスト工程に負荷のピークが集中することで、開発要員の許容負荷のオーバーや手戻りコストの変動による納期遅延のリスク³⁹を低減する効果が得られる。

一方、フォーマルメソッドの導入のための投資コストについては、主に以下のように分類される。

表 5-2:フォーマルメソッドの主な投資コスト

主な投資コスト	内容
(C1)フォーマルメソッド習得コスト	フォーマルメソッドごとに手法の基礎、ツールの使い方についての学習。技術指導・コンサルティング委託費用もある。
(C2)設計工数の増大	従来の設計にフォーマルメソッドを組み込む場合、設計工数が増加する場合がある。
(C3)ツール取得コスト	無償のツールは多いが、統合開発環境となっているツールで有償の製品がある。

フォーマルメソッドの習得コストは、利用する手法や対象システムにフォーマルメソッドをどの程度のレベルまで適用するかによって異なる。学習コストは、手法により異なるが、特定の手法に関する入門書、参考書の通読と例題等による実験等の時間から見積もることができる。また、設計工数の増大に関しては、従来の設計に比べて、フォーマルメソッドにより厳密で曖昧性のない仕様を作成し、設計工程で、仕様の矛盾や整合性等の検証を行う場合、従来の設計に比べて工数が増大する場合が多い。ツール取得コストに関しては、フォーマルメソッドの場合、無償のツールが中心であるが、統合的な開発環境として有償のものも存在する。これらの投資コストは、効果に挙げたリスク評価に比べ、見積もりは容易である。

フォーマルメソッドの効果は、開発現場の視点から工数や生産性向上などの短期的な観点のみならず、事業者や利用者などの視点からシステム障害による損害などビジネスリスクも考慮した広い視点から考えることが重要である。

以上のような効果とコストの要素の中には、定量化することが難しいものもある。しかし、要素ご

³⁷ 「(B1)品質」は、開発プロジェクトの成果物に関する価値であり、事故による損害リスクは、ソフトウェアの品質に依存するため(B1)に含める。「(B3)リスク」は、開発プロジェクトの失敗や遅延などのプロジェクトのリスクを対象とする。

³⁸ Marko Auerswald, EASIS Guidelines for the Development of Dependable Integrated Safety Systems, 2006

³⁹ (B3)のリスクには、プロジェクトの遂行におけるメンバーのコミュニケーションギャップ、技術者の意欲の低下など様々な要因によるプロジェクトの失敗や納期遅延が考えられるが、開発事例や組織に依存するため、本ガイドンスでは詳細には触れない。

とに適度な度合いでその規模の目安を評価することはフォーマルメソッドの導入判断において不可欠である。マネジメント分野において、ピーター ドラッカーは「測定できないものは管理できない。」⁴⁰と云い、また、ソフトウェア工学の分野においても、T. デマルコの有名な「計測できないものは制御できない」⁴¹と言っている。適度なレベルで評価をしなければ、フォーマルメソッドの費用対効果を適切に把握することはできない。

以上に挙げた効果とコストの主要要素(B1～B3, C1～C3)について、以下の章で具体的にどの程度の規模であるか、実績や具体例を示すことで、各組織における導入判断の参考情報を提供する。これらの全体像を把握することは、フォーマルメソッドの導入判断で重要である。

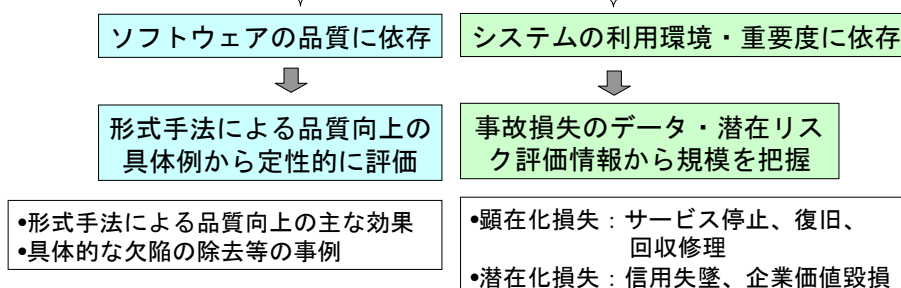
5.3. フォーマルメソッドの効果について

5.3.1. 品質向上

フォーマルメソッドの適用による効果のうち品質向上は、ソフトウェアの出荷・稼働後の不具合の顕在化やIT事故による損害の発生として始めて具現化する。情報セキュリティ経済学においては、損害リスクは、システム障害により発生する直接的な損失よりも、事故により企業の信用が失墜することで企業価値を毀損することによる潜在化損失の割合が大きいことが示されている⁴²。

情報セキュリティや一般の事故等による損害リスクは、大まかな概念としては事故の発生可能性と事故が発生した場合の損害規模(影響の度合い)という2つの要素の積の関係として捉えられる^{43, 44, 45}。ソフトウェアの場合においても、不具合に起因する事故(システム障害)の損害リスク(損害規模の期待値)についても概念的には以下のような関係で表わすことができる⁴⁶。

$$(\text{損害リスク}) = (\text{障害の発生可能性}) \times (\text{障害発生時の損害規模})$$



⁴⁰ Drucker, P., The Practice of Management. New York, NY: Harper Business “you can’t manage what you can’t measure”, 1993

⁴¹ “Controlling Software Projects: Management, Measurement, and Estimates,” Tom Demarco, “You can’t control what you can’t measure.”

⁴² Lawrence A. Gordon, Martin P. Loeb, Managing Cybersecurity Resources A Cost-Benefit Analysis, McGraw-Hill, 2006.

⁴³ 機能安全(IEC61508) part 5, リスク評価

⁴⁴ 島中伸敏, 情報セキュリティのためのリスク分析・評価 第2版, 日科技連出版社

⁴⁵ ISO/IEC Guide51

⁴⁶ ソフトウェアには、ハードウェアと異なり、故障という概念がない。ソフトウェアの不具合は設計や実装段階で確定している障害(Systematic Failure)のみで、不具合が潜在する確率を評価することは困難だが、ここではソフトウェアの不具合に起因する情報システムの障害を対象と考える。

この関係の「(障害の発生可能性)」(以下、「発生可能性」と呼ぶ。)は、ソフトウェアの品質に依存する⁴⁷。一方、「(障害発生時の損害規模)」(以下、「損害規模」と呼ぶ。)は、ソフトウェアの重要性や利用環境によって決まる。利用者が多ければ多いほど、また、システムの用途が重要であればあるほど、システムに障害が発生した場合の損害規模は大きくなる。前述のシステム事故による企業の信用失墜による企業価値の毀損もこれに含まれる。

表 5-3:ソフトウェアの不具合に起因する損害リスクの要素

要素	影響要因	フォーマルメソッドとの関係
システム障害の発生可能性 (発生可能性)	ソフトウェアの品質が高い程 障害の発生頻度は下がる。	フォーマルメソッドの適用によりソフトウェアの品質を高めることで、発生頻度を下げる効果が期待できる。
障害が発生した場合の損害規模 (損害規模)	ソフトウェアの利用状況、規模、重要性によって決まる。	フォーマルメソッドは直接影響しないが、フォーマルメソッドの導入判断で考慮すべき重要な点。

フォーマルメソッドの適用によって品質が向上すれば、「発生可能性」を下げる効果が期待できるが、事故が起きた場合の「損害規模」に直接的には影響を与えない。このようにリスク評価では、障害の発生可能性と損害規模を区別して考えることが一般的である。ソフトウェアによる障害の発生可能性は、ハードウェアの故障確率のような数量的に扱うことは困難である⁴⁸。そのため損害リスク全体を定量的に評価することは困難であるが、「損害規模」の目安を把握することは非常に重要である。障害の影響は最悪の規模を想定して、対策を検討することがリスク管理の基本的な考え方だからである。図 5-1 は、発生可能性と障害発生時の影響度とリスクの関係を示したものである。

⁴⁷ ソフトウェアによる事故の種類によって影響の度合いは異なるが、事故の種類ごとに見れば、大まかな概念として、障害の発生可能性と障害発生時の損害規模の積として捉えることができる。

⁴⁸ ソフトウェアによる障害の発生可能性を数量化することが困難な理由は以下のように考えることができる。障害の発生可能性は、利用環境における実行パスの実行頻度と実行パス上に不具合が潜在する可能性の重み付き総和として大まかに捉えられる。不具合の混入は、人のミス等によるもので、実行パスの実行頻度とは無関係であるため、それらの重み付き総和は、数量的な尺度とはならない。

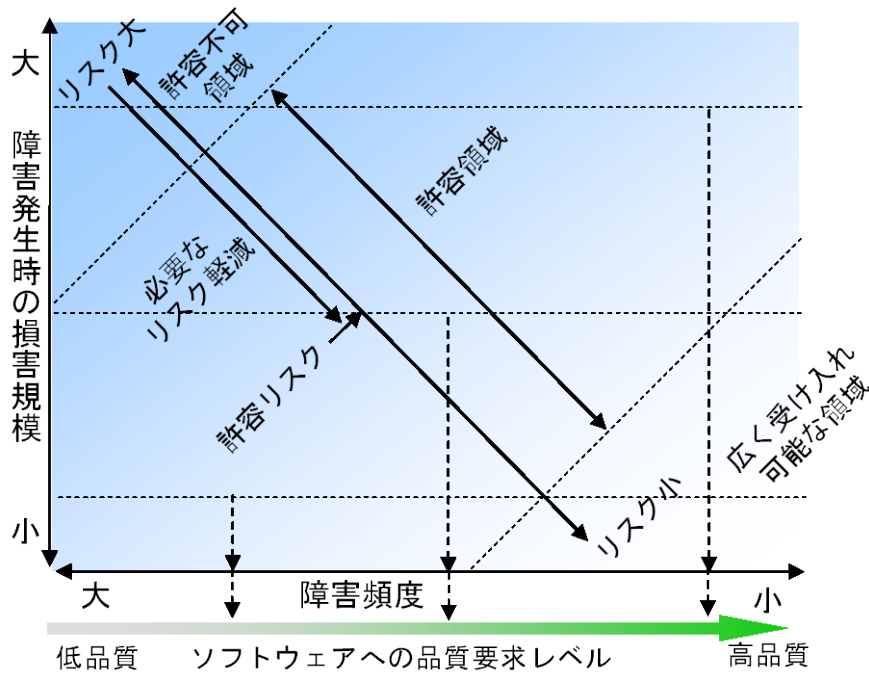


図 5-1: 発生可能性と障害発生時の影響度とリスクの関係

図 5-1 において、縦軸の「損害規模」と横軸の「発生可能性」の積によってリスクがきまるため、グラフの対角線上の左上ほどリスクは高く、右下ほどリスクは低い。情報システムは、あらかじめ求められるニーズや用途などのシステムの重要性に応じて「損害規模」が決まるため、リスクを一定レベルに抑えるためには、品質を向上させることにより、発生可能性を減らすことで、リスクを一定レベルに抑える必要がある。

フォーマルメソッドは、ソフトウェア全体のどの部分に適用し、どの程度検証するかにより、期待できるソフトウェアの品質向上やコストが大きく異なる。フォーマルメソッドによりソフトウェアの品質を高める場合、高い品質になるに従い、より多くのコストが発生する傾向にある。損害規模は、システムの用途や利用環境により、きわめて大きな幅を持つため、フォーマルメソッドの導入検討においては、損害規模を考慮することは不可欠である。ソフトウェアの品質を高めるためにいくらコストをかけても、不具合による障害の損害規模が小さければ、採算性がとれないという結果になる。一方、セーフティクリティカル・システムや重要インフラシステムなど影響度の大きなシステムには、それに見合った大きな投資コストをかけても、ソフトウェア品質の向上が求められる。ソフトウェアに求められる品質は、ソフトウェアの重要度(抱えるリスク)に応じて検討すべきであり、別々に議論することはできない。

一定の規模の複雑さをもつソフトウェアでは、どんなにクリティカルなものでも、完璧なものは期待できないことは世の中の事故事例が物語っている。完璧なソフトウェアを追求すれば、急激なコストの上昇を伴うことを覚悟しなければならない。機能安全の考え方では、許容可能リスク、許容不可

リスクの境界を定め、許容範囲内にリスクを抑えるための対策レベルを検討する⁴⁹。また、マーケットで受け入れられているソフトウェアの品質が必ずしも高いとは限らない。パソコン等一般のコンシューマーに利用されるソフトウェアでは、新しい機能をマーケットに早期に投入することが成功の要因となる場合が多い^{50,51}。現実的に考えれば、システムが利用される環境からその影響度を考慮し、コストとのバランスからフォーマルメソッドの導入を検討すべきである。日本学術会議の安全に関する緊急特別委員会報告「安全学の構築に向けて」⁵²においても、「絶対安全」から「リスクを基準とする安全の評価」への意識の転換が必要であることが述べられていることも、同様の考え方である。

5.3.1.1. 障害発生時の損害規模について

「損害規模」に関しては、システム障害によるサービスの停止、復旧コスト、製品回収修理コストなどの顕在化損失と、企業の信用やブランド価値に与える影響からくる潜在化損失がある⁵³。

$$(\text{損害規模}) = (\text{顕在化損失}) + (\text{潜在化損失})$$

顕在化損失については、前述の通り情報家電の不具合修理のための回収コストで 100 億円を超えるものがある。セーフティクリティカル・システムの例を挙げると、顕在化損失だけで、衛星1機の打ち上げ失敗の損害は 200 億円、航空機では数百億円以上、原子炉やITSやスマート・グリッドの構築等大規模なインフラシステムにおける致命的損失となれば兆円レベルにも及ぶことが指摘されている⁵⁴。顕在化損失の具体的な規模を示すものとして、経済産業省が実施する組込みソフトウェアに関する事故被害の額を示したものがある(図 5-2)。

⁴⁹ IEC61508 part 5

⁵⁰ Michael Cusmano, Critical Decisions in Software Development, 2009.

⁵¹ Michael. Cusumano et al., "Software Development Worldwide:The State of the Practice," IEEE Software, Dec. 2003, pp. 28-34.

⁵² 日本学術会議

⁵³ 経済産業省 リスク定量化ワークショップ, 2007年, "IT事故と情報セキュリティ対策が企業価値に与える影響分析", 石黒正揮(三菱総合研究所), 松浦幹太(東京大学), 田中秀幸(東京大学)

⁵⁴ 藤枝純教, 経営者はアーキテクチャと形式手法を忘れてはいけない, IPA SEC Journal

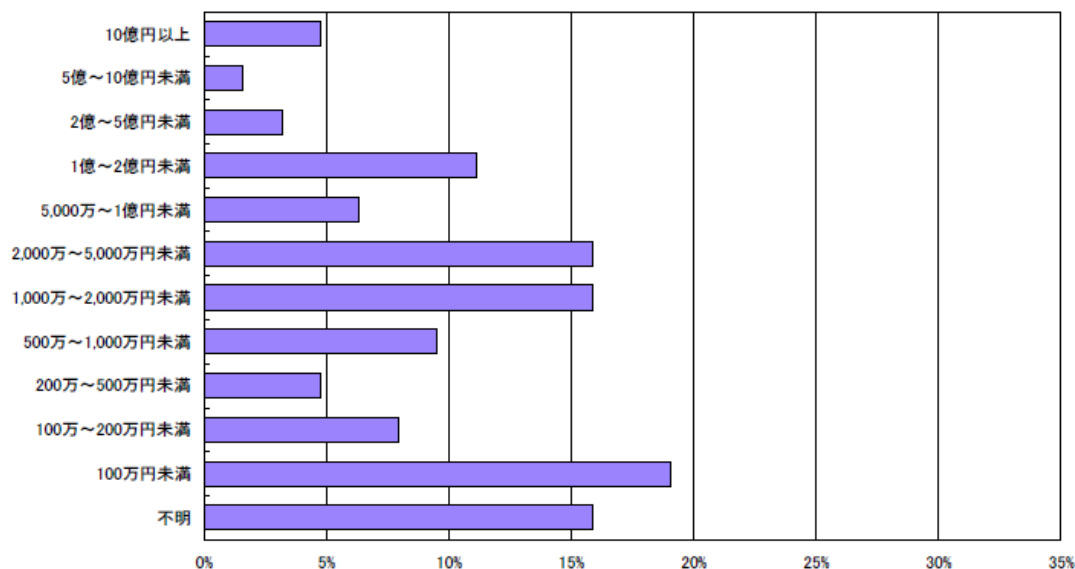


図 5-2: 不具合が発生したことによる対策費の総合計(2008 会計年度)⁵⁵

情報処理推進機構の報告書⁵⁶には、ソフトウェアに起因するシステム障害事例として、決済に関わる事故、情報漏えい、重要インフラに関わる障害、メール等のWEBサービスの障害などにおける顕在化被害の事例が多数挙げられており、その深刻さを認識する上で参考となる。

一方、潜在化損失については、まだ世の中では、その規模について余り認識されていない。その規模の具体例を知ることは非常に重要である。情報システムのセキュリティに関わる損害事故の規模については、情報セキュリティ経済学の分野である程度の評価を行う手法が確立されてきている。この分野では、事故の種別(不正アクセス、個人情報漏洩等)、企業規模等に応じて事故事例を分類し、それぞれの事例集合について、セキュリティ事故を要因とする企業価値の毀損額を評価し、セキュリティ事故によって企業価値が統計的に有意に毀損している場合の損害額を評価している。これらの結果に基づき、経済産業省の報告書では、情報システムの不具合等による情報セキュリティ事故(不正アクセス、機密情報漏洩等)に係わる日本の上場企業が抱える潜在リスクは 29 兆円と見積もっている^{57, 58}。この結果に基づけば、上場企業について情報セキュリティ事故だけでも、無形資産⁵⁹を含む企業価値の 5.4%のリスクを抱えていることになる。これは顕在化損失を上回る規模であり、上場企業1社当たり平均で 79 億円の潜在リスクを抱えているという計算になる⁶⁰。

顕在化損失と潜在化損失の規模の関係を把握するために、簡単な試算をすると以下のように

⁵⁵ 2010 年版 組込みソフトウェア産業実態調査:事業責任者向け調査、経済産業省

⁵⁶ IPA, 高信頼化のための開発手法ガイドブック-予防と検証の事例を中心に-, 2010

⁵⁷ 経済産業省, 「グローバル情報セキュリティ戦略」, 2007

⁵⁸ この推定値は、情報セキュリティ経済学の分野において、企業の市場価値(無形資産を含む)に関する統計的に有意な結果(信頼度 95%)にもとづいたものである。

⁵⁹ 企業の市場価値から有形の純資産を除いたもの。

⁶⁰ 定量的な数値に関しては、環境や組織によって異なるが、大まかな規模感を知ることは、その重要性を認識する上で大切である。

なる。日本ネットワークセキュリティ協会(JNSA)の報告書⁶¹によると、情報システムに関わる個人情報漏洩事故(人的事故 27%を含む)の総被害額は、利益 10 億円、売上げ 100 億円のインターネットショップ部門の場合、3.8 億円と試算されている。一方、同規模のサービス・情報通信企業の情報セキュリティ事故による損害は、11.7 億円と試算される。両者は、業種の違いなどはあるが、比較可能なものとして参考にすると、顕在化損失が 3.8 億円に対して、類似の事故で潜在化損失と顕在化損失を合計した企業価値全体の損失は、11.7 億円であるため、潜在化損失は、顕在化損失の大きめに2倍の規模になることが分かる。

情報セキュリティ以外のIT事故損害も考慮すれば、潜在リスクはさらに大きくなる。無形資産⁶²を含む企業価値の毀損は、企業に対する信用失墜により将来に渡って企業のビジネスに大きな影響を与えることによるものである。その影響は、通常、リコールや製品回収コストなど顕在化損失よりも、大きいことを示している。以上のような状況を考慮すれば、IT事故による企業の損失は、一般に認識されている顕在化損失だけでは、不十分であり、上記のような損害規模の認識は極めて重要であるといえる。

障害発生時の影響度は、利用分野ごとに異なり一般化できないため、分野ごとにユーザ、事業者、発注者、ベンダーなどのステークホルダーが集まりリスクの洗い出しを行う必要がある。その際に、従来の調査研究結果から以下のような項目を参考に、具体的な障害を複数想定し、それぞれについて評価することにより幅広い観点からその影響度を把握することが重要である。

表 5-4: 障害発生時の影響度を把握するための主な検討項目(参考例)

(文献^{63, 64, 65},を参考に作成)

区分	項目	内容	
人命への影響	死亡	死亡人数	
	負傷	負傷の程度×人数	
経済的な影響	顕在化損失	顧客の被害	被害内容(サービス停止、製品欠陥、料金誤請求、個人情報漏洩等)の重要度と件数の組合せ
		顧客対応コスト	問合せ・クレームの対応時間
		逸失利益	サービス停止による収益機会損失
		業務影響コスト	社内業務の低下・停止のコスト
		復旧コスト	復旧時間の人件費
		損害賠償・補償	損害賠償・補償費用

⁶¹ JNSA, 「2003年度情報セキュリティインシデントに関する調査報告書」

⁶² 無形資産: 企業における特許等の知的財産、従業員の持つ技術や能力、企業文化や経営管理プロセスなど物的な実態を伴わない資産。

⁶³ 経済産業省, 「企業における情報セキュリティガバナンスのあり方に関する研究会 リスク定量化に関する検討資料」

⁶⁴ IPA, 「国内・海外におけるコンピュータウイルス被害状況調査 被害額推進報告書」(2004 年4月)

⁶⁵ JNSA, 「2003 年度情報セキュリティインシデントに関する調査報告書」

	潜在化損失	企業ブランド 価値毀損 (信用失墜)	信用失墜による将来の製品・サービス売 上げ減少の正味現在価値(NPV : Net Present Value)
--	-------	--------------------------	---

障害発生時の損害規模について具体的に検討する場合、以下のような項目について、情報システムのドメインに関する知見をもっとも持っているユーザ(事業者)が検討しなければならない。

5.3.1.2. 障害の発生可能性について

損害リスクのもう一つの要素である「発生可能性」は、フォーマルメソッド等を適用してソフトウェアの品質を向上させることにより減少させることができる。フォーマルメソッドの適用により、ソフトウェアの品質向上にもたらす効果について、従来のソフトウェア・テストでは得られない効果として主に以下のようなものがある。

表 5-5: フォーマルメソッドによる品質の向上に関する主な効果⁶⁶

分類		フォーマルメソッドの効果	従来テストとの比較
性質保証		設計仕様が要求仕様を満たすこと ⁶⁷ 、与えた性質を設計仕様が満たすことを、形式検証が成功した場合、保証することができる。	保証したい性質について、いくらテストケースを増やしても、保証できない(ことが多い)。
不具合の検出	反例発見	与えた性質を満たさない反例を示し、その原因箇所の特を支援する情報を示す。(モデル検査)	反例を検出するとは限らない。
	デッドロックの検出	デッドロックが存在する場合検出できる。(モデル検査)	デッドロックを検出することは困難。
	仕様の矛盾検出	矛盾がある場合に検出できる。	人手によるため、検出できるという保証はない。
	仕様の定義漏れ検出	仕様の定義漏れがある場合検出できる。	人手によるため、検出できるという保証はない。
	デッドコードの検出	発生しない場合(デッドコード)が存在する場合検出でき	人手によるため、検出できるという保証はない。

⁶⁶ EASIS, Marko Auerswald, Guidelines for the Development of Dependable Integrated Safety Systems, NASA, FORMAL METHODS SPECIFICATION AND VERIFICATION VOLUME I, VOLUME II 等の文献をもとに整理

⁶⁷ 6章に示すとおり、Verification(検証)とValidation(妥当性確認)の区別が重要である。検証は、仕様どおりに正しく設計、実現されていることを検査し、妥当性確認は、仕様自体が正しいことを確認するものである。形式手法で性質を保証できるのは、Verificationの意味である。

		る。(モデル検査)	
曖昧性の除去		曖昧な仕様による誤解や実装の誤りを回避できる。 暗黙知を明確化する。	曖昧性から、誤った実装につながる場合がある。

これらの効果について、具体的なイメージをつかむために、付録のケーススタディについて具体的な例を以下に示す。

表 5-6: 効果の具体例

効果の分類 (一部)	具体例(ケーススタディの例)	予想される影響等
性質保証	ブルーレイディスクの操作制御に関するミドルウェアの設計に対して、「要求されたディスク操作(API)は、必ずデバイスで実行され、待機状態に戻る。」という性質を記述として与え、形式検証が成功したため、その性質が必ず満たされることを保証することができる。	ブルーレイディスク以外の制御系(例:電力、航空機、鉄道)で、ユーザが要求した操作が実行されない、それが安全に関わる処理であれば、大きな事故につながる可能性がある。
反例の発見	「全ての操作に関して要求された API は、必ずデバイスで実行される。」という性質には、再生操作 API に反例があることが発見された。	要求した操作が実行されない場合を発見して、改修することにより、そのような状況が発生することを減らすことができる。
デッドロック検出	操作APIの要求と、ドライブからの割り込みが同時に処理されるとき、制御状態を保持する共有変数に関してデッドロックが発生し、ミドルウェア全体が停止するという致命的な傷害が発生する。	デッドロックは、システム全体の処理が進行しなくなることを意味しており、重要インフラ系で同様なことが起きれば事故につながる可能性がある。
矛盾の検出	ミドルウェアを構成する2つのモジュールレイヤーがそれぞれ保持する動作に関する状態は整合性が取れているという性質は、満たされないことが示された。(当初から予想されていたが、厳密にそれが示された。)	あらかじめ想定された状況の発生が確認されたものであり、そのような状況でも不具合が発生しないことを検証する必要がある。
曖昧性の除去	自然言語による仕様書から、形式仕様記述を行う際に、59件の曖昧な記述を見つけた。	設計仕様の曖昧性により、設計仕様からコーディングの過程で、意図しない処理が行われる可能性がある。

これらの例は、いずれも従来のテストだけでは検出することが困難な不具合であり、フォーマルメソッドを適用することにより、ソフトウェアの品質向上の効果を特徴的に示している。ただし、フォーマルメソッドは、従来のテストを丸ごと置き換えるものではなく、従来のテストと補完関係にあり、双方の有効な部分を組み合わせることで、全体の品質向上を実現することが経済的にも効果的であることに留意しなければならない⁶⁸。

機能安全の分野では、ソフトウェアの設計や実装の不具合の存在確率については定量的な評価が困難であるため、開発の手順と適用する手法によるプロセスを認証することで安全性を確保するという考え方が一般的になっている。フォーマルメソッドの適用が、システム障害の発生確率の低減にどの程度の効果があるかについては、正確には定量化できないが、障害が発生した場合の損害額の大まかな規模をつかむことは重要である。機能安全標準 IEC61508 の考え方を参考にすれば、リスクの大きさに応じて、ソフトウェアのユーザ(事業者)が、ソフトウェアの品質レベルを設定し、その結果に対して責任を持つことが求められる。

フォーマルメソッドによる検証等による具体的な効果を整理したものが下図である。これらなども参考にその効果を現場で評価するとよい。

⁶⁸ Jonathan Bowen, Ten Commandments of Formal Methods, Ten Years Later, 2006

効果の分類	開発現場における現状の問題点	形式手法により期待される効果	具体例
不具合対策			
不具合の発見	<ul style="list-style-type: none"> ・並行プロセスのタイミングに依存する不具合は再現性が低く、発見が困難。 ・検査したい性質について、系統的に場合分けを網羅するテストケースの生成が困難。 ・そもそも、タイミングに係わる不具合の可能性さえも気が付かない場合が多い。 ・バグを減らすために、並行プロセスや条件分岐を減らすなど消極的なプログラミングを強いられる。 ・テストツールの自主開発は、コストがかさみ、また、転用が難しい。 	<ul style="list-style-type: none"> ・網羅的に検査するために、検査で不具合が見つからなければ、正しさが保証される。 ・モデル検査では、系統的、網羅的に自動検査が可能であるため、効率的。 ・不具合が存在する場合、不具合に至る実行トレースが提示されるため、原因を究明できる。 	<ul style="list-style-type: none"> ・ブルーレイディスクにおいて、操作要求とデバイス割込みが並行して発生する場合に、デッドロックが発生することがある。 ・再生実行が、処理されず停止に戻る場合がある。(通常の操作では発生しないことを確認)
不具合の原因特定	<ul style="list-style-type: none"> ・不具合の存在が分かっていても、再現性が低い場合、毎回挙動が異なるため、原因の特定が困難。 ・めったに再現しない不具合は、コストの制約から修正できないまま出荷せざるを得ない場合もある。 ・マイクロチェック数が30を越えると、設計した本人でも修正が困難となる。 	<ul style="list-style-type: none"> ・再現性が低い不具合であっても、網羅的な検査を行うため、不具合の原因特定が可能。 ・処理の分岐が複雑であっても、系統的に自動検査可能である。 	<ul style="list-style-type: none"> ・ブルーレイディスクの動作におけるデッドロックに至るトレースを追跡することで、原因が特定できた。
不具合の修正の影響分析	バグの修正の影響範囲は人手によるコードレビューに頼っている。	一度、検証性質を記述すれば、モデルの修正後、繰返し検査に利用することができる。	
検証			
与えられた性質を満たすことを保証	<ul style="list-style-type: none"> ・並行動作のタイミングに依存する不具合は、テスト回数をいくら増やしても、完全な保証が得られない。 	<ul style="list-style-type: none"> ・網羅的な検査を行うため、不具合が検出されなければ、与えられた性質に関して正しさが保証される。 	<ul style="list-style-type: none"> ・ブルーレイディスクの操作制御モデルに、デッドロックが発生しないことを保証できる。
検証結果の評価に対する科学的根拠	<ul style="list-style-type: none"> ・テストツールを開発しても、網羅性の保証は困難で、テスト結果に対する評価は、ベテランの感に頼ることが多く、科学的な根拠が示せない。 ・テストカバレッジを計測する方法が普及しておらず、パスを通った結果が正しいか人手に依存する。 	<ul style="list-style-type: none"> ・検査結果に網羅性が保証される。 ・形式仕様から、テストカバレッジを自動的に計測することができる(VDM)。 	<ul style="list-style-type: none"> ・ブルーレイディスクの制御モデルに関して、デッドロックが発生しないという性質を科学的に保証できる。 ・ファイルシステムにおいて、ファイルオープン操作のテストカバレッジが計測できる。(VDM)
従来テストでは扱えない性質(安全性、到達性等)の検査	従来テストでは、状態の網羅的な検査が必要な安全性や、実行の無限系列に関する到達性に関する性質が扱えない。	従来テストでは、扱いにくい性質を時相論理式として表現し、検証が可能。	「どのようなリクエストに対しても、いずれは待機状態に戻る」という性質は、従来のテストでは扱いが困難。
妥当性確認			
仕様の曖昧性排除	<ul style="list-style-type: none"> ・仕様書の曖昧性により、設計者と開発者の異なる思い込みにより、バグが入り込む。 ・要求仕様の曖昧性により、発注者の意図と設計者の理解にズレが生じることがある。 	<ul style="list-style-type: none"> ・性質に関する記述を、条件の論理的な組み合わせ(かつ、または、否定などの構造化)で表現することにより、条件の範囲や掛かり方が明確化される。 ・条件が満たされる時間の範囲や、検証対象とする実行経路の範囲や、係り受けが明確化され、誤解が回避される。 	<ul style="list-style-type: none"> 入室管理システムで「閲覧室が空室のとき以外、常に案内は変更されない」という表現は、「閲覧室が空室の時以外、決して案内は変更されない」の方が意味が明確。 「常に性質pを満たさない」という表現は、『「いずれは、性質pを満たす。」の否定』、または、『「常に性質pを満たす」の否定』のいずれであるか明確化が必要。

図 5-3: 開発現場が抱える問題点とフォーマルメソッドの適用で期待される効果の例

5.3.2. コスト削減

フォーマルメソッドの適用によるコスト削減に関する効果については主に以下のようなものが挙げられる:

- 設計、要求分析工程で、実装する前に検証やシミュレーションなどの解析を実行することにより仕様の不具合を検出でき、手戻りコストを削減する。
- 設計、要求を、曖昧性のない厳密な言語で記述することにより、コーディング等の後工程における誤解などによる不具合の混入を防ぐ。

形式仕様記述言語を用いて要求分析、設計を行うことにより、コーディングを行う前に上流工程で、それらの仕様の検証、シミュレーションが可能になり、早期の不具合除去に有用である。図 5-4 は、ソフトウェアプロジェクトに関するNISTの調査に基づきCMU/SEIが作成したソフトウェアの除去コスト等を定量的に評価したものである^{69, 70, 71}。

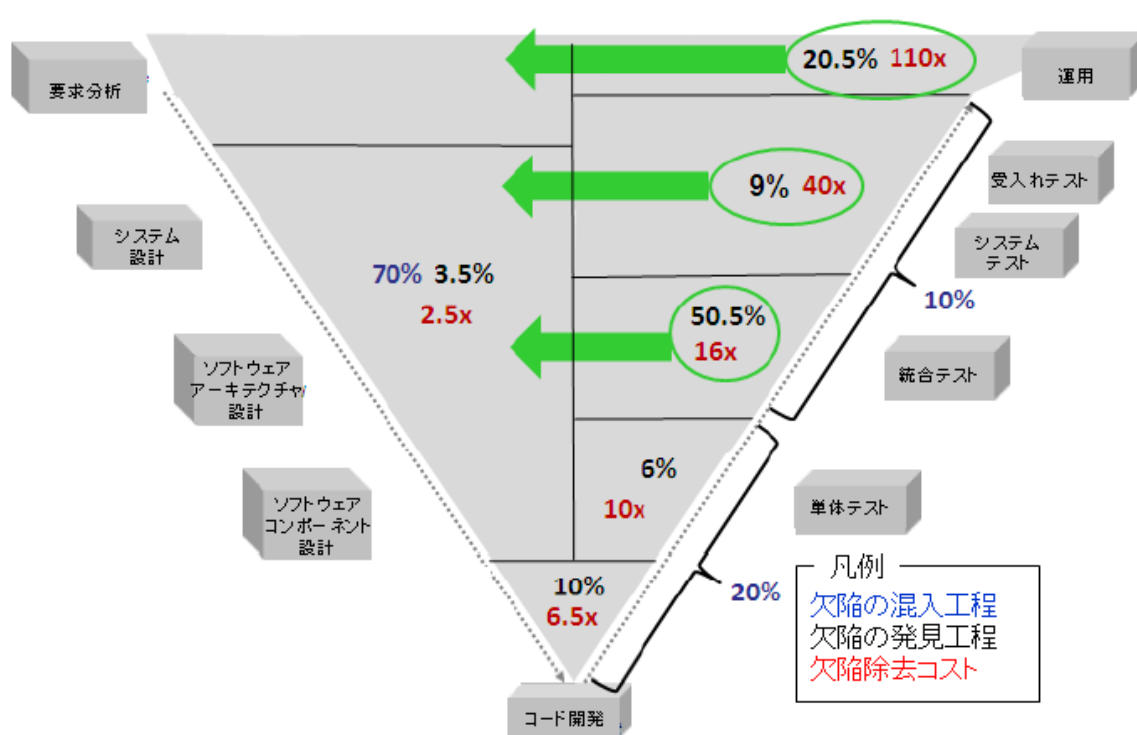


図 5-4: 欠陥の混入工程、発見工程、除去コスト

(Peter H. Feiler et al, System Architecture Virtual Integration: An Industrial Case Study,

⁶⁹ CAD/CAM/CAE/PDM(of computer-aided design/computer-aided manufacturing/computer-aided engineering/product data management software)の179ユーザーと10のソフトウェアベンダーに対するインタビュー調査結果。

⁷⁰ Peter H. Feiler et al, System Architecture Virtual Integration: An Industrial Case Study, November 2009, TECHNICAL REPORT CMU/SEI-2009-TR-017

⁷¹ NIST, The Economic Impacts of Inadequate Infrastructure for Software Testing, 2002.

これによれば、欠陥の 70%は、設計工程までに混入し、そのうち設計工程までに検出される欠陥は、3.5%に過ぎず、全体の 50.5%は、統合テスト工程で発見されることが示されている。欠陥を発見する工程が下流であればあるほど手戻りコストが増大し、統合テストでは、16 倍、システムテストでは、40 倍にかさむことが示されている。このようなことから、フォーマルメソッドを用い開発プロセスの要求と設計の上流に重点を置くことで、開発コストを下げる効果が期待できる。また、航空機、鉄道分野では、形式仕様記述言語を用いて設計を行うことにより、設計記述からプログラムソースコードを自動生成し、従来テストを大幅に減らす事例が多く、トータルでのコスト削減に効果が得られている例もある^{72,73}。例えば、航空機のフライト制御ソフトウェアの場合、設計にフォーマルメソッドを用い、自動コード生成を行うことにより、コーディング、レビュー、テストのコストを 70～90%削減できるという事例が示されている^{84,74}。

フォーマルメソッドの導入は、必然的に上流工程の要求分析、設計に重点を置くことになる。さらに、実装を記述することなく、要求仕様や設計仕様に対して検証や解析を行うことが可能になり、上流工程の成果物の品質を向上させることができる。McDonaldの文献⁷⁵では、上流工程を重点化する欠陥予防開発プロセス(Defect Prevention focused Process)は、下流工程のテストに時間をかけるテスト中心プロセス(Test-Centric Process)よりも生産性が2倍に達することを示している。

本書付録のケーススタディにおいては、フォーマルメソッドを適用しない場合のテスト工数に対して、モデル検査の適用により、制御アルゴリズムに関する不具合が設計時点で除去できることから計算して、テスト工程が、2ヶ月から1ヶ月に短縮できることが示された。この効果は、テスト工程における不具合の発見により、手戻りが削減されることを考慮すればさらに大きな効果になると言える。一般的なソフトウェア開発において、テスト工程が全体の50%以上を占める⁷⁶という実態からしても、テスト工程の短縮効果は大きいと言える。

一方で、フォーマルメソッドによる設計および検証は、プログラムを抽象化したレベルで実施し、検証された設計に基づき、人手によりコーディングを行うケースもある。このような場合、テスト工数は、幾分削減することは可能かもしれないが、トータルの工数としてはフォーマルメソッドを導入しない場合に比べて増大する場合が多い。しかし、このような場合であっても、設計が検証されていることによりソフトウェアの品質向上の効果が設計コストの増加を上回るならば採算性があると判断される。

従来の下流工程のテスト中心開発プロセスから、欠陥予防開発プロセスに移行するには、未経験の新しいプロセスを取り入れる際の不安や回避行動があるため、採用が進まないのが現状であ

⁷² Model-Based Development for Critical Embedded Software in Aerospace & Defense, Esterel Technologies.

⁷³ IPA, 形式手法適用調査, 2010

⁷⁴ ROI Analysis, Esterel Technologies, 2009

⁷⁵ The Practical Guide to Defect Prevention, Marc McDonald, Microsoft Corporation, 2008

⁷⁶ ROBERT M. HIERONS, et. al., Using Formal Specifications to Support Testing, ACM Computing Surveys, Vol. 41, No. 2, pp. 9-76, 200

るが、上流工程に重点をおいた開発プロセスの重要性が理解されれば、フォーマルメソッドも自然に受け入れられるものである。

5.3.3. リスク低減

リスク低減の効果については、見落とされがちであるが、重要な要素である。管理者の視点からは、プロジェクトが予算通りに実施されること、工数の大幅な変動が発生せず、納期が守られることは重要である⁷⁷。また、開発プロセス全体のうち、特定の工程に、負荷が集中するボトルネックが存在する場合、全体の生産性に悪影響を与えると共に、人員の稼働状況に非効率性が発生するなどの問題点が指摘されている⁷⁸。

フォーマルメソッドにより開発の上流工程に重点を置いた開発プロセスのフロントローディングを実現することで、従来のテスト工程のピーク負荷を緩和し、プロセス全体のピーク負荷を抑えるとともに、手戻り発生等による工数の大幅な変動リスクを抑えることにも役立つ。図 5-5 は、EU のフォーマルメソッド実践プロジェクト EASIS の資料に基づき、フォーマルメソッドを用いた開発のフロントローディングにより、ピーク負荷の工程は、従来のテスト工程から、上流の設計工程にシフトし、プロセス全体のピーク負荷が低減されること様子を示している。フォーマルメソッドを用いたこのようなフロントローディングは、航空機分野や鉄道分野など特定の分野においてはある程度実現されつつある。

このように、開発フロントローディングによる生産性の向上、納期遅延などのリスクの回避などの効果をもたらすものであるが、フォーマルメソッドの導入は、上流工程で厳密に設計し、検証を行うため、必然的に開発フロントローディングを実現することにつながるといえる。

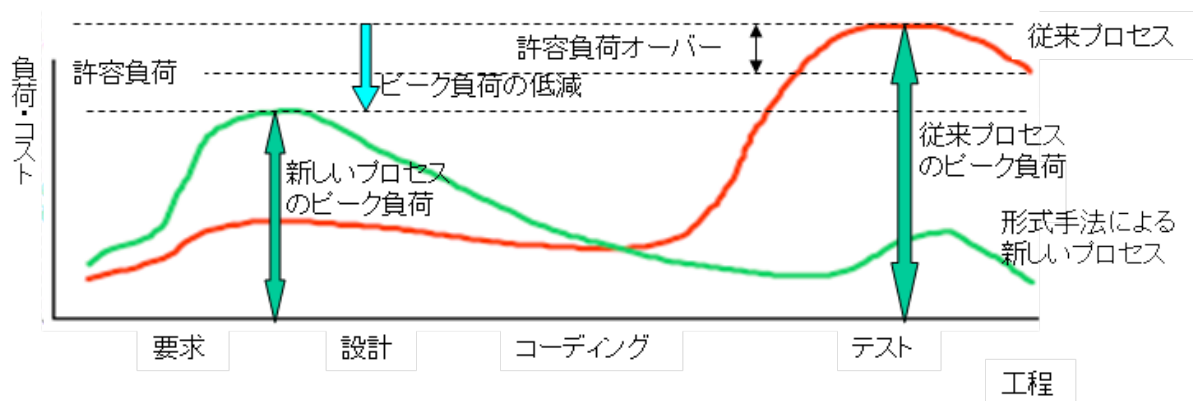


図 5-5: 開発プロセスのフロントローディングと許容負荷
(文献 66 を元に作成)

多くのフォーマルメソッドの文献に書かれているように、フォーマルメソッドを開発の上流工程に

⁷⁷ これら以外にも、Capers Johns は、ソフトウェア開発だけでなく保守も対象にしたリスクに関して洗い出し、顧客の要求仕様変更など60のリスクをあげている。これらについてもリスクを低減する上で参考になる。

⁷⁸ 制約理論(TOC)では、プロセス全体の生産性は、ボトルネックの改善によってのみ実現されるとされる。

適用することで工数の削減効果とともに、開発工数の変動リスクを低減する効果が期待できる⁷⁹。

5.4. フォーマルメソッドの投資コストについて

フォーマルメソッド導入のための投入コストには、主要要素として(1)学習コスト、(2)設計コスト、(3)ツール導入コストがある。以下では、それぞれのコストについて具体的な規模の例示と考え方について示す。

5.4.1. 学習コスト

手法の学習コストは、手法の種類と学習の達成レベルによって異なる。手法の種類による学習コストの違いとして、日本語の学習用入門書が比較的充実している手法は習得が容易になるというメリットがある。例えば、**SPIN**、**Bメソッド**、**NuSMV**、**VDM++**などの文献は比較的多い⁸⁰。これらの手法の基礎的な技術の習得は、入門書を数冊読むための時間から学習コストを見積もることができる。手法の習得は、必ずしも最初に上級レベルまで一気に習得する必要は無い。初級レベルに到達した段階で、試行的にフォーマルメソッドの適用を始められるが、実践を繰り返しながら、ノウハウを習得するとともに、上級参考書によりスキル向上を図らなければ、フォーマルメソッド適用により効果が得られるレベルに到達しない。また、成功事例とされる多くのプロジェクトは、フォーマルメソッドの適用時にフォーマルメソッドの専門家に定期的にアドバイスを受けられる環境にあることを挙げている^{81,82}。海外にはフォーマルメソッドの導入コンサルティングを行う企業も出てきている^{83,84}。成功している適用事例は、導入初期には、このようなコンサルティングサービスを利用しているものが多い。それは、高度で特殊なスキルをすべてマスターしてから適用するのでは初期コストが大きすぎるため、早い段階で実践を開始し、問題に直面した際に、外部専門家から指導を受けることで必要なスキルを効率的に習得することを目指す。

これらの手法の入門レベルの習得は、フォーマルメソッド特有の概念を理解する点で難しさはあるが、**Java**言語などプログラミング言語と比較して、特別多くの時間を要するという訳ではない。

本プロジェクトケーススタディでは、**SPIN**を用いた制御設計の検証に関連して技術習得のために以下のような工数が発生している。

⁷⁹ Abernethy, Technology Transfer Issues for Formal Methods of Software Specification, NASA, 2000.

⁸⁰ 学習者向け参考書は第 17.1.3 章等参照。

⁸¹ Jonathan Bowen, Ten Commandments of Formal Methods ... Ten Years Later, 2006

⁸² Jonathan Bowen, Ten commandments revisited a ten year perspective on the industrial application of formal methods, 2005.

⁸³ ClearSy, <http://www.clearsy.com/index-en.php>

⁸⁴ Esterel Technologies, <http://www.esterel-technologies.com/services/fast-ramp-up-services/methodology-preparation>

表 5-7: SPIN の技術習得コストの例

		学習時間 (h)	実践適用 時間(h)
被験者1	入門レベル	34	176
被験者2	入門レベル	63	370
被験者3	上級レベル	562	143

フォーマルメソッドの習得コストを、単一のプロジェクトで回収しようとするのは、合理的な考え方とは言えない。それは、プログラミング言語の習得を単一プロジェクトのためだけに行うわけではないことと同じである。フォーマルメソッドの初期の学習コストを考えると、極めて大規模なプロジェクトあるいはきわめてクリティカルなシステムでなければ、単一のプロジェクトで採算は取ることは難しい。プログラミング言語と比較して、フォーマルメソッドにおける採算性評価で難しい点は、フォーマルメソッドの効果に関して5.3章で示したように、損害リスクの低減がドメインや対象システム、利用環境に依存して評価が難しいためである。したがって、採算性の評価結果は、損害リスクの低減効果がどれだけ適切に行われるかによって大きく異なる。その点からも、ドメイン知識を持つ導入者が判断することが重要になる。

5.4.2. 設計コスト

フォーマルメソッドを用いた設計は、その詳細度により効果や影響が異なる。大まかに以下の2つのレベルに分けられる。

表 5-8: フォーマルメソッドを用いた設計レベル

レベル	内容	影響
基本設計レベル	アルゴリズムの一部などを抽象化した設計レベル ⁸⁵ 。	形式仕様記述言語で記述した基本設計をもとに、マニュアルによる詳細設計、コーディングを行う場合、基本設計レベルで形式検証を行った結果は、そのままでは保証されない。
詳細設計レベル	ソースコードが生成できる程度に詳細なレベル。	形式検証を行った詳細設計から、コード生成を行う場合、コード生成系が Qualification 済であれば、生成されたソースコードの信頼性は高い。

⁸⁵ モデル検査等のフォーマルメソッドの応用では、基礎設計レベルでは、アルゴリズムやそれを抽象化したものを対象とすることが多く、アーキテクチャ設計への応用はこれまであまり実績がないため、ここではアルゴリズムを中心に考える。

設計コストの増加は、これらの設計レベルおよび適用する手法により異なる。Bメソッド、VDMの場合、詳細設計レベルまで形式仕様を記述する場合が普通である。その場合、自動コード生成系を使うことが可能であり、生産性の向上が期待できる。

表 5-9 は、付録のケーススタディ(ブルーレイディスク、入退室管理システム)において、制御に係わる設計仕様について SPIN を用いた形式モデリングと検証を行った際の追加工数である。フォーマルメソッドでは、設計段階で、検証まで行えるためそれらの工数も含めたコストを考える。

表 5-9: ケーススタディにおけるフォーマルメソッドの設計付加コスト(時間)

	ブルーレイ ディスク	入退室管理 システム
設計書の理解	196	150
モデリング	40	70
モデルの形式記述	40	80
検証性質記述	3	20
不具合特定等	40	50

SPIN などモデル検査系では、状態爆発の問題があるため、抽象レベルの設計を対象とする場合が多い。この場合、形式検証済みの設計仕様に対して、人手により詳細設計やコーディングを行うことになり、人手によるプロセスから不具合が混入する可能性がある。

一方、設計と検証を比較的厳格に行う手法として代表的なBメソッドに関してフランスの鉄道分野で実践したプロジェクト⁸⁶における設計コスト、工程全体の比率を示したものとして以下のものが参考になる。

表 5-10: 鉄道システムに対する B メソッドの適用プロセスの工数比率

工程		工数比率	
プロジェクト管理・その他		13%	
抽象モデリング	対象分析等	55%	18%
	インスペクション		5%
	証明		16%
	その他		6%
具体モデリング	証明	24%	11%
	その他		13%
最終管理(文書、確認等)		8%	

⁸⁶ Using B as a High Level Programming Language in an Industrial Project: Roissy VAL, ClearSy and Siemens Transportation Systems

フォーマルメソッドの産業応用に関する調査によると、適用事例ごとに、適用した対象の範囲、適用工程、適用の深さ(適用した設計の詳細度)はまちまちであり、決して対象システムの全てのコンポーネント、全ての工程に一律に適用しているわけではない。対象ごとに、フォーマルメソッドの適性、限界などを考慮し、目的やリスクに応じて、適用範囲や適用の詳細度を選択することが成功の重要な要素とされている。

フォーマルメソッドの導入には、技術の習得に一定の初期コストが必要になるため、単一のプロジェクトで採算性をとるのではなく、複数のプロジェクトで利用することを想定した組織としての採算性を考えることが合理的と言える。実際、フォーマルメソッドの適用コストは、初回から、2回目で劇的に下がることがいくつも報告されている⁸⁷。

5.4.3. ツール導入コスト

フォーマルメソッドは、無償のツールが比較的多い。実践で比較的多く利用される B メソッド、VDM++に対応したツールやモデル検査系の SPIN, NuSMV は無償で提供されている。

一方、モデル検査系を含む SCADE や SAL など有償のツールも存在する。有償のツールの中には、エディタ、検証ツール、要求トレーサビリティ支援機能など開発に必要な一連の機能を統合した開発環境として提供するものもある。これらのツールの中には、さまざまな機能オプションやサービスと組合せとして価格がきまるものがあり、個別に費用の見積もりが必要になる場合がある。いずれにしても、ツール導入コストは、フォーマルメソッド導入前に、判断することができる。

5.5. 導入判断における留意点のまとめ

本章で示した考え方、留意点をまとめると以下の通りである。

- 5.2 章を参考に効果とコストの主要な要素と全体像を把握する。
- 効果とコストの主な要素について、5.3 章に示した具体例や考え方を参考に、ドメイン知識に基づき大まかに規模を評価・把握することが重要。特にソフトウェアの不具合による損害リスクのうち、情報システムに関する事故による企業価値の毀損を含めて、大まかな規模を把握することは重要。
- 組織として複数のプロジェクトで採算性を確保すると考える。プログラミング言語と同様に、単一のプロジェクトで、採算をとることは現実的な考え方ではない。
- コストと効果は、適用対象、適用手法、適用範囲(広さ)、適用レベル(深さ)などによって大きく異なるため、応用事例集(12 章)の情報なども参考に、対象システムとの類似性、適用レベル、効果の概要を把握して、効果とコストの規模に関する感覚をつかむ。

⁸⁷ Woodcock, J., Larsen, P. G., Bicarregui, J., and Fitzgerald, J. 2009. Formal methods: Practice and experience. ACM Comput. Surv. 41, 4 (Oct. 2009), 1-36.

6. フォーマルメソッドと関連技術の位置づけ

本章では、ソフトウェアに対する要求とそれを満たすために使われる技術の全体像をまとめ、その中でフォーマルメソッドの位置づけ、用途、制約、他の技術との関係などについて整理する。

対象読者	(1)ベンダー上級管理者 (2)開発プロジェクト管理者 (3)開発技術者、等
目的	ソフトウェアに対する要求としての機能要求、非機能要求、ディペンダビリティ等の関係や全体像について整理し、要求が満たされていることを確認するために用いられる技術の全体像とその中におけるフォーマルメソッドの役割、特徴、他の技術との関係を把握することで、フォーマルメソッドの位置づけの適切な理解を促進することを目的とする。
想定知識	ソフトウェア開発に関する基礎
得られる知見等	<ul style="list-style-type: none">● ソフトウェアに対する要求(機能性、安全性、信頼性、ユーザビリティ、ディペンダビリティ等)とそれを満たすための技術の全体像● 対策技術の全体におけるフォーマルメソッドの位置づけ● 開発プロセス(V字モデルなど)全体におけるフォーマルメソッドの適用箇所、用途● フォーマルメソッドに適した用途、フォーマルメソッドの制約や課題および他の技術との組合せの必要性

ソフトウェアに対する要求が満たされることを保証するためには、ソフトウェアを含むシステム全体で議論しなければならない⁸⁸。そのようなシステム全体におけるフォーマルメソッドの位置付けや用途について把握することは重要である。それらについて把握し、フォーマルメソッドと他の技術の組合せ利用について以下のようなことを検討する際の参考にするとよい。

- (1) 満たすべき要求の全体像を把握し、そのうちのどこにフォーマルメソッドが適しているか把握する。
- (2) 要件を満たすために利用される技術とフォーマルメソッドの関係を把握する。
- (3) フォーマルメソッドの適用箇所、およびその他の技術の補完的適用を検討する。

⁸⁸ Safeware: System Safety and Computers, Nancy G. Leveson, Addison-Wesley Professional, 1995

6.1. ソフトウェアに対する要求の把握

6.1.1. 機能要求と非機能要求

ソフトウェアに対する要求は、大きく以下の2つに分けられる⁸⁹。

表 6-1:ソフトウェアに対する要求

	内容
機能要求	ユーザがソフトウェアに対して求める中心的な機能で、ある入力に対して、出力(画面の表示変更等を含む。)を伴って提供されるもの。
非機能要求	機能要求に対する特性(制約や品質等)に関する要求で、具体的には、信頼性、安全性、性能、操作性、運用性などが含まれる。

機能要求は、ユーザが求める機能であるため、ソフトウェア開発の初期段階から比較的良好に検討される事項であるが、非機能要求の中には性能、利用性、セキュリティなどユーザが初期段階で、把握しにくいものもあるため、上流工程で十分な検討がなされないことが多い。人命に関わるようなセーフティクリティカル・システムや、基幹業務に関わるミッションクリティカル・システムなどにおいては、機能要求はもとより、非機能要求に対して高い水準が求められる。安全性等の非機能要求は、ソフトウェア・ベンダー側のステークホルダーよりも、ドメインにおける過去の経験や失敗情報を把握している事業者やユーザの方がより適切に指摘できる場合がある。「情報システムの信頼性向上に関するガイドライン」では、「非機能要求」を明確に定義することが情報システムの信頼性を向上する上で重要であると指摘している。

ソフトウェアに対する要求については、ソフトウェアの要求仕様や品質特性に関する国際標準⁹⁰等の様々な整理がなされている。

表 6-2:ISO/IEC 9126⁹¹ (ソフトウェアの品質特性)

要求	説明	特性
機能性 (functionality)	明示的及び暗示的の必要性に合致する機能を提供する	合目的性 正確性 相互運用性
信頼性 ⁹²	指定された達成水準を維持する特	成熟性

⁸⁹ IEEE Std 830 Software Requirement Specification Template

⁹⁰ ISO/IEC 9126 Software engineering — Product quality

⁹¹ ソフトウェアの品質に関する標準 ISO/IEC 25000:2005 Software Engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE に統合される。

⁹² 本ガイダンスでは、ソフトウェアの信頼性という用語を、ソフトウェア品質に関する国際標準 ISO9126, ISO25000 における信頼性や、Nancy Leveson のセーフウェアにおける信頼性の考え方に基づくものとして用いる。つまり、ある状況においてソフトウェアがどの程度機能するかに関する特性で、ソフトウェアが要求仕様に遵守している程度や、設計や実装に不具合が無い度合いを指すものとする。一方、通常、ハードウェアを含むシステムにおける信頼性は、与えられた期間、想定された条件の下で、求められた機能を実行する確率によって表される特性で、ハードウェアの故障が原因となる障害も含むもので、より広い概念である。

(reliability)	性	障害許容性 回復性
ユーザビリティ (usability)	理解、習得、利用でき、利用者にとって魅力ある特性	理解性 習得性 運用性 魅力性
効率性 (efficiency)	使用する資源の量に対して適切な性能を提供する特性	時間効率性 資源効率性
保守性 (maintainability)	保守(変更)作業のし易さに関する特性	分析容易性 変更容易性 安定性 試験容易性
移植性 (portability)	保守(変更)作業のし易さに関する特性	環境適応性 設置性 共存性

ソフトウェアの品質に関しては、表 6-1 のソフトウェア品質特性に関する国際標準 (ISO/IEC9126)が参考になる。従来、ソフトウェアの品質は、不具合がないことだけに偏っていたが、ISO/IEC9126により、信頼性やユーザビリティなど幅広い特性について認識が広まってきた。

非機能要求の要素については様々な提案がなされ、統一的な分類には至っていない⁹³が、Kotonya と Sommerville (KS)⁹⁴の非機能要求⁹⁵を例にとると以下のような要求を含むものとしている。

表 6-3:非機能要求の主な要素とその内容

非機能要求の要素	内容
安全性	ユーザや環境に、破壊的な影響を与えないこと。
セキュリティ	意図的な攻撃に対する耐性があること。
信頼性	意図した通りサービスを提供すること。
ユーザビリティ	美しさ、一貫性、単純性、文書の整備など
性能要求	応答時間、スピード、スループット、リソース使用率、効率性など。

フォーマルメソッドは、技術的な制約などにより、上記の全ての要素に対して適用できるわけで

⁹³ 非機能要求、山本修一郎、Business Communication, 2006年9月

⁹⁴ Gerald Kotonya and Ian Sommerville, Requirements Engineering, John Wiley & Sons, 2002

⁹⁵ Requirements Engineering : Processes and Techniques, Gerald Kotonya and Ian Sommerville., John Wiley 1998.

はない。そのため、フォーマルメソッドの適性を把握して、適した対象を選ぶことにより効果が得られると期待できる。具体的には、ソフトウェアの品質特性における機能性、信頼性や、非機能要求の主要素のうち、安全性、セキュリティ、信頼性などがフォーマルメソッドの対象として有望と言える^{66,111}。一方、フォーマルメソッドは、ユーザビリティ、性能、効率性、保守性、移植性に関しては、取り扱いが難しい。第7章や付録の応用事例集等を参考に、成功事例の多い対象を参考に、成功確度の高いものから試行するとよい。

6.1.2. ディペンダビリティの要素と対策

セーフティクリティカル・システムやミッションクリティカル・システムにおいては、高い信頼性や安全性が求められる。ソフトウェアを含むシステム全体の観点では、信頼性や安全性を含む広義の概念として、Jean-Claude Laprie などの提唱により⁹⁶、「ディペンダビリティ (dependability)」^{97, 98, 99}という言葉が使われるようになっていく。A. Avizienisらは、システムのディペンダビリティを以下の要素を含む広い概念として整理している^{100,101}：

表 6-4: ディペンダビリティの構成要素

要素	説明
信頼性(Reliability)	意図した通りサービスを提供すること。
安全性(Safety)	ユーザや環境に、破壊的な影響を与えないこと。
可用性(Availability)	サービスを適切な時に提供できる能力。
完全性(Integrity)	システムに対する不適切な変更が無いこと。
保守性(Maintainability)	変更や修正を行う能力。
機密性(Confidentiality)	権限の無い者への情報開示が無いこと。

ディペンダビリティのうち、機密性、完全性、可用性のことをセキュリティの要素としても捉えられる。

ディペンダビリティを阻害する原因は脅威と呼ばれ、以下のように分類される^{102, 103}：

⁹⁶ J. C. Laprie. "Dependable Computing and Fault Tolerance: Concepts and terminology," in Proc. 15th IEEE Int. Symp. on Fault-Tolerant Computing, 1985

⁹⁷ ディペンダビリティの概念は、(狭義の)信頼性 (reliability) を要素として含んでいるため、「信頼性」という言葉と区別して「ディペンダビリティ」と呼ぶ。

⁹⁸ これらの要素は、排他的ではなく互いに重なりを持っている。これらの要素のうち、信頼性と可用性は、定量的に計測可能だが、その他の要素はより主観的なものである。

⁹⁹国内規格 JIS においても、ディペンダビリティを、availability(可用性)、maintanability(保守性)、integrity (保全性)、confidentiality or security(機密性)、safety(安全性)などを包含する概念と定義している。

¹⁰⁰ A. Avizienis, V. Magnus U, J. C. Laprie, and B. Randell, "Fundamental Concepts of Dependability," presented at ISW-2000, Cambridge, MA, 2000.

¹⁰¹ Basic concepts and taxonomy of dependable and secure computing, Avizienis, A. et al, Dependable and Secure Computing, IEEE Transactions on Issue Date: Jan.-March 2004, Vol.1 Issue:1, pp.11 - 33, 2004

¹⁰² Basic concepts and taxonomy of dependable and secure computing, Avizienis, A. et al, Dependable

表 6-5:ディペンダビリティに対する脅威の分類

	分類	説明
脅威	エラー/誤り(error)	欠陥の原因となる(人間の)知識や行為のこと。行為が正しくても知識が間違っていれば、欠陥が生じる。正しい知識に基づいていても、行為が誤っていれば欠陥が生じる
	欠陥/バグ(fault)	エラーが原因で、プログラムや設計などの記述に埋め込まれた間違い。
	障害/故障(failure)	欠陥に基づいて発生した望ましくない事象。

情報セキュリティの分野では、脆弱性という用語が使われる。これは、仕様には明記されず、出荷時には明確になっていないもので、障害に至る原因となるもので、欠陥とは区別される。

さらにディペンダビリティの確保に関する対策は以下のように分類される。

表 6-6:ディペンダビリティに関する対策の分類

対策の分類	内容
欠陥予防 (Fault prevention)	欠陥が入り込まないための対策。仕様が正しく記述され、正しく実装されるための対策。
欠陥耐性 (Fault tolerance)	障害の発生を前提として、傷害が具現したときに、事故に至らないような対策。
欠陥除去 (Fault Removal)	欠陥がすでに含まれている時に、それらを検出し、除去する対策。
欠陥予測 (Fault forecasting)	欠陥の除去や回避を行うために、欠陥の存在の可能性を予測する対策。

ソフトウェアのディペンダビリティを確保するためには、人間の誤りによるバグの混入をいかに少なくするかという問題がある。テストによって欠陥/バグを見つけられるのは、テストケースにより障害が顕在化した場合のみであり、すべての欠陥を見つけられるものではない。

ディペンダビリティ対策の分類ごとの具体的な手法の例を挙げると以下ようになる。

表 6-7:ディペンダビリティ対策の手法例

対策の分類	手法の例
欠陥予防	<ul style="list-style-type: none"> ● 要求仕様書の作成 ● オブジェクト指向設計プログラミング手法

and Secure Computing, IEEE Transactions on Issue Date: Jan.-March 2004, Vol.1 Issue:1, pp.11 - 33, 2004

¹⁰³ J.C. Laprie. "Dependable Computing and Fault Tolerance: Concepts and terminology," in Proc. 15th IEEE Int. Symp. on Fault-Tolerant Computing

	<ul style="list-style-type: none"> ● デザインパターン ● モデルベース開発¹⁰⁴ ● フォーマルメソッド ● 要求管理・インパクト解析
欠陥耐性	<ul style="list-style-type: none"> ● エラー処理 ● 多重化 ● エラー処理を設計(アーキテクチャ記述言語 AADL 等)
欠陥除去	<ul style="list-style-type: none"> ● デバッガ ● 脆弱性チェッカー¹⁰⁵ ● テスト(ブラックボックス、ホワイトボックス) ● コードレビュー、コード・インスペクション ● フォーマルメソッド(不具合解析)
欠陥予測	<ul style="list-style-type: none"> ● ハザード・リスク分析¹⁰⁶ ➤ FMEA¹⁰⁷: 構成部品の故障モードを元にボトムアップで、影響や対策を分析(原因から結果) ➤ FTA¹⁰⁸: 問題となる事象の要因をトップダウンに分析する。(結果から原因にさかのぼる) ➤ HAZOP¹⁰⁹: システムの持つ危険事象を見つけ出し、致命度(Seriousness)の程度に従って区別する ● 信頼度成長曲線¹¹⁰

ディペンダビリティの対策分類とソフトウェアの開発工程ごとに、実際に利用される技術を整理すると図 6-1 のようになる^{111, 112, 113}。

¹⁰⁴ モデルベース開発: ソフトウェア開発においては、仕様をシミュレーションや検証可能なモデルで表現し、各工程内でモデルのシミュレーション等による検証と修正を繰り返し構成する開発手法。

¹⁰⁵ 脆弱性チェッカー: ソフトウェアのバッファオーバーフローなどネットワーク攻撃等に関する不具合を検出するツール。

¹⁰⁶ ハザード・リスク分析: 事故の原因とその影響の関係について分析する手法。

¹⁰⁷ FMEA: 設計の不完全や潜在的な欠点を見出すために構成要素の故障モードとその上位要素への影響を解析する技法。

¹⁰⁸ FTA: 事故などの事象について、発生経路、発生原因及び発生確率を原因の関係を木を用いて解析する手法。

¹⁰⁹ HAZOP: プロセスの目標値からのずれを想定し、そのずれの起こる原因と発生する危険事象を解析し、さらにその原因から危険事象に進展するのを防護する機能を評価し、対策を検討する技術。

¹¹⁰ 信頼度成長曲線: テスト工程においてテストケース数に応じて発見されたバグの累積件数をグラフ化したもの。

¹¹¹ Safeware: System Safety and Computer, Nancy Leveson, Addison Wesley

¹¹² EASIS D3.2 Part1: Guidelines for establishing dependability requirements and performing hazard analysis

¹¹³ Jackson, D. et al, Software for dependable systems. sufficient evidence?, NATIONAL RESEARCH COUNCIL, 2008

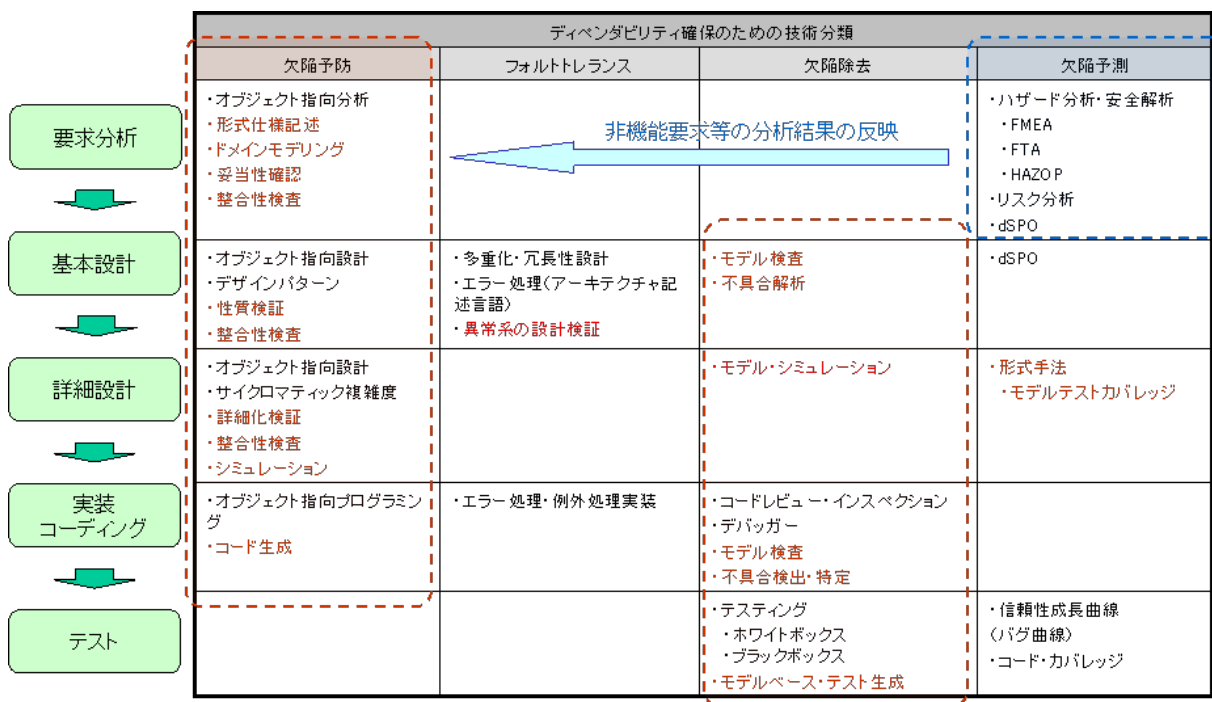


図 6-1: ディペンダビリティ確保の対策分類と利用される技術の関係

フォーマルメソッドは、欠陥予防、欠陥除去を中心に利用され、形式仕様記述による曖昧性の除去、仕様検証、不具合検出などに有効である。一方、図を見るとディペンダビリティを確保するためには、フォーマルメソッド以外の手法が必要な領域があることも示している。不具合から生じる障害や、障害の原因となる不具合などを解析するハザード・リスク分析(FTA, FMEA, HAZOP 等)は、高い安全性・信頼性を求められるシステムにとっては不可欠である。また、従来のソフトウェア・テストは、不具合が無いことを示すためには有効ではないが、コストの面で、フォーマルメソッドで全て置き換えられるわけではない。これらの技術を併用することで、目的とする要求を満たすことが現実的なアプローチと考えられる。

6.1.3. 信頼性・安全性・情報セキュリティの関係

従来から、ディペンダビリティの要素のうち、(狭義の)信頼性、安全性に対する要求は高いものであると認識されてきた。情報システムがネットワークに接続されることが一般的になり外部からの脅威に対して防御が求められるようになった今日では、信頼性、安全性に加えて、情報セキュリティに対する要求が高まっている。

信頼性、安全性、情報セキュリティの関係は、下図に示すような交わりを持つが必ずしも一致しない¹¹⁴。

¹¹⁴ 文献 111 をもとに、情報セキュリティの関係も加えて作成。

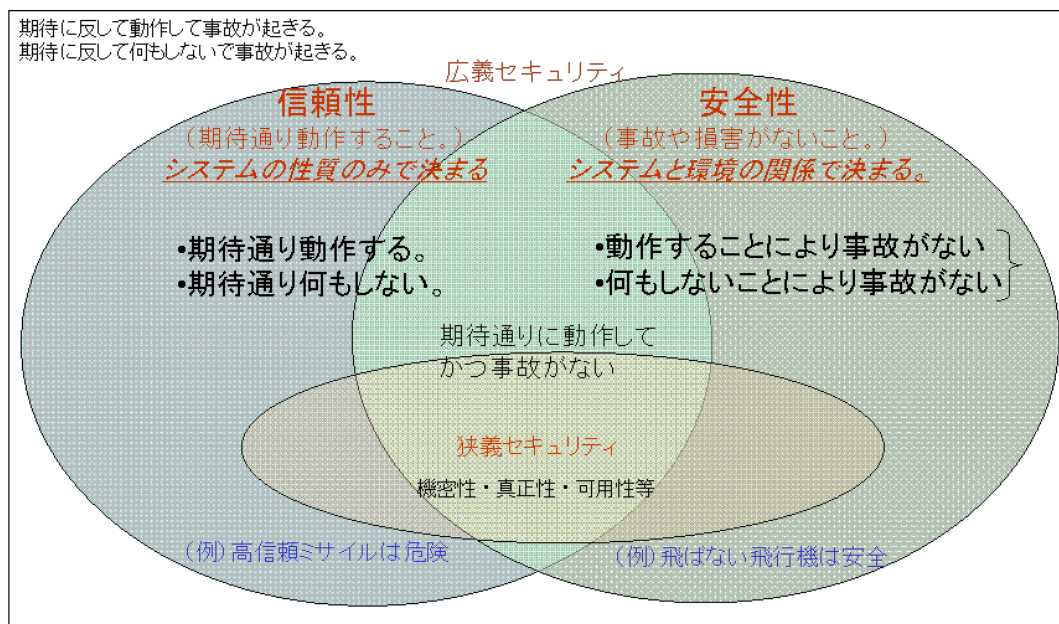


図 6-2:信頼性・安全性・セキュリティの関係

(狭義の)信頼性は、システムが仕様通りに動作することを意味し、安全性は、被害や損害を生じる事象に至らないことを意味する。信頼性と安全性は、互いに関係性を持つが、信頼性が高いシステムは、必ずしも安全とは限らない。実際、多くのセーフティクリティカル・システムでは、仕様通りに正しく動作している時に事故が発生している。逆に、仕様通りに機能を果たさない、何もしないシステムは、常に安全であるが、目的を達成しないシステムは意味がない。システムが本来期待される機能を提供するという(信頼性が確保されたという)前提の下で、初めて安全性を議論することに意味がある。一方、安全性は、システム単独で議論することはできず、システムと外部環境の関係において議論しなければならない。要求分析において、システム障害時の想定被害やその影響範囲なども分析して、要求に反映させる必要がある。

情報セキュリティは、信頼性と安全性と関係する部分もあるが、狭義の情報セキュリティは、意図的な攻撃による機密性、可用性、真正性が失われるようなことに対する耐性を指す。意図的な攻撃は、情報システムの企画・設計段階で網羅的に把握することが難しいため、要求分析の困難さがある。一般的な設計プロセスにおいては、正しい使い方を前提としたユーザであるアクターに基づきユースケースを用いた要求分析が行われてきたが、セキュリティを確保するためには、従来のユースケースに、攻撃者の視点で行動する攻撃アクターを加えたミスユースケースを作成し、セキュリティ要求を十分に洗い出すことが重要になっている。通常、システムの完全なセキュリティを保証することは経済性の面で現実的ではなく、トレードオフを考慮した脅威と攻撃のリスクに対するセキュリティ保証技術が必要である¹¹⁵。

¹¹⁵ Bashar Nuseibeh, Valuing Security: on risky requirements and expensive design, Software Security Symposium 2007

6.2. ソフトウェアの品質に関する対策技術とフォーマルメソッドの位置づけ

6.2.1. フォーマルメソッドの用途と特徴

ソフトウェア・テストでは、不具合の存在を発見することができても、不具合が存在し無いことを保証することができない¹¹⁶。つまり、いくらテストをしても、ソフトウェアの正しさを保証することができない。一方、フォーマルメソッドは、記述した性質に関して、形式検証に成功すれば、システム（設計や実装）が、与えられた性質を満たすことを保証できる点が、テストとの大きな違いである。

プログラムは、ある機能をどのように処理するか記述しなければならない。一方、フォーマルメソッドは、システムが機能を「どのように実現するか」(How)を記述することなく、「何を実現するか」(What)を記述することができる点に特徴がある。テストは、プログラムの実行を前提とするため、以下のような問題がある：

- プログラムが実装された後でなければ検査できない。
- 実行に依存するため、並列システムの動作タイミングなど再現性に依存する性質は、完全には検査できない。

フォーマルメソッドは、実現方法を記述することなく、何を実現するか記述でき、それらに対して検証も可能であるため、テストと比べて、要求分析・設計等の開発上流プロセスにおいて適用することが可能であり、早期に不具合を発見することで、開発手戻りを押さえる効果がある。安全性に関わるエラーの多くは、要求分析に原因であり¹¹⁷、要求仕様のレベルで検証を行えるフォーマルメソッドは、コスト低減の効果が大きいといえる。ただし、フォーマルメソッドは、特定の数学モデルに基づく制約の強い言語で記述するため、記述の自由が高いプログラムに比べて、記述の工数が大きくなることがある。したがって、フォーマルメソッドは、上流の設計工程で適用する方が高い費用対効果が得られる。

フォーマルメソッドの主な適用箇所と用途は図 6-3 のようになる¹¹⁸。

¹¹⁶ Edsger Wybe Dijkstra による言葉。

¹¹⁷ Abernethy, K. et al, Technology Transfer Issues for Formal Methods of Software Specification, Proceedings, 13th Conference on Software Engineering Education & Training, 2000, 6-8 March 2000, pp. 23 - 31

¹¹⁸ ソフトウェア開発の分野では、中間成果物に対して様々な用語が用いられる。「RFP(提案依頼書)」「要件定義書」「要求仕様書」は、ユーザがどんなシステムを開発したいのかを記述したものであり、立場や目的により異なる。RFP は、発注者がベンダーを選定するためのものであり、要件定義書は、システム開発プロジェクトにおける要件分析フェーズのアウトプットなどである。ソフトウェア工学の分野では、これらは Requirement(要求)という用語が用いられる。本ガイドンスでは、明確に仕様化されていないユーザ等のニーズを「要求」と呼び、仕様化されたものを「要求仕様」と呼ぶ。また、実現方法を仕様化したものを「設計仕様」(基本設計、詳細設計など、詳細度に応じて複数のレベルが想定される。)。また、「仕様」とは、異なるステークホルダー間の合意文書である。「要求仕様」は、発注者とベンダーの間の合意文書、「設計仕様」は、設計者とプログラマーの間の合意文書である。単に「仕様」というと、「設計仕様」を指す場合もあるが、曖昧であるため、本ガイドンスでは、「要求仕様」と「設計仕様」を明確に区別して表現する。

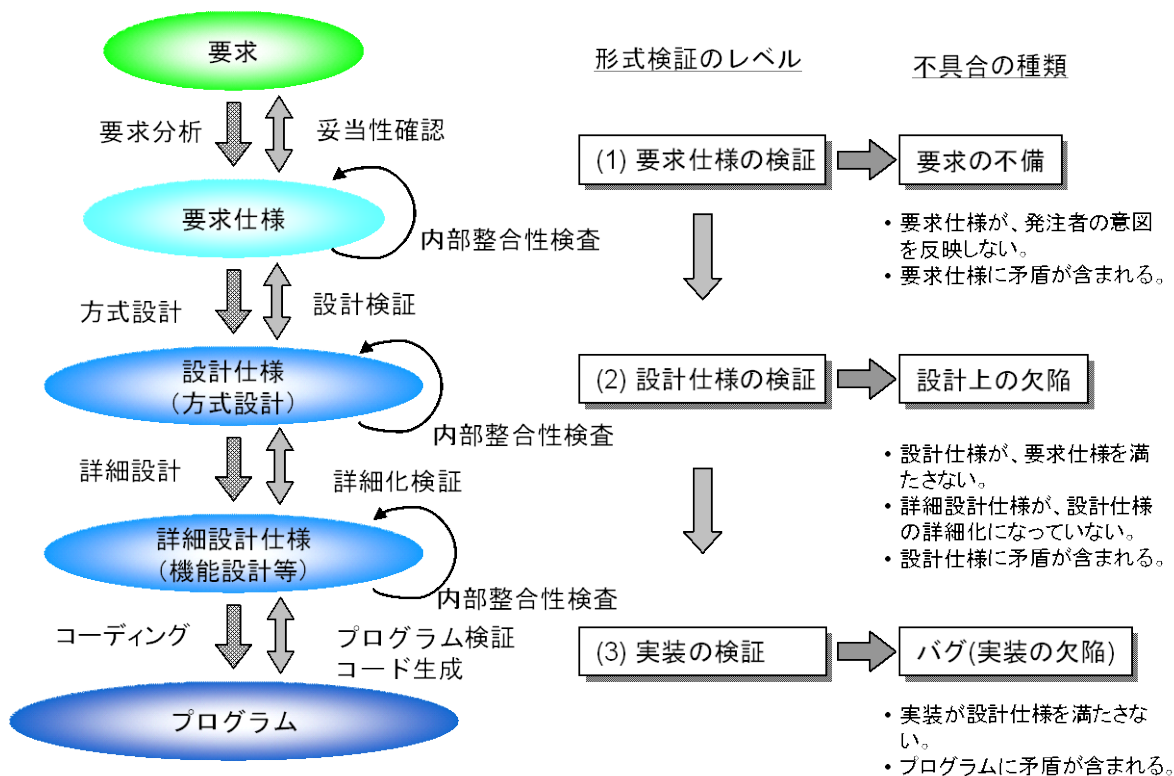


図 6-3: フォーマルメソッドの主な適用箇所と用途

図 6-3 に示したフォーマルメソッドの用途を整理すると、主に以下のようなレベルで適用されるものに分類される。

表 6-8: 検証のレベル

検証のレベル	内容
実装の検証	実装に不具合(バグ)がないか。実装が設計を満たしているか。
設計仕様の検証	設計上の不具合はないか。設計仕様が要求仕様を満たしているか。設計に矛盾、定義漏れがないか。
要求仕様の検証	要求仕様はだとうであるか。要求に矛盾がないか。要求は、ユーザの意図に沿っているか。

6.2.2. V字モデルにおける各手法の位置づけ

フォーマルメソッドとディペンダビリティ等を確保するためのその他の手法を、ソフトウェア開発V字モデルにおける位置づけを整理すると以下ようになる。

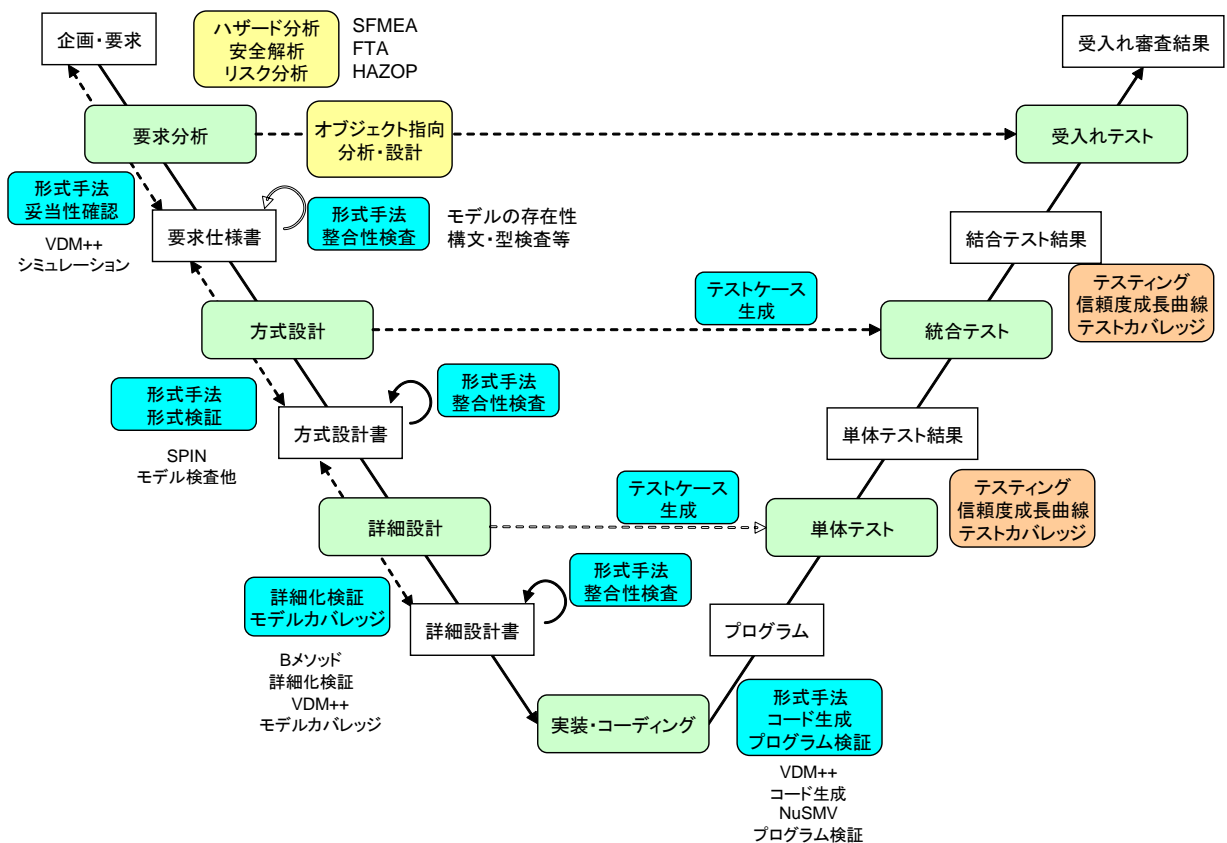


図 6-4: 開発 V 字モデルにおける各種手法の適用箇所

フォーマルメソッドは、主に、上流工程で適用される。フォーマルメソッドは、定義された仕様間の整合性や矛盾について解析することはできるが、ハザードの洗い出しを行うことには適していないため、セクティブ・クリティカル・システムに求められるリスクやハザードの洗い出しには、フォーマルメソッド適用の前に、ハザード分析などが使われる。

詳細設計やコード生成が可能なレベルの設計記述では、B メソッドや VDM++などのフォーマルメソッドが使われる。どちらもコード生成機能を持っており、形式仕様記述言語で記述した設計仕様のレビューや検証項目のレビュー、シミュレーション実行、形式証明により設計仕様の品質を高める。

一方、モデル検査法では、自動検証は可能であるが、検証できるシステム規模に制約が強く、通常、上流設計の検証や、ソースコードの一部のモジュールに対する検証に制限される。

テストは、システムに期待する動作が実際に行われるかという機能要求に適しているが、安全性などの非機能要求には適さないことがある。フォーマルメソッドは、陥ってはいけない状態など安全性に関わる検査も対象とできる点に特徴がある。

6.3. システム全体の一部としてのソフトウェアの位置づけ

ソフトウェアは、それが動作するシステムという状況の中でのみ、安全性や信頼性を評価すること

ができる。したがって、ソフトウェアを含むシステム全体について考えることが重要である。フォーマルメソッドは、エンタプライズ系やパソコンなど汎用のコンピュータシステムだけではなく、航空機、鉄道システム、電力システム、情報家電など組み込みシステムへの適用の方が多数を占めている^{119, 120}。また、近年、制御対象の物理システムと制御システムのハイブリッドシステム的设计モデリングと検証に対するニーズが高まっている^{121, 122, 123, 124}。これらに対するフォーマルメソッドの位置づけと役割を以下にまとめる。

6.3.1. 機能安全における安全性確保の考え方

機能安全とは、機能的な工夫により極力安全を確保する¹²⁵ための安全機能を実現するシステムに対する国際規格である。機能安全においては、システムの障害を、以下の2つに分類する。

障害の分類	内容
確率的ハードウェア障害	部品などハードウェアの劣化等により確率的に発生する障害
決定論的障害(系統的障害)	設計や実装の欠陥により決定論的に発生する障害

機能安全では、これらの障害に対して図 6-5 のように異なるアプローチで安全性の確保を目指すものである。

¹¹⁹ Woodcock, J., Larsen, P. G., Bicarregui, J., and Fitzgerald, J. 2009. Formal methods: Practice and experience. ACM Comput. Surv. 41, 4 (Oct. 2009), 1-36.

¹²⁰ IPA, 形式手法適用調査, 2010

¹²¹ Cyber-Physical Systems Research Charge, Jeannette M. Wing, Cyber-Physical Systems Summit, 2008

¹²² Cyber Physical Systems: Design Challenges, Edward A. Lee, Technical Report No. UCB/EECS-2008-8

¹²³ 連続系と離散系の統合モデリングと標準化, 制御設計研究会, JASA, 目時 伸哉, 2010

¹²⁴ EASIS, Marko Auerswald, Guidelines for the Development of Dependable Integrated Safety Systems, Formal Verification Techniques, 2006

¹²⁵ NECA 技術委員会報告 第三の波「機能安全」

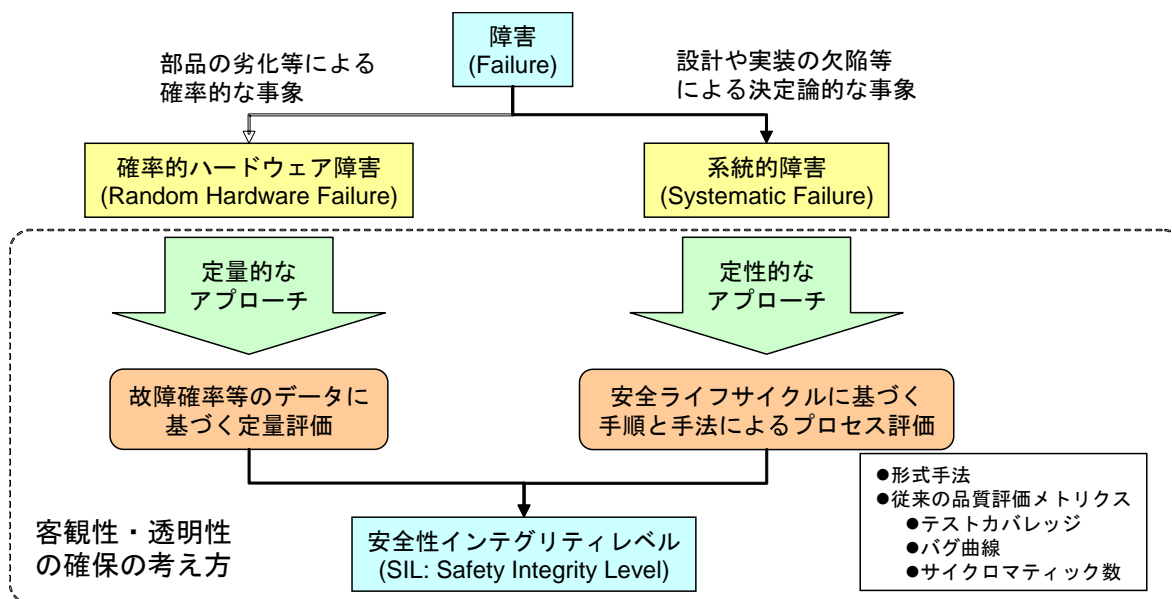


図 6-5: 障害の区別と評価対策アプローチ

確率的ハードウェア障害に対しては、故障確率等のデータに基づく定量的に評価する。一方、ソフトウェアを中心に系統的障害(決定論的障害)については、定量的に評価することは困難であるとの認識から、開発手順や適用手法によるプロセス評価によって安全性の確保を目指す。これらの両方のアプローチによりシステムの安全性レベル(SIL)を確保する。

SILは、システムのユーザである発注者が、自己責任を前提に設定する。SILは、モジュールごとにリスク評価に基づき設定される。もっとも高いレベル4では、フォーマルメソッドの適用が強く推奨され、フォーマルメソッドを適用しない場合は、認証機関に対して合理的な説明が求められる。

6.3.2. ハイブリッドシステムへの応用に関する現状

自動車等の制御システムでは、制御対象の物理システム(連続モデル¹²⁶)と制御システムの離散システム(離散モデル¹²⁷)のハイブリッドシステムの設計と検証が必要になる。近年、自動車のソフトウェア化が急速に進み、ハイブリッドシステムに対する設計や検証の要求が高まっている。また、医療機器、ビル管理、橋等のインフラ管理などにおいて、連続システムである物理システムと離散システムであるサイバーシステムのハイブリッドシステムに対する設計、検証のニーズが高まっている^{128, 129}。

ハイブリッドシステムにおいては、システムの重要な性質や要求は、システムと環境の連結したダイナミクスによって記述される。連続モデル(微分方程式)で表される環境からのフィードバックを扱う必要がある場合、離散モデルと連続モデルの統合モデルが必要になる。産業レベルの

¹²⁶ 物理法則に基づき微分方程式で表されるもの。

¹²⁷ 時間的に離散イベントに基づく状態遷移システムなど。

¹²⁸ Cyber Physical Systems: Design Challenges, Edward A. Lee, Technical Report No. UCB/EECS-2008-8

¹²⁹ Cyber-Physical Systems Research Charge, Jeannette M. Wing, Cyber-Physical Systems Summit, 24 April 2008

ツールとしては、Stateflow で拡張した Simulink、MatrixX、VisSim で拡張した Statemate などがあるが、これらはハイブリッドシステムのモデリング、シミュレーション、プロトタイピングを行うための機能を提供するが、ハイブリッドモデルに対する検証機能がない。

ハイブリッドシステムに対するフォーマルメソッドのアプローチとしては、ハイブリッド・オートマトンがある。ただし、ハイブリッド・システムに対する形式検証は研究途上にあり、以下のような研究が活発に進められている。

- ハイブリッドシステムを有限オートマトンあるいは時間オートマトンでマニュアルあるいは自動でエンコードし、モデル検査を可能にする。
- 厳密または近似したハイブリッドシステムの到達可能状態の直接生成
- 非線形制御からハイブリッド制御への証明ルールの拡張

これらの分野については、今後の動向を注視し、適用の可能性を検討していくことが重要である。

6.3.3. リアルタイムシステムへの応用に関する現状

リアルタイムシステムは、外部環境との相互作用において、ある入力に対する出力が一定の時間制約を満たすようなシステムである。多くの組込みシステム、特にセーフティクリティカル・システムは、リアルタイム性の要求が求められる。フォーマルメソッドによる自動検証を目的とした場合、時間オートマトンに基づくモデルが用いられる¹³⁰。時間オートマトンのモデル検査系で最も先進的なものの例としてUPPAALが挙げられる。いくつかの有効なケーススタディは見られるが、以下のような問題点がある：

- ツールは Boolean や制限された整数など単純なデータ構造のみしか扱うことができない。
- 状態爆発の問題を起しやすい。

時間オートマトンに関するモデル検査において、このような課題を克服するための研究が行われている¹³¹。また、モデル検査による自動検査に対して、PVSやIsabelleを用いたインタラクティブな証明により状態爆発を回避する方法についても研究されている。

これらの分野についても、今後の動向を注視し、適用の可能性を検討していくことが重要である。

6.4. まとめ

ソフトウェア・テストは、不具合の一部を発見することができても、不具合が無いことを保証することができない。フォーマルメソッドは、定義した性質についての検証が成功すれば、その性質に関して不具合が存在し無いことを保証できるという特徴がある。また、プログラムは、ある機能をどの

¹³⁰ EASIS Guidelines for verification and validation of dependability requirements, Formal Verification Techniques

¹³¹ E Asarin, Maler, and A. Pnuell, Data-Structures for the Verfication of timed automata, In Proc. Of the Int. Workshop on Hybrid and Real-Time Systems, 1997.

ように処理するか記述しなければならないが、フォーマルメソッドは、システムを「どのように実現するか」(How)を記述することなく、「何を実現するか」(What)を記述することができる点が特徴である。これにより、実現方法を記述する前に、開発の上流工程で矛盾を発見したり検証したりすることができる。また、ユーザに近い上位レベルの検証性質を記述することにより、本来の目的に近い検証が可能になる。

一方で、フォーマルメソッドの実践応用例では、フォーマルメソッドが全てのコンポーネントに対して、全ての工程で用いられていない。フォーマルメソッドは、モデル検査における状態爆発や、定理証明における証明の失敗、厳密な仕様を記述することによるコストの増加、記述できる範囲の制約などの問題もある。このようなことから、フォーマルメソッドは、従来のテストを置き換えるものではなく、補完的に組み合わせて利用されることが現実的である。また、フォーマルメソッドは、セーフティクリティカル・システムに求められるハザード分析による障害の原因などの求を洗い出しに適していない。これらの手法と組み合わせて利用されることも必要である。

フォーマルメソッドは、対象システムや手法の適性に応じて、検証範囲を特定し、従来のテストを部分的に置き換えるなどして、フォーマルメソッドと他の手法を組合せすることで、コストを抑えながら全体としてソフトウェアの信頼性・安全性を向上さるといった利用を検討することが重要である。

7. 手法の選択方法

本章は、形式手法の導入を検討する際、具体的にどの手法を導入するべきか、という手法の選択に関する参考情報を提示する。本章の概要は以下の通りである。

想定読者	(1)開発プロジェクト管理者 (2)開発技術者
目的	ソフトウェア開発に係わる管理者や技術者が、新たに形式手法を導入する際に、目的にあった形式手法を選択するための情報提供を目的とする。 用途と想定する適用の度合いに応じて、過去の事例に基づき適していると考えられる手法を絞り込んで提示する。また、主な形式手法の概要と特徴について情報を提示する。
想定知識	ソフトウェア開発に関する基礎、形式手法に関する基礎
得られる知見等	<ul style="list-style-type: none">● 主な形式手法の概要● 主な形式手法の用途● 成功事例における典型的な手法の選択

形式手法は、基礎とする数学モデルや用途に応じて多数の手法が存在する。また、仕様の記述力やツールにより提供される検証機能などに違いがあるため、用途に応じた使い分けが重要である。利用者の検証目的に適した形式手法を選択し、対象システムのうち形式手法に適した箇所に応用することにより、高い効果を得ることができる。以下では、比較選択のための絞り込みの観点と個々の手法の情報を提供する。

7.1. 形式手法の比較選択の手順

本節では、本章を通して提示する、形式手法の比較選択の手順について整理する。

まず、7.2 節では、主な形式手法について概観し、形式手法の想定用途と適用の度合いに応じて、手法と適用実績の関係について参考情報を提示する(図 7-1、図 7-2)。ただし、対象とする形式手法は、実用レベルの適用例の多い手法にあらかじめ以下の通り限定している。これら以外の手法でも実用上有益と考えられる一部の手法については、付録 17.1.2 節に示している。本章では、各手法について適用可能な範囲を全て示しているものではなく、手法の特徴を明確化するため典型的な適用例を示している。

以下、手法を比較する上で、モデル検査と形式仕様記述・定理証明と便宜的に分類している。

(1) モデル検査

- SPIN
- NuSMV
- UPPAAL
- SCADE

(2) 形式仕様記述・定理証明

- B
- Event-B
- VDM++,

また、これらの手法については、7.3 節にて各手法の概要を示すほか、「対象として適しているソフトウェア」の分野、「開発プロセスの中での位置づけ」を踏まえた処理の流れ、モデルの「記述力」、「大規模ソフトウェアへの対応」、「ツールサポートの状況」に関する情報を記す。

これらの情報を基に、導入候補となる形式手法を限定できることを意図して本章は構成されている。

7.2. 主な形式手法の概観

多くの形式手法は、適用対象に制限を明示してはいない。しかし、それぞれの手法の基礎となる数学モデルや提供ツールなどを考慮して、適切な対象に適切な手法を適用することが重要である。以下では、主要な形式手法の位置づけを概観するため、手法の用途と適用する開発工程による定性的な分類(7.2.1 小節)と、適用事例に基づく分類(7.2.2 小節)との2通りの分類を示す。

7.2.1. 用途と適用する開発工程による整理

形式手法の用途を、ここでは大別して(1)仕様記述によるプロトタイピング、(2)モデル検査による自動検証、(3)定理証明による検証、とする。仕様記述によるプロトタイピングでは、開発対象の動作モデルを形式言語で記述し、動作を確認する。モデル検査による自動検証では、システムの仕様が定義された検証性質を満たすことを保証するものである。定理証明による検証では、形式仕様記述言語で記述したシステムの仕様と、定義した検証性質との間で不具合や仕様の内部不整合を発見する。

形式手法の適用工程を大まかに分類すると、(1)基本設計、(2)詳細設計、(3)実装・コード、を扱う段階に分けられる。これは、形式手法で扱う対象の抽象度と対応している。基本設計の時には、抽象度の高いモデルを扱うのに対し、実装の段階では非常に具体的なコードを対象とする。

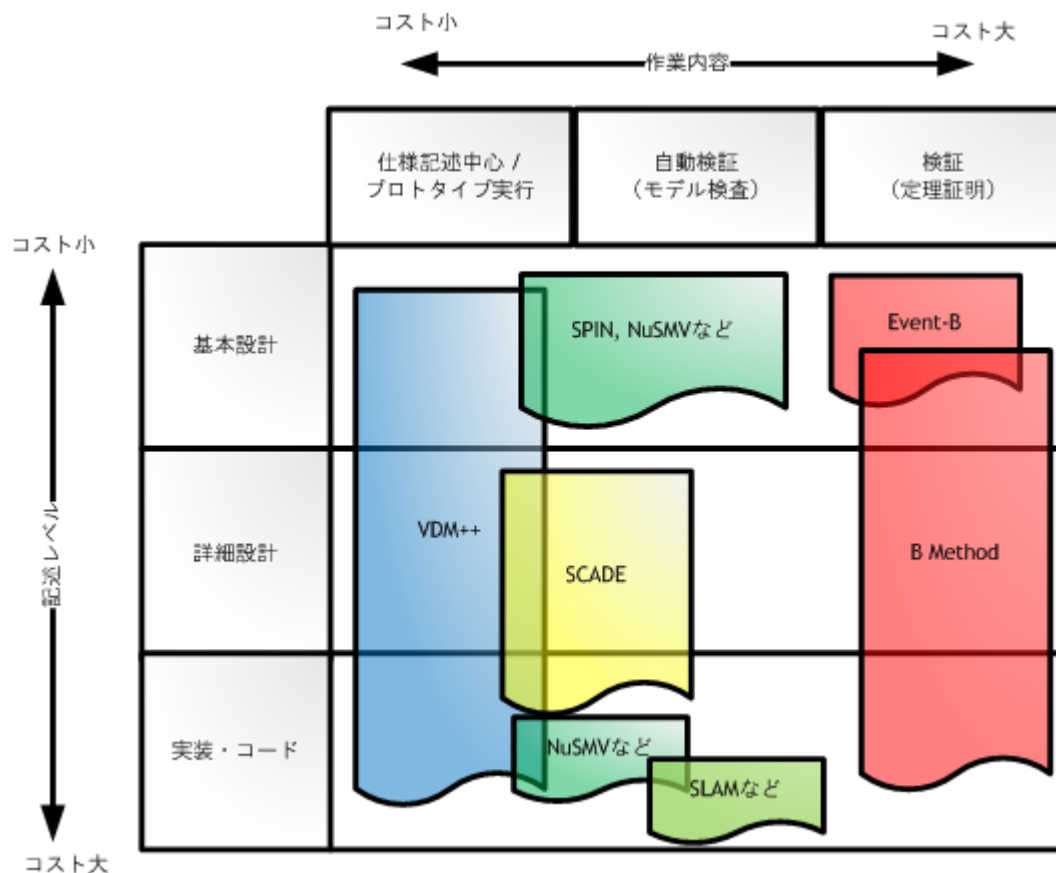


図 7-1: 主な形式手法の用途例の俯瞰

以上の分類に基づき、主な形式手法が適した用途の位置付けを示したものが、図 7-1 である。図 7-1 では、VDM++が各工程を通して仕様記述によるプロトタイピングに適しており、B メソッド / Event-B が定理証明による検証に適していることを示している。また、モデル検査については、SPIN、NuSMV が基本設計・実装レベルで利用されることと、Windows のデバイスドライバの検証で著名な SLAM も実装レベルで利用されていることを示している。SCADe 等は、詳細設計・実装レベルで利用される。なお、SCADe のようなモデルベース開発を支援する環境では、仕様記述や検証の作業が利用者からは隠蔽されているという特徴がある。

7.2.2. 適用事例に基づく整理

本節では、形式手法の成功事例を目的別に整理した情報を示す。

図 7-2 は、形式手法適用の目的、扱う対象(種別、範囲、詳細度)により、適していると考えられる手法を提示している。これは、産業界での適用事例が報告された事例などに基づく参考例を示すもので、必ずしもこれに制限されるものではない。

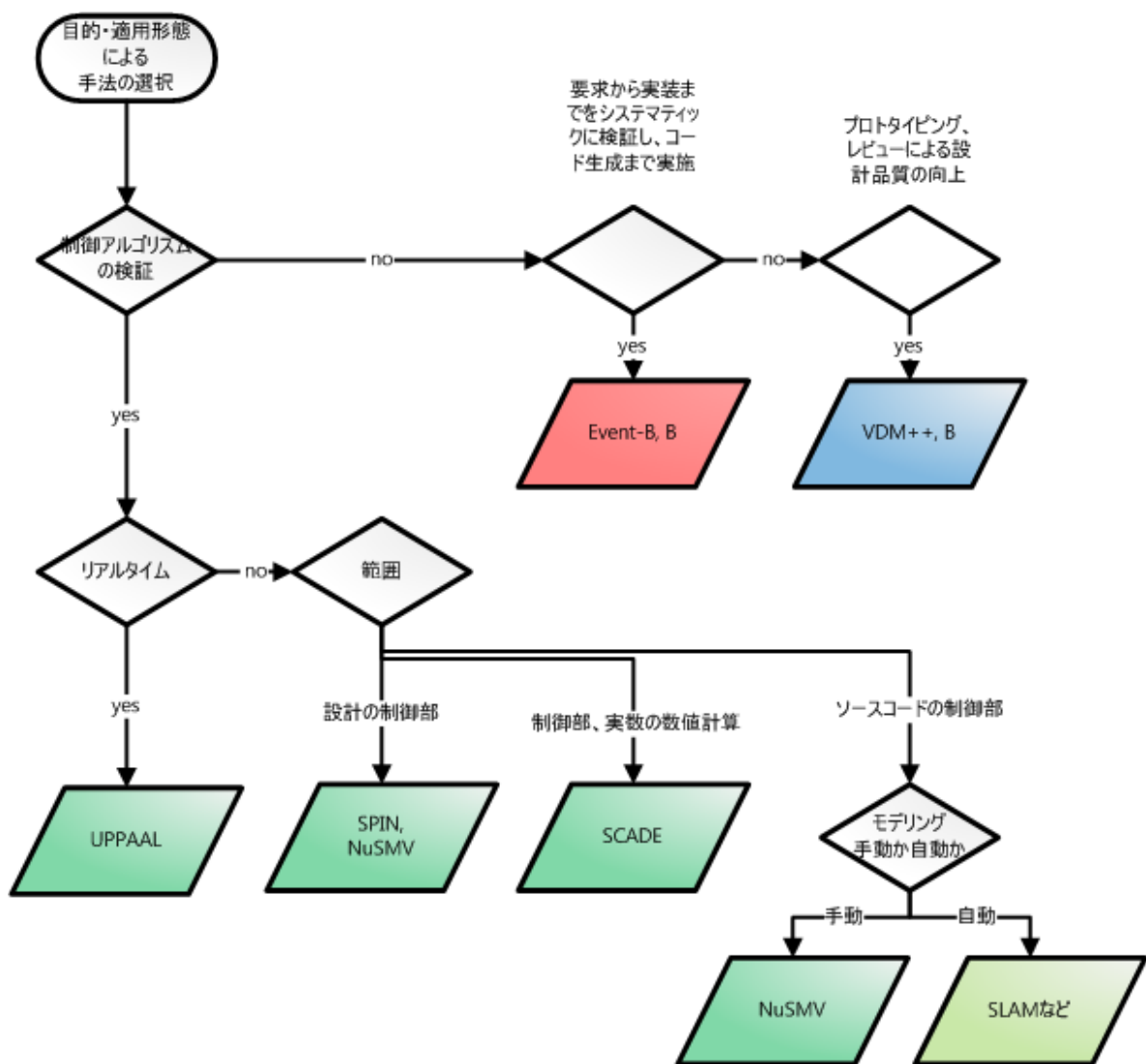


図 7-2: 成功事例に基づく目的と対象(レベル)による適用手法の典型例

適用の目的は、大別して以下の 3 分類としている。

1. 制御アルゴリズムの検証
2. 要求から実装までの自動検証、自動コード生成
3. プロトタイピング、レビューによる設計品質の向上

以下、対象について、適用の目的ごとに説明する。

1 の制御アルゴリズムでは、いわゆる制御系のシステムを対象としているため、リアルタイム性の有無が適用する形式手法の選択に影響する。また、適用範囲の広さは、リアルタイム性が無い場合、設計の制御部、ソースコードの制御部、制御部・実数の数値計算に分類できる。それぞれ開発工程が基本設計、実装、詳細設計と異なる。これらの組み合わせごとに、SPIN、NuSMV、

SLAM、SCADE といった実績のある手法が存在する。リアルタイム性がある場合は、UPPALL が適しているとされる。

2 の場合、ソフトウェア全般を扱うが、システム全体を記述するとコストがかかるため、実装の対象はセーフティクリティカルな個所に限定されている。この用途では、B メソッドが近年よく使われている。

3 の場合も同様に B メソッドが使われるほか、VDM++はプロトタイプングなど、要求や設計の整理に適している。

7.3. 主な形式手法の参考情報

本節では、手法の選択に際して参考となる手法の特徴などの情報を、より詳細に提示する。まず、表 7-1 に図 7-1、図 7-2 で挙げた主な形式手法の概要を示す。また、以下の 7.3.1 小節と 7.3.2 小節では、表 7-1 に記載した主要な各手法について整理する。整理の観点は、「対象として適しているソフトウェア」、「開発プロセスの中での位置づけ」、「記述力」、「大規模ソフトウェアへの対応」、「ツールサポートの状況」とする。なお、モデル検査の検証性質を記述する際に利用される時相論理のうち、特に LTL と CTL については 13.2 節に整理する。

表 7-1: 主な形式手法の概要

手法名	年代	開発組織	説明	関係サイト
SPIN	1980年代	Bell Labs' formal methods and verification group	<ul style="list-style-type: none"> ● SPIN は分散システムの形式検証に利用できるツールである。1980年にBell研究所で作られた。2002年4月に、ACMのSystem Software Awardを受賞している。 ● SPIN はシステムの仕様記述に PROMELA (PROcess MEta LAnguage)という高水準言語を利用する。 ● SPIN は分散システム設計における論理的なエラーの追跡に使われてきた。対象システムは、OS、データ通信プロトコル、スイッチシステム、並行アルゴリズム、鉄道の信号プロトコルなど。 ● ツールは、仕様の論理的な一貫性を検証する。デッドロック、明示されていない受信、競合条件などをレポートする。 	SPIN - http://spinroot.com/spin/whatispin.html
NuSMV	1990年代	CMU, IRST(Istituto per la Ricerca	<ul style="list-style-type: none"> ● NuSMV はシンボリックモデル検査ツールである。状態遷移モデルの状態空間をそのまま探索せずに、順序付き二分決定木(BDD)に基づいて論理関数で記号的にモデル検査をする。 	NuSMV - http://nusmv.fbk.eu/NuSMV/in

		Scientifica e Tecnologica)	<ul style="list-style-type: none"> ● NuSMV は CMU SMV を再実装、拡張したツールである。CMU SMV との違いとしては以下が挙げられる。 <ul style="list-style-type: none"> - 複数のインタフェース、LTL による性質記述の対応といったユーザビリティの向上と、効率化や状態爆発の部分的な制御といったヒューリスティクス - システムアーキテクチャがモジュール化されたオープンな構造 ● バージョン 2 以降、BDD だけでなく、SAT ベースの有界モデル検査もできるようになっているほか、CTL、LTL の双方を性質記述に利用できる。 	dex.html
UPPAAL	1990 年代	Uppsala University, Aalborg University	<ul style="list-style-type: none"> ● UPPAAL は、リアルタイムシステムのモデリング、V&V のための統合ツール環境である。V&V は、グラフィカルなシミュレーションによる妥当性確認とモデル検査による検証である。 ● UPPAAL が対象とするシステムは、時間オートマトンでモデル化される。時間オートマトンでは、状態遷移モデルにクロックと呼ばれる実時間的な振る舞いをする変数を追加する。 ● UPPAAL における検証は、CTL のサブセットを用いる。検証項目は、状態の定式化(デッドロックの検出など)、到達可能性、安全性、ライブネスである。 	UPPAAL.org - http://www.uppaal.org
SCADE	1990 年代	Esterel Technologies	<ul style="list-style-type: none"> ● SCADE (Safety Critical Application Design Environment)は、セーフティクリティカルな組み込みソフトウェアのための設計環境である。モデルベース開発と形式検証による開発支援がなされる。 ● Scade to C コンパイラは航空業界の開発プロセスに関する規範である DO-178B レベル A の認証を受けており、形式検証後のソースコードにも高い信頼性がある。 SCADE ツールは、 <ul style="list-style-type: none"> * シミュレータ * モデルカバレッジ分析 * 形式検証器 などからなる。 ● SCADE の基に Lustre という形式的に定義された同期言語がある。歴史的経緯として、原子力発電の監視と 	Esterel - http://www.esterel-technologies.com/company/history/

			緊急停止装置を行う Saga の設計に Lustre の概念が使われており、さらに Airbus A320 の飛行制御システムを開発するのに使われていた SAO というツールと統合することで SCADE となった。	
B	1980 年代	Jean Raymond Abrial	<ul style="list-style-type: none"> ● B Method は仕様記述からプログラム生成までを支援するソフトウェア開発手法である。Z、VDM と同様に、モデル規範型の形式手法として分類される。B は記法だけでなく、開発手法全体を含んでいる。 ● B Methodのソフトウェア開発スタイルは段階的詳細化に基づき、詳細化の各段階に合わせた仕様記述言語が提供されるとともに、各段階で仕様の整合性を検証する方法と、上位の仕様から下位の仕様への詳細化の正当性を検証する方法が提供される。仕様記述言語や検証項目は、この全ての過程でツールによる支援が受けられるように考えられている¹³²。 ● B では、B 抽象機械記法 (B Abstract Machine Notation: AMN) という形式仕様記述言語を利用して、抽象機械 (MACHINE) を記述する。抽象機械では、外部インタフェース仕様を定める。抽象機械を段階的にリファインメントすることで、内部設計を行い実装に近づけていく。実装では、B0 言語という言葉を用いて、非決定的な要素を除き、計算機によって実装可能な要素で記述する。 ● AtelierB などのツールがメンテナンスされている。 	B Method - http://www.bmethod.com/
Event-B	1990 年代	Jean Raymond Abrial	<ul style="list-style-type: none"> ● Event-B はシステムモデリングと分析のための形式手法である。主な特徴は、 <ul style="list-style-type: none"> - 集合論をモデリングの基底として利用していること - 抽象度の異なるシステムの表現にリファインメントという概念を使うこと - リファインメントの異なるレベル間での一貫性の検証に数学的な証明を用いること ● その名の示すとおり、Event-BはB Methodから派生している。考案者も同じ Jean-Raymond Abrial であり、上記の特徴も B の特徴を引き継いでいる。 	Event-B.or g http://www.event-b.org/

¹³² 出典：来間啓伸(著)，中島震(監修)，Bメソッドによる形式仕様記述，近代科学社，2007

			<ul style="list-style-type: none"> ● B はソフトウェアの開発を主な対象としているが、Event-B は、システム全体(ハードウェア、ソフトウェア、外部環境)のモデリングを目的としている点で異なる。 ● 具体的には、B の OPERATION に代わって EVENT の概念が導入され、リファインメントの過程でイベントの追加や統合ができるなど、B よりも柔軟な開発手法となっている。 ● Rodin などのツールが開発されている。 	
VDM++	1990年代 (VDMは1970年代)	IBM ウィーン 研究所	<ul style="list-style-type: none"> ● VDM++はVDM(Vienna Development Method)にオブジェクト指向と並行処理の拡張を加えた形式手法である。 ● VDM は、PL/I コンパイラの並列処理も含めた正しさを形式検証するために、表示的意味をつけたところから始まる。VDM は、抽象的なモデルを具体的な実装に段階的にリファインメントしていくことを念頭に置いたモデル規範型の形式手法である。VDM-SL という各種の抽象を含む形式仕様記述言語を含んでおり、VDM-SL は ISO 標準となっている。 ● VDM++では、クラスなどのオブジェクト指向の構成要素を導入し、スケーラビリティと複雑さへの対応をしている。 ● VDMTools や Overture などのツールが現在もメンテナンスされている。 	VDM Portal - http://www.vdmportal.org/wiki/bin/view

7.3.1. モデル検査

モデル検査の主な手法を以下の 5 つの観点から整理する、

- ① 対象として適しているソフトウェア
- ② 開発プロセスの中での位置づけ
- ③ 記述力
- ④ 大規模ソフトウェアへの対応
- ⑤ ツールサポート

はじめにこの 5 つの観点について説明する。

「① 対象として適しているソフトウェア」については、モデル検査ツールがモデリングに利用する言語の特性や過去の事例から、対象として適しているソフトウェア領域(および適していないとき

れる領域)についての言及を整理する。

「② 開発プロセスの中での位置づけ」については、図 7-1 の根拠となる情報と、ツールによる処理フローを示すことで、個々のツールの利用イメージを提示する。

「③ 記述力」については、モデリングに利用する言語および、検証性質を記述する際の記述力について、整理する。

「④ 大規模ソフトウェアへの対応」は、モデル検査ツールの動作原理による部分と、利用時の工夫による部分について整理する。

「⑤ ツールサポート」については、利用できるツールの種類、開発状況、普及状況などについて整理する。

7.3.1.1. SPIN

① 対象として適しているソフトウェア

SPIN は、並列分散処理ソフトウェアを対象としている。SPIN が作られた初期の目的はプロトコル検証であった。OS、データ通信プロトコル、スイッチシステム、並行アルゴリズム、鉄道の信号プロトコルなどに事例が多い。

② 開発プロセスの中での位置づけ

SPIN は、設計段階で主に利用される(図 7-3)。検証対象とするソフトウェアのモデルを Promela で記述し、各プロセスをランダムに実行するよるシミュレーションと、検証プログラムを C 言語で生成・コンパイル後に実行し、網羅的な探索により性質が成り立つかを検証する、2つの実行方式がある。

③ 記述力

モデルの記述に利用する Promela では、プロセス、ローカル・グローバルデータ、メッセージチャンネルという構成物を利用して、並列分散システムのコミュニケーションを表現する。文字列型、浮動小数点型は具備していない。

Promela は非決定性構文を持つため、非決定的な事象のモデリングに適している。例えば、パケットロスが時々起こること、ハードウェアのエラーが起きることなどをモデル化できる。

性質の検証では、到達可能性、進行性、安全性の検証ができる。手順として、assert や LTL 式を組み合わせで検証する。assert では、モデル中の適当な場所に assert 文を挿入することで、当該個所における性質の成立を検査できる。LTL 式では、すべての無限長の実行パスに対して、検証性質が成り立つかどうかを記述する。LTL では、全称記号(\forall)、存在記号(\exists)に加え、G(\square , Always, つねに)、F(\heartsuit , Eventually, いつか)、X(Next time, つぎに)、U(Until, ~まで) を利用できる。例えば、p と q をそれぞれ論理式とすると、次のような記述ができる。

- p が成り立つ後は常に、いつかは q が成り立つ。

$\square(p \rightarrow (\heartsuit q))$

- システムのある特定の状況が無限回発生する(公平性)

$$([\langle p \rangle] \rightarrow \phi)$$

なお、LTL の表現については、13.2 小節で CTL と併せて整理する。

- ④ 大規模ソフトウェアへの対応

SPIN は状態を明示的に扱うため、状態爆発への対応が必須となる。SPIN 自体は、半順序法、状態データ圧縮などメモリ使用量を減らすオプションを用意している(9.4 節を参照)。

- ⑤ ツールサポート

SPIN 本体、および GUI ツールの ispin、jSPIN などは SPIN の公式サイト¹³³からダウンロードできる。

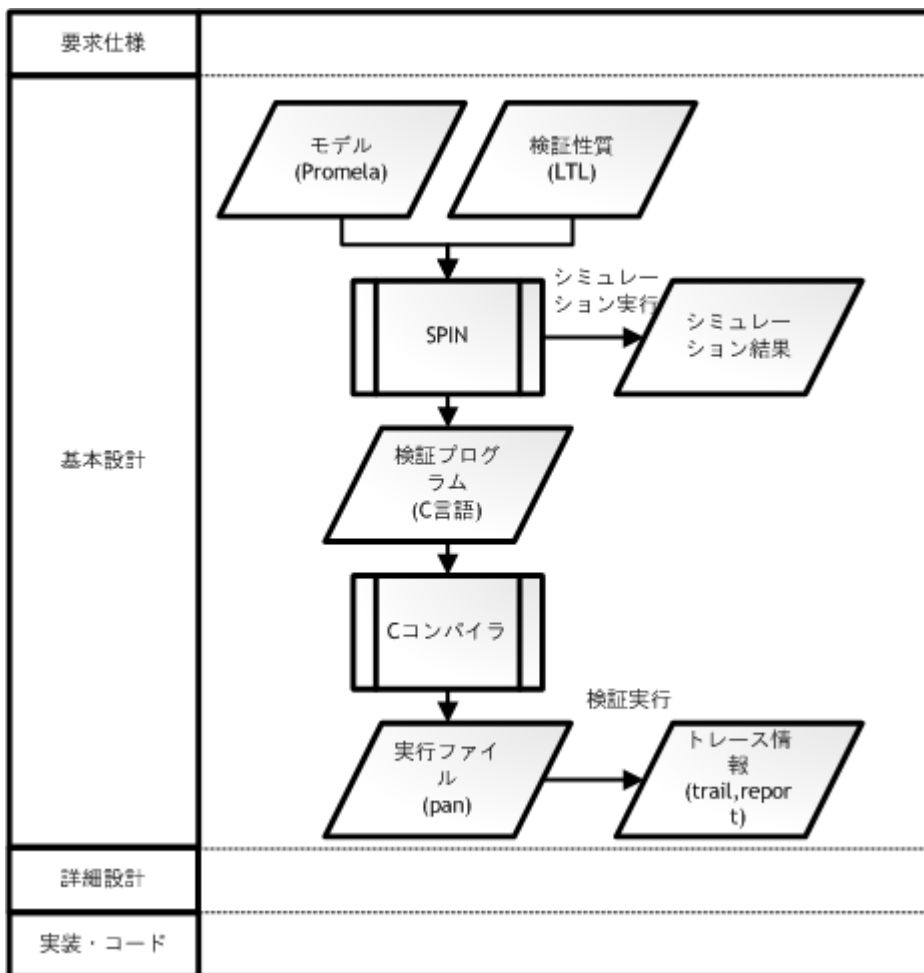


図 7-3: SPIN の開発プロセスにおける位置づけ

¹³³ <http://spinroot.com/>

7.3.1.2. NuSMV

① 対象として適しているソフトウェア

電子回路のようなシステムの記述に適しているとされる。同期・非同期の有限状態システムをモデル化できる¹³⁴。

リアルタイム性をモデリングすることは一般に不向きとされる。これは、同期的な状態機械は離散時間を扱うことによる。リアルタイムの振る舞いは、明示的にカウンター変数を使うなどして表現する必要がある。

② 開発プロセスの中での位置づけ

NuSMV は、設計段階で主に利用されるが、実装レベルの情報を基にモデルを作成してデバッグに利用されることもある(図 7-4)。

③ 記述力

モデルの記述力は、SPINのモデル記述言語のPromelaに近いとされる¹³⁵。ただし、検証性質にはLTLに加え、CTLとPSL¹³⁶を利用できる。また、モデル中に、不変条件、公平性を検査する予約語も提供されている。

④ 大規模ソフトウェアへの対応

NuSMV はシンボリックモデル検査の代表的な実装であり、BDD ベースと SAT ベースのモデル検査に対応している。BDD ベースのモデル検査は、無限パスを検査するのに対し、SAT ベースのモデル検査では、有限パスも検査できる。

SAT ベースのモデル検査により、比較的規模の大きなソフトウェアでのバグ検出ができる。

また、状態爆発への対応として、Model Simplification と Range Reduction.などの手段をオプションとして選択できる。

⑤ ツール

イタリア FBK-irst により開発が続いている。対話式シェルによるインタラクションモードとバッチモードが用意されている。

¹³⁴非同期についてはバージョン 2.5.0 以降非推奨となり、今後のバージョンではサポート外になる。

¹³⁵ Comparison of Model Checking Tools for Information Systems,
http://dx.doi.org/10.1007/978-3-642-16901-4_38

¹³⁶ <http://www.eda.org/vfv/docs/PSL-v1.1.pdf>

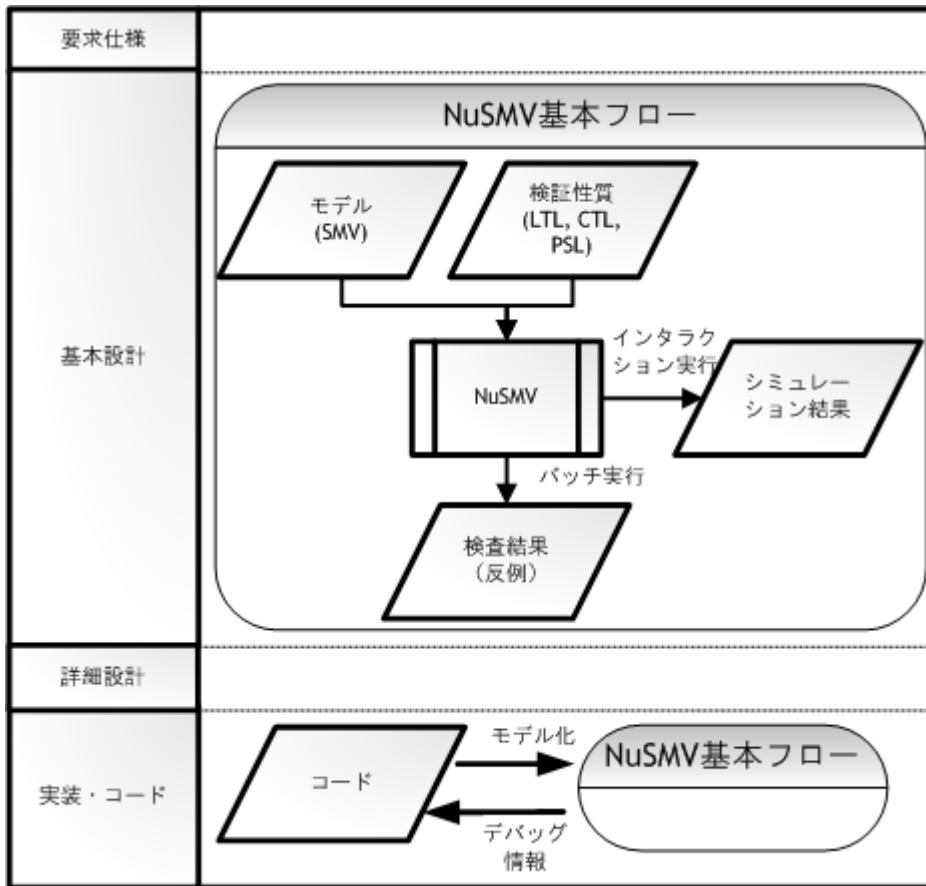


図 7-4: NuSMV の開発プロセスにおける位置づけ

7.3.1.3. UPPAAL

① 対象として適しているソフトウェア

リアルタイム制約のあるソフトウェアが対象として適しているとされる。リアルタイム制約とは、イベントが起きてから、反応をするまでのデッドラインがあることを指す。リアルタイム制約がある場合、論理的に正しい実行をするだけでなく、時間的にも正しいタイミングで実行する必要がある。

例として、映像・音声プロトコル、ギアボックス制御(自動車制御)、衝突回避プロトコル(ネットワーク)が挙げられる。

② 開発プロセスの中での位置づけ

設計時に、データ型を拡張した時間付きオートマトンのネットワークモデルをシステムのモデルとして作成する。作成したモデルに対して、シミュレータで動作の妥当性確認をし、CTL のサブセットである TCTL (timed computation tree logic)の簡略版を用いてモデル検査で性質の検証を行う(図 7-5)。

③ 記述力

GUI のモデリング環境を用いて、時間付きオートマトンのモデルを記述する。このモデルは、有限オートマトンに実数型のクロック変数を付与している。クロック変数は、システムの論理的な時間であり、0 から始まりグローバルに同期して進む。オートマトンの各状態からの状態遷移にはガード条件(**Guard**)をクロックの値を用いて記述できる。

オートマトンには、整数型変数、構造化データ型、チャンネル同期(**Synchronisation**)の拡張がなされている。また、代入(**Assignment**)もクロック変数や整数型変数に対して利用できる。また、クロック変数や整数型変数に対して不変条件(**Invariant**)も指定できる。性質の検証では、状態変数の値(デッドロック検出)、到達性、安全性、活性を検証できる。TCTL の簡略版であることの制約は、パス式のネストを許可しない点である。

④ 大規模ソフトウェアへの対応

モデル検査の検索手法は深さ優先と幅優先で切り替えられる。また、状態空間削減により、メモリにどれくらいの状態を含めるかを指定することや、状態空間の表現に、**Difference Bound Matrices (DBM)**を使うなどの工夫がなされている。

⑤ ツールサポート

Uppaal.orgより配布される。ただし、商用には別途ライセンスが必要である。CORA (cost-optimal reachability)、TRON(online testing)、Cover(offline test generation)、TIGA (timed game solver)、PORT(component based and partial order)、PRO (extension with probabilities)、TIMES(scheduling and analysis)などの周辺ソフトウェアも充実してきている¹³⁷。

¹³⁷ Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. 2011. Developing UPPAAL over 15 years. *Softw. Pract. Exper.* 41, 2 (February 2011), 133-142. DOI=10.1002/spe.1006 <http://dx.doi.org/10.1002/spe.1006>

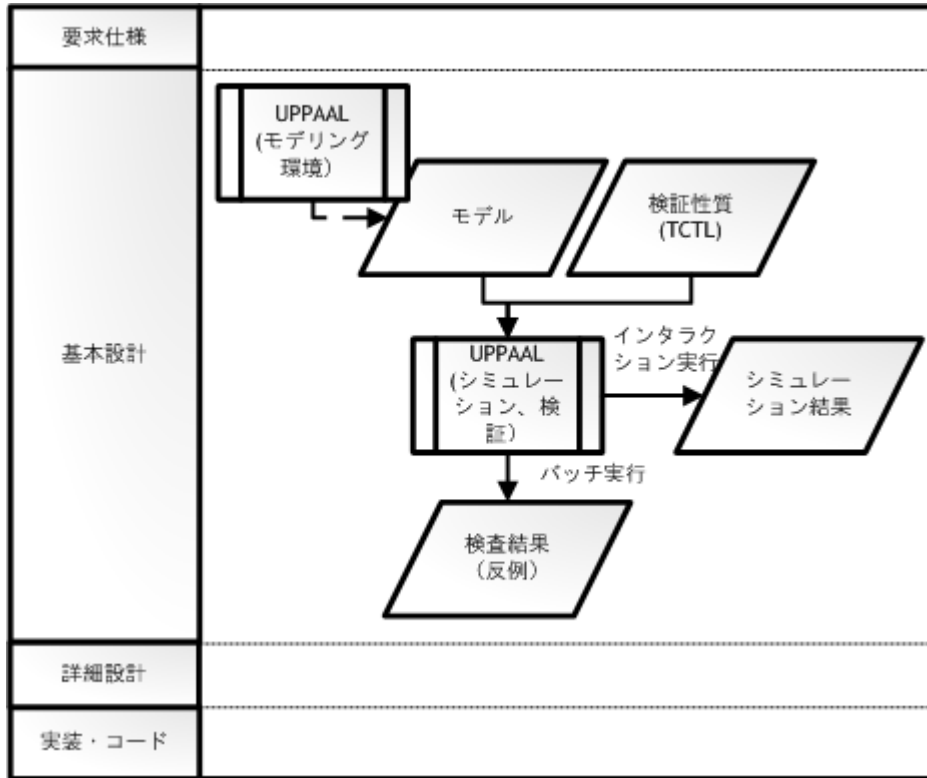


図 7-5:UPPAAL の開発プロセスにおける位置づけ

7.3.1.4. SCAD E

① 対象として適しているソフトウェア

連続モデルおよび離散モデルの記述が可能であるため、その両方のモデル化が必要なソフトウェアが適している。ただし、直接非線形な表現を取り扱うことはできないため、離散近似が困難であるモデルには向いていない。

また IEC-61508 SIL3 等の認証を取得した自動 C コード生成機能を持っているため、その認証レベルが必要なシステム構築に特に向いている。

② 開発プロセスの中での位置づけ

SCADE に基づく開発プロセスは図 7-6 の通り。SCADE の SCAD E Suite Design Verifier™ が形式的な検証機能を提供する。

③ 記述力

SCADE で用いられる言語は、データフロー図や状態遷移図を利用してモデルベースの記述をする同期型言語である。また、SCADE textual language を利用することにより、テキストベースでの記述もできる。

自動生成されるコードには、動的メモリアロケーションや無限ループ、再帰、OS のシステムコール、暗黙の型変換は含まれない。また、ポインタ演算が無く、関数を引数とし

て渡すことはできない。

基本型として **Boolean**、**Integer**、**Real**、**Character** が用意されている。またレガシーソフトウェアとの連携のために、**C** や **Ada** で定義された型をインポートすることができる。

④ 大規模ソフトウェアへの対応

データフロー図と状態遷移図をネストして記述することができる。

⑤ ツールサポート

SCADE は **Esterel Technologies** によりメンテナンスされている統合開発環境であり、検証機能もその一機能として提供されている。シミュレータや、モデルカバレッジ分析機能も具備している。

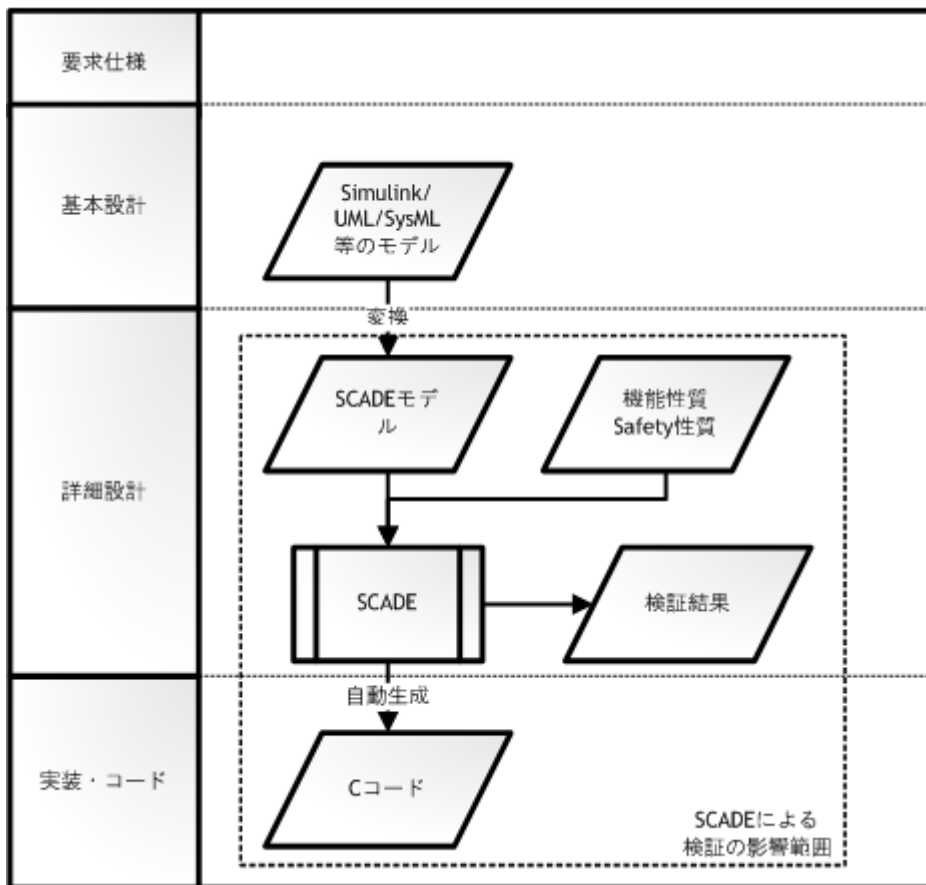


図 7-6: SCADE の開発プロセスにおける位置づけ

7.3.2. 形式仕様記述・定理証明

形式仕様記述言語を提供し、定理証明機能を備える主な手法について、整理する。整理の観点は、7.3.1 小節のモデル検査と同じ下記 5 通りである。

① 対象として適しているソフトウェア

- ② 開発プロセスの中での位置づけ
- ③ 記述力
- ④ 大規模ソフトウェアへの対応
- ⑤ ツールサポート

過去 25 年の形式手法の変遷としては、図 7-7 が Jean-Raymond Abrial 博士により提示されている。本小節では、このうち B(7.3.2.1)、Event-B(7.3.2.2)、および VDM の拡張である VDM++(7.3.2.3)を対象とした。これは、企業や政府の資金によりツールの開発が継続されていることと、最近の応用事例も考慮している。

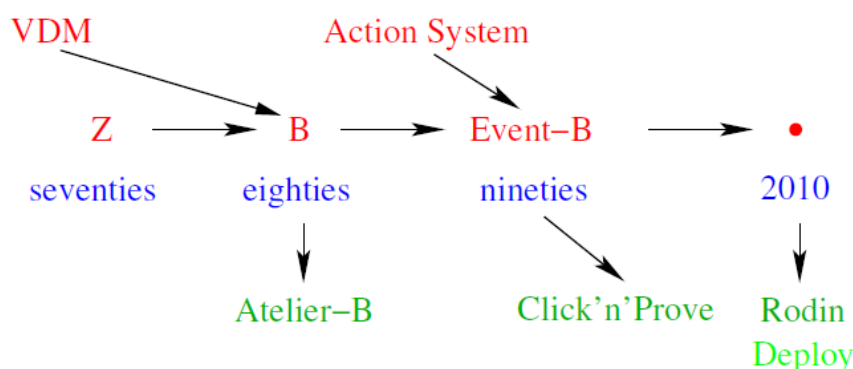


図 7-7: 過去 25 年の形式手法の変遷¹³⁸

7.3.2.1. B

- ① 対象として適しているソフトウェア

B による抽象機械の記述には、離散論理に基づくソフトウェアに適している。特に、集合論表現に落とし込み易い仕様を持つソフトウェアが適している。これには、応用事例としてあるような電車の駅におけるプラットフォームのドア開閉制御などが例としてあげられる。

並行システムや分散システムには不向きとされる。これは後述の Event-B が開発される背景でもある。

- ② 開発プロセスの中での位置づけ

B に基づく開発プロセスは図 7-8 に示す通り。基本的に、ソフトウェアライフサイクルの中心的な側面を扱う。技術的な仕様記述、連続する段階的な詳細化による設計、階層アーキテクチャ、および実行可能コード生成である。

B では、段階的に詳細化を進めていくなかで、不変条件を満たすことの証明も同時に行う。最終的には、非決定性を除去して実装コードへの自動変換が行われることが想定

¹³⁸ 出典: “Formal Methods in the Last 25 Years”, Jean-Raymond Abrial, 2011

されている。

③ 記述力

B の記述単位の抽象機械は、プログラムの仕様を記述することが目的であり、後の工程で命令型プログラミング言語によって記述されることを想定している。擬似的なプログラミング言語のような記法が提供されている。

抽象機械の記法では、すべての命令型プログラミング言語が持つような、代入と条件分岐がある。しかし、それ以外に事前条件、複数代入、束縛なし(あり)選択、ガードなどの実行、さらには実装されない要素も含んでいる。

抽象機械の段階では、シーケンス実行(一般に";"というオペレータで表現されることが多い)やループ(**While** 文)は使わない。詳細化の段階ではじめて一般化代入の一部として使われる。これは、抽象機械がプログラムの **how** ではなく **what** を記述することを目的としているためである。

抽象機械は、段階的詳細化を経て、非決定性を除去していき、最終的に **B0** 言語により **IMPLEMENTATION** という記述単位で仕様は表現される。これは、ツールによる自動変換を考慮するためである。

④ 大規模ソフトウェアへの対応

B では、モジュールの積み重ねにより大規模なソフトウェアシステムを構築する。**B** に基づく開発において、抽象機械の数学的なモデルは何回かの詳細化を経て、最終的には詳細化の最終段階から直接コンピュータ上で実行できるようになるところまで詳細化される。最終的な詳細化は、他の機械(モデル)を"**IMPORT**"し、次第に実装されていく。

⑤ ツールサポート

統合環境として **Atelier B**、モデル検査機能も提供する **ProB**、アニメータ機能を提供する **Brama** など、主に **EU** の基金により開発が進められている。また、詳細化の自動化、コード生成の自動化などのツールの開発も進められている。

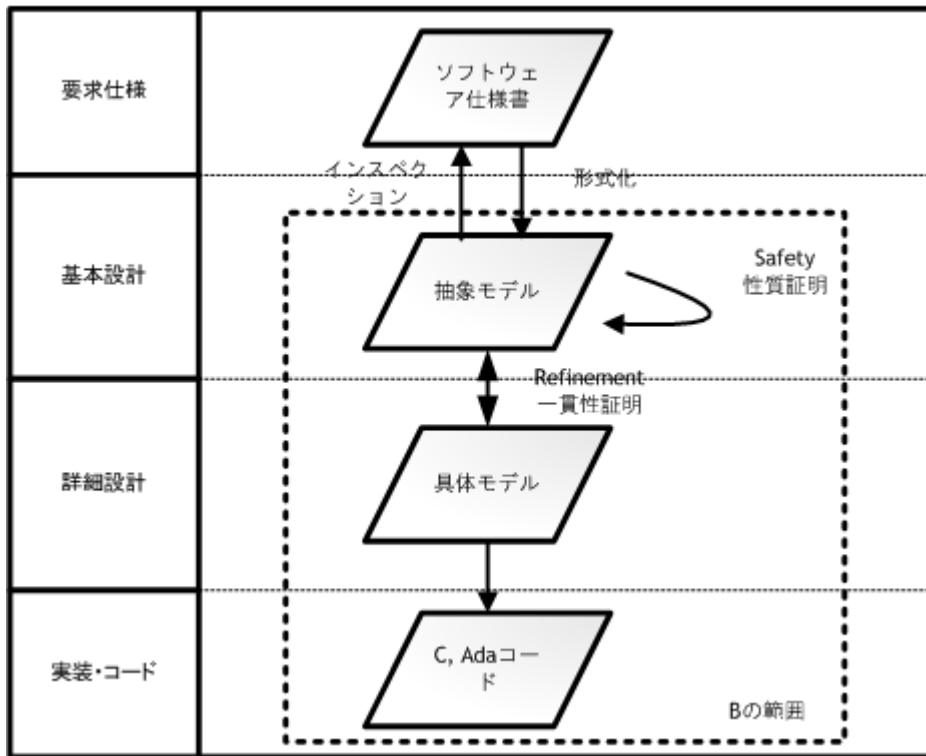


図 7-8: B の開発プロセスにおける位置づけ

7.3.2.2. Event-B

① 対象として適しているソフトウェア

B が対象とするソフトウェアに加え、並行システムやイベントベースの分散システムの記述にも向くとされる。

② 開発プロセスの中での位置づけ

図 7-9 に示す通り、システム要求から、段階的詳細化、分割、証明責務の証明を経て抽象アーキテクチャとソフトウェア要求仕様書

図 7-8 の B の開発プロセスにおいて、ソフトウェア仕様書に記載された非形式的な要求ドキュメントから段階的に情報を抽出し、抽象モデルを構築する。

③ 記述力

Event-B で使われる言語は、基本的に B で使われる言語に内包される。しかし、B のオペレーションに代わって、イベントの概念が導入されている。イベントには、事前条件と対照的なガード条件があり、ガード条件が満たされた時にイベントは発生する。このイベントの仕組みは、分散システムのモデリングに向いている。

また、B では、最初に定義した抽象機械により外部インターフェースを決定し、詳細化によって内部設計を行うのに対し、Event-B では、詳細化によってモジュール分割や外部

インタフェースの定義を進める。

仕様の性質の検証は、システムのあらゆる状態について不変条件が成り立つこと、全てのイベントが有限時間に起こり得ること(到達可能性)などに対して行うことができる。

④ 大規模ソフトウェアへの対応

分割とインスタンス化の仕組みなどを有する。分割の仕組みは、一つのモデルを系統的に複数のモデルに分割し、それぞれを独立に検証、具象化できる仕組みである。また、詳細化の過程でイベントの追加や統合もできる。インスタンス化は、基本集合や定数をパラメータとしたモデルのテンプレートに具体的な基本集合や定数を与えてインスタンス化することで、モデルを構成する。

⑤ ツールサポート

Rodin Platform¹³⁹がEUの研究開発基金により継続的に開発されている。Rodinプロジェクト(2004-2007)および、現在はDeploy(2008-2011)プロジェクトにてEclipseベースのオープンソースソフトウェアとしてメンテナンスされている。Rodinには開発中のプラグインが数多く存在する¹⁴⁰。モデリング(UML-Bなど)、アニメーション(ProBなど)、およびコード生成(B2C)などを利用できる。

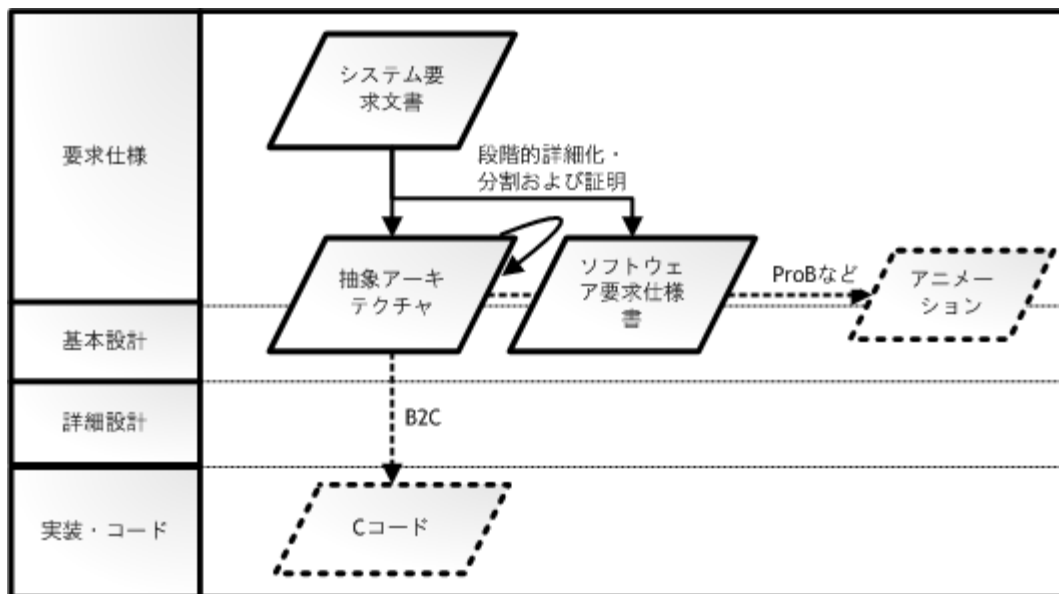


図 7-9: Event-B の開発プロセスにおける位置づけ

7.3.2.3. VDM++,

① 対象として適しているソフトウェア

クリティカルなコンピュータシステムのモデリングを指向している。例えば、航空、鉄道、

¹³⁹ <http://rodin.cs.ncl.ac.uk/deliverables.htm>

¹⁴⁰ http://wiki.event-b.org/index.php/Rodin_Plug-ins

自動車、原子力、軍事といったドメインのシステムが挙げられている。分散アーキテクチャの評価などに産業界での実績がある。

② 開発プロセスの中での位置づけ

VDM++は要求仕様を策定する段階と設計段階において活用される。テストケースを用いた動作シミュレーションや、外部 UML ドローツールとの連携などが想定されている。VDMTools などのツールには、Java や C++のコードを生成する機能も用意されている(図 7-10)。

③ 記述力

VDM-SL の記述とオブジェクト指向拡張、並行処理の拡張による記述力を持つ。

VDM はモデル規範型の仕様記述であり、シンプルで抽象的なデータ型(集合、リスト、写像など)を扱う。モデルにおける機能は、関数と操作という概念で抽象化される。また、双方に対し、事前条件、事後条件を追記できる。

オブジェクト指向の拡張により、Java 言語などのオブジェクト指向プログラミング言語にあるような、クラス、オブジェクトなどの概念を持つ。また、クラスの継承や、宣言や定義に `public`、`private` といったアクセス修飾子を設定できる。なお、関数型言語の特徴も持ち、`Let` 文や `Lambda` 文も利用できる。

並行性について、VDM++はスレッドに基づいている。スレッドは、共有オブジェクトを通して通信をする。共有オブジェクトに対する同期は、`permission` 述語により記述する。

④ 大規模ソフトウェアへの対応

大規模ソフトウェアへの対応には、継承などの言語そのものの特徴もあるが、むしろ VDM++の開発環境や、UML ツールなどとの連携によるところが大きいと考えられる。⑤でも触れるが、VDMTools や Overture は統合開発環境としての機能を有し、VDM++プロジェクトの管理を支援する。また、UML エディタの出力を取り込む機能なども、大規模ソフトウェアへの対応に役立つと考えられる。

⑤ ツールサポート

VDM++のツールとしては、IFAD A/S社が開発し、CSK Systems 社が現在はメンテナンスする VDMTools(<http://www.vdmtools.jp/en>)と、EU の研究開発基金で開発されている Overture (<http://www.overturetool.org>)が著名である。

VDMToolsは以下の機能を持つ¹⁴¹。

- 仕様の構文、型チェック機能
- 証明課題生成機能
- 実行可能仕様のインタプリタとデバッグ機能
- 実行可能仕様のコードカバレッジ計測機能
- Rational Rose との連動機能

¹⁴¹ 出典: VDM とは, VDMTools.jp, <http://www.vdmtools.jp/modules/tinydl/index.php?id=1>

- VDM-SL、VDM++の清書機能
- 実行可能仕様から Java や C++への生成機能(オプション)
- Java から VDM++への変換機能(オプション)
- CORBA API との連動機能(オプション)

Overture は Eclipse のプラグインとして実装された統合開発環境であり、GPL で公開されている。

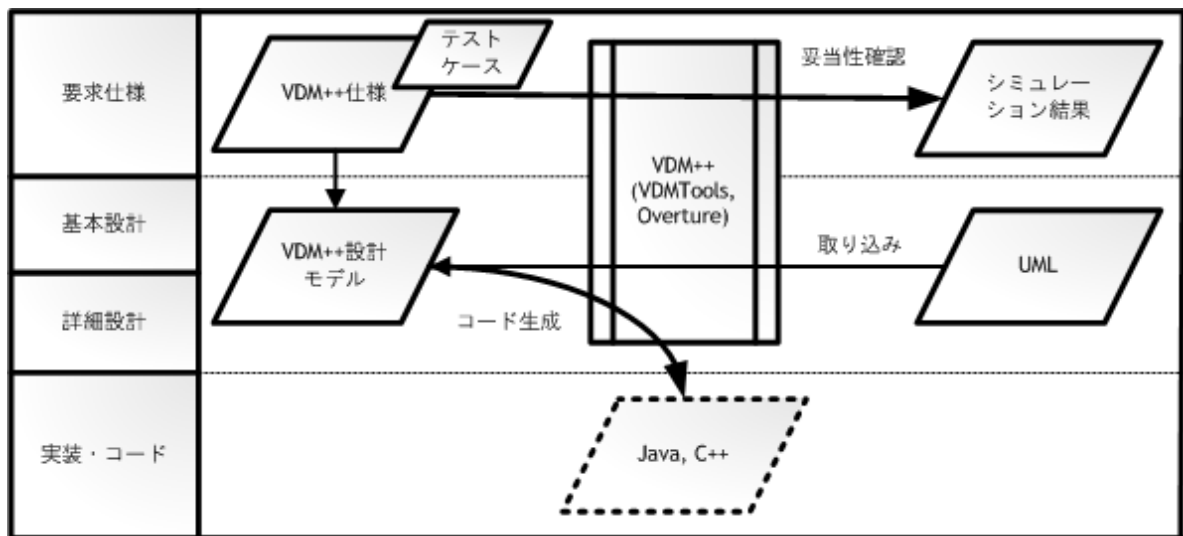


図 7-10: VDM++の開発プロセスにおける位置づけ

第3部 技術編

概要

第3部は、主に開発技術者を対象として、フォーマルメソッドを適用する上で技術やスキルの面で障害となる部分で、既存の文献ではあまり扱われないものを対象に、考え方やノウハウをまとめる。本パートは、本文中などで示す既存の参考書を理解した上で、それらと併用して読むことを想定する。本パートは、既存の参考書で扱われる個々の要素技術の知識を想定して、それらを基盤とした広い視点からの技術的なノウハウや考え方などを示すものである。

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	<ul style="list-style-type: none">● 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、ポジションペーパー、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	<ul style="list-style-type: none"> ● 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。 ● 要求からシステム的设计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	<ul style="list-style-type: none"> ● 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	<ul style="list-style-type: none"> ● 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。

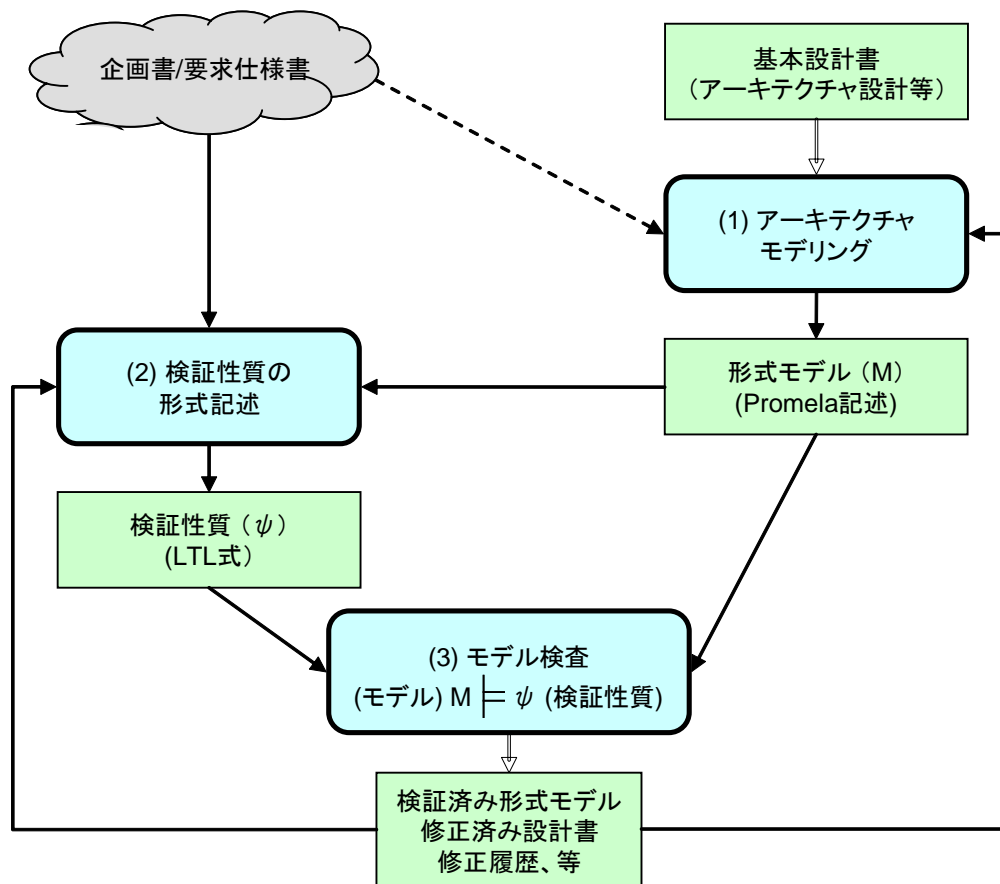


図 8-1:コンポーネント駆動パターンの概要

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。

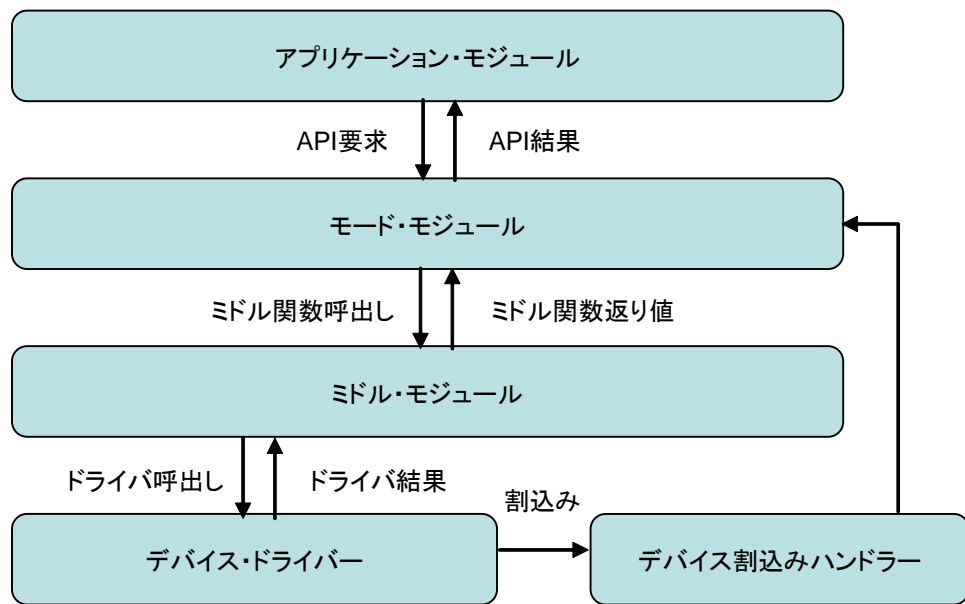


図 8-2:システムの構成要素と相互作用の例

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。

モデル形式記述の手順

(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

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

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

active proctype <状態遷移表 1>() {
  mttype state;
  mttype event;
  do ::<入力チャンネル> ? event ->
    if
      ::(event == Event1) ->
        if
          ::(state == State1) -> state == <遷移後の状態>;
                                <出力チャンネル>! <送信イベント>
          ::(state == State1) -> state == <遷移後の状態>
                                <出力チャンネル>! <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り返し、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		$\square(\text{request} \rightarrow \langle \rangle \text{response})$
	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 \square は重要である。 \square 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 が成立した時、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		$\square \neg(\text{critical1} \wedge \text{critical2})$
	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		$(\square \neg \text{pre-condition}) \vee (\neg \text{pre-condition} \text{ U second})$
	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		$\square \langle \rangle \text{condition}$
	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		$\langle \rangle \square \text{condition}$
	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を行うが、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

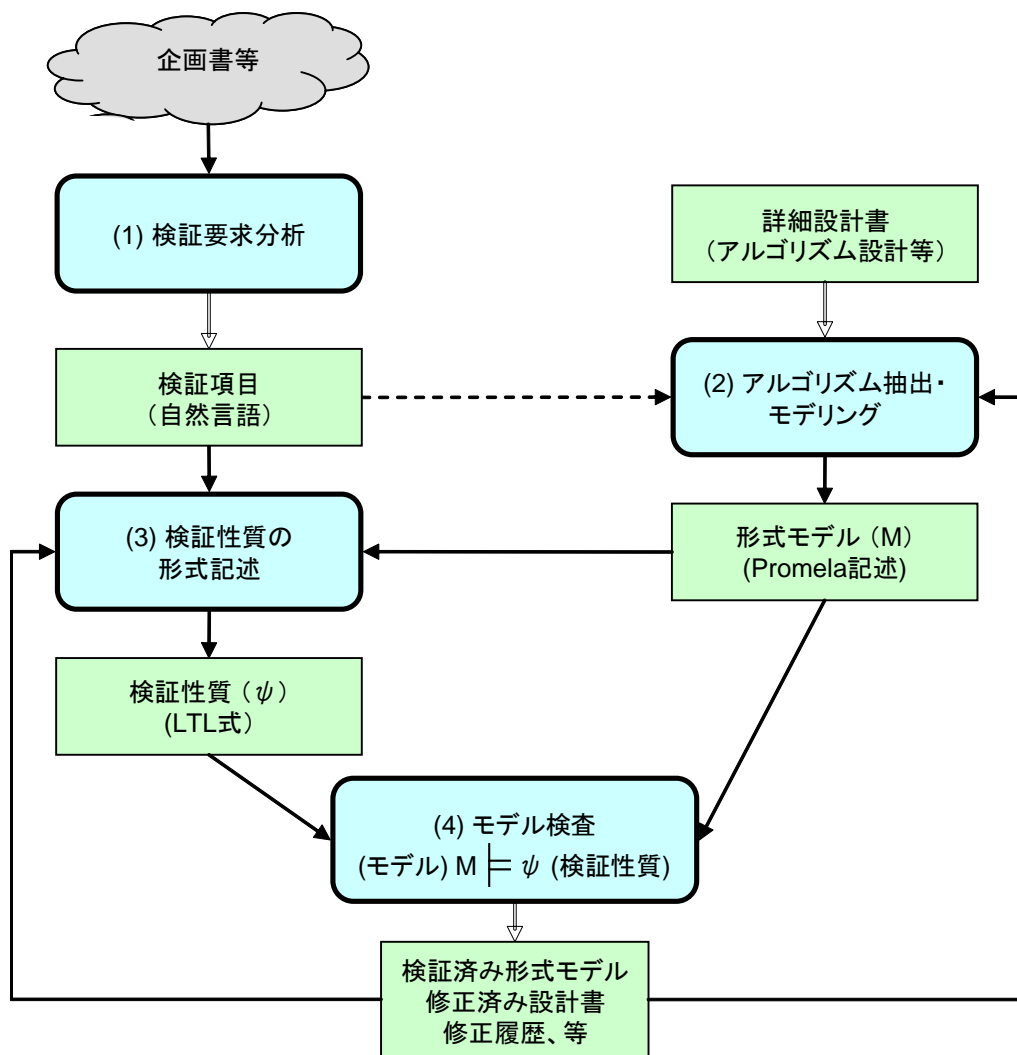


図 8-4: アルゴリズム駆動プロセスの概要

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。

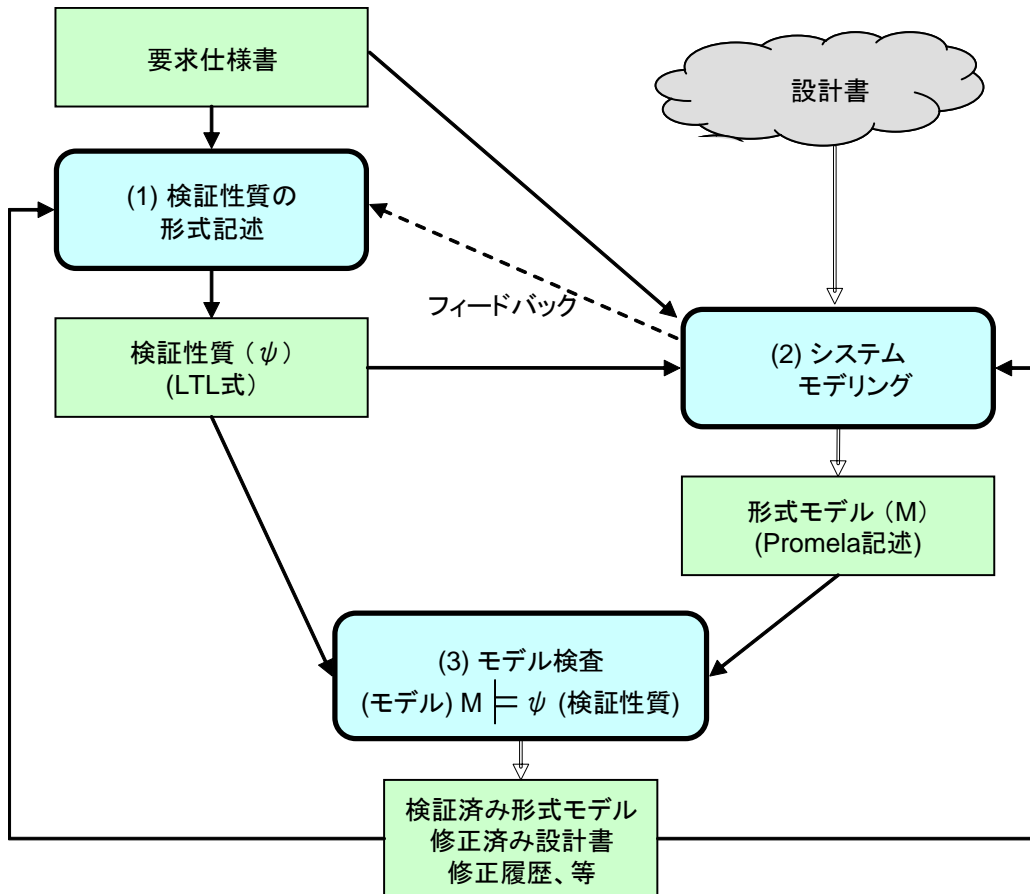


図 8-5: 要求性質駆動プロセスの概要

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る利用が想定される。

9. モデルの抽象化と状態爆発の対策法

本章では、状態爆発の全体像を示し、その原因と解決策について具体的な方法を例示することにより、状態爆発を回避する方法を示す。本章の概要は以下のとおりである。

対象読者	開発技術者、(開発プロジェクト管理者)
目的	モデル検査における状態爆発の技術的問題点を示し、その解決策として、モデルの抽象化等の対策法を示す。特に、既存の書籍ではあまり書かれていない、Promela のコードレベルでの具体的な対策について示す。
想定知識	SPIN の入門書などによりモデル検査の入門知識を持つソフトウェア技術者。
得られる知見等	<ul style="list-style-type: none">● 状態爆発の問題点と原因の具体例● 抽象化の概念と留意点● 状態爆発の対策の全体像● Promela コードレベルを中心に具体的な対策法

9.1. 状態爆発の問題点

モデル検査における技術的な制約として、「状態爆発」と呼ばれる状況で、自動検査が長時間終了しない問題が発生することがある。大規模なソフトウェアの詳細設計をそのままの詳細度でモデリングし、モデル検査を行うと、状態爆発を発生する可能性が高い。

SPIN の場合、モデルの状態は、大まかに変数の値、並行プロセスのステートメントの実行位置の組合せ、通信チャンネルのバッファ内の値などの組合せによって決まる。Promela のコード量が少なくても、これらの組合せが莫大になればすぐに状態爆発を発生するため、必ずしもコード量だけからでは判断が出来ない。

例えば、以下のような単純な2つの並行プロセスを考えた場合、SPIN においては、それぞれのプロセスの状態遷移は、非同期に並行実行されるとみなされる。

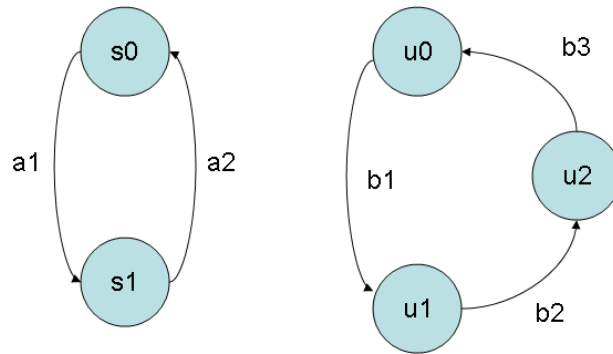


図 9-1: 2つの並行プロセスの状態遷移図(例)

プロセスが非同期並行実行されると、各プロセスの実行ステートメントは、任意の組合せで実行(インターリービング)されるため、全体のプロセスはその組合せにより状態が増大する。上記の2つの並行プロセスから構成される全体の状態遷移システムは、下図のようになる。

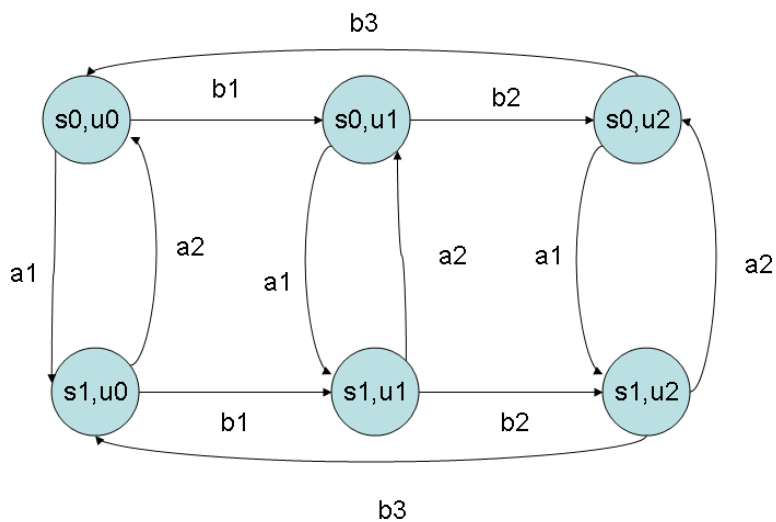


図 9-2: 非同期並行実行による状態遷移図

一方、変数の数が増えれば、変数を取りうる値の数を変数分掛け合わせた変数の数の指数のオーダーで状態数が増大する。例えば、整数 (2^{32}) の変数が n 個使われると、大まかに $(2^{32})^n$ のオーダーで増加する。

以上のような指数爆発が発生するようなモデルは容易に状態爆発が発生し、モデル検査が終了しない。このような状況に対処するために、モデルを抽象化することで、状態爆発を回避する必要がある。ただし、モデルの抽象化は、状態爆発を回避することだけが目的ではない。それについて以下の節でまとめる。

9.2. 抽象化に関する留意点

モデルの抽象化は、検証したい性質などのある観点に注目して、それに無関係な情報を捨象

することである。例えば、簡単な例で言えば、システムのうち検証したいコンポーネントと無関係なコンポーネントを除外することは一つの抽象化である。また、もう少し難しい例を挙げると、ソフトウェアのうち、何を実現するか(What)を表す「要求」に注目して、機能をどのように実現するか(How)を表す実現方法を、取り除くことも抽象化である。また、ある制御アルゴリズムにおいて、注目したデータに影響を与えるステートメントや制御構造のみを抽出するスライシングと呼ばれるものも、抽象化である。

抽象化は、状態爆発を回避することなどを含め、以下のような目的に分けられる。

目的	内容
状態爆発の回避 (技術上の制約回避)	検証性質に関係の無いモデルの要素を捨象することで、状態数を削減する。
検証の効率化 (作業コストの低減)	検証したい内容に関係の無い部分を捨象して、作業コストを低減する。
ユーザ要求レベルの記述	実装に依存しないユーザの要求の視点に近いレベルで性質を記述し、要求とのギャップによる不具合の混入を防止する。

プログラミングでは、あまり抽象化の概念は出てこないため、習得の難しい概念であるが、状態爆発という技術的制約を回避するだけでなく、問題の本質を捉え、ユーザ要求など上位の概念を記述することで、本来の目的に近い観点から検証を行うことによる効果が期待できる。

状態遷移システム(モデル)の抽象化の具体例を示すために、非常にシンプルな以下の例を考える。

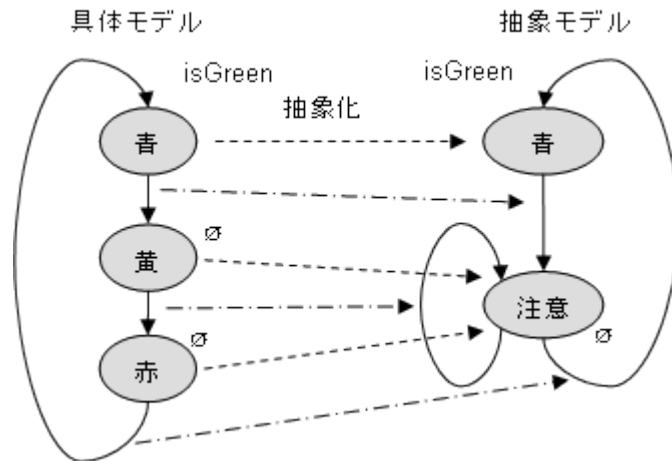


図 9-3: 交通信号モデルの抽象化(例)¹⁵⁰

状態遷移システムにおける抽象化は、状態と遷移の縮退として捉えられる。図 9-3 の左の具体モデルは、交通信号の「青」「黄」「赤」の状態遷移を表している。それを抽象化した右側の抽象モデルは、「黄」「赤」の2つの状態を「注意」という1つの状態に抽象化し、それに対応して状態遷移を修正したものである。

モデル検査を行う場合、モデルの抽象化において、検証したい性質が保存されるかどうかが必要になる。抽象化したモデルを検証する際には、通常、以下のようなことが期待される：

- 抽象化の健全性
モデル検査により抽象モデルで成立つことが示された性質は、具体モデルでも成立つ。
- 反例の保存性
モデル検査により抽象モデルで反例が検出された場合、具体モデルでも反例が存在する。

上記のモデル抽象化について以下のような性質について考える。

「いつでも信号は、いずれ「青」以外に切り替わる。」 (9-1)

図 9-3 において、状態「青」でのみ成立つ命題を `isGreen` とすると、この論理式は、以下のよう記述される。

$\square \Leftrightarrow \text{isGreen}$

この性質は、抽象モデルで成立ち、具体モデルでも成立つことが、簡単な思考によりすぐ分か

¹⁵⁰ 田辺、高井、高橋：抽象化を用いた検証ツール，コンピュータソフトウェア，Vol. 22，No. 1，pp. 2-44，2005.

る。

一方、以下のような性質は、具体モデルでは、状態「注意」への自己ループにより無限遷移し、「青」に遷移しない場合もあるため、具体モデルで成立ちませんが、抽象モデルでは成立つことが分かる。

「いつでも信号は、いずれは「青」に切り替わる。」(9-2)

このようなことから、例のモデルの抽象化では、少なくとも、「抽象モデルに反例がある場合、具体モデルでも反例がある。」という性質は成り立たない。

一般に、抽象化の健全性が成立つためには、以下のような3つの条件が成立てば良いことが証明されている¹⁵¹。

図 9-4: 抽象化の健全性の条件 (言葉による説明)

抽象化の健全性の条件

- (1) 具体モデルの初期状態を抽象モデルに写像した状態は、抽象モデルの初期状態集合に含まれる。
- (2) 具体モデルで遷移関係のある2つの状態は、抽象モデルに写像した先でも、遷移関係にある。
- (3) 具体モデルの状態で成立つ命題全体は、抽象モデルに写像した状態で成立つ命題全体と一致する。

この条件が成立てば、特定の時相論理式のクラス(CTL*)の任意の論理式について、抽象モデルで成立つことは、具体モデルでも成立つことが示されている^{152, 153}。条件の正確な定義は、それらの文献を参考にされたい。

健全性の条件が成立っていても、反例の保存性は一般には成り立たない。その例が、数式(10-2)の例が示している。抽象モデルで反例が見つかった場合、通常、以下のような対処を行う。

- (1) 抽象モデルの反例の実行系列から、具体モデルの実行系列で反例になるか分析する。
- (2) 抽象モデルの反例から、具体モデルの実行系列が反例ではない時(偽反例)、偽反例を除くように抽象モデルを修正する。

¹⁵¹ 正確な条件は、Clarke, E., Grumberg, O. and Peled, D.: Model Checking, MIT press, 1999 の Theorem 16 CTL*の健全性に示されている。ここでは、直感的な表現で条件を示している。

¹⁵² 状態遷移システムに関する等価性の関係として双模擬関係(Bisimilar relation)がある。この関係が成立つ状態遷移システムは、見かけ上区別できないため、抽象化の健全性も成立つ。

¹⁵³ 抽象モデルと具体モデルが双模擬関係にあるとき、健全性と反例の保存性の両方が成立つ。Blackburn 他編, Handbook of Modal Logic, Theorem 4.3

モデル検査における抽象モデルと具体モデルの関係は以下のように分類される。

表 9-1: 抽象モデルと具体モデルの関係と用途

分類		説明	用途
(1)	(A) 健全性	抽象モデルで検証した性質は、具体モデルでも成立つことが言える。	性質の保証に有用
(2)	(B) 反例の保存性	抽象モデルで反例がある場合、具体モデルでも反例が存在する。	不具合検出に有用
(3)	(A)かつ(B)	—	上記両方に有用
(4)	(A)または(B)が経験的に成立つと分かる	—	実際この場合が多い

分類(1) は、抽象モデルである性質が検証されれば、元の具体モデルでも成立つことが保証されるため有用ではあるが、実際には、健全性を示すことが難しい場合が多い。また、健全性の条件を満たすためには、状態数をあまり減らすことができず、状態爆発の回避という観点で効果が低い場合が多い。分類(2)は、抽象モデルで反例が見つかった場合、具体モデルでも反例があることが示される。これも実際には、抽象モデルで反例が存在しても、具体モデルでは性質が満たされる場合があり、そのような反例を「偽反例」と呼ぶ。

分類(3)は、検証性質に関して、抽象モデルと具体モデルが同じであることを示しており、非常に強い関係にあるが、実際に適用できるのは稀である。現実的には、(4)の場合が多く、アドホックに検証に関係ない部分を捨象する方法が使われることが多い。この場合、抽象モデルの検証結果と、具体モデルの性質が厳密には保証されないが、その場合でも反例発見やソフトウェアの品質に対する確信度を向上させるという点で効果が得られる。

9.3. 抽象化および状態爆発の対策の全体像

モデリングのプロセスに従い、抽象化と状態爆発の対策法について全体像を示したものが図 9-5 である。

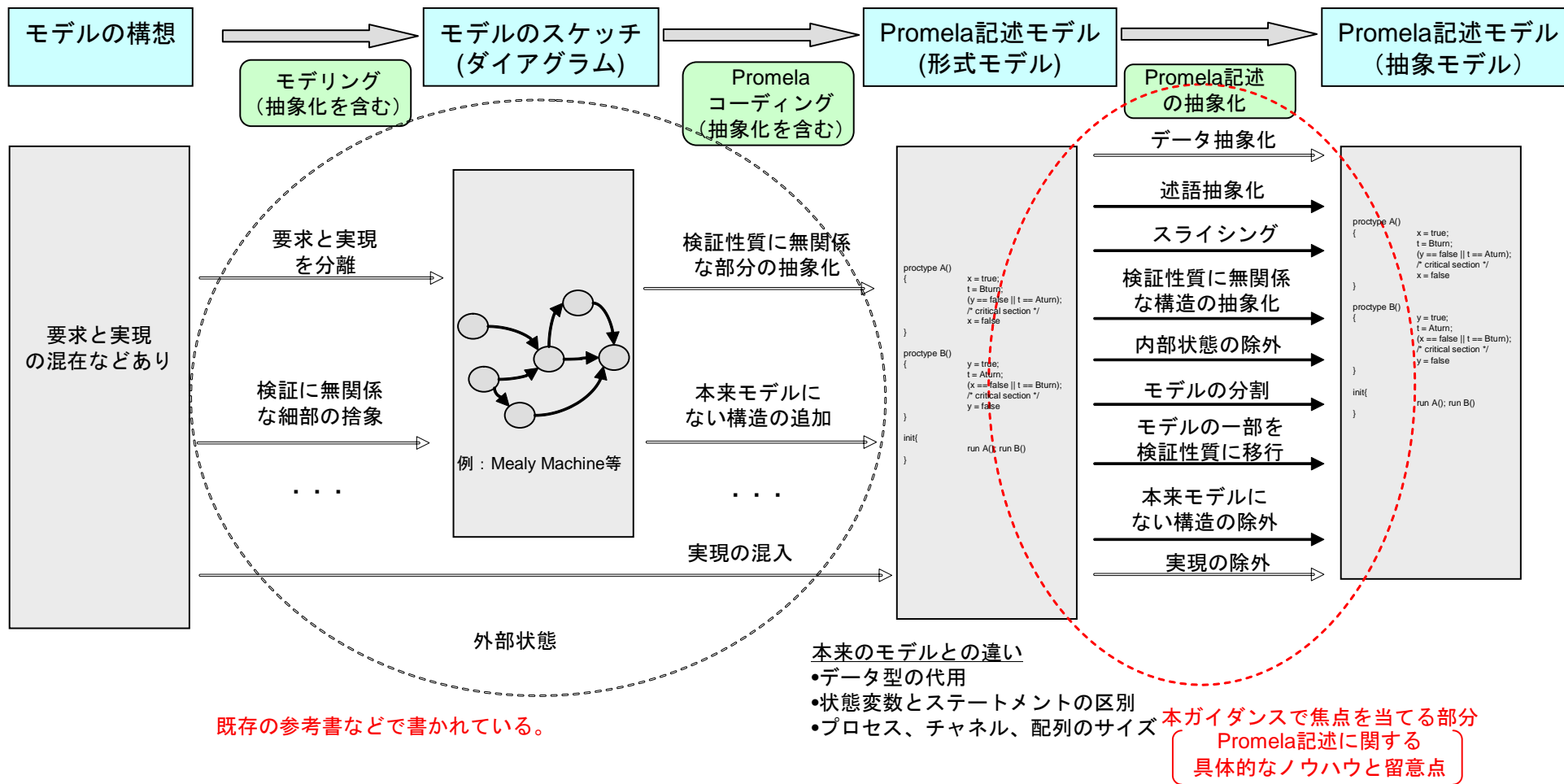


図 9-5: モデルの抽象化と状態回避策に関する全体像

抽象化のプロセスは、モデルの構想から始まり、ステートマシン図などのダイアグラムを用いてモデルのスケッチを行い具体化させる。さらに、Promela 等のフォーマルメソッド記述言語を用いて厳密なモデルを定義する。これに対して、状態爆発等の問題が発生する場合、次節に示す対策法を用いて抽象モデルを作成する。

モデルの構想からダイアグラムによる表現の段階で、検証範囲に入らないコンポーネントを除外したり、要求と実現の分離をおこなうなどアドホックな抽象化を行う。ダイアグラムから形式モデルの記述においては、具体的なデータ型の定義やチャンネル通信のセマンティクスを決定する。これにより、ダイアグラムのレベルでは曖昧であった意味を厳密に規定する。

次節では、Promela 形式モデルから抽象化された形式モデルへのプロセスにおいて、Promela のコードレベルで具体的な対策について示す。

9.4. Promela 記述に関する状態爆発の対策

この節では、Promela により記述された形式モデルに関して状態爆発を回避するためのコードレベルでの具体的な対策法を示す。

Promela の構成要素に基づき以下のレベルで対策法を示す。

(1) 並行プロセスに関する対策

検証性質に影響を与えない並行プロセスを統合、除去することで状態爆発を回避する。

(2) 変数に関する対策

不要な変数や変数の使い方の不注意による状態爆発を回避する。

(3) ステートメントに関する対策

内部状態の抽象化などにより状態爆発を回避する。

(4) モジュール分割等に関する対策

検証に無関係なモジュールの除外などにより状態爆発を回避する。

9.4.1. 並行プロセスに関する対策

9.4.1.1. 問題の原因

第9.1節に示した並行プロセスの非同期実行により、各プロセスの実行ステートメントがインターリーピングされ、プロセス数の組み合わせのオーダーで状態爆発が発生する。一般に、状態数 N のプロセスと状態数 M のプロセスが非同期平行動作すると、 $N \times M$ のオーダーの状態からなる積状態遷移システムができる。並行プロセスが増えると、プロセス数の指数オーダーで爆発する。このようなことから、プロセス数の削減は状態爆発抑止に非常に効果大きいことが分かる。

9.4.1.2. 対策

9.4.1.2.1. メッセージ送受信が主体のプロセスを統合

メッセージの送受のみや単純なフィルタのみを行うプロセスを、複数のプロセスで行うことがよくあるが、これらは必ずしも検証上は別プロセスにする必要がない場合が多い¹⁵⁴。文献に書かれている例を参照するとよい。メッセージの送受に複数のプロセスを利用している場合、まずこれをより簡潔に記述できないかを考えてみることは検討に値する。

ブルーレイディスクのケーススタディの場合、アプリケーションに対して、ディスク操作要求を任意の順序で、任意回実行するユーザプロセスは、その要求を受け取るアプリケーションのプロセスと統合できる。

9.4.1.2.2. 変数値管理のプロセスを除去

オブジェクト指向モデリングの影響などをうけて、1 つあるいは複数の変数値を管理するだけのプロセスをモデルに含める場合がある。しかし、モデル検査の立場から言うと、これは避けた方がよい。このような場合は、通常、プロセス間で共有されるグローバル変数を用意して、それを、それぞれのプロセスが必要な時点でアクセスする形でモデル化する方が不要な状態を増やさずに済むのが普通である。

9.4.1.2.3. 同期通信型のプロセスの統合

シーケンス図で書いたときに「縦線」になるものをそれぞれ別個のプロセスとしてモデル化してしまう場合がある。例えば図 9-6 のようなシーケンス図がある場合、A は B の返却値があるまで実質的に動作を停止し、B は C の返却値があるまで実質的に動作停止するというのであれば、A, B, C をそれぞれ別プロセスにする必要はなく、全体を 1 つの逐次的なプロセスとすればよい。

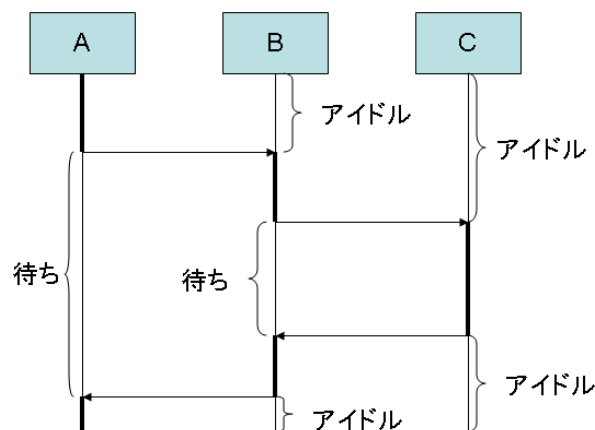


図 9-6: 実質的に逐次的な処理のシーケンス図

¹⁵⁴ The SPIN MODEL CHECKER, 第 5 章“Sink, Sources, and Filters”

これに関連して、特に SPIN/Promela の場合に、慣れないうちは、単なるサブルーチンを独立プロセスにしてしまう場合があるが、これは Inline 定義を利用すべきである(なお inline 定義については必要な記述上の注意などもあるので例えば文献の inline 定義の注意を参照のこと)。

9.4.1.2.4. 検証性質に基づくプロセスの統合

これは上述の(1) から (3) のまとめであるが、これら以外にも、検証項目(検証式で表されるもの)との関係では関係ない部分は 1 つのプロセスと考えることができる場合もある。場合によっては、検証式自体を書きかえることで、そのことが可能になる場合もある。

9.4.1.2.5. 多重プロセスの試行的調整

この方法は、上述の(1)から(4)とは異なり、主として不具合発見の目的でのみ有効な方法である。しかし、実際の検証ではモデル上の変更量が小さくて済み、またもとのモデルに戻すのも簡単であることから、しばしば利用される。例えば、顧客が N 人いて、それぞれ別個にシステムにアクセスしてくる、という状況をモデル化するとき、それぞれの顧客を別プロセスにする必要があるとする。このとき、小さい N で示すべき性質が成立するかをまず試してみて、次第に N を必要な大きさまで大きくしていく、という手法をとることができる。

通常、 N が m のときに反例が見つかったとすると、 N を $m+1$ にしたときもその反例は有効である。特に最後に追加された $m+1$ 個目のプロセスが完全にインターリーブで動くとなると、たまたまそのプロセスが動く順番がまわってこない場合を考えられるが、これは N が m のときの動作とほぼ同一と考えられる。したがって、多くの場合、 N が m のときの反例は、そのまま N が $m+1$ のときの反例にもなる。しかし逆に N が m のときに反例が出なかったからといって N が $m+1$ の時にも反例が出ないことにはならないのが通常である。新しく加わったプロセスのせいで別の処理の組合せが発生するので、そこで反例が発生する可能性があるからである。

このことから、この N を小さくする方法は、特に、不具合発見の目的に有効な方法であるということになる。

ただ、ここで注意しなければいけないことは、反例についても、 N を大きくしたときに必ずそのまま保存されるとは限らないということである。一般にこの手法を使うときには、小さな N で反例が出たら、反例が出た原因を調べて、大きな N ではどうなるかを推定することが必要である。

9.4.2. 変数に関する対策

9.4.2.1. 問題の原因

変数ごとに取る値のサイズを掛け合わせたオーダーで状態数が増加する。したがって変数の数の指数オーダーで急激に増加する。

たとえば、Promela で、以下の処理を考える。

```
bool flag;
active proctype pr1() {
...
}
```

このとき、pr1 の中に記述された表面的なコードは同一でも、モデル検査系が調べる状態は、大域変数 flag が true のときと false のときで別の物である。したがって、flag の値が pr1 の中で両方の値をとりうるならば、Promela コード上に表面的に見える状態数の 2 倍の状態をモデル検査器は調べる必要がある。

ここでもし変数 flag が不用ならば、それを消してしまえば、別の物と扱われていた 2 つの状態は縮退して 1 つになる。これによって、大まかに、状態数は半減することになる(実際には pr1 の中での flag の使い方にも依存する)。

基本的に変域の大きさ R の変数が V 個あるときには、モデル検査器内部には R×V 個の状態が生じることがあると考えるのが良い。従って両者をなるべく減らすべきである。表 9-2 に Promela の場合の型と変域の大きさの例を示す。他のモデル検査系を利用する場合も同様に考えることができる。

表 9-2:型と変域の大きさ

Promela の型(あるいは変数宣言)	変域の大きさ
mtype = { red , blue, yellow }	3 + 1 (例外値)
bool	2(True or False)
Boolean a[5]	2 の 5 乗=32
byte	2 の 8 乗=256
short	2 の 16 乗=65536
Int	2 の 32 乗≒43 億
Typedef Record { byte b[3] short f }	256 の 3 乗×65536≒280 兆

9.4.2.2. 対策

9.4.2.2.1. 必要最小限のサイズの型を用いる

Promela なら、なにげなく int 型を使っている部分があったら、mtype や, byte, bit で十分ではないか、あるいはせめて short ではどうかを検討することは有効である。

9.4.2.2.2. 構造型の定義はより単純な構造を用いる

typedef で定義されたレコード型などを利用している場合、表 9-2 からわかるように、全体の変

域のサイズは各フィールドの変域のサイズの積となる。従って、不要なフィールドを削ることを考えること、あるいはそのフィールドの型をよりシンプルなものにできないかを検討することも有効である。このことは、その型を利用する変数の変域を小さくすることに貢献する。

9.4.2.2.3. 検証に関係ない状態変数は削減する。

実装に整合させることを意識して、不要な変数を入れてしまうことは考えられる。しかし、それが検証に関係がなければ、入れる必要はない。

なお、変数に関係あるかないか正確に判定するには影響範囲をたどっていくCOI(Cone of Influence)と呼ばれる方法があるが、まずは、明らかに関係ないものを排除することを考えるべきである。なお、COIについて詳しくは専門的になるが文献¹⁵⁵の 13.1 節を参照すると良い。

また、使用する変数との関連を考慮して、検証項目自体を変形する(分割するなど)ことも検討対象となり得る。

9.4.2.2.4. グローバル変数をプロセスローカル変数にする。

検証に関係の無い大域変数を局所変数にすることで、局所変数はスコープの外で使われないことが分かるため、SPINの処理系で最適化される¹⁵⁶。

9.4.2.2.5. モニタリング用変数、デバッグ用の変数は削除する。

通常のデバッグ時に状態をモニターする変数を挿入する感覚で、モニタリング用の変数を挿入している例があるが、このような変数は状態数を不要に増大させる可能性があるため、避けるべきである。

文献¹⁵⁷第 5 章の"COUNTERS"には、このような例として実行ステップを数えるカウンターを除去する例が説明されている。

9.4.2.2.6. データ抽象化

データの変域を小さくする抽象化法としてデータ抽象化(データマッピング)が代表的である。これは、例えば、整数として宣言された変数が、検証においては、マイナス、0、プラスに注目すれば十分である場合など、整数をこれらの3つの値に縮退させる方法である。

データ抽象化については以下のような文献で豊富に説明があるため、ここではそれらへのポイントを示す。代表的な参考書に以下のようなものがある：

- 文献¹⁵⁸の 9.2.1 節
- 文献¹⁵⁹の 2.3 節および 2.6 節

¹⁵⁵ Clarke, E., Grumberg, O. and Peled, D.: Model Checking, MIT press, 1999.

¹⁵⁶ 萩谷, 吉岡, 青木, 田口, SPIN による設計モデル検証, 2008

¹⁵⁷ Holzmann, G.: THE SPIN MODEL CHECKER, Addison Wesley, 2004.

¹⁵⁸ 中島震: SPIN モデル検査 - 検証モデリング技法, 近代科学社, 2008 .

¹⁵⁹ 田辺, 高井, 高橋: 抽象化を用いた検証ツール, コンピュータソフトウェア, Vol. 22, No. 1, pp. 2-44, 2005.

さらに、データ抽象化のいろいろな種類については、文献の 9.2 節に詳しい。

9.4.2.2.7. 述語抽象化

データ抽象化を拡張したより専門的な手法として、述語抽象化という手法がある。述語抽象化は、複数の変数の組合せに対して、それらを単純な構造に抽象化するものである。これらについても文献、などに書かれているため参照すると良い。

9.4.2.2.8. キュー、スタックのサイズは小さいものから試す

これは上述の 9.4.1.2.5 節と同様に、主として不具合発見の目的の際のみに使用できる方法である。キューやスタックが配列として実装されている場合、その配列のサイズを小さく設定してとりあえず反例が出ないかどうかを確認し、次第に本来のサイズまで広げていくという方法である。これも 9.4.1.2.5 節と同様に、キュー、スタックなどを小さくして検証した結果反例が出たからと言って、大きい場合でも反例が出るとは一般には言えない。反例の出た原因から推定することが必要になる。

なお、例えばキューに関して、9.4.2.2.6 節のデータ抽象化の応用として、実際のキューを作らずに、例えば「キューに空きがあるか満杯か」だけを表す二値変数だけを用意することでも検証目的によっては十分な場合がある。実際にこの方法でデッドロックを検出した例も存在する。本手法は 9.4.2.2.1 節から 9.4.2.2.6 節と比較すると最後の手段であるので、9.4.2.2.1 節から 9.4.2.2.6 節までを先に検討するとよい。

● 注意点

1. ローカル変数への着目

変数を削減する、あるいは変数の変域を削減することを検討する際に、ローカル変数をまず見直してみることは有効である。

ローカル変数は、モデル記述上一つであっても、実行時にそれぞれの実行主体ごとに別空間を利用する場合があるので、実際数は増幅する場合があるからである。ローカル変数がどのようにインスタンス化されるかを正しく理解した上でその削減を検討できれば一番ではあるが、そうではなくても削減対象としてローカル変数に着目することは有効である。

2. 変数削減時の処理の変形方法

上記(2-3)のように変数を削減しようとする、その変数を参照して処理を変えている部分をどのように変形すれば良いかが問題になる。例えば、Promela 記述内に次のような部分があるとすると、

```
if
  :: flag = true -> ACTION1;
  :: flag = false -> ACTION2;
```

ただし、ACTION1, ACTION2 は実際にはそれぞれ何らかの処理であるとする。このとき、flag が全体から見ると検証には関係がないので削除するということになったとすると、これをどのように変形すべきか、という問題になる。

この一般的な方法は、次のように ACTION1 または ACTION2 への非決定的な遷移にする方法である。

```
do
  :: ACTION1;
  :: ACTION2;
```

この方法で反例が見つかったからといって、それがもとのモデルでの反例になるとは限らない。もちろん本当に flag が検証項目と関係がないならば、反例にも関係ないはずであるが、一応、この方法で発見された反例がある場合は、それを解析してもとのモデルでも反例となるかを調べる必要がある。

9.4.3. ステートメントに関する対策

9.4.3.1. 問題の原因

Promela で記述したモデルでは、1つの文が1つの状態を構成する。複数の文が1つのステップとしてアトミックに実行されたと見なしても問題が無い場合、複数の文を **atomic** を用いて1ステップと定義した方が、他のプロセスのステップとの間のインターリーピングが発生しない分状態を抑えることができる。

処理の遷移の途中で中間状態があり、本来1ステップで良いところが2ステップになっているとする。そして、そのプロセス以外にも他にいくつものプロセスがあるとする。すると、中間状態があるがゆえに、インターリーブの考え方のもとで他のプロセスとの間の相互作用の組合せが膨大に増えることになり、結果的に、モデル検査対象の経路数が巨大化する。

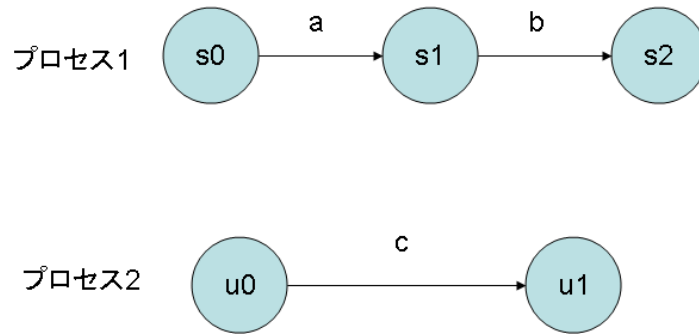


図 9-7: 冗長な中間状態 s1 がある場合

例えば、図 9-7 のように 2 つのプロセスがあるとする。このとき、2 つのプロセス内のアクション a,b,c のインターリーブによる実行順の組合せは、

- c; a; b
- a; c; b
- a; b; c

の三種類である。しかし、もしプロセス 1 の中間状態 s1 が冗長であるとする、これを削除して、a と b をまとめて一つのアクションとしてしまうことが考えられる(図 9-8)。

すると、a,b,c の可能な実行順の組合せは、

- c; a; b
- a; b; c

の二種類に減少する。これに応じて、モデル検査器の探索空間も小さくなる。

この例では他のプロセスは 1 つだけであり、アクションも 1 つだけを考えてが、通常は他にたくさんプロセスがあり、いくつものアクションが関係してくるので、1 つの中間状態を削除しただけでも多くの余分な組合せの排除につながる。

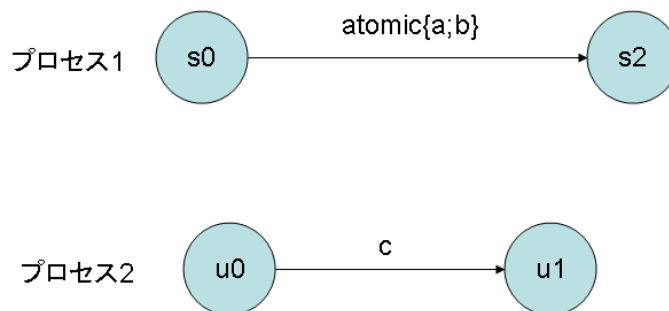


図 9-8: 状態 s1 を省いて、a;b を 1 ステップにした場合

9.4.3.2. 対策

9.4.3.2.1. 内部状態の遷移系列を 1 ステップとなるように atomic 宣言する。

Promela の場合、まとめて 1 ステップとしたい処理を atomic{} で囲うことでこれを実現できる。これにより劇的な効果をもたらす場合がある。ただし、とにかく atomic をつけると状態爆発に効果が出るということを知ると、atomic をつけてはいけない部分にまで付けてしまうことがある。1 ステ

ップにするということは、その処理が始まって終わるまでの間は他のプロセスが動けないということである。**atomic** をつける際に、そこで他のプロセスが動く場合を考えなくてよいかどうかを考慮する必要がある。

ここまでの説明は **SPIN/Promela** に沿って行ったが、それ以外のモデル検査ツールを利用する場合も基本的な考え方は同様である。

9.4.3.2.2. スライシング

ステートメント単位で、注目したスライス基準の依存関係を辿り、無関係なステートメントを除外する。スライシングについては文献¹⁶⁰に具体的に書かれているので、それを参照するとよい。また、**SPIN**では自動スライシングの機能がありオプション機能(-A)を指定することで利用できる。

9.4.4. モジュール分割に関する対策

9.4.4.1. 問題の原因

モデル全体に対して複数の検証性質を順に検証しようとする、モデルの状態数が大きすぎる場合が生じる。検証性質ごとに、必要最小限のモデルに分割して検証することが可能である。

9.4.4.2. 対策

9.4.4.2.1. モジュール分割

モジュール分割は、モデリング言語の種類によっては、強力なサポートがある。**Promela** の場合には特に言語的なサポートはないが、処理を意味的な関連度の高さとグルーピングした後、分割するのが適当である。

9.4.4.2.2. 初期状態による分割

初期状態がいくつもある場合、全体の状態の中には、特定の初期状態からしか辿りつかないものが含まれている可能性がある。このため、初期状態ごとに関係がある状態だけをモデル化することも効果がある場合がある。

また本質と関係ない初期値の未定値(例えば配列でキューやスタックを表現するときの、値の格納前の配列の値など)は確定しておくことで、不要な初期値についての計算を避けることができる。例えば、図 9-9 が左から使用するスタックを表すとして、スタックポインタより右の部分の値(図の **e1** から **e7**)は不定値の可能性がある。これを確定しておかないと、モデル検査器は **e1** から **e7** に入りうるすべての値の組合せについて調べてしまう場合がある。したがって、スタックの空の部分の初期値は何らかの値に確定しておくべきである。

¹⁶⁰磯部, 桑野, 櫻庭, 田口, 田原: ソフトウェア科学基礎, 近代科学社, 2008

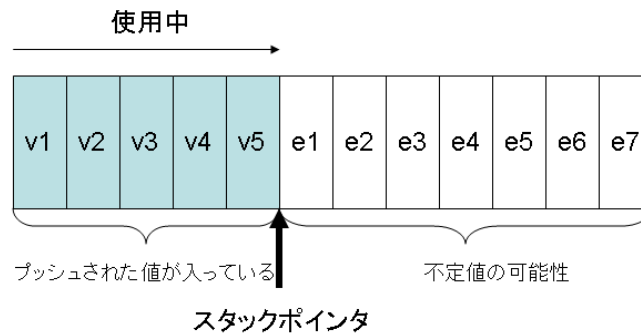


図 9-9:スタックの利用前部分の不定値

9.4.4.2.3. 検証項目による分割

検証項目が複数ある場合、それぞれに、9.4.1 節～9.4.3 節の手法の適用可能性が変化する場合がありますので、それぞれ別個にモデルを作って検証することも効果がある。

9.4.4.2.4. 例外的処理の分離

システムは通常、標準的な処理と例外的な処理を含んでいる。これをすべて一体化して検証しようとする状態数が増大することになるのが一般的である。不具合発見が目的であれば、例えばまず例外処理部分を除いたモデルで標準的な処理について検証(不具合調査)し、その後、例外処理を含むモデルを検証することも考えられる。あるいは、例外処理への脱出までのモデルと、例外処理自体のモデルを別に作り、それぞれ別個に検証する方法も考えられる。

9.4.4.2.5. 仮定を利用した検証条件の分割

性質 B を示すのに、まず A を示し、次に「A ならば B」を示すことで、結果的に B を示すという方法を使うこともできる。これも一種の分割と言えるであろう。

一部の性質は、モデル化するよりも、検証条件に仮定(⇒の前提)として与えた方が扱いやすい場合がある。例えば、公平性と呼ばれる性質(インターリーブで考えたときに、どのプロセスにもいつかは動く順番が回ってくる、という性質など)はモデル上には表現しにくい場合が多いので、これを仮定にするということはよく行われる。

ただし、検証性質にモデルの性質を入れた場合、モデル検査の状態数が大きくなるため、これは状態爆発に対する対策というよりも、モデルの記述の困難性に対する対策と言える。

● 適用上の注意点

上記の 9.4.4.2.1 節から 9.4.4.2.5 節のいずれについてもあてはまるが、分割検証できるようにするために、検証項目の記述自体を工夫する(分割するなど)も考える余地がある。通常細かい検証項目ほど、ローカルに検証できるので、検証すべき項目を細かく詳細化することも一つの方法である。

9.5. 本章のまとめ

本章では、状態爆発の問題とその対策法としてモデルの抽象化のうち、**Promela** のコードレベルでの方法を中心にまとめた。データ抽象化や述語抽象化など、一般的な手法に関しては、既存の文献に具体的に書かれているため、それらの紹介にとどめた。

本章により状態爆発の問題その解決策としてモデルの抽象化に関する全体像を把握するとともに、問題の原因の理解を通じた、実践での解決のためのヒントを示した。

第4部 付録

10. ケーススタディ1:ブルーレイディスク

10.1. 検証の概要

10.1.1. 検証対象システムの概要

検証対象システムは、再生専用の Blu-ray Disc プレイヤのディスク操作を制御するミドルウェア（以下、モード制御ソフトウェアと呼ぶ。）である。Blu-ray Disc プレイヤは、Blu-ray や CD、DVD などのディスクメディアを、プレイヤ本体の挿入口に入れ、ユーザのリモコン操作を通して入力される命令に応じて、メディアに記録されている映像の再生、停止、一時停止、早送り、スロー再生などを行う。

モード制御ソフトウェアは、図 10-1 に示す Blu-ray 系インタフェースと Blu-ray ミドルウェアモジュールで構成される。Blu-ray 系インタフェースは、Blu-ray アプリケーションと Blu-ray ミドルウェアモジュールのインタフェースであり、Blu-ray アプリケーションが要求する Blu-ray 用ドライブの制御（再生、停止等）を、Blu-ray 用ドライブの状況に応じて Blu-ray ミドルウェアモジュールに伝える。Blu-ray 系インタフェースから要求された制御命令を、Blu-ray 用ドライブの状況に応じて、ドライバを通して制御する。

CD、DVD など Blu-ray 以外のメディアについては、Red 系インタフェースを通して制御する。

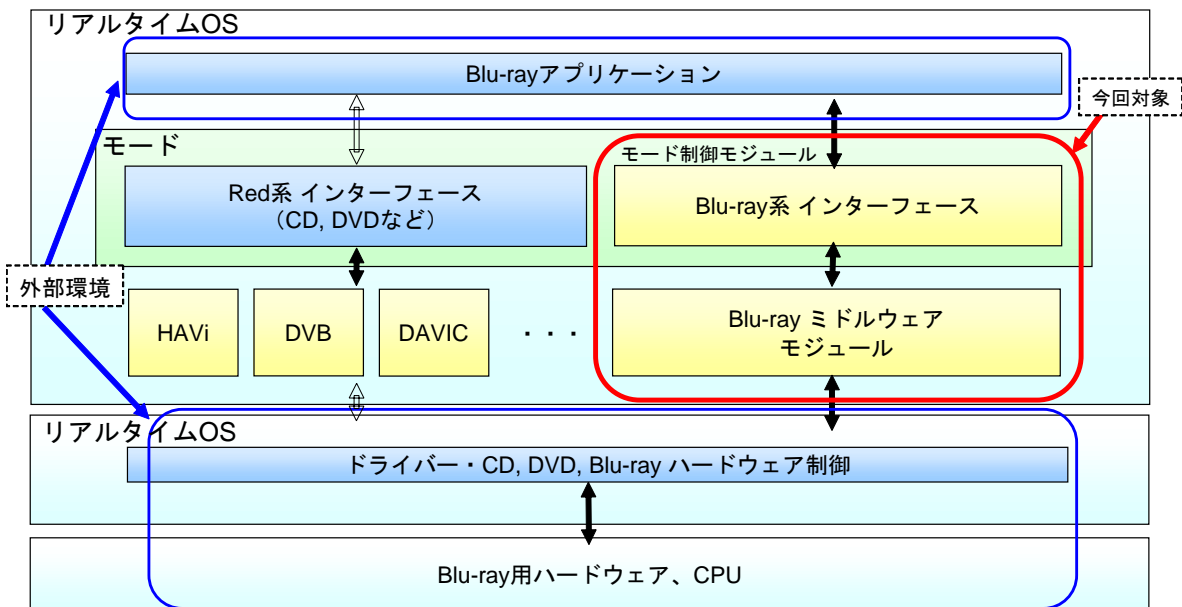


図 10-1: 検証対象システムの構成

ユーザがリモコンなどを通して Blu-ray ディスクを操作する命令は、Blu-ray アプリケーションがインタフェースとなって受け付ける。Blu-ray アプリケーションは、モード制御ソフトウェアの API を通してユーザから受け付けた命令を、モード制御ソフトウェアに伝達する。モード制御ソフトウェアは、

Blu-ray アプリケーションから伝達されたユーザの命令を受信して、ディスクが現在再生中か、停止中かなどの状態を、ドライバを通して観測し、その状態を基にドライバに受け付けた命令を伝達する。例えば、ディスクが再生中の場合には、再生命令をモード制御モジュールが受け付けてもドライバには転送しないが、停止命令を受け付けた場合、モード制御モジュールはドライバに停止命令を要求する。また、モード制御モジュールは、Blu-ray ドライブから発せられる「再生終了」や「読み込みエラー」(ディスクに傷などがついていて、読み込みに失敗した場合に発せられるエラーイベント)などのイベント割り込みを受け付けて、停止状態への移行などモード制御を行う。

10.1.2. 検証の目的と検証内容の概要

モード制御ソフトウェアはBlu-ray系インタフェースとBlu-rayミドルウェアモジュールで構成されている。Blue-rayミドルウェアモジュールは、ディスクの制御状態を細かく把握しており、一方のBlue-ray系インタフェースはBlu-rayミドルウェアモジュールを通してディスクの制御状態をより抽象的に把握する。例えば、Blu-rayミドルウェアモジュールは、ディスクの制御状態を「再生移行中」、「再生」、「早送り」、「正スロー再生¹⁶¹」などのように詳細に把握しているが、Blu-ray系インタフェースはこれらの状態を1つの「再生」状態として把握する。このように 2 つのモジュールが異なる状態把握を行っていることに起因して、状態把握の不整合が生じ、結果としてソフトウェア設計時に意図していなかった制御やデッドロックが発生する可能性がある。そこで、今回の検証では、以下の点に着目して検証を行うこととした。

- モード制御ソフトウェア内の Blu-ray 系インタフェースと Blu-ray ミドルウェアモジュールのそれぞれが把握する状態の不整合などにより、意図しない制御やデッドロックが発生しないことを確認する。

SPIN によるモデル検査では、モード制御ソフトウェアをモデリングするとともに、それ以外の部分は外部環境としてモデル化した。

検証の結果、当初の設計において、API 要求とデバイスからの割り込み処理が同時に発生した場合に、デッドロックが発生することが検出され、その修正を行うことができた。

10.2. 検証対象システムの仕様

Blu-ray プレイヤ全体のシステム構成を図 10-2 に示す。検証を行うソフトウェアは、図 10-2 中の Blu-ray 系インタフェースと Blu-ray ミドルウェアモジュールで構成されるモード制御ミドルウェアである。

¹⁶¹ 正スロー再生とは、再生方向へのスロー再生のこと。

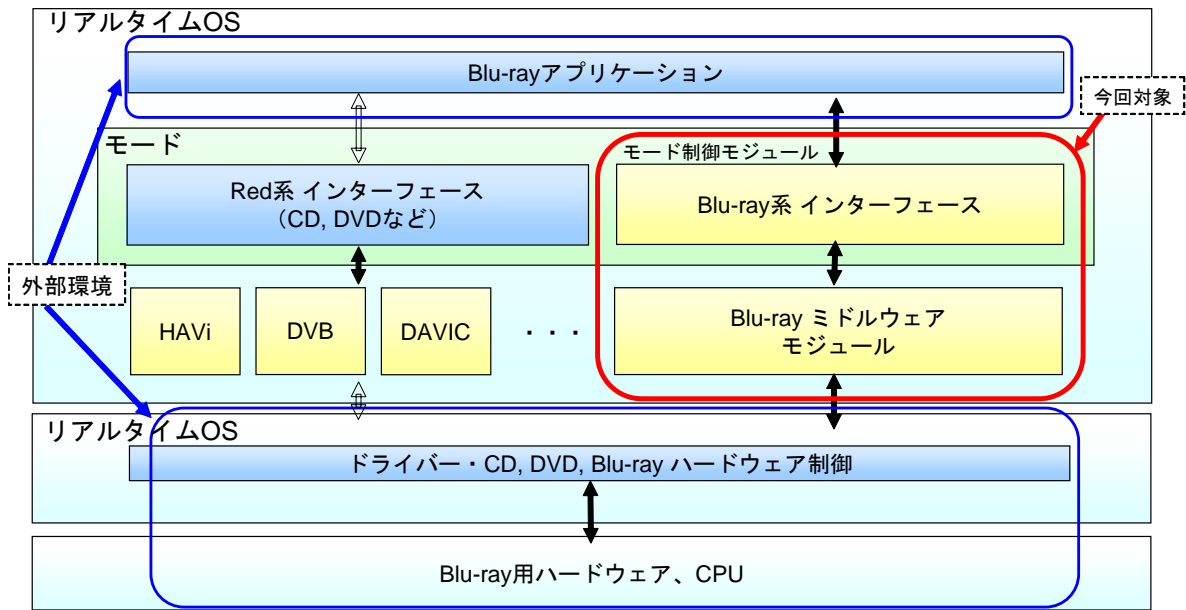


図 10-2: 検証対象システムの構成(再掲)

Blu-ray プレイヤのシステムは、ハードウェアモジュールとソフトウェアモジュールで構成されている。Blu-ray や CD、DVD などのディスクメディアを扱う Blu-ray 用ドライブ、Blu-ray 用ドライブを制御する制御プロセッサはハードウェアで構成されている。制御プロセッサとのインタフェースとして、ドライブが実装されており、ドライブを通して、モード制御ミドルウェアが Blu-ray 用ドライブを制御する。また、Blu-ray ディスクが最後まで再生された場合や、Blu-ray ディスクに傷などがありディスク上に記録されているデータの読み込みができなかった場合には、それぞれ「再生終了」、「読み込みエラー」のイベントを制御プロセッサが発生させ、割り込み処理としてモード制御モジュールに伝達する。なお、Blu-ray 用ドライブおよび制御プロセッサ以外のコンポーネントは全てソフトウェアで構成されている。

リモコン操作などを通してユーザから Blu-ray プレイヤに送信されてくる命令は、Blu-ray アプリケーションと呼ばれるスレッドで蓄積、管理される。このスレッドは複数並列処理可能である。Blu-ray アプリケーションに蓄積された命令は、モード制御ミドルウェアに送信される。モード制御ミドルウェアは、Blu-ray 用ドライブの状態(例えば、「停止」や「再生中」など)の状態に応じて、Blu-ray アプリケーションから送信されてきた命令を、ドライブを通して、制御用プロセッサに伝達する。

ディスク操作モード制御ミドルウェアは、Blu-ray 系インタフェース(以下、「モードモジュール」と呼ぶ。)と Blu-ray ミドルウェアモジュール(以下、「ミドルモジュール」と呼ぶ)で構成されている。Blu-ray 系インタフェースと Blu-ray ミドルウェアモジュールは、それぞれ Blu-ray 用ドライブの現在の状態を記憶しているが、その詳細さが異なる。例えば、Blu-ray ミドルウェアモジュールは Blu-ray 用ドライブが「再生移行中」、「再生」、「早送り」、「正スロー再生」などのように詳細に把握しているが、Blu-ray 系インタフェースはこれらの状態を1つの「再生」状態として把握する。

Blu-ray 系インタフェースは、Blu-ray 用ドライブの現在の状態を Blu-ray ミドルウェアモジュールに問い合わせを行い、Blu-ray ミドルウェアからの戻り値によって Blu-ray 用ドライブの状態を把握する。つまり、Blu-ray インタフェースは Blu-ray 用ドライブの状態を直接観測できず、Blu-ray ミドルウェアモジュールを通して把握する。

Blu-ray 系インタフェースは、Blu-ray アプリケーションが呼び出すことを想定した API (アプリケーションプログラミングインタフェース) を提供しており、Blu-ray アプリケーションがユーザから受け付けた命令に対応する API 関数を呼び出すことで、ユーザの命令を Blu-ray 系インタフェースに伝達する。Blu-ray 系インタフェースは、Blu-ray アプリケーションから呼び出された API 関数と、自身が把握している Blu-ray 用ドライブの状態に応じて、Blu-ray ミドルウェアモジュールの関数を呼び出す。なお、Blu-ray 系インタフェースと Blu-ray アプリケーションは 1 対 1 に対応し、複数のスレッドが並列で動作するようになっている。

Blu-ray ミドルウェアモジュールは、自身が把握している Blu-ray 用ドライブの状態に応じて、呼び出された関数に応じて、Blu-ray 用ドライブの制御をドライバを通して行う。

10.2.1. 検証対象システムに求められる条件（要求仕様）

検証を行うモード制御ミドルウェアに求められる要求仕様を表 10-1 に示す。検証対象のモード制御ミドルウェアが「必ずしなければならないこと」を「機能要求」、「絶対にしてはならないこと」を「非機能要求」として、分類して表記してある。

表 10-1 に示した要求仕様が、後述するモデル検査における検証内容の出発点となる。

表 10-1: モード制御ミドルウェアの要求仕様

	要求仕様
機能要求	モードモジュールは、要求された API に対して、モード状態に基づき操作可能な API については、ミドルモジュールに対して、API に対応する要求を出す。
	ドライブから割り込みが発生した場合、ミドルモジュールは、割り込みの種類に応じてミドル状態とモード状態を遷移させる。
非機能要求	デッドロックが発生しない。
	モードモジュールは、要求された API のうち、現在のモード状態で実行できない API は、ミドルモジュールに要求を出さない。
	モード状態、ミドル状態の間に不整合は生じない。

10.2.2. 検証対象システムの設計仕様

検証対象となるモード制御ミドルウェア (Blu-ray 系インタフェースと Blu-ray ミドルウェアモジュール) を中心として、その周辺のコポーネントを含めた設計仕様をまとめる。

10.2.2.1. プロセス構成

検証を行うモード制御ミドルウェアに関連するコンポーネントは、Blu-ray系アプリケーション、ドライバ、および、割り込み処理を通知するコールバック¹⁶²である。ここでは、各コンポーネントがBlu-rayディスクシステム内で実行時に起動されるスレッド数をプロセスとして考える。

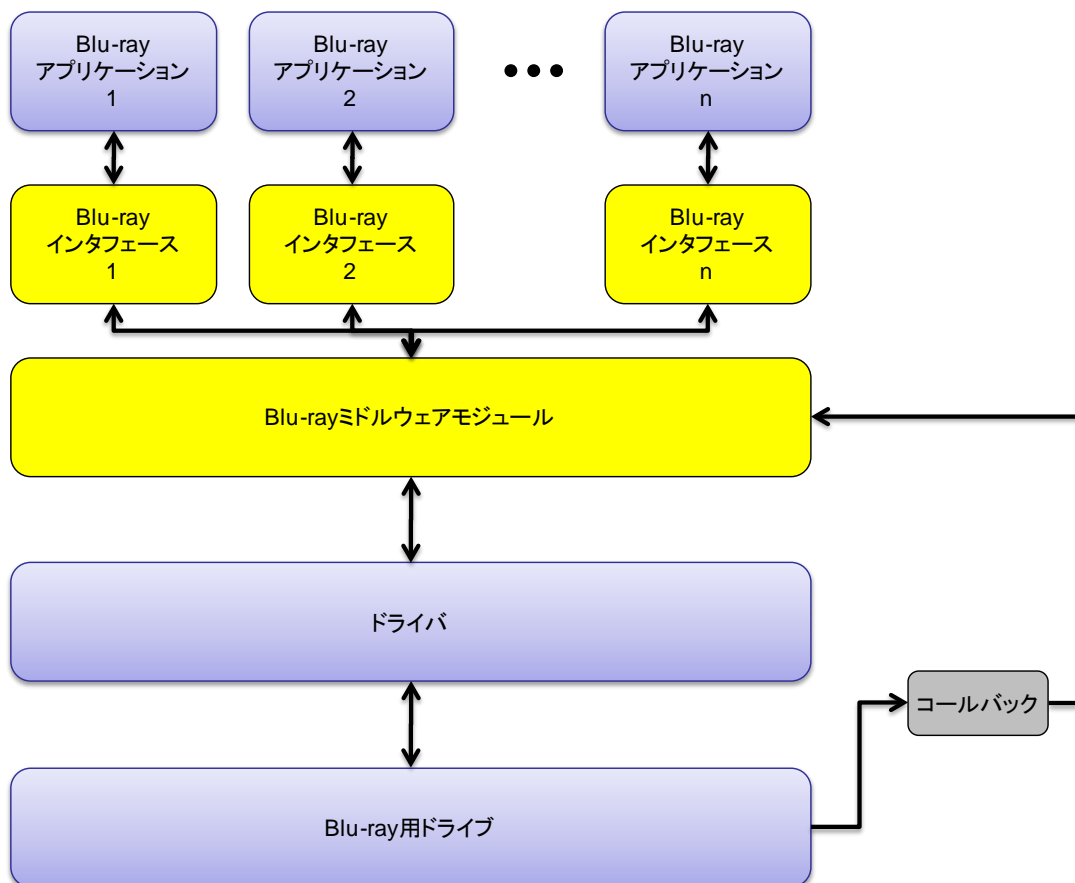


図 10-3: プロセスの構成図

Blu-ray アプリケーションは、複数のスレッドとして並列に動作するため、 n 個のプロセスとなる。モード制御ミドルウェアの一部である Blu-ray 系インタフェースは、1 つの Blu-ray アプリケーションに対応して、1 つのスレッドが動作する。従って、Blu-ray 系インタフェースのプロセス数は、Blu-ray アプリケーションのプロセス数と同じ n 個である。1 つの Blu-ray アプリケーションプロセスとそのプロセスに対応する Blu-ray 系インタフェースがメッセージ通信を行うことで、ユーザ操作内容(命令)のモード制御ミドルウェアへの伝達や、Blu-ray 用ドライブなどのハードウェア部分の割り込み処理の結果等の Blu-ray アプリケーションプロセスへの伝達が行われる。

Blu-ray ミドルウェアモジュールは、Blu-ray ディスクシステムに 1 つ付属する Blu-ray 用ドライブをドライバを通して観測する。このため、Blu-ray ミドルウェアモジュールのプロセスは、システム内

¹⁶² プログラム中で、呼び出し先の関数の実行中に実行されるように、あらかじめ指定しておく関数。

で1つのスレッドが起動され、プロセスは1つとなる。同様にドライバのプロセスも1つである。

一方、コールバックは、Blu-ray用ドライブから発生する予測不能なイベントをキャッチする。もし、イベントが発生した場合、当該イベントをキャッチした1つのコールバックのスレッドは、割り込み処理を行うようにBlu-rayミドルウェアモジュールに通知を行う。

10.2.2.2. モード制御モジュールの内部状態

モード制御モジュールを構成するBlu-ray系インタフェースとBlu-rayミドルウェアモジュールはそれぞれ異なる抽象度で、Blu-ray用ドライブの状態を把握する。

10.2.2.2.1. Blu-ray系インタフェースの状態

Blu-ray系インタフェースは、Blu-rayミドルウェアモジュールに問い合わせを行い、その結果を基にBlu-ray用ドライブの状態を把握して、自身の内部状態として管理する。

Blu-ray系インタフェースの内部状態には、「停止中」と「再生中」の2種類である。

表 10-2: Blu-ray系インタフェースの内部状態

内部状態	意味
停止中	Blu-ray用ドライブは、挿入されているBlu-rayディスクを回転させておらず、記録されているデータの読み込み等が不可能な状態である。
再生中	Blu-ray用ドライブは、挿入されているBlu-rayディスクを回転させており、記録されているデータの読み込み等が可能な状態である。

10.2.2.2.2. Blu-rayミドルウェアモジュールの内部状態

Blu-rayミドルウェアモジュールは、ドライバを通して、Blu-ray用ドライブの状態を把握し、自身の内部状態として記憶する。

Blu-rayミドルウェアモジュールの内部状態は、「停止状態」、「再生移行中」、「再生状態」、「ポーズ中」、「早送り中」、「早戻し中」、「正スロー中」、「逆スロー中」の8種類がある。

表 10-3: Blu-rayミドルウェアモジュールの内部状態

内部状態	意味
停止状態	Blu-ray用ドライブに挿入されているBlu-rayディスクは停止しており、回転していない状態。
再生移行中	Blu-ray用ドライブに挿入されている、停止しているBlu-rayディスクの回転を開始し、再生状態になるまでの間の状態。
再生状態	Blu-ray用ドライブに挿入されており、回転しているBlu-ray

ポーズ状態	ディスクから、記録されているデータを読み出し、デコーダに転送している状態。
早送り中	Blu-ray 用ドライブに挿入されている Blu-ray ディスクは回転しているが、データの読み出しを停止している状態。
早戻し中	Blu-ray 用ドライブに挿入されている Blu-ray ディスクは回転しており、データを単位時間ごとに逆向きに読み出し、デコーダへの伝送している状態。
正スロー中	Blu-ray 用ドライブに挿入されている Blu-ray ディスクは回転しており、データの読み出しタイミングを通常の再生の場合より遅く行い、デコーダへの伝送している状態。
逆スロー中	Blu-ray 用ドライブに挿入されている Blu-ray ディスクは回転しており、データの読み出しタイミングを通常の再生の場合より遅く行いながら、データを逆向きに読み出して、デコーダへ伝送している状態。

10.2.2.2.3. Blu-ray系インタフェースとBlu-rayミドルウェアモジュールの内部状態の関係

Blu-ray 系インタフェースの内部状態と Blu-ray ミドルウェアモジュールの内部状態の対応関係を表 10-4 に示す。

表 10-4:内部状態の対応関係

Blu-ray 系インタフェースの内部状態	Blu-ray ミドルウェアモジュールの内部状態
停止中	停止状態
	再生移行中
再生中	再生状態
	ポーズ状態
	早送り中
	早戻し中
	正スロー中
	逆スロー中

10.2.2.3. Blu-ray系インタフェースのAPI

Blu-ray 系インタフェースには、Blu-ray アプリケーションがユーザの操作命令を伝達するための API を実装している。

表 10-5: Blu-ray 系インタフェースの API

	API	内容
1	FirstPlay	Blu-ray ディスクに記録されているデータを最初から再生することを要求する
2	Resume	再生中の Blu-ray ディスクの一時停止を要求する。 Blu-ray ディスクに記録されているデータ列の中で、停止位置を記録する。
3	Chapter_Search	ユーザが指定するチャプターの再生を要求する。 チャプターは Blu-ray ディスクに記録されているコンテンツごとに異なる。
4	Normal_Stop	再生中の Blu-ray ディスクの停止を要求する。
5	Switch	Blu-ray ディスクが再生中なら一時停止を要求する。 一時停止中の場合は、一時停止の解除を要求する。
6	Fast_Play	早送りを要求する。
7	Fast_Back	早戻しを要求する。
8	Slow_Play	
9	Slow_Back	
10	Flame_Advance	コマ送りを要求する。
11	Cancel_Trick	一時停止・早送り・早戻し・正スロー・逆スローといった特殊再生状態から解除して、通常の再生をすることを要求する。
12	Get_Info	Blu-ray ディスクの再生状態の取得を要求する。
13	Change_Angle	メディアのアングルの切り替えを要求する。
14	Set_Code	音声の言語コード設定を要求する。
15	Get_CallBack_Info	この API は呼び出しが特殊である。コールバック部がデバイスからのコールバックがあったことをアプリケーションプロセスに通知するために用いる。この API では、モード状態の変更が行われる。

10.3. 検証対象システムのモデリング

10.3.1. モデリングの前提

10.3.1.1. ケーススタディの方針(モデリングの方針)

今回の事例では、検証対象となるモード制御モジュール(**Blu-ray** 系インタフェースおよび **Blu-ray** ミドルウェア)以外にも様々なコンポーネントがそれぞれ 1 つ以上のスレッドとして並行動作する。例えば、**Blu-ray** アプリケーションは n 個、コールバックは 1 個、ドライバおよび **Blu-ray** 用ドライブはそれぞれ 1 個のスレッドとして並行動作する。このため、状態爆発やモデルの複雑さに起因する検証結果のトレースが困難になるなどの問題が想定される。そこで、今回のケーススタディでは、検証の目的を明確化した上で、検証に必要な部分を詳細にモデル化する一方、その他の部分は簡略化および統合により、1 つのコンポーネントにするなどして、プロセスを減らすと共に検証対象のモデルを簡略化していく。

今回のケーススタディでは、複数並行動作する **Blu-ray** 系インタフェースと単一スレッドで動作する **Blu-ray** ミドルウェアモジュールの内部状態の間で不整合が生じることはないか、不整合が生じた場合に **Blu-ray** 用ドライブの制御やデッドロックは発生しないか、といった点を検証することを目的とする。このため、**Blu-ray** 系インタフェースのプロセスと **Blu-ray** ミドルウェアモジュールのプロセス間の通信を検証対象の中心に据える。従って、**Blu-ray** 系インタフェースおよび **Blu-ray** ミドルウェアモジュールの内部状態とその制御をなるべく詳細にモデリングする。一方、それ以外の部分については、外部環境とみなして簡略化してモデリングする。

10.3.1.2. 検証対象の範囲

Blu-ray 用ドライブに関連する、**Blu-ray** 系アプリケーション、**Blu-ray** 系インタフェース、**Blu-ray** ミドルウェアモジュール、ドライバ、**Blu-ray** 用ドライブと、**Blu-ray** ディスクプレイヤを操作する人間を検証対象の範囲としてモデル化する。

Blu-ray ディスクプレイヤ内に付属する DVD、CDなどを再生するため制御機構は除外する。

10.3.1.3. モデリングの前提条件

今回の検証では、その検証目的から、**Blu-ray** ディスクプレイヤに何らかの **Blu-ray** ディスクは挿入済みであるという前提で考える(前提1)。また、**Blu-ray** ディスクプレイヤが通常動作している際の、モード制御モジュールの検証を行うため、停電や不慮の電源コンセントの取り外しなどは起こらない前提で考える(前提2)。

10.3.1.4. モデリングの考え方

上述のとおり、本ケーススタディのモデリングの方針では、「**Blu-ray** 系インタフェースと **Blu-ray** ミドルウェアモジュールの内部状態と通信についてなるべく忠実にモデル化する」とともに、「それ以外のコンポーネントは、簡略化または統合によりプロセス数を削減する」ということであった。

ここでは、モデルを構成するために、実際の **Blu-ray** ディスクプレイヤのプロセス構成の単位を

図 10-4 に分けて考えることとした。図 10-4 の赤枠で囲まれた Blu-ray インタフェースと Blu-ray ミドルウェアモジュールは、今回の検証における主要なターゲット部分であるため、実際のシステムの仕様に対して、必要部分を選別後、なるべく忠実にモデル化する。一方で、その他の部分(図 10-4 の青枠部分)については、簡略化または統合してプロセス数を減らす。

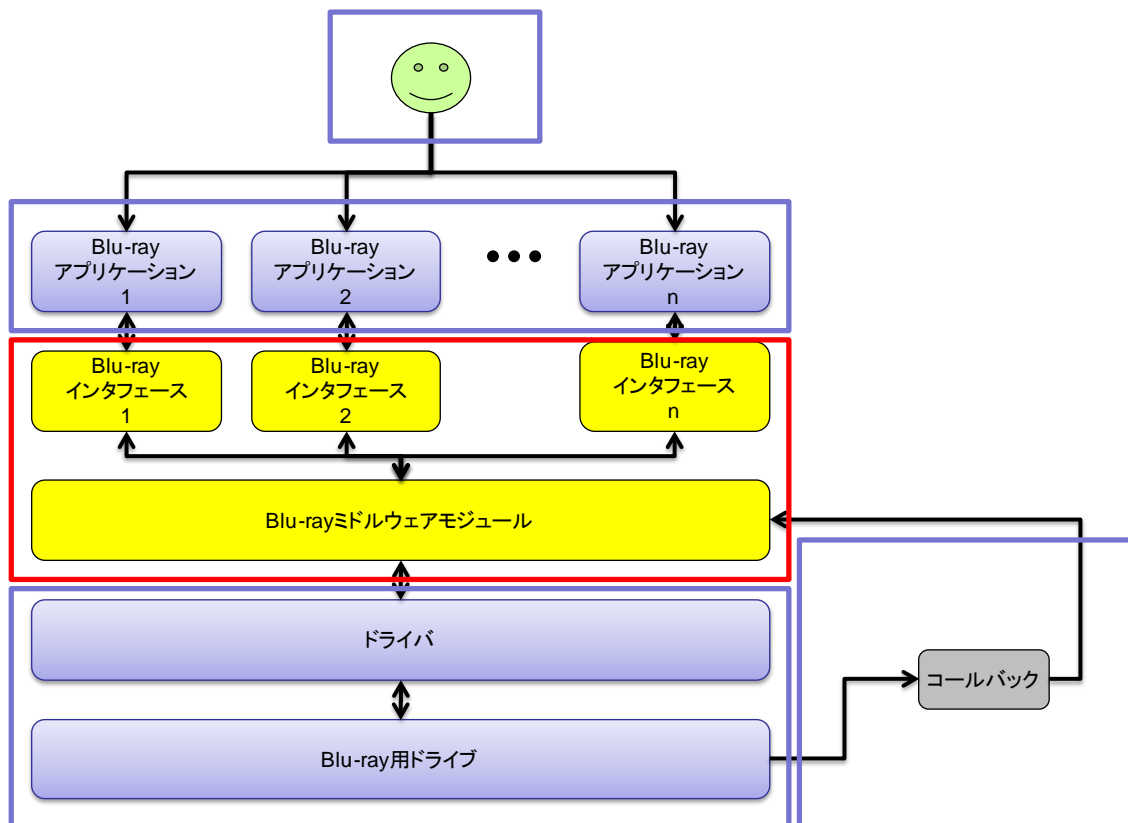


図 10-4: 実際のシステムのプロセス構成とモデリングを行う単位

以下、表 10-6 に、図 10-4 のそれぞれの部分についてのモデリングの方針とその理由を概説した。

表 10-6: における各部分のモデリングの方針とその理由

部分	モデリングの方針	理由
ユーザ	外部環境 Blu-ray ディスクドライブに対して、任意の順番で選択可能な命令を送信する	あらゆる命令の組に対して検証を行うため
Blu-ray アプリケーション	Blu-ray インタフェースに統合する	Blu-ray アプリケーションは検証目的の主要部分を構成せず、Blu-ray インタフェースと

Blu-ray インタフェースとBlu-ray ミドルウェア ドライバと Blu-ray 用ドライブ コールバック	必要部分を割り出した上で、実際のシステムの制御方法になるべく忠実に記述する	Blu-ray アプリケーションのプロセスは1対1で対応しているため 主要な検証対象部分であるため
	1つのプロセスとして統合する	主要な検証対象部分ではなく、1対1に対応したプロセスであため
	サイズ m のバッファを持つ 1 つのプロセスとする	割り込みイベントの発生確率は非常に小さいと考えられるため、発生した順番で逐次処理を行うことで十分と考えられるため

モデリングの結果と実際の Blu-ray ディスクプレイヤのプロセス構成の対比を図 10-5 に示す。

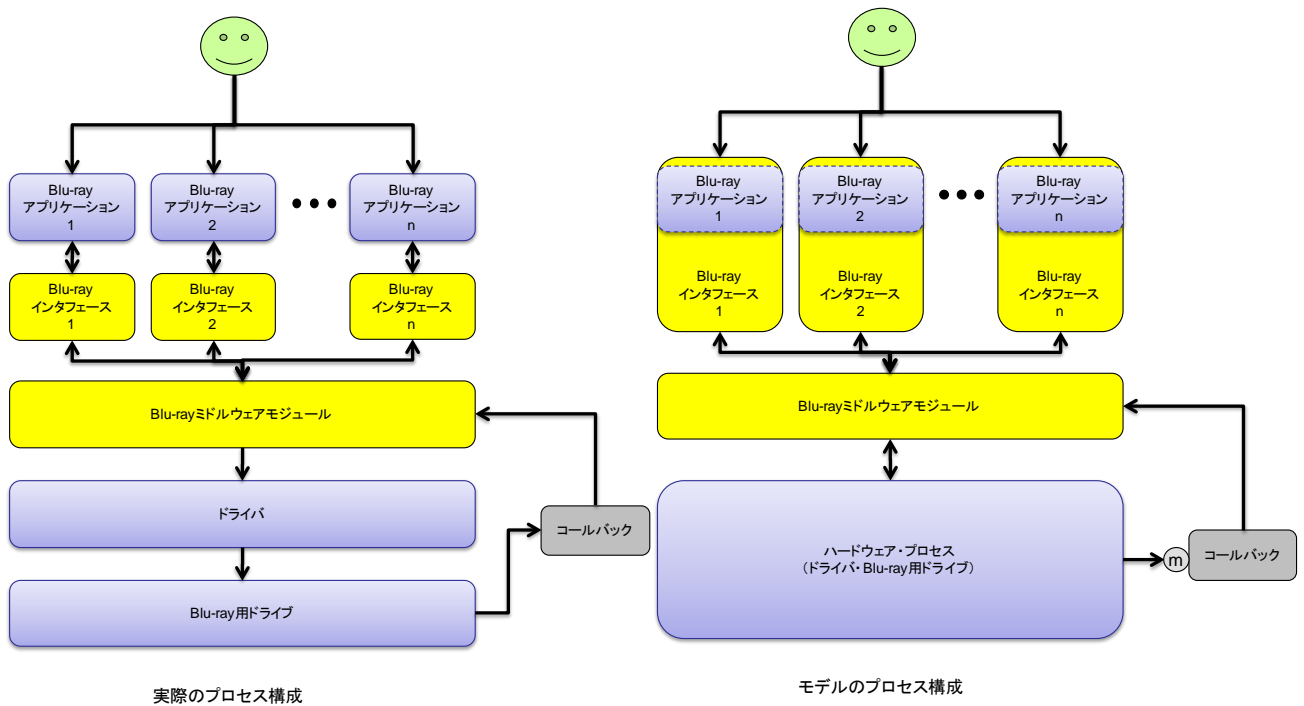


図 10-5: 実際のシステム構成とモデルのプロセス構成

10.3.2. モデリングの詳細

10.3.2.1. ユーザのモデリング

Blu-ray ディスクプレイヤからみて、ユーザは自身を操作する命令を、(リモコンなどを通して)送信してくる主体である。ユーザは選択可能な命令を自分の好きな順番、任意のタイミングで選択して送信してくる。このため、検証対象であるモード制御モジュールからみると、ユーザは外部環境として捉えることが可能である。

検証の目的である「Blu-ray系インタフェースの内部状態とBlu-rayミドルウェアのモジュールの内部状態の不整合の有無」を考える場合、ユーザが選択可能な命令のあらゆる組み合わせにおいて検証することが求められる。後述のように、Blu-rayアプリケーションはBlu-ray系インタフェースに組み込んで考えるため、ユーザのモデルではBlu-ray系インタフェースのAPI関数を直接呼び出すことができるとみなし、これをユーザが選択可能な命令(操作)であると考ええる。

10.3.2.2. Blu-rayアプリケーションのモデリング

Blu-rayアプリケーションは、今回のケーススタディにおいて主要な検証部分ではないため、Blue-ray系インタフェースに組み込み、プロセス数の削減を行うこととした。

10.3.2.3. Blu-ray系インタフェースのモデリング

モードモジュールの状態遷移を定義する状態遷移表を以下に示す。モードモジュールが保持するモード状態は、設計仕様にある通り、「停止中」と「再生中」からなる。モードモジュールの状態遷移は、モード状態に加え、ミドル状態にも基づくため、状態遷移表において状態を示す列には、モード状態とミドル状態の一覧が記載される。状態遷移表の行には、イベントの種類を表すAPIの一覧が示される。

	状態に関わらず行う	status.b.disc_play	play_st (AVC API PLAY ST系) ただし、BDMVの再生状態取得で扱わない状態は除く	停止状態	再生状態	再生移行中	ポーズ中	早送り中	正スロー中	早戻し中	逆スロー中
再生 FirstPlay再生	first_playback 再生処理 CALL disc_playを再生中に アプリに成功を返す	再生中 (灰色部は状態によって 処理に違いがないことを示 す)									
再生 Resume再生 (Resume無し)	first_playback 再生処理 CALL disc_playを再生中に アプリに成功を返す	再生(再開)処理CALL disc_playを再生中に アプリに成功を返す (ミドル回数からエラーが 返ったときはdisc_playを 変更しない、アプリにエラーを 返す)									
再生 Resume再生 (Resumeあり)	first_playback 再生処理 CALL disc_playを再生中に アプリに成功を返す (ミドル回数からエラーが 返ったときはdisc_playを 変更しない、アプリにエラーを 返す)	再生(再開)処理CALL disc_playを再生中に アプリに成功を返す (ミドル回数からエラーが 返ったときはdisc_playを 変更しない、アプリにエラーを 返す)									
再生 Chapterサーチ	Chapter Search実行時 処理CALL disc_playを再生中に アプリに成功を返す (ミドル回数からエラーが 返ったときはdisc_playを 変更しない、アプリにエラーを 返す)	Chapter Search実行時 処理CALL disc_playを再生中に アプリに成功を返す (ミドル回数からエラーが 返ったときはdisc_playを 変更しない、アプリにエラーを 返す)									
停止 通常停止	disc_playを停止中に アプリに成功を返す	disc_playを停止中に アプリに成功を返す									
再生一時停止	Pause押下処理CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	Pause押下処理CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)									
早送り	順方向再生速度変更処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	順方向再生速度変更処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)									
早戻し	逆方向再生速度変更処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	逆方向再生速度変更処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)									
正スロー	順方向再生速度変更処理 (スロー)CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	順方向再生速度変更処理 (スロー)CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)									
逆スロー	逆方向再生速度変更処理 (スロー)CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	逆方向再生速度変更処理 (スロー)CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)									
コマ送り	順方向コマ再生処理CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	順方向コマ再生処理CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)									
スキップアップ	NextPoint(チャプターサ ーチ)処理CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	NextPoint(チャプターサ ーチ)処理CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)									
スキップダウン	PrevPoint(チャプターサ ーチ)処理CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	PrevPoint(チャプターサ ーチ)処理CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)									
トリックプレイ解除	トリックプレイ解除処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	トリックプレイ解除処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)									
リポート設定 Chapterリポート	Chapterリポート回数CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	Chapterリポート回数CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)									
リポート解除	リポートクリアCALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	リポートクリアCALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)									
レジュームポイント の有無	resume 情報存在チェック CALL アプリにレジューム有無を 返す	Player 再生情報Register セット処理CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)									
レジュームポイント の設定	Resume 情報クリア処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	Resume 情報クリア処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)									
再生情報取得	disc_playを停止中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す
アングル切り替え	アングル変更処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	アングル変更処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	アングル変更処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	アングル変更処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	アングル変更処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	アングル変更処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	アングル変更処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	アングル変更処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	アングル変更処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	アングル変更処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)	アングル変更処理 CALL アプリに成功を返す (ミドル回数からエラーが 返ったときはアプリにエ ラーを返す)
音声言語コード設 定	Audio Language セット CALL アプリに成功を返す	Audio Language セット CALL アプリに成功を返す									
コールバック状態取 得回数(正常系通 知用)	disc_playを停止中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す
コールバック状態取 得回数(異常系通 知用)	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す	disc_playを再生中に アプリに再生情報を返す

図 10-6: モードモジュールの状態遷移表

10.3.2.4. Blu-rayミドルウェアモジュールのモデリング

ミドルモジュールの状態遷移を定義するミドル状態遷移表を以下に示す。ミドルモジュールは、基本的には、ミドル状態に基づき、モードモジュールから呼出された関数をイベントとして、状態遷移表に示す通りモデリングされる。ただし、ミドルモジュールが受け取るイベントは、モードモジュールからの関数呼び出しに加え、外部環境に相当するドライバからの割込みがある。状態遷移表では、それをモデリングするために、イベントとして、割込みとモードモジュールからの関数呼出しの一覧を示している。

	ミドル状態(表現はモードが解釈したと仮定したもの)							
	停止状態	再生状態	再生移行中	ポーズ中	早送り中	正スロー中	早戻し中	逆スロー中
ミドル側処理(モードから見たら発生するかわからない)	／	停止状態に遷移(再生終了)	再生状態に遷移	／	停止状態に遷移	停止状態に遷移	停止状態に遷移	停止状態に遷移
first_playback 再生処理CALL	再生移行中状態に遷移 モードに成功を返す	再生移行中状態に遷移 モードに成功を返す	／	再生移行中状態に遷移 モードに成功を返す	再生移行中状態に遷移 モードに成功を返す	再生移行中状態に遷移 モードに成功を返す	再生移行中状態に遷移 モードに成功を返す	再生移行中状態に遷移 モードに成功を返す
再生(再開)処理CALL	再生移行中状態に遷移 モードに成功を返す	×	×	×	×	×	×	×
Chapter Search実行時処理CALL	×	再生移行中状態に遷移 モードに成功を返す	／	再生移行中状態に遷移 モードに成功を返す	再生移行中状態に遷移 モードに成功を返す	再生移行中状態に遷移 モードに成功を返す	再生移行中状態に遷移 モードに成功を返す	再生移行中状態に遷移 モードに成功を返す
停止処理CALL	／(状態遷移なしで成功を返す)	停止状態に遷移 モードに成功を返す	停止状態に遷移 モードに成功を返す	停止状態に遷移 モードに成功を返す	停止状態に遷移 モードに成功を返す	停止状態に遷移 モードに成功を返す	停止状態に遷移 モードに成功を返す	停止状態に遷移 モードに成功を返す
Pause押下処理CALL	×	ポーズ中に遷移 モードに成功を返す	×	再生状態に遷移 モードに成功を返す	ポーズ中に遷移 モードに成功を返す	ポーズ中に遷移 モードに成功を返す	ポーズ中に遷移 モードに成功を返す	ポーズ中に遷移 モードに成功を返す
順方向再生速度変更処理CALL	×	早送り中に遷移 モードに成功を返す	×	早送り中に遷移 モードに成功を返す	／	早送り中に遷移 モードに成功を返す	早送り中に遷移 モードに成功を返す	早送り中に遷移 モードに成功を返す
逆方向再生速度変更処理CALL	×	早戻し中に遷移 モードに成功を返す	×	早戻し中に遷移 モードに成功を返す	早戻し中に遷移 モードに成功を返す	早戻し中に遷移 モードに成功を返す	／	早戻し中に遷移 モードに成功を返す
順方向再生速度変更処理(スロー)CALL	×	正スロー中に遷移 モードに成功を返す	×	正スロー中に遷移 モードに成功を返す	正スロー中に遷移 モードに成功を返す	／	正スロー中に遷移 モードに成功を返す	正スロー中に遷移 モードに成功を返す
逆方向再生速度変更処理(スロー)CALL	×	逆スロー中に遷移 モードに成功を返す	×	逆スロー中に遷移 モードに成功を返す	逆スロー中に遷移 モードに成功を返す	逆スロー中に遷移 モードに成功を返す	逆スロー中に遷移 モードに成功を返す	／
順方向コマ再生処理CALL	×	／	×	／	／	／	／	／
トリックプレイ解除処理CALL	×	／	×	再生状態に遷移 モードに成功を返す	再生状態に遷移 モードに成功を返す	再生状態に遷移 モードに成功を返す	再生状態に遷移 モードに成功を返す	再生状態に遷移 モードに成功を返す
アングル変更処理CALL	×	／	×	×	／	×	／	×
Audio Language セットCALL	／	×	×	×	×	×	×	×

図 10-7:ミドルモジュールの状態遷移表

10.3.2.5. ドライバとBlu-ray用ドライブのモデリング

Blu-ray ミドルウェアモジュールは、ドライバを通して制御する Blu-ray 用ドライブを制御する。つまり、ドライバが適切に動作しているのであれば、Blu-ray 用ドライブはドライバに従って動作すると考えられる。このため、ドライバと Blu-ray 用ドライブを一体とみなし、1 つのプロセスとする。

Blu-ray 用ドライブから発生する割り込みイベントは、挿入されている Blu-ray ドライブに傷がある、Blu-ray ディスクに記録されているデータを終端まで読み込んだなどにより発生する。つまり、上位の Blu-ray 系インタフェースや Blu-ray ミドルウェアモジュールが現在の状態について下位のモジュールに問い合わせても把握できないものである。従って、割り込みイベントは Blu-ray 用ドライブから任意のタイミングで発生するように、つまり、非決定的に発生するものとしてモデリングする。

通常の Blu-ray ディスクでは、1 枚のディスクにおいて最後までデータを読み込んだ場合に「再生終了」の割り込みイベントが発生する。つまり、Blu-ray ディスクに書き込まれているデータ系列の最後まで到達した場合に発生し、Blu-ray ディスクプレイヤーが動作している時間軸で考えると発生確率は非常に小さいと考えられる。また、ディスクに傷などがある場合、Blu-ray 用ドライブがデータの読み込みを行っている途中で、ディスクの傷などによりデータが破損している箇所に到達した時に「読み込みエラー」のイベントが発生する。つまり、ディスクに傷があり、データ破損箇所に到達した場合に「読み込みエラー」の割り込みイベントが発生し、その発生確率は非常に小さいと考えられる。従って、割り込みイベントの発生確率はいずれの場合も非常に低いと考えてモデリングすることができる。

10.3.2.6. コールバックのモデリング

Blu-ray 用ドライブから発生する割り込みイベントは、予測不能な任意のタイミングで発生する。コールバックは、この割り込みイベントを捕捉し、上位の Blu-ray 用ミドルウェアモジュールに通知する。ここで、上述の通り「個々の割り込みイベントは非常に小さな確率でしか発生しない」というようにモデリングしているため、同時に複数の割り込みイベントが発生する確率も非常に小さくなる。従って、同時に複数の割り込みイベントが発生する場合を捨象しても大きく現実のシステムと乖離するわけではない。そこで、割り込みイベントを発生した順番に m 個格納して逐次処理するようにコールバックをモデル化する。つまり「 m 個のバッファを持つ 1 つのプロセス」としてモデル化する。

10.4. 検証性質の形式記述

10.4.1. 検証性質の抽出

本ケーススタディでは、モード制御ソフトウェアが API リクエスト等を処理する過程で、モードモジュールとミドルモジュールの2つのモジュールが独自に保持する状態の不整合などが原因で、意図しない制御やデッドロックなどの発生が懸念されていた。

しかし、設計工程では、どのような不具合が発生するか想定することが難しく、検証すべき性質を明確化することが出来なかった。そのため、形式モデリングの後に、検証性質の形式記述を実施した。

10.4.2. 検証性質の形式記述

設計に関する問題が懸念される点を検証性質の形で記述し、モデル検査により検証する。具体的に検証性質を以下に示す。

10.4.2.1. 状態のズレによる不具合の存在について

モードモジュールとミドルモジュールの状態がずれることにより、予期しない制御の発生やデッドロックが発生しないかどうか検証したい。

このような性質、モデル検査により検証するためには、LTL 式として記述しなければならない。そのため、上記の性質の一部を示すものとして、「ミドルが処理できない要求をモードが流しても、いつかは待機状態に戻る。」を考える。さらにこれを、ミドル状態と、モードからミドルへの要求イベ

ントの具体例について考えることで、検証式を記述する。たとえば、「いつでも、ミドル状態が「停止中」で、かつ、ミドルに「ポーズ」要求が来ても、いつかは待機状態に戻る」を検証すればよい。この性質を構成する要素となる命題には、「ミドル状態が「停止中」」、「ミドルに「ポーズ」要求が来る」、「待機状態に戻る」の3つの条件がある。これらは順に、Promela で記述したモデルにおいては、ミドル状態(Mid_state)が、「停止中」(mid_stop)、モードモジュールからの関数呼出が一時停止(midf_pause)、モード状態(Mode_state)は、「停止中」(mode_stop)になる、ということで記述できるため、それぞれ、Mid_state==mid_stop、Mode_req ? [midf_pause(ch1)、Mode_state == mode_stop のように記述される。また、性質全体のうち、「いつでも」や「いつかは」などのように、時間の順序を表す性質は、時相演算子[], <>を用いて記述できる。したがって、この性質は形式的には以下のような式で記述することができる。

```

[]((Mid_state==mid_stop) && (Mode_req ? [midf_pause(ch1)]) -> <>(Mode_state == mode_stop))

```

10.4.2.2. モードからミドルへのリクエストの制約

次の検証性質は、モードモジュールは、ミドルモジュールで処理できない関数呼出しを行わないかどうか検証するものである。実際には、設計時に、反例が存在することが予想されていたため、ここでは反例を発見することを目的とした検証である。具体的には、「ミドル状態(mid_state)が「再生移行中」(mid_to_playing)で、かつ、モードモジュールへの要求がポーズである Mode_req ? [midf_pause(ch1)]、ことがない。」ことを検証すればよい。

```

[]!((mid_state == mid_to_playing) && (Mode_req ? [midf_pause(ch1)]))

```

10.4.2.3. モード状態とミドル状態の一致性

モード状態には、「停止中」と「再生中」の 2 種類がある。一方、ミドル状態には、「停止状態」、「再生移行中」、「再生状態」、「ポーズ中」、「早送り中」、「早戻し中」、「正スロー中」、「逆スロー中」の 8 種類がある。モード状態とミドル状態の一致とは、設計上は、モード状態が停止中の時、ミドル状態が停止状態にあり、かつ、モード状態が再生中の時、ミドル状態は、停止状態以外の状態であることと定義する。実際には、設計時点、このような性質には反例があることが予想されており、本検証では、その反例を見つけることが目的である。LTL 式は以下のように記述される。

```

#define s1 (Mode_state == mode_stop)
#define s2 (Mid_state == mid_stop)

#define p1 (Mode_state == mode_play)
#define p2 (Mid_state == mid_play)
#define p3 (Mid_state == mid_to_playing)
#define p4 (Mid_state == mid_pause)
#define p5 (Mid_state == mid_forward)
#define p6 (Mid_state == mid_fslow)
#define p7 (Mid_state == mid_rewind)
#define p8 (Mid_state == mid_bslow)
/*** LTL ***/
 || (p1 && (p2 || p3 || p4 || p5 || p6 || p7 || p8)))

```

s1, s2 は、モード状態とミドル状態がそれぞれ停止状態であることを示す命題である。また、p1 は、モード状態が再生中である命題を示し、p2~p8 は、ミドル状態が停止状態以外の各状態であることを定義する。これらの命題を用いて、LTL式は、最終行のように定義できる。

10.4.2.4. 機能の実行

ある操作を実行したときに、いずれはその機能が実行されるという、通常の機能要求を検証したい。この性質は、状態やイベントを具体化してそれらの組合せとしてインスタンスかされる性質を順に検証する。具体化した一つの例は、「モード状態が「停止中」の時、いつでも「再生」を要求すれば、いずれは再生ミドル関数が実行される。」となる。LTL式で記述すると以下ようになる。

```
[]((Mode_state == mode_stop) && (Mode_req ? [midf_play(ch1)]) -> <>(Mid_state == mid_play))
```

LTL 式の読下すと以下の通りである。モード状態(**Mode_state**)が、停止状態(**mode_stop**)でかつ、モードモジュールからミドルモジュールへの関数呼出しが再生(**midf_play**)である時、いずれはミドルの状態(**Mid_state**)は、再生(**mid_play**)となる。

10.4.2.5. その他

モデル検査では、以上のように LTL 式で記述したもの以外に、到達性解析によりデッドロックやデッドロックの存在について検証する。

10.5. モデル検査と結果

10.5.1. 検証結果

検証の結果、以下の(1)~(5)の項目のうち、(4)、(5)に関して想定外の問題を発見することができた。(1)~(3)に関しては、予想通りの結果を示すことができた。

10.5.1.1. 状態のズレによる不具合の存在について

以下の性質に関してモデル検査を行った。

「ミドル状態(**Mid_state**)が、「停止中」(**mid_stop**)で、かつ、モードモジュールからの関数呼出しが一時停止(**midf_pause**)の時、いずれは、モード状態(**Mode_state**)は、「停止中」(**mode_stop**)になる」

モデル検査を実施したところ、反例が見つかったが、それは、LTL 式におけるイベントと状態の値の記述ミスであることが判明した。LTL 式を修正することにより、上記の性質を検証することができた。

10.5.1.2. モードからミドルへのリクエストの制約

予想通りの反例の存在を確認することができ、具体的な反例に至る過程を特定できた。

10.5.1.3. モード状態とミドル状態の一致性

予想通りの反例の存在を確認することができ、具体的な反例に至る過程を特定できた。この LTL 式に関しても、記述に誤りがあることが判明し、それを修正することにより、期待通りの検証が行われた。

上記の(1)、(3)の2つの性質に関する LTL 式の誤りは、要求イベントと状態変数の値の混同によるもので、Promela の記述において状態やイベントを型で区別できないために、構文チェックが行えないことによる。このような点は注意が必要である。

10.5.1.4. 機能の実現性

機能要求として成り立つことが期待されるものであるが、ある操作を要求した際に、いずれそれが実行されることを保証する性質に反例を発見された。これにより、ユーザ操作の系列によっては、ライブロックの問題が発生することが判明した。ユーザの操作イベントによっては、期待した機能に到達しない。具体的には、停止状態から「再生移行中」に移行した段階で、別の操作リクエストが発

生した場合に、再生状態に移行することなく停止状態になる。この状態が繰り返されることにより、永久に再生に至らないことが発見された。

10.5.1.5. その他の検証

10.5.1.5.1. 到達性解析によるデッドロックの発見

到達性解析によりデッドロックを発見した。デッドロックは、ミドル処理とデバイス割込み処理の並行プロセスに関して共有変数に対する排他的なアクセスにより発生していることが判明した。

これは、一方が、共有変数をロックし、それを解除する前に、そのプロセスから呼出された別のプロセスが、共有変数のロックを必要とし、デッドロックが発生する。この点の設計仕様を修正することによりデッドロックを解決することができた

11. ケーススタディ2: 入退室管理システム

11.1. 検証の概要

11.1.1. 検証対象システムの概要

本ケーススタディで取り扱うボックス管理システムは、各利用者にボックス(図中 box)が割り当てられており、IC カード認証を利用して、いくつかの部屋を経由してボックスに至るアクセスの安全性を確保するシステムである。利用者(閲覧者)は必要に応じて、自身に割り当てられているボックスにアクセスして、物品を出し入れすることができる。ボックス管理システムの全体像を 図 11-1 に示す。部屋は、待合室、閲覧室 A、閲覧室 B、ボックス格納室、および外部のスペースに分かれており、利用者はそれらの部屋を移動する。

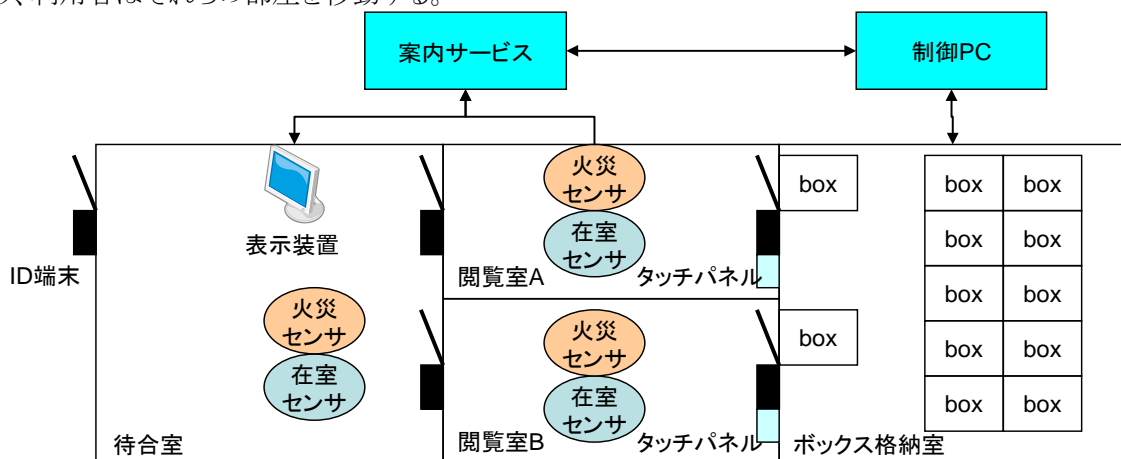


図 11-1: ボックス管理システム全体像

11.1.2. 検証の目的と検証内容の概要

閲覧者は人間であるため、システム設計時には想定しない動作を行う可能性がある。このため、システム設計時に想定しないエラーはこの部分に含まれる可能性が高い。そこで、このケーススタディでは、設計時に想定しない人間の動作を含めてシステムの検証を行い、仕様上のバグを発見することを目的とする。

モデリングでは、対象システムをなるべく忠実かつシンプルに捉えるとともに、そのモデルに合わせて網羅的な閲覧者(人間)のモデルを構築する。構築したモデルを基に、要求仕様を基に検証を行う。主な検証内容としては、以下である。

- 「閲覧室に案内されていない人が、タッチパネル脇 ID 端末で認証を行っても、常に認証をパスしない
- 閲覧室 X が空室のとき以外、常に案内は変更されない
- 待機者リストは常にオーバーフローすることはない

検証結果は、3 番目の「待機者リストは常にオーバーフローすることはない」を除き、反例は見つからなかった。

11.2. 検証対象システムの仕様

11.2.1. 検証対象システムの全体像

11.2.1.1. 検証対象システムの構成要素

検証対象システムでは、ボックス格納室、閲覧室 A および B、待合室と、案内サービスプログラム、制御 PC、PC インタフェースで構成されている。各構成要素の概要を表 11-1 に示す。

表 11-1: 検証対象システムの構成要素

構成要素	概要
------	----

構成要素	概要
待合室	システムを利用する閲覧者が一番初めに入る部屋。 入室の際には、扉の脇に設置されている ID カード端末(以下、待合室扉脇 ID 端末)に、閲覧者は自身の ID カードをかざして入室する。 閲覧室の使用状況に応じて、個別の閲覧者に対して閲覧室 A または閲覧室 B への案内情報を表示する表示装置が設置されている。 また、在室センサ、火災センサが取り付けられており、人の存在の有無、火災の感知ができるようになっている。
閲覧室 A/B	各閲覧者が自身のボックスにアクセスするための部屋。 物理的なスペースの制限により、閲覧者は 1 名だけしか入れないようになっている。 閲覧室 A/B への入室の際には、各閲覧室の扉脇に設置されている ID カード端末(以下それぞれ、閲覧室 A 扉脇 ID 端末、閲覧室 B 扉脇 ID 端末)に、閲覧者は自身の ID カードをかざして入室する。 暗証番号を入力するためのタッチパネルが設置されており、その脇に正当な閲覧者であることを認証するための ID 端末(以下、タッチパネル脇 ID 端末)が設置されている。 待合室と同様に、在室センサ、火災センサが設置されている。
ボックス格納室	複数のボックスを格納し、案内サービスプログラムからの要求に応じて、該当するボックスを要求のあった閲覧室に移動したり、戻したりする。
案内サービスプログラム	新規の閲覧者の登録、待合室や閲覧室 A/B の状態管理等、システム全体の管理を行うソフトウェア・サービス。
制御 PC	ボックス格納室内のボックスの移動等を制御する機器。
PC インタフェース	ID カード端末、在室センサ、火災センサの管理を行い、カード認証とセンサ反応を案内サービスプログラムに通知するソフトウェア。

11.2.1.2. 閲覧者の基本的な動作と対象システムの動作

閲覧者は、ボックスにアクセスするために、待合室に入室し、自身が閲覧室 A または閲覧室 B のいずれかに案内されるまで待機する。待合室の入室の際には、待合室扉脇 ID 端末にて、あらかじめ配布されている ID カードをかざして、認証を行う。認証をパスすると待合室の扉のロックが解除され、待合室への入室が可能となる。閲覧室 A または閲覧室 B への案内は、待合室内に設置されている表示装置(ディスプレイ)にて表示され、閲覧者に通知される。

閲覧室 A/B のいずれかに案内された閲覧者は、該当する閲覧室に移動し、各閲覧室の扉脇に設置されている ID 端末に、ID カードをかざして認証を行う。認証をパスすると、扉のロックが解除され、閲覧室への入室が可能となる。

閲覧室に入室した閲覧者は、閲覧室内のタッチパネルの操作を行い、自身のボックスを呼び出す。ボックスを呼び出す際には、タッチパネル脇に設置された ID 端末に持っている ID カードをかざして認証を行う。認証をパスすると、ボックスの取出し口のロックが解除され、ボックスを開くことができる。

ボックスにアクセスした閲覧者は、閲覧室の扉を開いて待合室に移動し、待合室の扉から外に出て行く。閲覧室および待合室の扉は、内側からは ID 端末を用いての認証は行わない。

11.2.2. システムに求められる条件 (要求仕様)

システム開発者とその発注者(ボックス管理システム所有者)が、システム開発当初に要求分析を行い、要求仕様として取り纏めた結果を表 11-2 に示す。ここで示す要求仕様は、一般の要求仕様のうち形式検証を想定したもののみ抽出したものである。この表 11-2 に示された要求仕様を、モデリングした検証対象システムに応じて修正し、検証要求とする。

表 11-2: ボックス管理システムの要求仕様

	要求仕様
「すること」	ID カードを登録している人は、待合室に入ることができる
	待合室に入った人は、必ず案内される
	閲覧室の扉認証は、閲覧室内が空室なら誰でも成功する
「しないこと」	閲覧室に人が案内されると、その人以外は、一定時間、閲覧室のボックスの認証に成功しない。 (最終的に保証すべき安全性)
	閲覧室に人が入室すると、その人が退室するまで、閲覧室のドア認証に成功しない。 (副次的な安全性。閲覧室の安全性)
	閲覧室が空室の時以外、閲覧室に人を案内しない。
	利用者は、どの部屋にも閉じ込められない。(常に、退出することが出来る。)

11.2.3. 検証対象システムの設計仕様

システム開発がボックス管理システムの要求仕様を基に作成した設計仕様について以下で説明を行う。ここで示す設計仕様から、システムのモデリングを行う。

11.2.3.1. 案内サービスプログラム

案内サービスプログラムは、PC インタフェースを通して、各部屋の ID 端末、室内センサ等から送信されてくるメッセージを基に、待合室および閲覧室 A、B の状態を管理すると共に、必要に応じて制御 PC、データベース等に動作の指示を行う。案内サービスプログラムは、待合室に入室した閲覧者のリスト(待機者リスト)を持ち、閲覧者の管理を行う。

以下では、案内サービスプログラムを実現する構成要素についてまとめる。

11.2.3.1.1. 待機者リスト

案内サービスプログラムは、システムが把握できる個々の閲覧者の位置を待機者リストにより管理する。

待機者リストは、待合室に入室した閲覧者ごとに、カード ID 番号、ボックス格納室 ID、案内閲覧室 ID、案内時刻で構成され(表 11-3)、リスト形式で管理する。

表 11-3: 待機者リストのデータ型

項目	内容	備考	初期化時の値
カード ID 番号	待合室に入室した閲覧者が保有する ID カード番号。	閲覧者待合室入室時(待合室扉脇 ID 端末で認証 OK となったとき)に 入力され、キーとして利用する。	待合室に入室した閲覧者の ID 番
ボックス格納室 ID	カード ID 番号に対応するボックス格納室の ID 番号。 1 つ以上のボックス格納室を想定。	1 つ以上のボックス格納室が存在する場合を想定。 閲覧者待合室入室時(待合室扉脇 ID 端末で認証 OK となったとき)に、 データベースに問い合わせを行い、 カード ID 番号に対応したボックスが格納されているボックス格納室 ID を記録する。	待合室に入室した閲覧者の ID 番号に対応するボックスが格納されたボックス格納室の ID
案内閲覧室 ID	ID 番号の閲覧者を案内した閲覧室の ID	閲覧室は 1 つ以上を想定。 空室となった閲覧室からの案内要求メッセージに応じて、当該 ID 番	NULL

		号の閲覧者とその閲覧室に案内する場合、閲覧室の ID 番号が記録される	
案内時刻	ID 番号の閲覧者を閲覧室 A または B に案内した時刻。	閲覧者が待合室入室時には、NULL が代入される。	NULL

11.2.3.1.2. 待合室管理プロセス

待合室管理プロセスは、PC インタフェースを介して受信するメッセージに応じて、待合室の状態を管理する。PC インタフェースを介して受信するメッセージは、待合室扉脇 ID 端末、待合室に設置されたセンサからのメッセージである。

待合室扉脇 ID 端末、待合室在室センサからのメッセージを PC インタフェースを通して取得し、待合室の状態を管理する。また、待合室扉脇 ID 端末で認証 OK となった閲覧者について、新規の閲覧者として待機者リストに登録を行う。

表 11-4: 待合室管理プロセスの受信メッセージと対応するアクション

メッセージ	内容	アクション
待合室扉脇 ID 端末認証 OK	待合室扉脇 ID 端末において認証をパスしたことを示す	新規閲覧者として初期化し、待機者リストに登録する。
待合室在室センサ ON	待合室に(正当な閲覧者とは限らない)人間が存在することを示す。	—
待合室在室センサ OFF	待合室に誰もいないことを示す。	—
案内タイムオーバー	待機者リストに登録されているが、制限時間内に案内されなかった閲覧者が存在することを示す	待機者リストの該当項目(制限時間オーバーの閲覧者のデータ)を削除する。

11.2.3.1.3. 閲覧室A/B管理プロセス

閲覧室 X 管理プロセス(ただし、X=A or B を表す。)は、PC インタフェースと制御 PC を介して受信するメッセージに応じて、閲覧室 X の状態を管理する。PC インタフェースを介して受信するメッセージは、閲覧室 X 扉脇 ID 端末、閲覧室 X 扉、閲覧室 X 内に設置されたセンサ、閲覧室 X 内タッチパネル、閲覧室 X 内タッチパネル脇 ID 端末から送信される。制御 PC を介して受信するメッセージは、ボックス取出口の状態である。

表 11-5: 閲覧室 X (X=A or B) 管理プロセスが受信するメッセージと対応するアクション

メッセージ	内容	アクション
閲覧室 X 扉 ID 端末認証 OK	閲覧室 X 扉脇に付属した ID 端末に入力された ID カード番号を持って、ID 端末が認証 OK としたことを示す。	<ul style="list-style-type: none"> 入力されたカード ID 番号を基に、閲覧室 X の入室者リストを更新する 閲覧者が一名以上在室する場合は「緊急事態」として、閲覧室 X の入室者リストに、新たな閲覧者を追加する。
閲覧室 X 扉閉	閲覧室 X の扉が閉まったことを示す。	<ul style="list-style-type: none"> 閲覧室 X の入室者リストの最後尾のデータに、メッセージを受信した時刻を記入する。 ボックスが格納済みであれば、閲覧室 X に閲覧者は存在しないとして、閲覧室 X の退出処理を行う。

メッセージ	内容	アクション
閲覧室 X 扉施錠	閲覧室 X の扉に鍵が施錠されたことを示す。	・緊急事態を除いて他の閲覧者が入室できないように、閲覧室 X の扉を施錠する
閲覧室 X 扉内側解錠	閲覧室 X 内の閲覧室 X の扉が解錠され、その要因が内側から開いたことを示す。	・扉の開閉が入室によるものか、退室によるものかを判断するために用いる
閲覧室 X 在室センサ OFF	閲覧室 X 室内の在室センサが OFF であり、閲覧者が 1 人もいないことを示す。	・ボックスが格納済みであれば、閲覧室 X に閲覧者は存在しないとして、閲覧室 X の退出処理を行う。 ・閲覧室 X が空室の場合、待機者リストから待合室に待機中の閲覧者を走査し、未案内の閲覧者の案内閲覧室 ID を閲覧室 X の ID にセットし、データベース(DB)に待機者リストのコピーを行う。
閲覧室 X 在室センサ ON	閲覧室 X 室内の在室センサが ON であり、閲覧者が 1 人以上いることを示す。	—
閲覧室 X 周期監視:在室センサ OFF	在室センサの状態を周期的に観測し、その際在室センサが OFF であったことを示す。	・ボックスが格納済みであれば、閲覧室 X に閲覧者は存在しないとして、閲覧室 X の退出処理を行う。 ・閲覧室 X が空室の場合、待機者リストから待合室に待機中の閲覧者を走査し、未案内の閲覧者の案内閲覧室 ID を閲覧室 X の ID にセットし、データベース(DB)に待機者リストのコピーを行う。
閲覧室 X 周期監視:在室センサ ON	在室センサの状態を周期的に観測し、その際在室センサが ON であったことを示す。	—
閲覧室 X 内タッチパネル脇 ID 端末認証 OK	閲覧室内のタッチパネル脇に設置された ID 端末に入力された ID カード番号が認証をパスしたことを示す	・ID 端末から入力された ID カード番号と、待機者リストに記録されている閲覧室 X に案内した ID カード番号を照合する(閲覧室内認証処理)
ボックス格納受信	ボックスが格納されたことを示す。	・閲覧者の所定の処理が終了したものとして、閲覧室 X の入室者リストから当該閲覧者のデータを削除すると共に、待機者リストを更新する(閲覧室退室処理)。
在室時間オーバー	閲覧者が定められた時間以上に閲覧室内にいることを示す。	・閲覧室 X の在室時間超過フラグを ON にする
閲覧室 X タッチパネル暗証番号照会	タッチパネルに暗証番号が入力されたことを示す。	・制御 PC に閲覧室内認証の結果を通知する。 ※制御 PC は、認証 OK を受信した場合、ボックスのロックを解除

11.2.3.1.4. PCインタフェース

PC インタフェースは、ID 端末、タッチパネル、在室センサ、火災センサなどから送信されてくる信号を案内サービスプログラムで解釈可能なメッセージに変換して、案内サービスプログラムに送信する。

以下では、PCインタフェースを構成する要素についてまとめる。

11.2.3.1.5. 待合室扉脇ID端末

待合室の入り口に附属する扉の脇に設置された ID 端末である。正規の閲覧者は、所持している ID カードを本 ID 端末にかざして認証を行い、室内に入る。

待合室扉脇 ID 端末内部には、正規の閲覧者の ID カード番号を記録しており、閲覧者が所持する ID カードから入力された ID 番号と照合を行う。ID 端末内部に記録されている ID カード番号の中の一つにマッチすると、認証成功とし、認証 OK の信号を PC インタフェースに送信すると共に、待合室の扉のロックを解除する。ID 端末内部に記録されている ID カード番号のいずれにもマッチしない場合、認証失敗とし、待合室の扉のロックは解除されない。なお認証失敗の場合には PC インタフェースを介して案内サービスプログラムに認証 NG のメッセージが送信されるが、案内サービスプログラム内部ではこのメッセージを利用しない。

11.2.3.1.6. 閲覧室X扉脇ID端末

閲覧室 A および閲覧室 B のそれぞれの入り口に附属する扉の脇に設置された ID 端末である。正規の閲覧者は、所持している ID カードを本 ID 端末にかざして認証を行い、室内に入る。

閲覧室 X 扉脇 ID 端末内部には、正規の閲覧者の ID カード番号が記録されており、閲覧者が所持する ID カードから入力された ID 番号と照合を行うと共に、閲覧室 X 内に誰もいない場合、閲覧室 X の扉が開錠され、入室が可能となる。閲覧室 X 内に誰かがいる場合には、ID カード番号の照合をパスしても、閲覧室 X の扉は開錠されず、閲覧者は入室できない(表 11-6)。

表 11-6: 閲覧室 X の扉の開錠条件

		閲覧室 X 内の在室者の有無	
		無し	有り
照合	アンマッチ	解錠しない	解錠しない
	マッチ	解錠する	解錠しない

11.2.3.1.7. 閲覧室X扉

閲覧室 A および閲覧室 B の扉である。電子錠を使用しており、錠の解錠、施錠状態と扉の開閉状態の変化を通知する。通知メッセージには、発生要因が含まれている。解錠であれば、閲覧室 X 扉脇 ID 端末の認証成功が要因なのか、部屋の内部からカード認証を行わずに直接開けたことが要因なのかが区別できる。

表 11-6 の制御は閲覧室 X 扉に、施錠モードを設定することで実現する。閲覧室 X 内の在室者が無しの場合は、通常モードで動作させ、カード認証による解錠を許可する。閲覧室 X 内の在室者が有りの場合は、解錠禁止モードで動作させ、カード認証による解錠を禁止する。

11.2.3.1.8. 閲覧室X内タッチパネル及びID端末

各閲覧室 A および B 内部には、ボックスの開閉を行うための認証装置として、ID 端末とタッチパネルが設置されている。閲覧者は、閲覧室内の ID 端末に ID カードをかざすと共に、タッチパネルにて暗証番号を入力し、認証をパスするとボックスの開閉が可能となる。

閲覧室 X 内部の ID 端末から入力された ID カード番号は、案内サービスプログラムの閲覧室 X 管理プロセスにおいて、正規に案内された閲覧者の ID カード番号との照合を行う。照合結果が OK である場合には、制御 PC がボックスの制御を行い、閲覧室 X の取り出し口までボックスが移送される。照合結果が NG の場合には、ボックスの移送は行われず、認証失敗が閲覧者に伝えられ

る。

移送されたボックスの取り出しに際しては、閲覧室内のタッチパネルからの暗証番号の入力を行い、タッチパネル内部で暗証番号の照合を行い、OK であればロックが解除され、閲覧者はボックスへのアクセスが可能となる。NG であれば、ロックは解除されず、閲覧者はボックスへのアクセスができない。

11.2.3.2. センサ

待合室、閲覧室 A および閲覧室 B には、在室センサおよび火災センサが取り付けられている。以下、在室センサおよび火災センサについて説明する。

11.2.3.2.1. 在室センサ

在室センサは、人の存在を検知し、センサ内部にその状態を記録する。PC インタフェースは、定期的に各部屋の在室センサを監視し、閲覧者の存在の有無を確認する。

在室センサは、人間が存在することを感知すると、内部状態を OFF から ON にスイッチする。人間の存在の有無だけを感知し、人数の把握はできない。また、部屋に人間が 1 人もいなくなったとき、在室センサは内部状態を ON から OFF にスイッチする。

11.2.3.2.2. 火災センサ

火災センサは、火災を検知し、センサ内部にその状態を記録する。PC インタフェースは定期的に各部屋の火災センサを監視し、火災の有無を確認する。

火災センサは、火災の有無を感知すると、内部状態を OFF から ON にスイッチする。

11.2.3.3. データベース(DB)

データベース(DB)は、登録されている閲覧者と閲覧者に割り当てたボックスが格納されているボックス格納室の対応関係を管理する静的なデータと、待機者リストを逐次複製し、待合室および閲覧室 X の在室状況を管理する動的なデータを保有する。

案内サービスプログラムの待合室管理プロセスは、PC インタフェースを通して、待合室扉脇 ID 端末から認証 OK を受信した際に、新規の閲覧者として待機者リストに登録する。この際、当該閲覧者に対応するボックスが格納されているボックス格納室の ID の問い合わせを DB に行う。DB は待合室管理プロセスの問い合わせに応じて、当該閲覧者のボックスが保管されているボックス格納室の ID を待合室管理プロセスに返す。なお、今回のケーススタディでは、ボックス格納室が 1 つであるため、どの閲覧者に対しても同じボックス格納室 ID がデータベース(DB)から待合室管理プロセスに返される。

案内サービスプログラムの閲覧室 X(X=A or B)管理プロセスは、PC インタフェースを通して、閲覧室 X 内の在室センサが OFF であることを感知すると、待機者リストにおいて未案内の閲覧者を走査し、当該閲覧者の案内閲覧室 ID を閲覧室 X にセットして、DB に待機者リストのコピーを行う。

11.2.3.3.1. 表示装置(Display)

表示装置(Display)は、一定周期ごとにデータベース(DB)が待機者リストのコピーから最新の案内者を走査し、表示内容を切り替える()。表示内容の切り替えの際には、一旦表示内容をリフレッシュして、DB 内の待機者リストのコピーを参照して、閲覧室 A/B それぞれへの最新の案内者を探索して表示する。

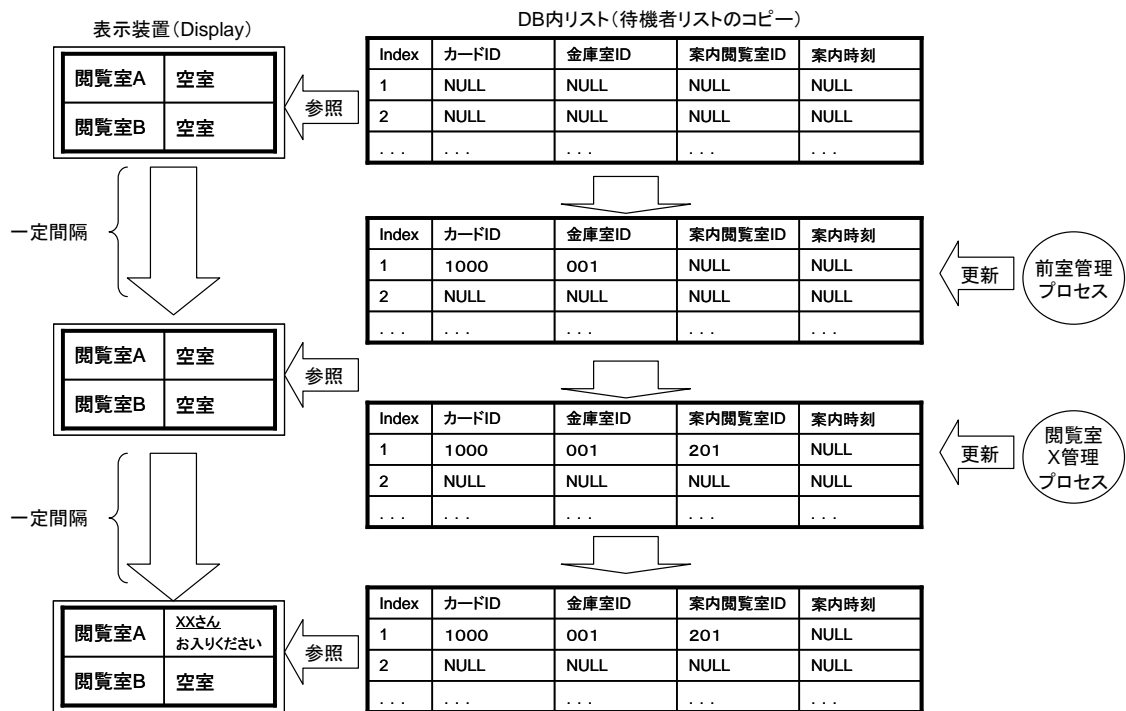


図 11-2: 表示装置の更新タイミング

11.2.3.4. 閲覧者

形式検証を行うためには、対象システムだけでなく、システムの周りの環境もモデリングし、全体として閉じた系を定義しなければならない。そこで、システムにとって外部の環境に相当する閲覧者の動きを説明する。閲覧者は人間であるため想定外の動きすることが考えられる。ここではシステムとして想定する閲覧者の動作を述べる。

正当な閲覧者は、システム管理者またはサービス提供者から付与または貸与された ID カードを保有している。ID カードには、当該閲覧者に対応する ID 番号が記載されており、この ID 番号によりシステム上は一意に特定される。

閲覧者は、待合室の入り口に設置されている待合室扉脇 ID 端末に所持している ID カードをかざし、本端末で認証をパスして、待合室に入室する。

待合室では、表示装置に自身の名前が表示されるまで待機する。表示装置に自身への案内 (入室する閲覧室と自身の名前) が表示されたら、その閲覧室の方向に進む。

案内された閲覧者は、閲覧室 X (A or B) の扉脇に設置された閲覧室 X 扉脇 ID 端末において、保有する ID カードをかざし、本端末で認証をパスして、閲覧室 X に入室する。

閲覧室 X に入室した閲覧者は、閲覧室内 ID 端末に所持している ID カードをかざし、タッチパネルに暗証番号を入力して、自身のボックスにアクセスする。アクセスが終了した後、ボックスを格納して、閲覧室 X の扉から待合室に移動する。

ボックスを取り出した後に閲覧室から待合室に移動した閲覧者は、待合室の扉から外部に行く。

11.3. 検証対象システムのモデリング

11.3.1. モデル化の前提

11.3.1.1. ケーススタディの方針

今回のケーススタディでは、対象システムの外部環境は人間 (閲覧者) の動きに相当する。人間の動きに関して、システム開発時に想定していなかった動作も含めたシステムの検証を行うことを

主目的とする。これは、必ずしも人間はシステムが想定する動作だけを行うわけではないため、実際のシステムでは想定外の動作も含めた形で検証を行う必要があるためである。一方で、人間の動作は無限大であることから、検証対象システムをなるべくシンプルなものとして捉え、閲覧者の動作をそのモデルに合わせて構築する。

11.3.1.2. 検証対象とするシステムの範囲

システム構築者がシステムの構築を請け負った案内サービスプログラムを中心にモデリングを行う。つまり、ボックス格納室並びにボックスの制御を行う制御 PC はモデリングの範囲外とする。

11.3.1.3. モデリングの前提条件

検証対象システムのモデリングを行う際の前提を表 11-7 に示す。

システムの要求仕様が満たされることを示すことが目的であるため、通常時のシステム動作を検証範囲とし、非常事態等は取り扱わない。つまり、火災等の非常事態や、閲覧者の怪我、病気等の事象も取り扱わないこととする(前提 0-1、0-2)。

また、上述 6.2.2 に示したシステムの取扱範囲から、ボックス格納室は取り扱い範囲外とする(前提 0-3)。関連して、閲覧室 X 内におけるボックスの取り出しに関する処理は対象外とし(前提 1-2)、付随して、タッチパネルは取り扱わない(前提 1-2-1)、および、閲覧者や閲覧室 X でボックスに関連する状態は取り扱わない(前提 1-2-2)。ただし、検証対象システムでは、正規の閲覧者が閲覧室 X 内でボックスにアクセスすることが主要なサービスであることから、前提 1-2、1-2-1、1-2-2 を考慮し、閲覧室 X 内 ID 端末にアクセスして認証 OK となることでボックスの取り出しが終了したものとみなし簡略化する(前提 1-2-3)。

SPIN では時刻に関連する取扱が困難であるため、時刻は対象外とし、タイムオーバー関連処理は対象外とする(前提 1-1)。また、モデル化の単純化のために閲覧室 X に附属する扉はとりあつかわない(前提 1-3)、全ての閲覧者は登録済み(前提 1-5)とする。最後に、前提 0-1 から、災害等の非常事態は取り扱わないため、閲覧室 X における緊急入室、案内停止は取り扱わない(前提 1-4)。

表 11-7: 検証対象システムのモデリングの前提

		前提の内容	理由
大前提	0-1	災害関連の事象・システムは取り扱わない	システム構築者は正常状態の検証をニーズとして持っているため
	0-2	閲覧者の怪我、病気等の事象とそれに対応するためのシステムは取り扱わない	システム構築者は正常状態の検証をニーズとして持っているため
	0-3	ボックス格納室については取扱い範囲外とする	ボックス格納室はシステム構築者の取扱い範囲外
モデリングの際の前提	1-1	時刻は対象外とし、タイムオーバー関連処理は対象外とする	SPIN では時刻の取扱が困難なため
	1-2	ボックスの取り出し等に関する処理は対象外とする	ボックス格納室はシステム構築者の取扱い範囲外(前提 0-3)のため
	1-2-1	閲覧室内のタッチパネル関連処理は対象外とする	ボックスの取り出しに関連する処理であるため(前提 1-2)
	1-2-2	ボックスに関連する状態は取り扱わない	ボックスの取り出しに関連する処理であるため(前提 1-2)
	1-2-3	閲覧者が閲覧室内のタッチパネル脇の ID 端末で認証 OK となった時点で、ボックスの取り出しが終了したものとみなす	閲覧室における閲覧者の動作を簡略化してモデル化するため
	1-3	閲覧室の扉は取り扱わない	モデルの単純化のため
	1-4	閲覧室内における緊急入室、案内停止は	災害関連の事象・システムは取り扱わな

	前提の内容	理由
	取り扱わない	いため(前提 0-1)
1-5	全ての閲覧者は登録済みとする	モデルの単純化のため

11.3.2. モデリングの考え方

11.3.2.1. モデリングの要件

今回のケーススタディの要件としては、上述の通り、

- 閲覧者の動作について設計時に想定しない動作も含めて網羅的に検証できるように構築すること
- 閲覧者のモデルに合わせて、原因個所が分かるようになるべくシンプルに検証対象をモデリングすること

の2点があげられる。

また、今回検証対象としたシステムでは、多くの異なるプロセスが存在する。例えば、検証対象システムだけで、大きく分けて、待合室管理プロセス、閲覧室 A 管理プロセス、閲覧室 B 管理プロセスとDBプロセスの4つのプロセスが考えられ、この他にもセンサ等プロセスとして実装可能なものが存在する。また、これらに加えて複数個の閲覧者プロセスが考えられる。従って、

- 必要最小限の閲覧者プロセスを確保しつつ、全体のプロセス数を削減し、現実的な時間内で検証が終了するようにモデリングすること

が必要である。

11.3.2.2. モデリングの要件に関する対処方針

11.3.2.2.1. 閲覧者モデリングの方針

閲覧者のモデリングは、システム設計時に想定していた閲覧者の動作(11.2.3.4参照)をベースに有り得る状態を割り出し、現実的に有り得る状態遷移を構成することで、設計時に想定していなかった閲覧者の動作もモデリングする。

11.3.2.2.2. 検証対象システムのモデリングの方針

検証対象システムは、複数のプロセスが考えられ、それぞれのプロセスが複数の状態を有し、各プロセスが関連しあいながら状態遷移を行う。また、個々の状態で待機者リストの中からある条件に該当するレコードを検索するなど、比較的複雑な処理を行う場合がある。これらを考慮し、以下の方針でモデリングを行うこととする。

- 各プロセスが有する状態をなるべく少なくする
- 待機者リスト等を検索するなどの処理は単純化し、必要最小限にとどめる

11.3.2.2.3. プロセス数の削減の方針

検証対象システムでは、案内サービスプログラム内の待合室および閲覧室の各部屋の閲覧者の有無を管理するプロセス、待機者リストをコピーし定期的に表示装置に表示される内容を更新するDBのプロセスなど、複数のプロセスが同時並行的に動作している。これらを検証対象システムの実態に即してモデル化することは、不可能であり、また、状態爆発から検証を行うことができなくなる可能性が高い。そこで、以下の方針で検証対象システムをモデル化する。

- システムの要求仕様(表 11-2)から、主な検証対象となる案内サービスプログラムの待合室、閲覧室 A および閲覧室 B の管理を行う部分を主要プロセスとしてモデリングする
- その他のプロセスは、主要プロセスまたは閲覧者プロセスの一部に含めるか、チャネルまたはグローバル変数とする

検証対象システムでは、複数の閲覧者が想定されている。理想的にはなるべく多くの閲覧者プロセスに対して検証を行うことが望ましいと考えられるが、一方で、閲覧者プロセスが多くなるほど状態爆発により検証ができなくなる可能性が高まる。そこで、検証においては、閲覧者プロセスの

数を 2 以上で実施することとし、状態爆発が起こらない範囲で閲覧者プロセスの数を増やしなが
 検証を行うこととする。

11.3.3. モデリングの概要

検証対象システムをモデリングする際に、実際のシステムに比較的近いと考えられる詳細なモ
 デルを構築した後で、上記の 11.311.3.2.2 に示したモデリングの要件に関する対処方針に従って
 モデルを単純化し、状態爆発の発生を抑え検証を行い易くする。

検証対象システムの単純化したモデルを構築した上で、検証者のモデリングを行う。

11.3.3.1. 実際に近いモデルの概要

実際の検証対象システムに近いモデルのイメージ全体像を図 11-3 に示す。

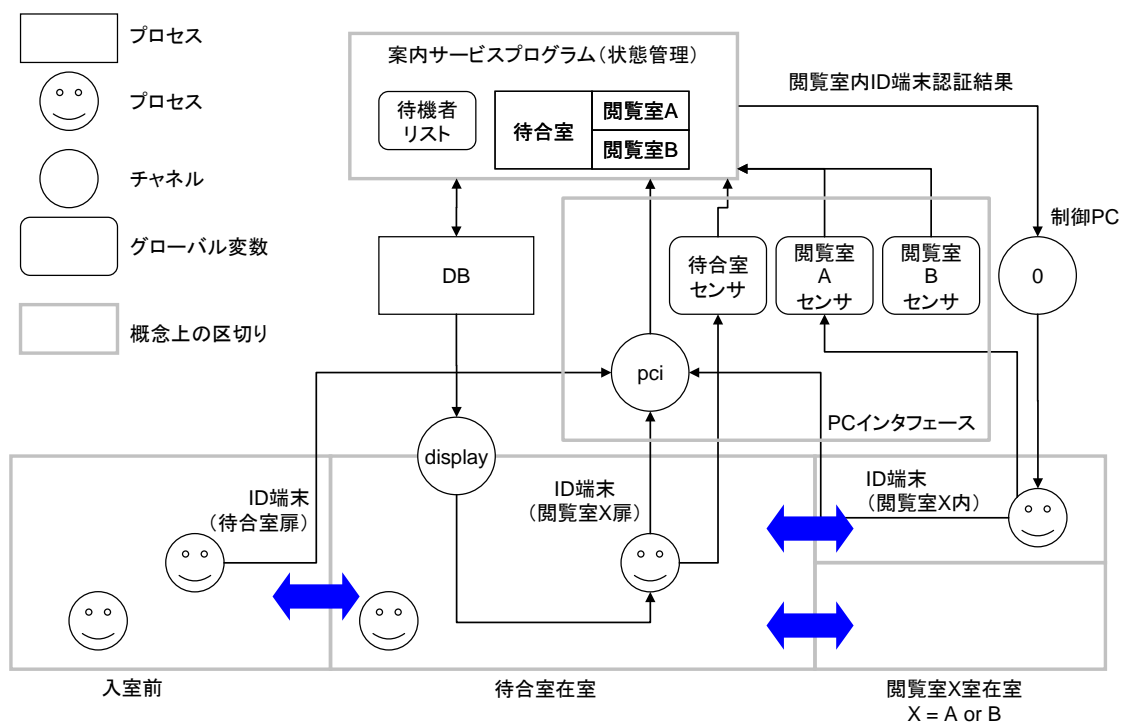


図 11-3: 実際に近いモデル(全体像)

案内サービスプログラムは、待合室管理プロセス、閲覧室 X(X=A or B)管理プロセスおよび待
 機者リストで構成する。待機者リストは、グローバル変数として定義する。

PC インタフェースは、待合室扉脇 ID 端末や在室センサなどの信号を案内サービスプログラム
 で解釈可能なメッセージに変換し、案内サービスプログラムに送信する役割を担うことから、チャネ
 ル(図 11-3 pci)およびグローバル変数として表現する待合室センサ、閲覧室 A センサおよび閱
 覧室 B センサで構成する。なお、ここでのセンサとは在室センサである。

データベース(DB)は、プロセスとして定義する。DB プロセスは、待合室管理プロセスや閲覧室
 X プロセスの要請に応じて、待機者リストのコピーや表示装置の切り替えを行う。

表示装置は、DB の要請に応じて表示内容を変更するだけであり、能動的に動作する主体では
 ないため、チャンネルとして表現し、プロセス数の削減をはかることとした。

また動作 PC は、検証の範囲外であるが、実際は閲覧者が動作 PC の反応、つまり、ボックスの
 開閉を通して、閲覧室内 ID 端末の認証結果を知ることから、チャンネルとして表現し、案内サービ
 スプログラムの閲覧室 X 管理プロセスから閲覧者がメッセージを受け取るようにモデリングした。

11.3.3.2. 単純化した検証対象システムモデルの概要

単純化した検証対象システムのモデルの全体像を図 11-4 に示す。

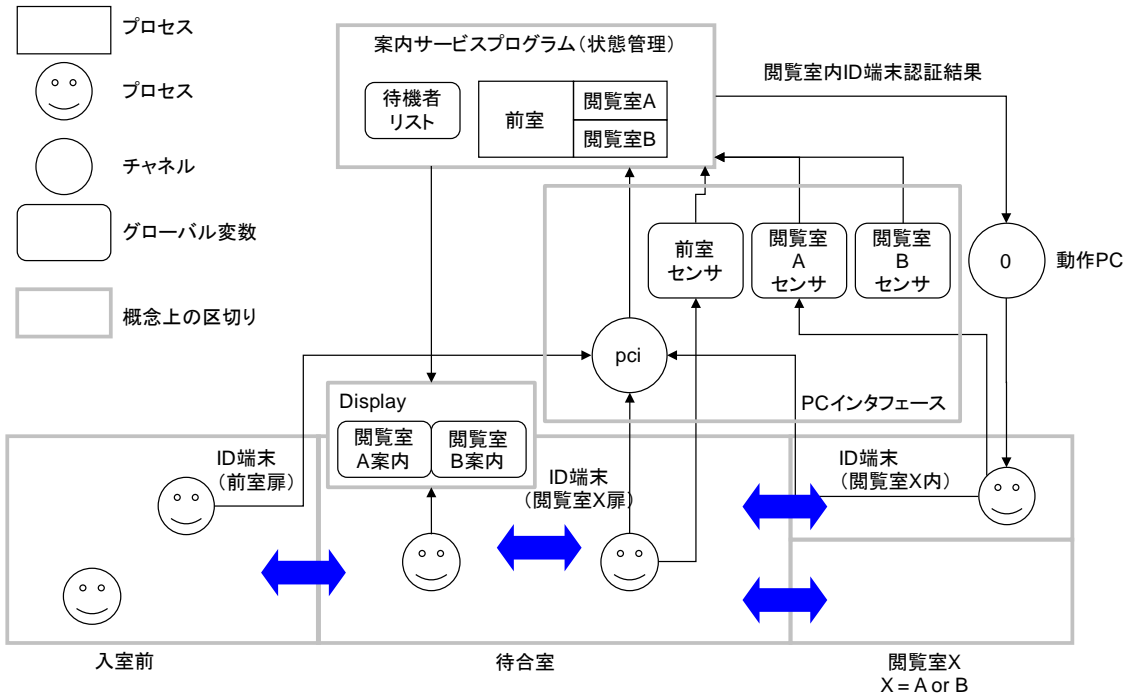


図 11-4: 単純化した検証対象システムのモデル(全体像)

単純化したモデルでは、図 11-3 における DB プロセスを削減するとともに、表示装置 (Display) を 2 つのグローバル変数に変更した。これは上述したプロセス数の削減を行うと、表示装置に係る制御を単純化するためである(11.3.2.2 参照)。

ここで留意すべき点として、モデルの単純化に伴い検証結果に差異が生じないかという点である。今回のケーススタディでは、検証対象システムの動作の方が閲覧者の動作よりも十分に早いことを考慮し、案内サービスプログラムと DB のインタラクションを一体とみなし、案内サービスプログラムと DB 間のインタラクションで生じる不具合は無視することとした(11.2.3.3 参照)。

11.3.3.3. 閲覧者のモデルの概要

システム設計時に想定した、個々の閲覧者の動線を以下に示す。

閲覧者は、以下のステップで各部屋を移動する。

システム設計時に想定された閲覧者のモデル

- ① 閲覧者は、自分で保有する ID カードを待合室扉脇 ID 端末にかざして認証を受ける。認証をパスした場合には、待合室の扉のロックが解錠され、閲覧者は②に進む。
- ② 閲覧者は、待合室の扉を開けて、待合室内に移動する(③に進む)。
- ③ 閲覧者は、待合室内に設置された表示装置を確認する。自分の氏名(または、ID カード番号)が表示されている場合には、④に進み、表示されていない場合には、待機する(つまり、③を繰り返す)。
- ④ 閲覧者は、表示装置に表示されていた氏名(または、ID カード番号)とともに表示されていた閲覧室 X (A または B) に移動し、閲覧室 X 脇 ID 端末に自身の ID カードをかざして認証を受ける。閲覧室 X の扉のロックが解錠された場合、室内に移動する。閲覧室 X の扉のロックが解錠されなかった場合、③に戻る。

- ⑤ 閲覧者は、閲覧室 X 内にあるタッチパネルに ID カード番号を入力するとともに、閲覧室 X 内 ID 端末に自身の ID カードをかざす。認証をパスした場合、閲覧室 X ない ID 端末脇にあるボックスとり出し口のロックが解錠され、ボックスへのアクセスを行う。
- ⑥ 閲覧者は待合室に移動する。閲覧室 X でボックスにアクセスできた場合には⑦へ、閲覧室 X でボックスにアクセスできなかった場合には③へ進む。
- ⑦ 待合室の扉を開けて、外に出る。

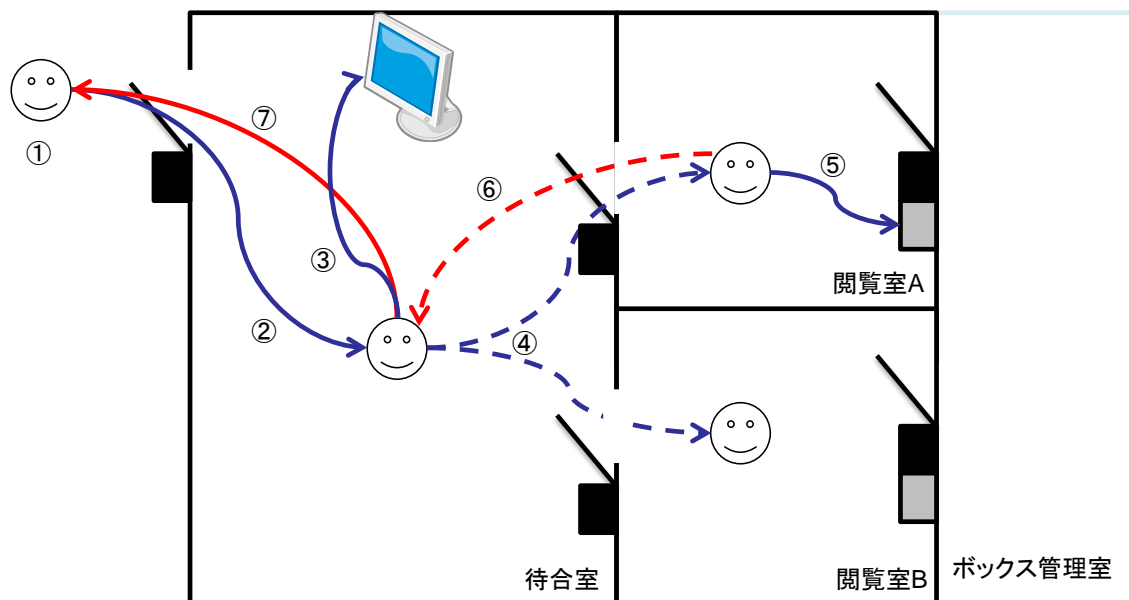


図 11-5: 閲覧者の動線 (システム設計時)

閲覧者のモデルには、システム設計時に想定していない動作も含めてモデリングを行う。これにより、システム設計時に想定しなかった閲覧者の動作に起因するエラーを検出する。

具体的には、上記の閲覧者の動作ステップを以下のように修正した。

システム設計時に想定していなかった動作も含めた閲覧者のモデル

- ① 閲覧者は、自分で保有するIDカードを待合室扉脇ID端末にかざして認証を受ける。認証をパスした場合には、待合室の扉のロックが解錠され、閲覧者は②、または、①に進む。
- ② 閲覧者は、待合室の扉を開けて、待合室内に移動する(③に進む)。
- ③ 閲覧者は、待合室内に設置された表示装置を確認する。閲覧者は、表示装置の表示内容によらず、以下のいずれかに進む。
 - A) ③に進む。
 - B) 閲覧室Aに移動する(④に進む)。
 - C) 閲覧室Bに移動する(④に進む)。
 - D) 閲覧室から出る(①に進む)。
- ④ 閲覧者は、表示装置に表示されていた氏名(または、ID カード番号)とともに表示していた閲覧室 X (A または B) に移動し、閲覧室 X 脇 ID 端末に自身の ID カードをかざして認証を受ける。閲覧室 X の扉のロックが解錠された場合、室内に移動する。閲覧室 X の扉のロックが解錠されなかった場合、③に戻る。
- ⑤ 閲覧者は、以下のいずれかの動作を行う。
 - A) 閲覧室Xから退出する(③に進む)。
 - B) 閲覧室X内タッチパネルを操作し、閲覧室X内ID端末にIDカードをかざす。認証をパスした場合、閲覧室XないID端末脇にあるボックスとり出し口のロックが解錠され、ボッ

- クスへのアクセスを行い、⑥に移動する。
- ⑥ 閲覧者は待合室に移動する。閲覧者は以下のいずれかの動作を行う。
 - A) ③に進む。
 - B) 閲覧室Aに移動する(④に進む)。
 - C) 閲覧室Bに移動する(④に進む)。
 - D) 閲覧室から出る(⑦に進む)。
 - ⑦ 待合室の扉を開けて、外に出る。

11.3.4. 各プロセスの詳細

11.3.4.1. 案内サービスプログラム

案内サービスプログラムは、待機者リスト、待合室管理プロセス、閲覧室 A 管理プロセスおよび閲覧室 B 管理プロセスから構成した。閲覧室 A 管理プロセスと閲覧室 B 管理プロセスは、同一の機能を有する閲覧室 A と閲覧室 B の状態を管理するプロセスであるため、Promela コード上の記述は同一とし、閲覧室 A、B それぞれに割り当てた ID 番号で識別するようにモデリングした。また、DB の一つの機能である、表示装置の更新に関して、待合室管理プロセス、閲覧室 A/B プロセスが、待機者リストを DB へ登録するタイミングで行うようにモデリングした。

11.3.4.1.1. 待機者リスト

待機者リストは、カード ID 番号、ボックス格納室 ID、案内閲覧室 ID、案内時間を 1 つのレコードとするリストである(表 11-3)。Promela のコード上では、1 つのレコードを parsonData 型として定義し、これを配列として扱うようにモデリングした(図 11-6)。なお、parsonData 型の案内時間 guide_time は Promela コード上では利用していないダミーコードである。

```

typedef parsonData {
    int card_id;           /* カード ID */
    int box_room_id;      /** ボックス格納室 ID */
    int booth_room_id;    /** 案内閲覧室 ID*/
    int guide_time;       /** 案内時刻 */
};

```

図 11-6: parsonData 型の定義

11.3.4.1.2. 待合室管理プロセス

待合室管理プロセスの状態遷移表を表 11-8 に示す。
 待合室は閲覧者が誰もいないとき(不在状態: ABSENCE)と、誰か 1 名以上いるとき(在室状態: PRESENCE)の 2 つの状態を定義した。この理由として、待合室に設置されている在室センサは、人の存在の有無(「誰もいない」または「1名以上人がいる」)を検知することしかできないため(11.2.3.2 章参照)、待合室管理プロセスは、待合室に閲覧者が1名以上存在するすなわち「在室」状態と、待合室に閲覧者が誰もいない「不在状態」に分けられると考えられるためである。
 イベントには、チャンネル pci を通して受信する「待合室扉脇 ID 端末認証 OK」と待合室のセンサ状態 ON と OFF の3つを設定した(表 11-4)。

表 11-8: 待合室管理プロセスの状態遷移表

	待合室扉脇 ID 端末認証 OK id_terniminal_ok	在室センサ ON PREVIOUS_ROOM_SENSOR_STATE =SENSOR_ON	在室センサ OFF PREVIOUS_ROOM_SENSOR_STATE =SENSOR_OFF
不在	新規顧客登録	遷移: 在室	

ABSENCE	ボックス格納室検索 DB 登録 遷移: 在室		
在室 PRESENCE	新規顧客登録 ボックス格納室検索 DB 登録 遷移: 在室		遷移: 不在

待合室扉脇 ID 端末は、内部に記録されている閲覧者 ID 番号と入力された値との照合を行い、マッチするものが存在した場合にだけ PC インタフェースに認証 OK のメッセージを送信する(11.2.3.1.5 参照)。従って、認証 OK の場合のみ、待合室管理プロセスにメッセージが送信されてきてイベントが発生する。逆に、認証をパスしなかった場合、待合室管理プロセスにはメッセージは送信されない。このため、待合室管理プロセスを駆動させるイベントの一つは、「待合室扉脇 ID 端末認証 OK」のメッセージであると特定できる。「待合室扉脇 ID 端末認証 OK」のメッセージは、チャンネル pci として表現される PC インタフェースに{ PREVIOUS_ROOM_DOOR_ID_TERMINAL, card_id, id_terminal_ok} としてバッファされるようにモデリングされるため(11.3.4.2.1 参照)、チャンネル pci のバッファの中から PREVIOUS_ROOM_DOOR_ID_TERMINAL に一致するメッセージが存在するか確認した後、存在する場合はそのメッセージだけを受信する。

待合室に設置されているセンサの状態は、グローバル変数 PREVIOUS_ROOM_SENSOR_STATE として、この変数に ON(==1)、OFF(==0) がセットすることで、在室および不在のセンシングをモデリングしている(11.311.3.4 11.3.4.3 11.3.4.3.1 参照)。待機室管理プロセスは、PREVIOUS_ROOM_SENSOR_STATE を参照し、閲覧者の存在の有無を確認する。

□ 新規顧客作成処理

新規顧客作成処理は、待機者リスト waitingList[] に対して、待合室に新たに入室した閲覧者の parsonData 型のデータを作成して格納する。

□ ボックス格納室検索処理

ボックス格納室検索処理は、データベース(DB)プロセスに対して、新規の閲覧者のカード ID に対応するボックス格納室 ID に問い合わせを行い、戻り値を parsonData.box_room_id に代入する処理である。ただし、今回の検証対象システムではボックス格納室は 1 つであり、形式的な問い合わせとなるため、Promela コード上では、何の処理も行わないように、記述を行わなかった。

□ DB 登録処理

DB 登録処理は、DB プロセスに対して待機者リスト waiting_list[] を DB 内部にコピーすることを要求する処理である。

モデル上では、DB を削減しているため、DB 登録処理のタイミングで waiting_list[] から、次に案内すべき閲覧者 (waiting_list[i]. booth_room_id == BOOTH_ROOM_A または waiting_list[i]. booth_room_id == BOOTH_ROOM_B となっているレコード) を検索し、グローバル変数でモデル化される表示装置 (display_A, display_B) に代入することで、表示装置の切り替えを行う。

11.3.4.1.3. 閲覧室 X (X=A or B) 管理プロセス

閲覧室 X 管理プロセスの状態遷移表を表 11-9 に示す。

閲覧室 X も閲覧者が在室する場合(在室状態: PRESENCE)と、誰も不在している場合(不在状態: ABSENCE)で動作が異なるため、2 つの状態を定義した。これは、特に在室センサが OFF のときに、案内者決定処理を行う部分が顕著に異なるためである。

チャンネル pci を通して受信するイベントとしては、閲覧室 X 扉脇 ID 端末認証 OK (BOOTH_ROOM_DOOR_X_ID_TERMINAL, card_id, id_terminal_ok)、及び、閲覧室 X 内タッ

チパネル脇 ID 端末認証確認 (INSIDE_BOOTH_ROOM_X_ID_TERMINAL, card_id, check_id_number) がある。また、グローバル変数で定義した BOOTH_ROOM_X_SENSOR_STATE を参照することで受信するイベントとして、閲覧室 X 在室センサ ON (BOOTH_ROOM_X_SENSOR_STATE_SENSOR == ON) 及び閲覧室 X 在室センサ OFF (BOOTH_ROOM_X_SENSOR_STATE_SENSOR == OFF) がある。

表 11-9: 閲覧室 X プロセスの状態遷移表

	閲覧室扉脇 ID 端末認証 OK id_terniminal_ok	閲覧室 X 内タッチ パネル脇 ID 端末 認証確認	在室センサ ON PREVIOUS_ ROOM_SENSOR _STATE =SENSOR_ON	在室センサ OFF PREVIOUS_ ROOM_SENSOR _STATE =SENSOR_OFF
不在 ABSENCE	閲覧室扉認証処理 遷移: 在室			案内者決定 (待機者リスト更新) DB 登録
在室 PRESENCE		閲覧室内認証		閲覧室退出処理 遷移: 不在

□ 閲覧室扉認証処理

閲覧室扉認証処理は、閲覧室 X に入室した閲覧者を管理するために、入室した閲覧者のカード ID 番号、入室時刻を記録し、閲覧室 X 入室者リストとして管理を開始する処理である。Promela のコード上では、カード ID 番号 card_id、入室時刻 time で構成される BoothRoomParsonData 型 (図 11-7) を定義し、閲覧室 X 管理プロセスの内部変数として管理する形とした。なお、time は形式的に定義しているのみであり、初期化時は NULL 値を代入することとし、処理の過程では使用していない。

```

typedef BoothRoomParsonData      {
    int card_id;      /* カード ID 番号 */
    int time;        /* 入室時刻 */
};

```

図 11-7: BoothRoomParsonData 型の定義

□ 閲覧室内認証処理

閲覧室 X 内タッチパネル脇 ID 端末に入力されたカード ID 番号と、実際に案内サービスプログラムが表示装置を通して案内した閲覧室 X とカード ID 番号 (つまり、待機者リストの該当するカード ID 番号と記録されている案内閲覧室 ID) が一致しているか否かを検査する。検査をパスすれば、id_terminal_ok のメッセージをチャンネル pc_controller (制御 PC に相当、後述) を通して、閲覧室 X に入室した閲覧者に送信する。検査をパスしない場合、id_terminal_ng のメッセージを pc_controller を通して、閲覧室 X に入室した閲覧者に送信する。

□ 閲覧室退出処理

閲覧室 X 管理プロセスの内部変数である入室者リストに記録されている情報、つまり、閲覧室 X に入室していると考えられる閲覧者の card_id 番号を基に、待機者リスト waitingList[] から該当する閲覧者のレコードを削除すると共に、閲覧室 X 管理プロセスの入室者リストを初期化する。

□ 案内者決定処理

待機者リスト waitingList[] から未案内の閲覧者、つまり、案内閲覧室 ID (booth_room_id) に

閲覧室 ID がセットされておらず、NULL 値 (Promela コード上は NL) となっているレコードを、待機者リストの先頭から探索する。該当するレコードが存在する場合には、そのレコードの booth_room_id に当該閲覧室の ID をセットする。該当するレコードが存在しない場合には、何もしない。

11.3.4.2. PC インタフェース

モデリングの上では、PC インタフェースは、各 ID 端末からのメッセージを送信するためのチャンネルと、各在室センサの状態を記録するための変数によりモデリングした。

11.3.4.2.1. チャンネル pci

各 ID 端末からの信号を、メッセージに変換して案内サービスプログラムに送信する PC インタフェースの機能は、単純なメッセージ変換機能であるため、Promela コード上ではモデリングではチャンネル pci としてモデル化をおこなった (図 11-8)。なお、pci チャンネルのサイズを 256 としたのは、待合室や閲覧室 A、B、閲覧者のプロセスからのメッセージを十分に捕捉できるようにし、チャンネルのオーバーフローが起こらないようにするためである。

```
#define N_PCI      256
chan pc_interface = [N_PCI] of { int, int, mtype };
```

図 11-8: チャンネル pci の定義

チャンネル pci の第 1 引数は各 ID 端末に割り振った ID 番号 (表 6 10)、第 2 引数は閲覧者が保有するカード ID 番号、第 3 引数は ID 端末から送信されるメッセージ (表 11-1) となっている。チャンネル pci をこの形式としたのは、待合室管理プロセスおよび閲覧室 X プロセス (X=A or B) のそれぞれが受信するべきメッセージを自動で判別するために、端末の ID 番号を識別子として用いるためである。これは、Promela/SPIN では、チャンネルの第 1 引数を条件として指定することで、該当するメッセージのみを受信する制御が行えるためである。

表 11-10: 各 ID 端末の ID 番号割り振り

ID 端末	定数名称	端末 ID 番号 (識別子)
待合室扉脇 ID 端末	PREVIOUS_ROOM_DOOR_ID_TERMINL	101
閲覧室 A 扉脇 ID 端末	BOOTH_ROOM_A_DOOR_ID_TERMINAL	201
閲覧室 B 扉脇 ID 端末	BOOTH_ROOM_B_DOOR_ID_TERMINAL	301
閲覧室 A 内 ID 端末	INSIDE_BOOTH_ROOM_A_ID_TERMINAL	202
閲覧室 B 内 ID 端末	INSIDE_BOOTH_ROOM_A_ID_TERMINAL	302

各 ID 端末から送信可能なメッセージ (表 11-11) は、id_terminal_ok と check_id_number の 2 通りを定義した。待合室扉脇 ID 端末および閲覧室 X 扉脇 ID 端末では、ID 端末内部で ID カード番号の照合を行い、扉のロック/アンロックを制御し、成功の場合についてのみ、待合室管理プロセスまたは閲覧室管理プロセスで処理を行うため (11.2.3.1.5 節、11.2.3.1.6 節)、id_terminal_ok のみを定義した。check_id_number は、閲覧室 X 管理プロセスにおける閲覧室内認証処理 (11.3.4.1.3 節) において利用する。閲覧室内認証処理では、閲覧室 X 内 ID 端末から入力された ID 番号が、待機者リストにおいて当該閲覧室に案内されたとされる ID 番号と同一であるか検査するため、check_id_number を定義した。

表 11-11:ID 端末から送信可能なメッセージ

メッセージ	意味	送信可能 ID 端末
id_terminal_ok	ID 端末において認証 OK であったことを示す。	待合室扉脇 ID 端末 (101) 閲覧室 A 扉脇 ID 端末 (201) 閲覧室 B 扉脇 ID 端末 (301)
check_id_number	入力されたカード ID 番号の検証を行うことを要求する。	閲覧室 A 内 ID 端末 (202) 閲覧室 B 内 ID 端末 (302)

11.3.4.2.2. 待合室扉脇ID端末

案内サービスプログラムでは待合室扉脇 ID 端末で認証成功した場合のメッセージ id_terminal_ok の場合のみ処理を行うため(11.2.3.1.5 節)、プロセス数の削減を目的として、閲覧者のモデルに含めて実装することとした(図 11-9)。

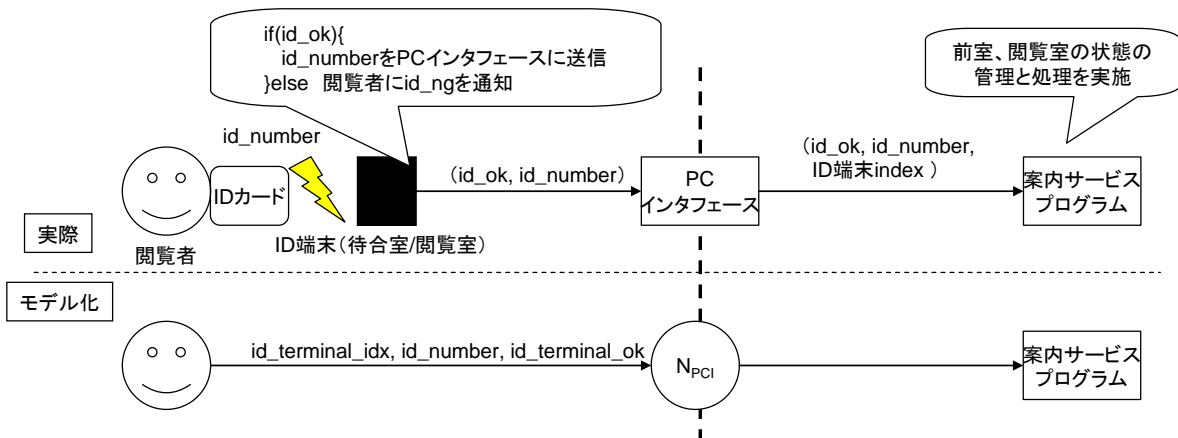


図 11-9:扉脇 ID 端末認証装置のモデル化

11.3.4.2.3. 閲覧室X扉脇ID端末

閲覧室 X 扉脇 ID 端末は、基本的には待合室扉脇と同様のモデリング(11.3.4.2.2 節)を行っている。ただし、閲覧室内 X に誰か別の閲覧者が存在する場合には、閲覧室 X の扉は開錠されない(表 11-6: 閲覧室 X の扉の解錠条件表 11-6)ため、閲覧室 X の在室センサ BOOTH_ROOM_X_SENSOR_STATE==SENSOR_ON の場合は、チャンネル pci に対してメッセージ id_terminal_ok は送信しない。逆に

BOOT_ROOM_X_SENSOR_STATE==SENSOR_OFF の場合は、pci に対して id_terminal_ok を送信する。

11.3.4.2.4. 閲覧室X内タッチパネル

閲覧室 X 内タッチパネルは、前提 1-2-1 よりモデリングに含めず、前提 1-2-3 から閲覧室 X 内 ID 端末の処理に含めることとした。

11.3.4.2.5. 閲覧室X内ID端末

閲覧室 X 内 ID 端末は、待合室扉脇 ID 端末および閲覧室 X 扉脇 ID 端末のモデリング(各 11.3.4.2.2 節、11.3.4.2.3 節)と同様に閲覧者のモデルに含めるが、閲覧室 X 管理プロセスにおいて閲覧室内認証処理(11.3.4.1.3 節)を行うため、チャンネル pci には、{閲覧室 X 内 ID 端末 ID 番号、id_card_number、check_id_number}を送信する。

11.3.4.3. 室内センサ

11.3.4.3.1. 在室センサ

在室センサは、人感センサであり、設置された室内に閲覧者が 1 人以上存在する場合、センサ内部の状態を ON に、誰もいなくなった場合 (OFF) に OFF にし、PC インタフェースに信号を送信する(11.2.3.2.1 節)。

Promela/SPIN におけるモデリングにおいては、周期的に待合室管理プロセスおよび閲覧室 X プロセスが各々に対応する在室センサの状態を監視することもあり、それぞれの在室センサの状態を PREVIOUS_ROOM_SENSOR_STATE および BOOTH_ROOM_X_SENSOR_STATE (X=A or B)として定義し、入室時に閲覧者プロセスが SENSOR_ON とすることとした。

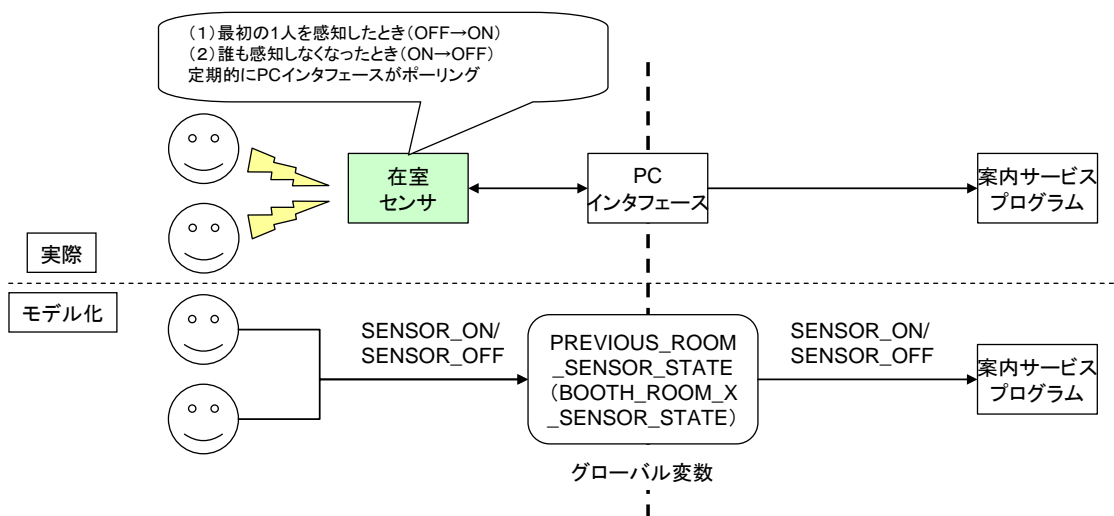


図 11-10: 在室センサのモデリング

退出時には待合室と閲覧室 X で処理内容が異なる。待合室は複数人在室可能であるが、閲覧室 X は仕様上、1 名のみ在室可能である。すなわち、待合室から退出する閲覧者プロセスに含まれる在室センサ処理では、待合室に在室している閲覧者の人数が、自身が退出することで 0 以下となる場合には、PREVIOUS_ROOM_SENSOR_STATE を SENSOR_OFF にセットし、1 以上の場合には何もしない。待合室の在室人数については、グローバル変数として HEAD_COUNT_OF_PREVIOUS_ROOM を定義している。一方で、閲覧室 X からの退出処理では、退出する場合 BOOTH_ROOM_X_SENSOR_STATE を SENSOR_OFF にセットし、在室人数の確認は行わない。

11.3.4.3.2. 火災センサ

火災センサは、前提 0-1 からモデルには含めない。

11.3.4.4. データベース(DB)

データベース(DB)は、待合室管理プロセスおよび閲覧室 X (X=A or B) 管理プロセスからの要求メッセージに応じて、受動的に処理を行うプロセスである。このため、プロセス数を削減するために、待合室管理プロセスおよび閲覧室 X 管理プロセスに組み込むこととした(図 11-11)。

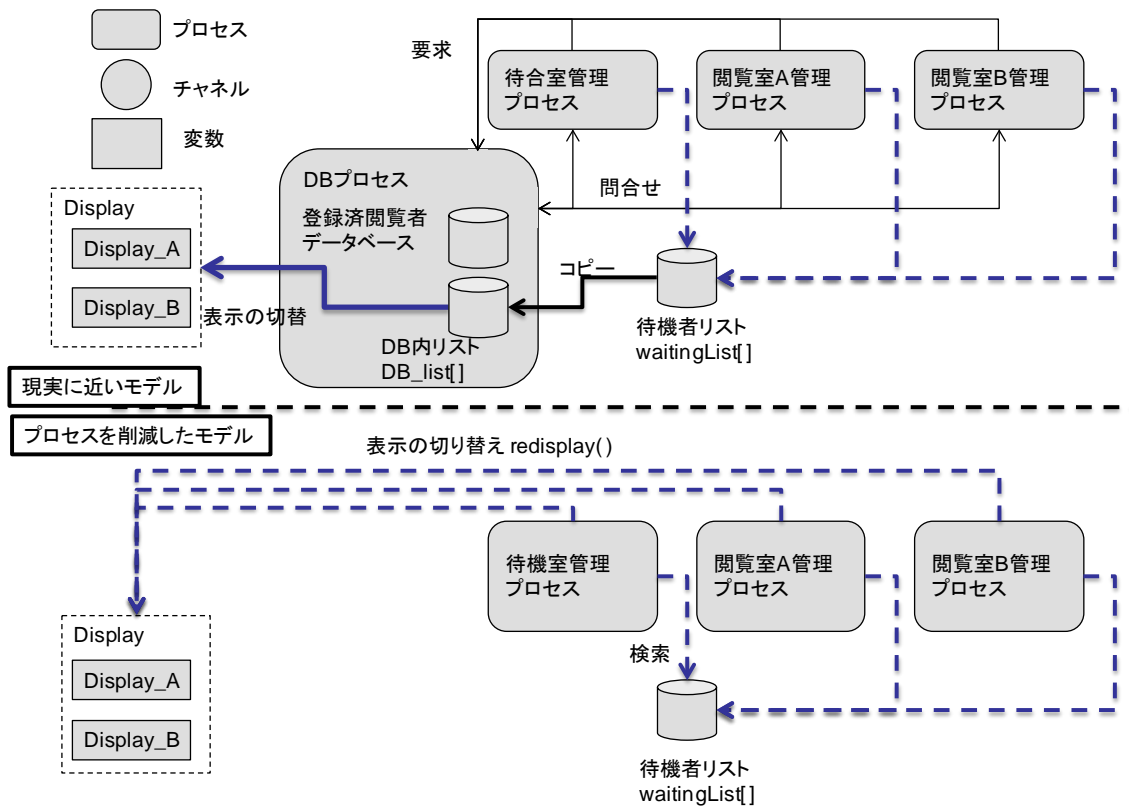


図 11-11: データベース(DB)と表示装置のモデル化

実際のシステムでは、各管理プロセス(待合室管理プロセス、閲覧室 X 管理プロセス)からの問合せや処理要求に応じて、DB は内部で管理している 2 つのデータベースそれぞれに対して、参照や操作を行う。各部屋の管理プロセスにおける処理と処理対象をまとめた表を表 11-12 に示す。

表 11-12: 各管理プロセスの処理と処理対象

処理主体	処理内容	対象			
		待機者リスト		DB(内データベース)	
		参照	操作	参照	操作
待合室管理プロセス	新規顧客登録		○		
	ボックス格納室検索			○	
	DB 登録				○
閲覧室 X 管理プロセス	閲覧室扉認証				
	閲覧室内認証			○	
	案内者決定	○			
	DB 登録				○
	閲覧室退出		○		
DB	ボックス格納室検索			○	

プロセス	DB 登録				○
	表示装置表示内容更新			○	

DB プロセスを削減する際に問題となるのは、以下の点である。

- 待機者リスト `waitingList[]` の内容をコピーした DB 内リスト `DB_list[]` の更新タイミングの間にずれがあり、検証結果に影響はないか（つまり、DB プロセスを捨象したために、検証結果に違いが出ないか）

DB 内部の DB 内リスト `DB_list[]` は、待合室管理プロセスまたは閲覧室 X プロセスから DB 登録処理を要求された時点での待機者リスト `waitingList[]` の内容をコピーしたものである。また、表示装置の表示内容の更新は一定間隔で `DB_list[]` の内容を基に行われる。従って、DB プロセスを捨象するに当たって、`DB_list[]` が削減されるに当たり、モデル上では以下が上記の懸念に対する具体的な内容となる。

- DB 内リスト `DB_list[]` は待機者リスト `waitingList[]` の内容が更新された時点でその内容を正確に反映するものとみなすことができるか

DB 内部の DB 内リスト `DB_list[]` は、待機者リスト `waitingList[]` の (DB 登録が要求された時点での) 内容をコピーしたリストである。待機者リスト `waitingList[]` の操作更新が行われた後、即座に DB 登録処理が行われるのであれば（つまり `atomic` として実装できるのであれば）、`waitingList[]` と `DB_list[]` はほぼ等価であると言えるだろう。ここで、`waitingList[]` の更新（登録、削除）は、人間である閲覧者が待合室に入室した時点（厳密には、閲覧者が待合室扉脇 ID 端末に認証をパスしたタイミングで実施される新規顧客登録の処理過程）、および、閲覧者が閲覧室 X から退出した時点（厳密には、閲覧室 X の在室センサが OFF であることを検知した時点で実施される閲覧室退出処理の処理過程）で行われる。このため、それぞれの処理（新規顧客登録および閲覧室退出処理）が行われる頻度は、閲覧者が待合室に入出するタイミングと閲覧室 X から退出するタイミングにほぼ等しく、`waitingList[]` が操作されたタイミングと DB 登録が実施されるタイミングの間に、これらの処理が行われることはほぼないだろう。従って、`waitingList[]` の操作の後、即座に DB 登録処理が行われる（つまり、`atomic`）であると仮定してモデリングしても今回の検証では問題ないと考えられる。

表示装置 (Display)

案内の表示を行う表示装置は、DB 内に保管されている待機者リスト `waitingList` をコピーしたリストから、定期的に関覧室 A および B への最新の案内者を表示する。

表示装置のモデリングでは、閲覧室 A へ案内する閲覧者の ID 番号をグローバル変数 `display_A` に、閲覧室 B へ案内する閲覧者の ID 番号をグローバル変数 `display_B` に、それぞれ代入するようにした。`display_A` および `display_B` への値の代入は、待合室管理プロセス、閲覧室 A 管理プロセスおよび閲覧室 B 管理プロセスの 3 つのプロセスが、インライン関数として実装した `redisplay()` の処理を通して行う。`redisplay()` は待機者リスト `waiting_list[]` から次に案内すべき閲覧者の ID 番号を検索し、`display_A` または `display_B` にその値を代入する。

各閲覧者は、`display_A` または `display_B` に代入されている値を参照し、自身に割り当てられている ID 番号が `display_A` の値と一致する場合は閲覧室 A へ、`display_B` の値と一致する場合には閲覧室 B へ移動する。`display_A` および `display_B` に自身に割り当てられた ID と一致する値がなかった場合、待機室にとどまる。

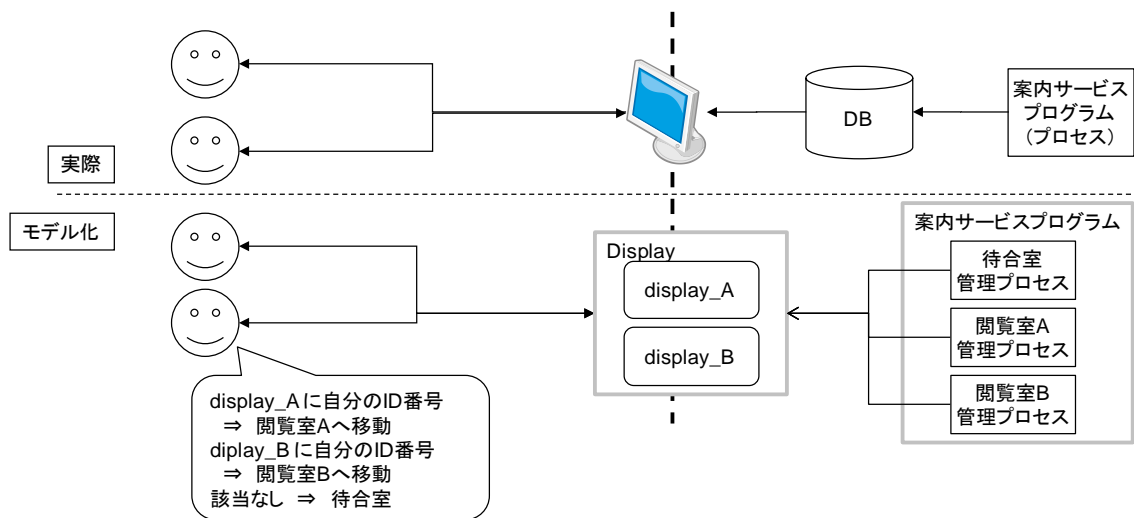
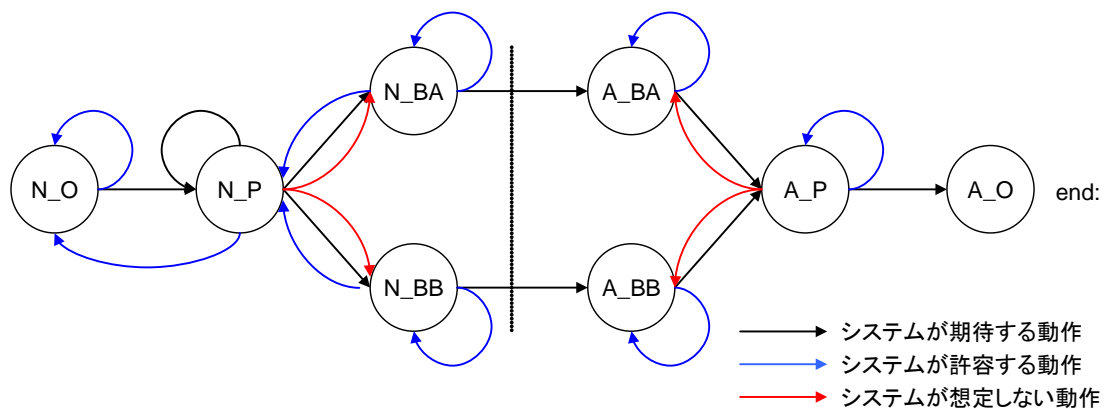


図 11-12: 表示装置のモデリング

11.3.4.5. 閲覧者

閲覧者は、検証対象システムの外部環境としてモデリングする。外部環境は、物理的に起こりえるあらゆる状態を網羅した形でモデリングする。ここではシステムの想定以外の動作を閲覧者が行うことを前提として、システムで保証できる検証性質とそうでないものを区別することを目的として、閲覧者をモデリングすることとした。このため、閲覧者のモデルでは、検証対象システムのモデルに合わせて状態を細分化し、その状態間の遷移によって閲覧者の動作を **Promela** コード上で制御する方針とした。

実際のシステムで期待する閲覧者は、システム外部から待合室に入り、表示装置に案内される案内にそって閲覧室 **X** に移動し、閲覧室内 **ID** 端末およびタッチパネルの認証をパスして自身のボックスを取得し、閲覧室 **X** を退出して待合室に戻り、システムの外部に出て行き、一連の処理を終了する。ここで、例えば、表示装置の内容を無視して閲覧室 **X** に移動したり、ボックスを取得した後に閲覧室 **X** に戻るなどは、システムが想定しない閲覧者の動作である。他方、ボックスにアクセスしていないときにシステムの外に出ることや、閲覧室 **X** 内に留まること、ボックス取得後に待合室内に留まることなどは、システムが期待はしていないが許容されるべき閲覧者の動作である。これらの考察を基にして構成した、大枠での閲覧者のオートマトンを図 11-13 に示す。



ラベル	状態		ラベル	状態	
	ボックス	部屋		ボックス	部屋
N_O	×	システムの外	A_O	○	システムの外
N_P	×	PREVIOUS_ROOM(待合室)	A_P	○	PREVIOUS_ROOM(待合室)
N_BA	×	BOOTH_ROOM_A(閲覧室A)	A_BA	○	BOOTH_ROOM_A(閲覧室A)
N_BB	×	BOOTH_ROOM_B(閲覧室B)	A_BB	○	BOOTH_ROOM_B(閲覧室B)

図 11-13: 閲覧者のオートマトン(大枠)

また、待合室における閲覧者を詳細に見ていくと、待合室入室後 (INTO)、表示装置を見て (WATCH)、自分が案内されている場合はその閲覧室の扉にむかい (TO_DOOR_A または TO_DOOR_B)、閲覧室 X 扉脇 ID 端末で認証の処理を行う。案内されていない場合には、案内されるまで待機する (WAIT)。待機状態 WAIT から TO_DOOR_A または TO_DOOR_B に直接移動することや、表示装置閲覧 WATCH において表示装置 display の内容を無視して TO_DOOR_A または TO_DOOR_B に直接移動することもありえるが、この動作はシステムが想定していない動作である。他方、WAIT の際に所要を思い出すなどして外にある (LEAVE) や閲覧室 X の扉まで行って戻るとは、システムとしては期待していないが許容できる動作である。異常の考察から、待合室における閲覧者のオートマトンを図 11-14 とした。

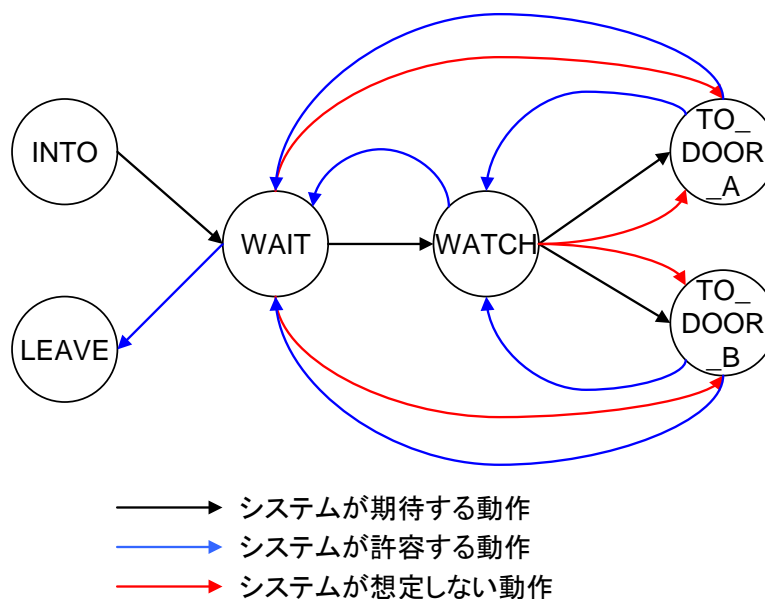


図 11-14: 待合室における閲覧者のオートマトン

11.4. 検証性質の形式記述

11.4.1. 要求仕様と検証項目

上述0節で述べた検証対象システムの要求仕様と、11.3節で述べた Promela/SPIN によるモデルに適用する検証性質との関係を表 11-13 に示す。検証性質については、モデルの検証に適した形に翻訳している。ここに挙げた検証性質のうち、「しないこと」に分類される検証性質を中心に検証を行った。

表 11-13: 本ケーススタディにおける要求仕様と検証性質

分類	要求仕様	No.	検証性質	備考	検証対象
すること	ID カードを登録している人は、待合室に入ることができる			簡略版モデルでは、全ての閲覧者は登録済み(前提 1-5)	×
	待合室に入った人は、必ず案内される	L01	待合室に入室した人は、いつか必ず案内される	—	○
	閲覧室の扉認証は、閲覧室内が空室なら誰でも成功する			センサを閲覧者のモデルに組み込んでいるため	×
	案内された人は、一定時間、閲覧室のボックス取り出しの認証を自分の ID カードでパスできる。	L02	閲覧室に案内された人だけが、タッチパネル脇 ID 端末で、いつか必ず認証 OK となる	簡略版モデルでは、時間を取り扱っていない(前提 1-1)	○
		L03	閲覧室から退出した閲覧者は、いつか必ず待機者リストから削除される	システム構築ベンダー殿からの要請	○
しないこと	閲覧室に人が入室すると、その人が退室するまで、閲覧室のドア認証に成功しない。(最終的に保証すべき安全性)	S01	閲覧室に案内されていない人が、タッチパネル脇 ID 端末で認証を行っても、常に認証をパスしない	簡略版モデルでは、時間を取り扱っていない(前提 1-1)	◎
	閲覧室が空室の時以外、閲覧室に人を案内しない。	S02	閲覧室 X が空室のとき以外、常に案内は変更されない	—	◎
	利用者は、どの部屋にも閉じ込められない。(常に、退出することが出来る。)			簡略版モデルでは閲覧室の扉を取り扱っていない(前提 1-3)	×
		S03	待機者リストは常にオーバーフローすることは無い	コードデバック中に index エラーを見つけたため	◎

◎: 検証済み

○: 検証対象だが、未検証

×: 検証対象外

11.4.2. 検証方法

11.4.2.1. 検証内容 S01

検証内容「閲覧室に案内されていない人が、タッチパネル脇 ID 端末で認証を行っても、常に認証をパスしない」について、その検証方法を説明する。

S01 の表現を変え、「常に閲覧室 X に案内された人だけが、閲覧室 X 内タッチパネル脇 ID 端末の認証をパスする」を示すことにする。

閲覧室は閲覧室 A および閲覧室 B の 2 つがあり、閲覧室 X (A または B) に案内された閲覧者が、閲覧室 X 内タッチパネル脇 ID 端末で認証行為を行うと、いつか必ず制御 PC (チャンネル pc_controller) を通して id_number_ok を受信する。従って、Promela コード上の閲覧者プロセスにおいて、チャンネル pc_interface に check_id_number メッセージを送信する部分と、pc_controller から閲覧者プロセスが id_number_ok を受信する部分に、それぞれ SEND_CHECK_ID_X および GET_ID_OK_X のラベルを付す。

案内表示装置 (チャンネル display) において閲覧室 X へ案内している ID については、display に送信後 DISPLAY_ID_A にその id_number を記録しておき、閲覧者プロセスが所持している id と比較する。閲覧者プロセスが所持する id への参照は「プロセス名[_pid]:id」の形式で行うことができる。

閲覧室 X は、閲覧室 A または閲覧室 B であることから、それぞれに関して LTL を構成し、最終的には下記のようになる。

```
! □ (((Human[6]:id == DISPLAYING_ID_A) &&  
□ (Human[6]@SEND_CHECK_ID_A → ◇ Human[6]@GET_ID_OK_A)) ||  
((Human[6]:id == DISPLAYING_ID_B) &&  
□ (Human[6]@SEND_CHECK_ID_B → ◇ Human[6]@GET_ID_OK_B)))
```

図 11-15: 検証内容 S01 の LTL 式

11.4.2.2. 検証内容 S02

検証内容「閲覧室 X が空室のとき以外、常に案内は変更されない」について説明する。

ここでは、「閲覧室 X が在室状態になったとき、いつか必ず案内が変更されることは常でない」を示すことにする。

閲覧室 X の在室については、在室センサが ON のとき、つまり、BOOTH_ROOM_X_SENSOR_STATE==SENSOR_ON の場合を考える。このとき、「いつか必ず案内が変更される」という命題を、応答性を用いて LTL の記述を行う。応答性は、命題 p, q を用いて $p \rightarrow \Diamond q$ (p が成立したならば、いつか必ず q が成り立つ) で記述する。

次に、「案内は変更される」という命題について検討する。モデル上では、案内装置 (display) は DB 登録処理のタイミングでチャンネル display をリフレッシュ (チャンネル display に格納されているメッセージを全て DB プロセスで破棄) して、最新の案内 (閲覧室 A および閲覧室 B、それぞれに対応する待機者リストのカード ID の表示) をチャンネル Display に送信することとなり、厳密な意味では、DB 登録が行われると常に案内の内容は変更されることになる。ここでの「案内が変更される」は、「チャンネル Display の内容が更新前後で変っていること」である。閲覧室 X の案内内容の変更の有無を DISPLAY_X_CHANGED として定義した上で、DB 登録処理実施前後の閲覧室 X への案内閲覧者 ID を変数 new_booth_X, old_booth_X に記録して差異を比較する。異なれば DISPLAY_X_CHANGED に true を代入、同じであれば DISPLAY_X_CHANGED に false を代入するように DB における DB 登録処理直後に記述を追加する。

```
! □ ((BOOTH_ROOM_X_SENSOR_STATE==SENSOR_ON) →  
◇ (DISPLYA_X_CHANGED==true))
```


図 11-16: 検証内容 S02 の基本となる LTL 式

閲覧室 X は閲覧室 A および閲覧室 B の 2 つがあるため、閲覧室 A または閲覧室 B のいずれかが在室のとき案内が変更されることはないことを示す必要がある。従って最終的には、下記のようになる。

```
!□(((BOOTH_ROOM_A_SENSOR_STATE==SENSOR_ON)
  → ◇(DISPLAY_A_CHANGED==true)) ||
  ((BOOTH_ROOM_B_SENSOR_STATE==SENSOR_ON)
  → ◇(DISPLAY_B_CHANGED==true)))
```

図 11-17: 検証内容 S02 の LTL 式

11.4.2.3. 検証内容 S03

検証内容「待機者リストは常にオーバーフローすることはない」の検証方法について、説明を行う。

本検証を行う方法として、上記と同様に検証記述を行う方法と `assert` 文を用いて不具合を自動検出する方法が考えられる。本検証内容は、「待機者リスト `waiting_list[]` への登録を行う処理では、常に待機者リスト `waiting_list[]` の大きさ未満である」と言い換えることができる。つまり、`waiting_list[]` への登録を行う(データの登録を行う)時点で、エラーの検出ができればよいため、検証記述を用いるよりも簡単な `assert` 文を用いた自動検証により検証を行う方針を採用することとする。

待合室における Promela 記述を図 11-18 に示す。これに図 11-19 に示すように、待機者リストへの登録(`registrate_waitingList()`)を実行する直前に、

`assert(waiting_list_idx < MAX_WAITING_LIST_SIZE)` を挿入する(図 11-19、16 行目、39 行目)。`waiting_list_idx` は、待機者リスト `waiting_list[]` に登録を行うために利用するインデックスを記憶する。待機者リスト `waiting_list[]` は配列で表現していることから、インデックスは 0 から始まるため、待機者リストの最大サイズ (`MAX_WAITING_LIST_SIZE`) - 1 までは、待合室に入室した閲覧者の登録が可能である。このため、`waiting_list_idx` が `MAX_WAITING_LIST_SIZE` 未満である場合は不具合ではなく、`assert` 文はエラーを検出しない。逆に `waiting_list_idx` が `MAX_WAITING_LIST_SIZE` 以上となる場合、オーバーフローの不具合であり、`assert` 文はエラーを検出し、トレイルファイルにその実行パターンを出力する。

```

1 active proctype Previous_Room(){
2     int room_number = PREVIOUS_ROOM;
3     int waiting_list_idx = 0;      /* 待機者リストのインデックス */
4     int list_idx = 0;             /* 待機者リスト検索 ( Display 表示時 ) 用 */
5     mtype pci_m;                 /* pc_interface から受信するメッセージ */
6     int input_id;                 /* pc_interface から受信した card_id 番号 */
7
8     ABSENCE:      /* 不在状態 */
9         if
10            :: (pc_interface?[PREVIOUS_ROOM_DOOR_ID_TERMINAL,input_id,pci_m])->
11                /* 前室扉脇 ID 端末からのメッセージを検索 */
12                pc_interface??PREVIOUS_ROOM_DOOR_ID_TERMINAL,input_id,pci_m;
13                /* 前室扉脇 ID 端末からのメッセージを受信 */
14            if
15                :: (pci_m==id_terminal_ok) ->      /* 認証 OK なら前室入室処理 */
16                    registrate_waitingList(waiting_list_idx, input_id);
17                    redisplay( );
18                    goto PRESENCE;
19                :: else -> goto ABSENCE;
20            fi;
21            :: (PREVIOUS_ROOM_SENSOR_STATE == SENSOR_OFF) ->
22                /* 周期監視在室 OFF */
23                goto ABSENCE;
24            :: (PREVIOUS_ROOM_SENSOR_STATE == SENSOR_ON)      ->
25                /*      周期監視在室 ON      */
26                goto PRESENCE;
27            :: (HC_AO == HEAD_COUNT) -> goto TERMINATE;      /* 全員 A_O なら終了*/
28            :: else -> goto ABSENCE;
29        fi;
30
31    PRESENCE:      /* 在室状態 */
32        if
33            :: (pc_interface ?? [PREVIOUS_ROOM_DOOR_ID_TERMINAL,input_id,pci_m]) ->
34                pc_interface ?? PREVIOUS_ROOM_DOOR_ID_TERMINAL,input_id,pci_m;
35                /* 前室扉脇 ID 端末からのメッセージを受信 */
36            if
37                :: (pci_m==id_terminal_ok) ->      /* 認証 OK なら前室入室処理 */
38                    /* 新規顧客作成 */
39                    registrate_waitingList(waiting_list_idx, input_id);pr:
40                    redisplay();
41                    goto PRESENCE;
42                :: else -> goto PRESENCE;
43            fi;
44            :: (PREVIOUS_ROOM_SENSOR_STATE == SENSOR_OFF) ->      /* 周期監視在室 OFF */
45                goto ABSENCE;
46        fi;
47
48    TERMINATE: end:
49        skip;
50 }

```

図 11-18:待合室の Promela 記述

```

1 active proctype Previous_Room(){

```

```

2   int room_number = PREVIOUS_ROOM;
3   int waiting_list_idx = 0;      /* 待機者リストのインデックス */
4   int list_idx = 0;             /* 待機者リスト検索 ( Display 表示時 ) 用 */
5   mtype pci_m;                  /* pc_interface から受信するメッセージ */
6   int input_id;                 /* pc_interface から受信した card_id 番号 */
7
8   ABSENCE:      /* 不在状態 */
9       if
10          :: (pc_interface?[PREVIOUS_ROOM_DOOR_ID_TERMINAL,input_id,pci_m])->
11              /* 前室扉脇 ID 端末からのメッセージを受信 */
12              pc_interface??PREVIOUS_ROOM_DOOR_ID_TERMINAL,input_id,pci_m;
13              /* 前室扉脇 ID 端末からのメッセージを受信 */
14          if
15              :: (pci_m==id_terminal_ok) ->      /* 認証 OK なら前室入室処理 */
16                  assert(waiting_list_idx<MAX_WAITING_LIST_SIZE);
17                  registrate_waitingList(waiting_list_idx, input_id);
18                  redisplay( );
19                  goto PRESENCE;
20              :: else -> goto ABSENCE;
21          fi;
22          :: (PREVIOUS_ROOM_SENSOR_STATE == SENSOR_OFF) ->
23              /* 周期監視在室 OFF */
24              goto ABSENCE;
25          :: (PREVIOUS_ROOM_SENSOR_STATE == SENSOR_ON)
26              /*          周期監視在室 ON          */
27              goto PRESENCE;
28          :: (HC_AO == HEAD_COUNT) -> goto TERMINATE;      /* 全員 A_O なら終了*/
29          :: else -> goto ABSENCE;
30      fi;
31
32   PRESENCE:      /* 在室状態 */
33       if
34          :: (pc_interface ?? [PREVIOUS_ROOM_DOOR_ID_TERMINAL,input_id,pci_m]) ->
35              pc_interface ?? PREVIOUS_ROOM_DOOR_ID_TERMINAL,input_id,pci_m;
36              /* 前室扉脇 ID 端末からのメッセージを受信 */
37          if
38              :: (pci_m==id_terminal_ok) ->      /* 認証 OK なら前室入室処理 */
39                  assert(waiting_list_idx<MAX_WAITING_LIST_SIZE);
40                  /* 新規顧客作成 */
41                  registrate_waitingList(waiting_list_idx, input_id);pr:
42                  redisplay();
43                  goto PRESENCE;
44              :: else -> goto PRESENCE;
45          fi;
46          :: (PREVIOUS_ROOM_SENSOR_STATE == SENSOR_OFF) ->      /* 周期監視在室 OFF */
47              goto ABSENCE;
48          fi;
49
50   TERMINATE: end:
51       skip;
52 }

```

図 11-19: assert 文を追記した待合室の Promela 記述

11.5. モデル検査と結果

11.5.1. 検証内容 S01 の検証結果

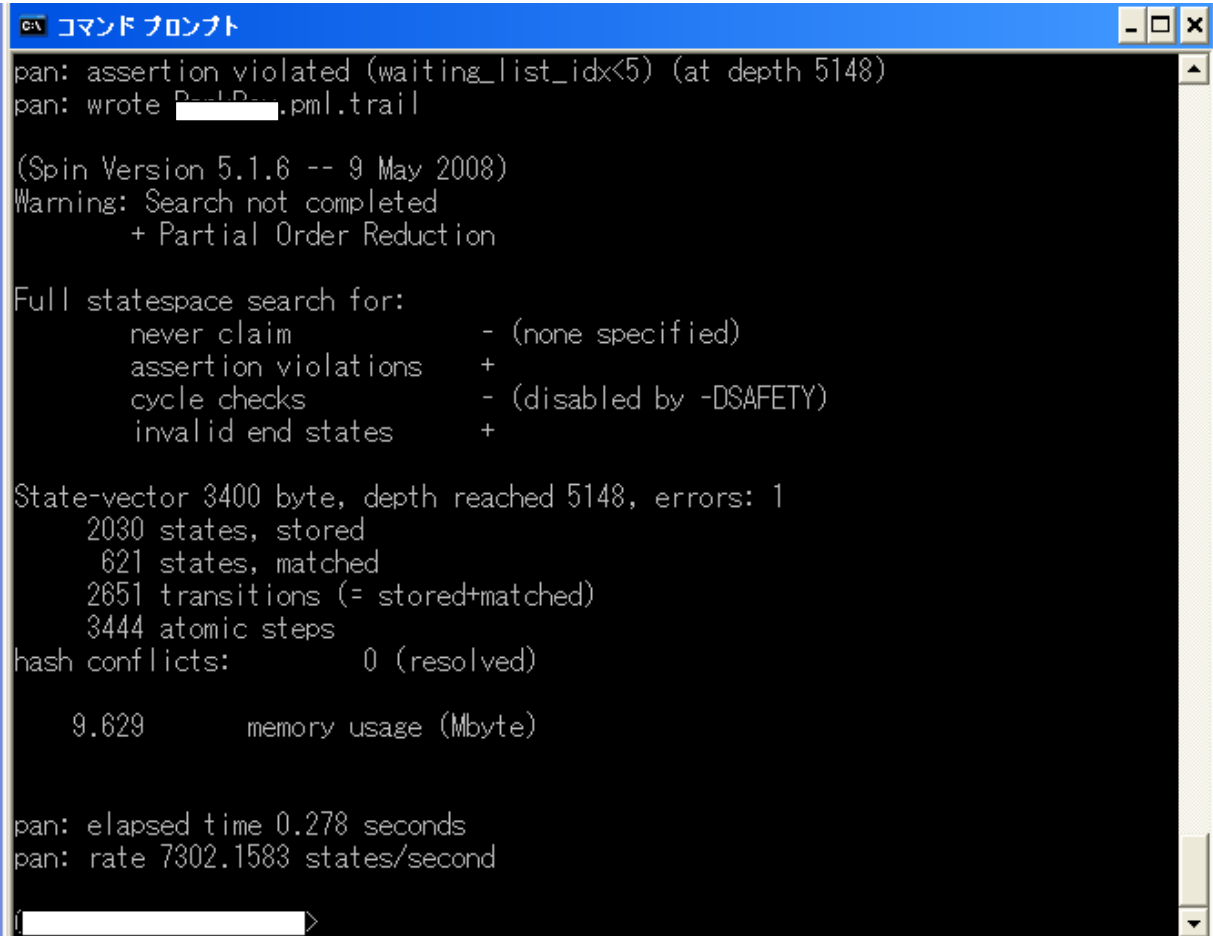
反例は見つからなかった。

11.5.2. 検証内容 S02 の検証結果

反例は見つからなかった。

11.5.3. 検証内容 S03 の検証結果

閲覧者の人数(つまり、閲覧者プロセス数)を 1、待機者リスト `waiting_list[]` の最大サイズ (`MAX_WAITING_LIST_SIZE`) を 5 として、検証を行った結果、`assert` 文においてエラーを検出した(図 11-20)。



```
コマンドプロンプト
pan: assertion violated (waiting_list_idx<5) (at depth 5148)
pan: wrote P-1-P-.pml.trail

(Spin Version 5.1.6 -- 9 May 2008)
Warning: Search not completed
+ Partial Order Reduction

Full statespace search for:
  never claim           - (none specified)
  assertion violations   +
  cycle checks          - (disabled by -DSAFETY)
  invalid end states    +

State-vector 3400 byte, depth reached 5148, errors: 1
  2030 states, stored
   621 states, matched
 2651 transitions (= stored+matched)
 3444 atomic steps
hash conflicts:      0 (resolved)

  9.629      memory usage (Mbyte)

pan: elapsed time 0.278 seconds
pan: rate 7302.1583 states/second
```

図 11-20: S03 の検証結果

図 11-20 の 11 行目に“State-Vecotor 3400byte, depth reached 5148, errors:1”とあり、エラーが検出されている。また、1 行目に“pan: assertion violated (waiting_list_idx<5) (at depth 5148)”とあることから、`assert` 文のところでエラーが検出され、次の 2 行目でその結果を“.trail”ファイルに書き込んでいる。

この“.trail”ファイルに基づき Spin のシミュレーション実行を行う。具体的には、「-t」オプションと共に、「-p」オプションを用いて、“`spin -t -p <promela_code.pml> > <result.txt>`”とプロンプト

から入力し、実行する。<promela_code.pml>には、検証に用いている Promela で記載されたファイル名を代入する。<result.txt>は実行結果の出力先ファイル名を記載する。

<result.txt>には、実行パターンが出力されている。この結果を分析することで、エラーが起きる実行パターンを解析する。

trail ファイルの分析の結果、閲覧者プロセスが外部から待合室への入室を待機者リスト `waiting_list[]` の最大サイズ `MAX_WAITING_LIST` 回繰り返すと、`waiting_list_idx > MAX_WAITING_LIST` となり、反例となることが分かった。

このエラーは、同一の閲覧者が複数回待合室への入退出を繰り返すことにより発生する。つまり、同一の閲覧者を複数回待機者リストに登録することによって発生する。従って、待機者リストへの登録の処理 (`registrate_waitingList()`) を実行する直前に、登録処理の対象となる閲覧者が待機者リストに登録されているか確認し、登録されていない場合だけ `registrate_wiatingList()` の処理を行うようにすれば、解決する。この関数をインライン関数として、実装した待合室の Promela 記述を図 11-21 に示す。図 11-21 における `check_waitingList(input_id)` がこの対処に相当する。この処理により、当該エラーは回避されることを、同様に検証器を作成して実行することで確認できた。

```

1 active proctype Previous_Room(){
2     int room_number = PREVIOUS_ROOM;
3     int waiting_list_idx = 0;    /* 待機者リストのインデックス */
4     int list_idx = 0;           /* 待機者リスト検索 ( Display 表示時 ) 用 */
5     mtype pci_m;               /* pc_interface から受信するメッセージ */
6     int input_id;              /* pc_interface から受信した card_id 番号 */
7
8     ABSENCE:    /* 不在状態 */
9         if
10            :: (pc_interface?[PREVIOUS_ROOM_DOOR_ID_TERMINAL,input_id,pci_m])->
11                /* 前室扉脇 ID 端末からのメッセージを受信 */
12                pc_interface??PREVIOUS_ROOM_DOOR_ID_TERMINAL,input_id,pci_m;
13                /* 前室扉脇 ID 端末からのメッセージを受信 */
14            if
15                :: (pci_m==id_terminal_ok) -> /* 認証 OK なら前室入室処理 */
16                assert(waiting_list_idx<MAX_WAITING_LIST_SIZE);
17                check_waitingList(input_id);
18                registrate_waitingList(waiting_list_idx, input_id);
19                redisplay();
20                goto PRESENCE;
21            :: else -> goto ABSENCE;
22            fi;
23            :: (PREVIOUS_ROOM_SENSOR_STATE == SENSOR_OFF) ->
24                /* 周期監視在室 OFF */
25                goto ABSENCE;
26            :: (PREVIOUS_ROOM_SENSOR_STATE == SENSOR_ON) ->
27                /* 周期監視在室 ON */
28                goto PRESENCE;
29            :: (HC_AO == HEAD_COUNT) -> goto TERMINATE; /* 全員 A_O なら終了*/
30            :: else -> goto ABSENCE;
31        fi;
32
33    PRESENCE:    /* 在室状態 */
34        if
35            :: (pc_interface ?? [PREVIOUS_ROOM_DOOR_ID_TERMINAL,input_id,pci_m]) ->
36                pc_interface ?? PREVIOUS_ROOM_DOOR_ID_TERMINAL,input_id,pci_m;
37                /* 前室扉脇 ID 端末からのメッセージを受信 */
38            if
39                :: (pci_m==id_terminal_ok) -> /* 認証 OK なら前室入室処理 */
40                assert(waiting_list_idx<MAX_WAITING_LIST_SIZE);
41                check_waitingList(input_id);
42                registrate_waitingList(waiting_list_idx, input_id);pr;
43                redisplay();
44                goto PRESENCE;
45            :: else -> goto PRESENCE;
46            fi;
47            :: (PREVIOUS_ROOM_SENSOR_STATE == SENSOR_OFF) -> /* 周期監視在室 OFF */
48                goto ABSENCE;
49        fi;
50
51    TERMINATE: end:
52        skip;
53 }

```

図 11-21: 対処を行った待合室の Promela 記述

12. 応用事例情報

本章では、形式手法の導入を検討する上で参考となる実システムに対する応用事例について、費用対効果や導入時の課題・工夫点等を中心にまとめる。

想定読者	CIO, 組織管理者、開発技術、発注者等
目的	ユーザが関心を持つ類似事例から、形式手法の実用度、費用対効果、課題・工夫点等の情報を提供することにより、形式手法の導入を支援する
想定知識	ソフトウェア開発プロセスの概略
得られる事	<ul style="list-style-type: none"> ● 形式手法に関する実用度および費用対効果 ● 形式手法導入時の課題とその解決策 ● 手法・ツール選択のための指針 ● 形式手法導入時における教育方法の指針

12.1. 事例の整理項目

各事例は、表 12-1 に示す項目で整理した。

表 12-1: 応用事例情報の整理項目

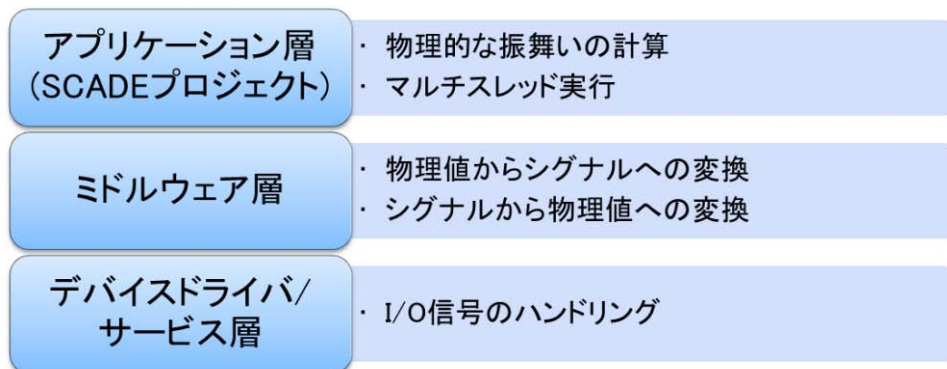
項目	概要
ドメイン	開発対象のドメイン
開発対象	形式手法を利用して開発した対象
国	開発された国
開発組織	開発を行った組織
形式手法(言語、ツール)	利用した形式手法の言語およびツール
適用範囲・規模(形式手法)	形式手法を適用した規模、全体のシステムに対する割合
適用対象のソフト種別	制御系、リアルタイム制御系、エンタプライズ系の区別
適用目的・工程	形式手法を適用した工程
実装言語	実装に利用した言語
規模(実装)	開発対象の実装コード行数
効果	形式手法を利用して得られた効果
検証内容	検証性質の具体例
検証規模	証明課題、要求仕様に対する検証率
期間	開発期間
形式手法を利用した動機	形式手法を利用して開発を行った理由
手法・ツール選択理由	具体的な手法・ツールを選択した理由
障害と工夫	導入および開発時に生じた障害とその対策方法
体制	開発を行った組織体制
教育	形式手法に関して開発メンバーに対して行った教育
その他(外部リソース他注目点)	外部リソースの利用等、その他注目すべき点
参考資料	参考文献

12.2. 事例の調査結果

以下に形式手法を利用した応用事例を示す。

12.2.1. 富士重工業 - ECU ソフトウェア開発

ドメイン	自動車
開発対象	モータ制御 ECU
国	日本
開発組織	富士重工業
形式手法(言語、ツール)	SCADE
適用範囲・規模(形式手法)	制御ソフトウェアをアプリケーション層、ミドルウェア層、デバイスドライバ/サービス層の3層に分け、このうちアプリケーション層全体に対して SCADE を適用した。
適用対象のソフト種別	リアルタイム制御系
適用目的・工程	設計、自動コード生成
実装言語	C
実装規模	不明
効果	ヒューマンエラーの排除及び生産性の向上



(出典: Masaru Kurihara, Automotive ECU software development with SCADE Suite, 2009, http://www.esterel-technologies.com/files/Automotive_ECU_Development-2010.pdf より作成)

図 12-1: ソフトウェアアーキテクチャ設計

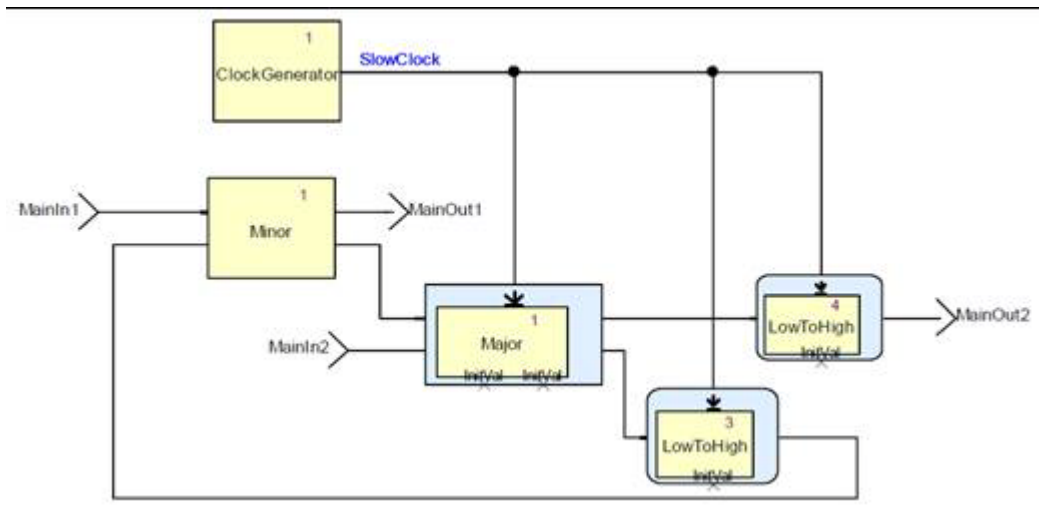
- 詳細情報
 - 検証内容
 - ◇ 異なるスレッド間におけるデータハンドリングについて一貫性が保証されることを検証した。
- 判断
 - 形式手法を利用した動機
 - ◇ 製品開発時間の短縮に対応する必要があり、従来の開発手法とは異なる開発方法が要求された。
 - 障害と工夫
 - ◇ モデルベース開発の導入をスムーズに行うため、既存のソフトウェアアーキテクチャを再設計して3層に分け、上層のアプリケーション層に SCADE を適用した。アプリ

ケーション層からは、ターゲットハードウェアや OS を隠蔽した。

- ◇ アプリケーション層と第 2 層のミドルウェア層とのインタフェースを人手で記述するとエラー混入の可能性が生じるため、スクリプトを作成して **SCADE** 内で実行することにより、インタフェース部分を自動生成した。
 - ◇ 開発初期はミドルウェアやハードウェアの仕様も頻繁に変更されるため、ミドルウェア層以下で定義され、アプリケーション層でも利用されるデータ型を手動で管理するとエラーが混入する可能性が高まる。そこで、ミドルウェア層で利用されるデータ型を自動的に抽出し、型定義ファイルを **SCADE** に提供する **Td** スクリプトを作成し、実行した。
 - ◇ **SCADE** はマルチスレッドのモデリングを対象としていないため、**SCADE** 言語を十分に理解してマルチスレッドに適用可能な安全なコードを自動生成した。
- ・ 情報源
 - CDAJ CAE Solution Conference 2008 | 講演概要, ECU ソフトウェア開発における **SCADE** の適用例(富士重工業), http://www.cdaj.co.jp/ccsc2008/lecture/scade_04.html
 - Masaru Kurihara, Automotive ECU software development with **SCADE** Suite, 2010, http://www.esterel-technologies.com/files/Automotive_ECU_Development-2010.pdf

12.2.2. Intertechnique - 航空機の燃料管理システム

ドメイン	航空運輸
開発対象	Airbus A380 の燃料管理システム
国	フランス
開発組織	Intertechnique, Esterel Technologies
形式手法(言語、ツール)	SCADE
適用範囲・規模(形式手法)	不明
適用対象のソフト種別	リアルタイム制御系
適用目的・工程	設計、自動コード生成
実装言語	不明
実装規模	不明
効果	開発工程の早い段階でシミュレーションによる検証を行うことができたため、機能の不具合が改善されたことにより、ハードウェアとの統合にかかる時間を 60% 程度削減することができた。



(出典: A. Jean-Louis Camus, Pierre Vincent, Olivier Graff, Sebastien Poussard, A verifiable architecture for multi-task, multi-rate synchronous software, 2008)

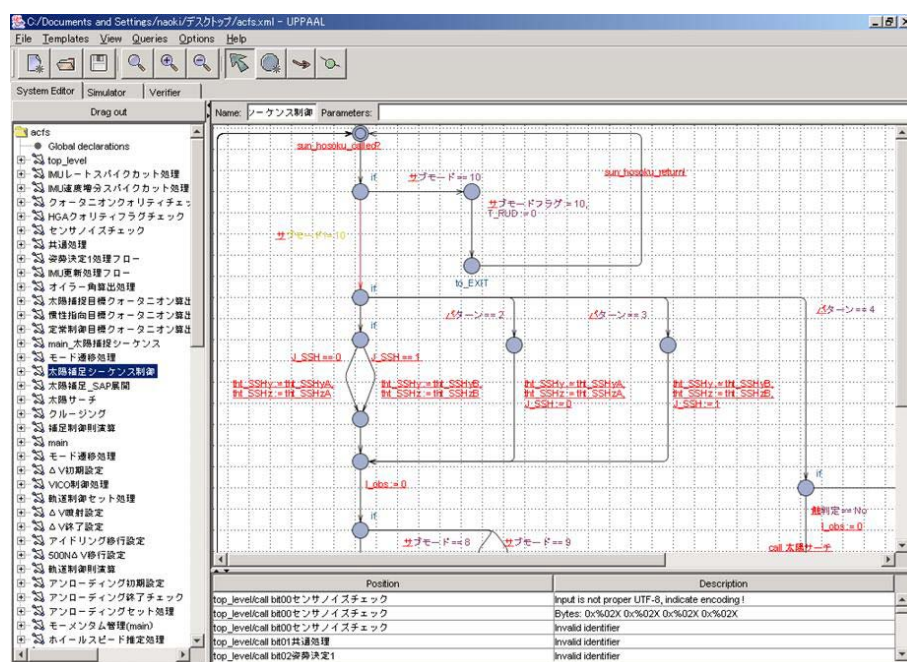
図 12-2: SCADE により作成されたスケジューリングとコミュニケーションモデル

- ・ 詳細情報
 - 検証内容
 - ◇ サブシステム間のデータ交換が正しく行われることを検証した。
- ・ 判断
 - 形式手法を利用した動機
 - ◇ 安全を保証した設計とその検証を早い段階で行い、組み込みソフトウェアのコードを生成する必要がある。
 - ◇ DO-178B 標準の要求を満たす必要がある。
 - 手法・ツール選択理由
 - ◇ 以前の A340 電子ロード管理システムの実装に成功し、その後 2 年間、Esterel Technologies の協力を得て、SCADE の利用を社内標準とした。
 - ◇ SCADE は DO-178B 標準を満たす C コード生成機能を有している。
 - ◇ SCADE は並列動作や機能の依存性の簡潔で明確な表現が可能である。
 - 障害と工夫
 - ◇ SCADE は単スレッドのコードを生成するように設計されているため、当初考案したモデル(図参照)に対応する正しいコードを生成することができなかった。そこで、自動コード生成可能な意味的に同一なモデルを作成した。
- ・ 情報源
 - Intertechnique :: Success Stories, Esterel Technologies, <http://www.esterel-technologies.com/technology/success-stories/intertechnique>
 - A. Jean-Louis Camus, Pierre Vincent, Olivier Graff, Sebastien Poussard, A verifiable architecture for multi-task, multi-rate synchronous software, 2008

12.2.3. JAXA - 人工衛星の姿勢制御ソフトウェア

ドメイン	航空運輸
開発対象	人工衛星である SELENE および WINDS の姿勢制御ソフトウェア
国	日本
開発組織	JAXA (旧 NASDA)
形式手法(言語、ツール)	UPPAAL

適用範囲・規模(形式手法)	不明
適用対象のソフト種別	リアルタイム制御系
適用目的・工程	仕様記述
実装言語	C
実装規模	不明
効果	時間制約を考慮した検証を行うことができた。



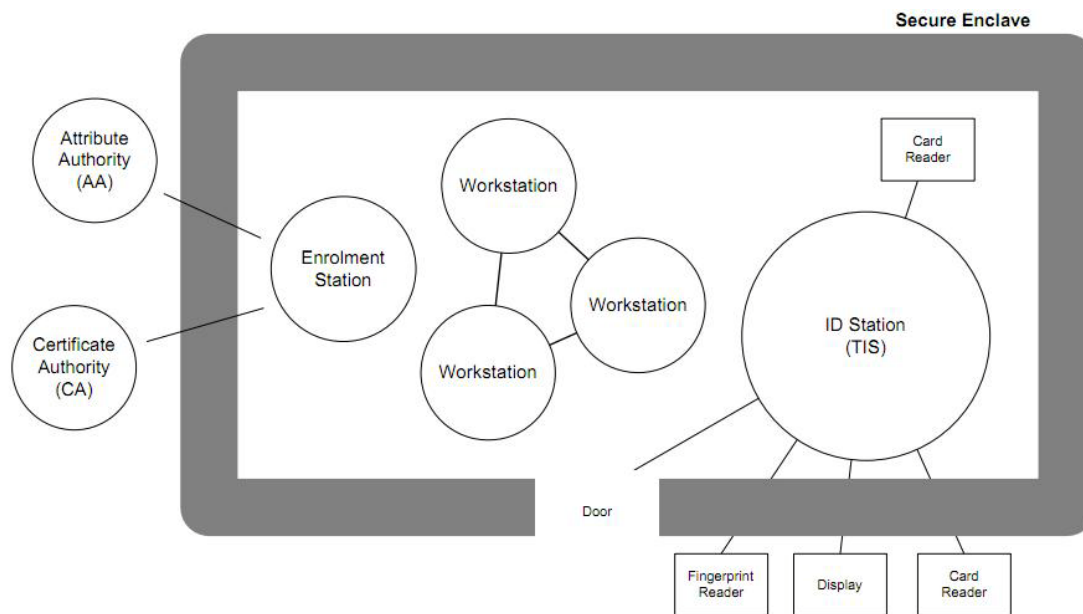
(出典: Naoki Ishihama, Hideki Nomoto, Haruka Nakao, NASDA IV&V, NASA IV&V Facility, 2003)

図 12-3: UPPAAL によるシーケンス制御モデル図

- 詳細情報
 - 検証内容
 - モデルの不変条件や到達性を検証した。
- 判断
 - 形式手法を利用した動機
 - ◇ 高信頼のシステムソフトウェアを実現する必要があった。
 - 手法・ツール選択理由
 - ◇ ソフトウェアの要求がデータフローチャートで記述されており、UPPAAL モデルはデータフローチャートであるから、UPPAAL の適用を決定した。また、時間制約を記述する必要があり、UPPAAL は時間オートマトンを拡張したネットワークをモデル化することができる。
- 組織
 - 体制
 - ◇ ソフトウェア成果物の妥当性を検証するチームが IV&V(独立検証および妥当性確認)の一環として実施した。
- 情報源
 - Naoki Ishihama, Hideki Nomoto, Haruka Nakao, NASDA IV&V, NASA IV&V

12.2.4. NSA - バイオメトリクス ID 認証を利用した入退室管理システム

ドメイン	その他
開発対象	バイオメトリクス ID 認証ツールのアクセス管理等に利用されるセキュリティソフトウェアである Tokeneer を利用した入退室管理システム
国	米国
開発組織	Praxis High Integrity Systems
形式手法(言語、ツール)	Z (fuzz type checker)、SPARK (SPARK Examiner)
適用範囲・規模(形式手法)	Z 及び英語テキスト: 形式仕様・・・118 ページ セキュリティプロパティ・・・11 ページ 形式設計・・・171 ページ
適用対象のソフト種別	制御系
適用目的・工程	仕様記述、設計、コード検証
実装言語	SPARK
実装規模	宣言・・・4964 実行行数・・・4975 SPARK flow アノテーション・・・6036 SPARK proof アノテーション・・・1999 コメント・・・8529 合計・・・30278
効果	多くの領域で EAL5 の要求以上を達成した。形式手法を利用したことが結果的に効率的であった。



(出典: David Cooper and Janet Barnes, Tokeneer ID Station EAL5 Demonstrator: Summary Report, 2008)

図 12-4: 入退室管理システムの全体像

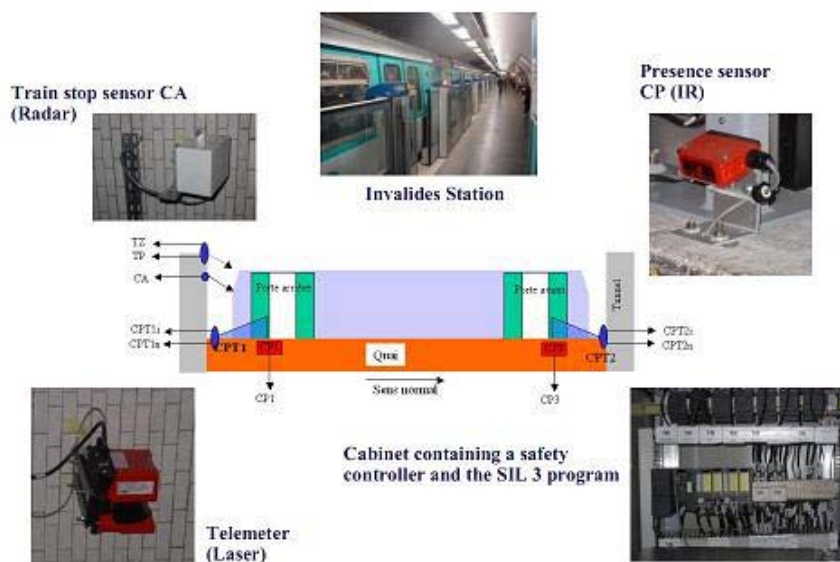
- 詳細情報
 - 検証内容
 - ◇ 部屋に入る権限を持つユーザがトークンリーダーを通して自分のトークンを TIS (Tokeneer ID Station) に提供したとき、もし、ユーザのトークンが正しいものであり、トークン内の認証と認可のデータがユーザの指紋と一致する指紋テンプレートを保有していたら、ユーザが部屋に入ったらドアが閉まりロックされ、ユーザのトークンに権限付与証明書が与えられる。等。
 - 検証規模
 - ◇ SPARK Examiner が生成した Verification Condition 292 個のうち、223 個は Examier/Simplifer を利用して自動的に証明された。14 個は Interactive Proof Checker を利用して証明し、55 個はレビューにより証明した。
 - 期間
 - ◇ 2003
- 判断
 - 形式手法を利用した動機
 - ◇ EAL5 レベルのシステムを、従来の開発プロセスよりもコストを抑えて開発することができるかどうかを実証するために形式手法を利用した。
 - 手法・ツール選択理由
 - ◇ Z を利用した理由:
 - notation がチェック可能であり、仕様作成時の小さい不具合も除去できる。
 - 仕様作成時に詳細な設計について考える必要がない。
 - 巨大なシステムを管理可能なサブコンポーネントに分割できる。
 - 型チェックや検証のためのツールサポートがある。
- 組織
 - 体制

- ◇ 3名の技術者(2人はZの読み書きができ、セキュリティ等の専門知識を有していた。1人はZの解読はできるが記述スキルは不十分であった)

- ・ 情報源
 - David Cooper and Janet Barnes, Tokeneer ID Station EAL5 Demonstrator: Summary Report, 2008
 - 形式手法適用調査 調査報告書、情報処理推進機構、2010

12.2.5. ClearSy - パリ地下鉄プラットフォームドアの制御

ドメイン	鉄道
開発対象	プラットフォームドアのコマンドコントローラ
国	フランス
開発組織	RATP、ClearSy
形式手法(言語、ツール)	B (COMPOSYS, B4Free, Atelier B, Brama Animator)
適用範囲・規模(形式手法)	1300 ページのドキュメント セーフティケースは 30 のドキュメントがあり、600 ページ B モデルは 3500 行
適用対象のソフト種別	リアルタイム制御系
適用目的・工程	システム仕様設計、ソフトウェア仕様設計、コード検証
実装言語	LADDAR
実装規模	不明
効果	テスト工程において、ソフトウェアの異常が一つも発見されなかった。



(出典: Guilhem Pouzancre, A Formal Approach in the Implementation of a Safety System for Automatic Control of "Platform Doors", 2006)

図 12-5: プラットフォームドアの構成

- ・ 詳細情報
 - 検証規模
 - ◇ 1000 の証明責務があり、90%を自動証明、残りを 2 日間で Atelier B interactive

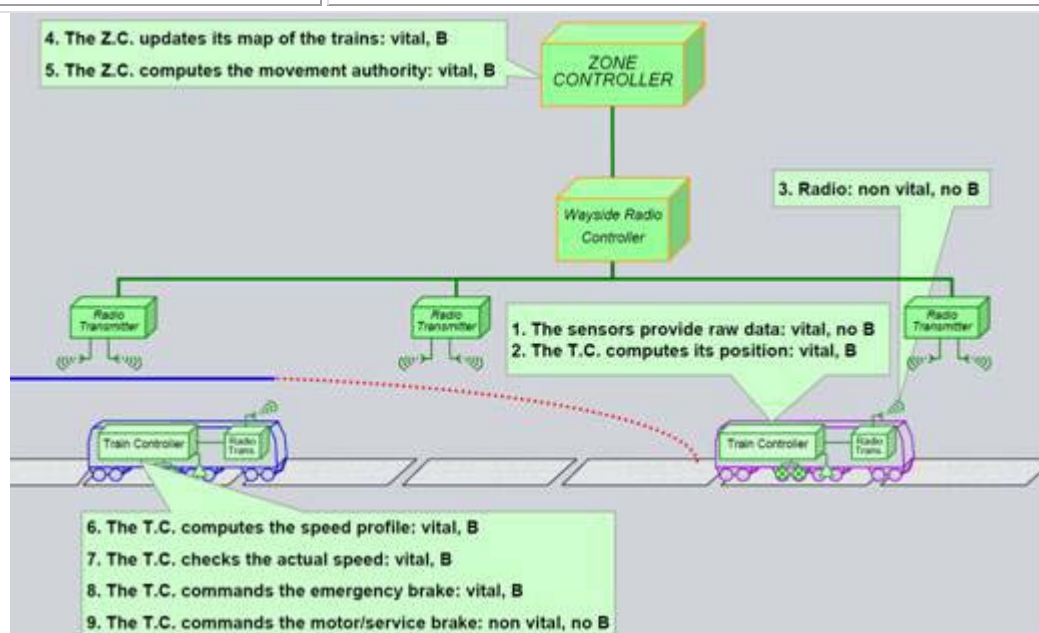
実装言語	C
実装規模	あるコンポーネントにおいて自動生成された C コードは 785,000 行である。
効果	多くのコードが自動生成されたため、コーディングエラーが大幅に削減された。 仕様変更に対応できた。 トレーサビリティが改善された。 生産性が大幅に改善した。

- ・ 詳細情報
 - 検証規模
 - ◇ 各システムの LOC のうち、自動コード生成された割合は、以下のとおりである。
 - 航空制御システム 70%
 - fly-by-wire 制御 70%
 - 表示機器 50%
 - 警告およびメインマシン 40%
 - 期間
 - ◇ 1990 年代後半 - 2005
- ・ 判断
 - 手法・ツール選択理由
 - ◇ **SCADE** は、開発者が意識することなく形式手法を利用することができる。したがって、**SCADE** の操作を習得した開発者がいれば開発可能である。しかし、生成されたコードの検証を行うためには、数学的素養のあるエンジニアが必要である。
 - 障害と工夫
 - ◇ 内製ツールである **ACG** を利用することで、**SCADE KCG** で生成された C コードを後処理し、モデルに影響を与えることなくターゲットに最適化されたコードを生成した。
- ・ 組織
 - 体制
 - ◇ 重要なコンポーネントは自社開発を行った。サブコンポーネントのうち 1/3 は自社開発であり、残りは社外開発である。
- ・ 情報源
 - 形式手法適用調査 調査報告書、情報処理推進機構、2010

12.2.7. STS - ニューヨーク地下鉄列車制御システムの最新化

ドメイン	鉄道
開発対象	地下鉄カナーシ線列車制御システム
国	米国
開発組織	STS (Siemens Transportation System)、ニューヨーク市都市交通
形式手法(言語、ツール)	B (Atelier B, Edith B)
適用範囲・規模(形式手法)	設計の早い段階で重要な部分とそうでない部分に分け、全ての重要な機能について B を適用した。(ただし、低レイヤの入出力、設定ファイル、main 関数の無限ループを除く。) B の抽象モデルを 125,000 行、具体モデルを 38,000 行手動で作成し、EDiTh B を利用することにより自動的に 110,000 行を生

	成した。
適用対象のソフト種別	リアルタイム制御系
適用目的・工程	仕様設計、詳細設計
実装言語	Ada
実装規模	不明
効果	簡潔で曖昧性の無い、高品質なソフトウェア仕様を作成することができた。 妥当性検証チームは詳細コードの開発ではなく、仕様の開発に集中することができた。



(出典: Daniel Dolle, B in Large-Scale Projects: The Canarsie Line CBTC Experience, Siemens Transportation Systems, http://www.clearsy.com/pdf/b2007_b_in_large_scale_projects_siemens.pdf)

図 12-6: 列車制御システムの構成

- ・ 詳細情報
 - 検証規模
 - ◇ 証明責務は、抽象モデルが 38,500 個、手動による具体モデルが 19,000 個、自動生成による具体モデルが 25,000 個であった。
 - 期間
 - ◇ 2001-2005
- ・ 判断
 - 形式手法を利用した動機
 - ◇ パリ地下鉄 14 号線で形式手法 (B) を利用した経験があった。
 - 手法・ツール選択理由
 - ◇ B を利用することにより、簡潔で曖昧性の無い、高品質なソフトウェア仕様を作成することができる。
 - ◇ B はコードが形式仕様のプロパティを維持していることを保証することができる。
 - ◇ Atelier B は数万の証明責務を取り扱うことができるほどロバスタなツールである。

- 障害と工夫
 - ◇ STS で開発された半自動詳細化ツールである EDiTh B を利用することにより、パリ地下鉄 14 号線の開発時と比較して大幅に開発効率が向上した。
- 組織
 - 体制
 - ◇ 4 人のチームが 1 年で車載ソフトウェアを開発した。4 人のうち 2 人はパリ地下鉄 14 号線における B を利用した開発の経験者であった。その他 1 人は B の理論についての知識はあるが開発経験はなく、もう一人は B の知識もなかった。形式手法の専門家は必要としなかった。
- 情報源
 - 形式手法適用調査 調査報告書、情報処理推進機構、2010
 - Daniel Dolle, B in Large-Scale Projects: The Canarsie Line CBTC Experience, Siemens Transportation Systems, http://www.clearsy.com/pdf/b2007_b_in_large_scale_projects_siemens.pdf

12.2.8. ClearSy - 北京地下鉄の自動列車停止システム

ドメイン	鉄道
開発対象	地下鉄の自動列車停止システム
国	中国
開発組織	ClearSy、Alstom
形式手法(言語、ツール)	B (Atelier B)
適用範囲・規模(形式手法)	不明
適用対象のソフト種別	リアルタイム制御系
適用目的・工程	仕様設計、詳細設計、自動コード生成
実装言語	Ada
実装規模	不明
効果	仕様が満たされることが数学的に証明された。 リファインメントを繰り返し、最終的に Ada コードを生成した。

- 詳細情報
 - 検証内容
 - ◇ 安全が保証されない状態になった際に、緊急ブレーキがかかることを検証した。
 - 期間
 - ◇ 2006-2008
 - 組織
 - 体制
 - ◇ Alstom 社のチームが形式的に仕様設計を行った。この仕様を、ClearSy 社のチームが Alstom 社のチームと連携して B を用いて形式化した。
 - 情報源
 - ClearSy 社 Web ページ, <http://www.fersil-railway.com/php/projet-urbalis-evolution-en.php>

12.2.9. 日立ソリューションズ - Blu-Ray ディスク

ドメイン	その他(家電)
------	---------

開発対象	Blu-Ray ディスク制御ミドルウェア
国	日本
開発組織	日立ソリューションズ
形式手法(言語、ツール)	SPIN
適用範囲・規模(形式手法)	制御ミドルウェアの 20%程度を抽象化したもの
適用対象のソフト種別	制御系
適用目的・工程	制御アルゴリズム検証・基本設計
実装言語	C
実装規模	200kLOC
効果	<ul style="list-style-type: none"> • テストでは発見が困難なデッドロックの検出。 • 制御アルゴリズムの性質の保証、想定動作の確認。 • テスト工程半減(2ヶ月→1ヶ月)

- 詳細情報
 - 検証内容
 - ◇ 要求されたディスク操作は、必ず実行され、待機状態に戻る。
 - ◇ デッドロックが存在しない。
 - ◇ モジュール間の制御状態のズレが有っても、動作停止などの問題は発生しない。
 - 検証規模
 - ◇ 機能要求、安全性に関する4つの検証性質とデッドロックに関して検証した。(100%)
 - 期間
 - ◇ 2008年10月～2009年3月
- 判断
 - 形式手法を利用した動機
 - ◇ 制御の動作に関して、テストでは保証できない性質に確信を得たかった。
 - 手法・ツール選択理由
 - ◇ デバイスを含む並行システムの制御アルゴリズムの検証のため SPIN が適していると判断した。
 - 障害と工夫
 - ◇ 制御の基本設計をそのままモデリングすると並行プロセス数が多く状態爆発を発生した。
 - ◇ 抽象化のヒューリスティックスを用いて並行プロセスを減らし、対象 API を減らすことにより、API を限定した検証が可能になった。
- 組織
 - 体制
 - ◇ システム分析、設計の分析は日立ソフトウェアが実施し、形式記述によるモデリングおよび検証は MRI が実施した。NTT データ、NII、JAIST から、形式モデリングに関する助言を得た。
 - 教育
 - ◇ SPIN に関する和書、洋書のテキストを5冊程度精読した。モデリング、検証作業で、オンラインマニュアルなどを利用した。
 - その他
 - ◇ NII 等からモデリングにおいて助言を得たことが効果的であった。
- 情報源
 - 本ガイダンス 10 章 ケーススタディ1:ブルーレイディスク

- 石黒 正揮,中島 震,梅村 晃広,田中 一之、組込みソフトウェアの制御機構に対するモデル検査の適用に関する評価実験、コンピュータセキュリティシンポジウム CSS2009
- Masaki Ishiguro, Kazuyuki Tanaka, Shin Nakajima, Akihiro Umemura, Tomoji Kishi, A Guidance and Methodology for Employing Model-Checking in Software Development, APESER: Asia-Pacific Embedded Systems Education and Research Conference, December 14-15, 2009
- 石黒正揮,ソフトウェア開発における形式手法導入に関する課題と解決アプローチ, 先端ソフトウェア工学に関する Grace 国際シンポジウム,形式手法の産業応用ワークショップ 2010,WIAFM2010: Workshop on Industrial Applications of Formal Methods, p.41-48, 2010年3月15日, 国立情報学研究所, Grace-TR-2010-03

12.2.10. 日立ソリューションズ - 入退室管理システム

ドメイン	その他(入退室管理)
開発対象	入退室管理システム
国	日本
開発組織	日立ソリューションズ
形式手法(言語、ツール)	SPIN, VDM++
適用範囲・規模(形式手法)	入退室管理システムの制御アルゴリズム
適用対象のソフト種別	制御系
適用目的・工程	制御アルゴリズムの検証、要求の妥当性確認
実装言語	C
実装規模	30kLOC
効果	待合室待ちリストのオーバーフローの可能性の発見。

- ・ 詳細情報
 - 検証内容
 - ◇ 待合室に入室した人は、いつか必ず案内される
 - ◇ 閲覧室に案内された人だけが、タッチパネル脇 ID 端末で、いつか必ず認証 OK となる
 - ◇ 閲覧室から退出した閲覧者は、いつか必ず待機者リストから削除される
 - ◇ 閲覧室に案内されていない人が、タッチパネル脇 ID 端末で認証を行っても、常に認証をパスしない
 - ◇ 閲覧室 X が空室のとき以外、常に案内は変更されない
 - ◇ 待機者リストは常にオーバーフローすることはない
 - 検証規模
 - ◇ 要求仕様7件のうち4件検証。その他設計関連の性質2件検証。
 - 期間
 - ◇ 2009年6月～2010年3月
- ・ 判断
 - 形式手法を利用した動機
 - ◇ 要求仕様の妥当性、運用上の条件などを明確化する。制御アルゴリズムの正しさを検証する。
 - 手法・ツール選択理由
 - ◇ 制御系の検証を行うため SPIN を用いた。また妥当性確認のため VDM++を使った。
 - 障害と工夫

- ◇ 時間に関わる性質は扱えなかった。状態爆発が発生したためメッセージの送信のみを行うプロセスを統合するなどして、並行プロセスの数を減らした。
- ・ 組織
 - 体制
 - ◇ システム分析、設計の分析は日立ソフトウェアが実施し、SPIN 形式記述によるモデリングおよび検証は MRI が実施した。VDM++の妥当性確認は NII で実施した。
 - 教育
 - ◇ SPIN に関する和書のテキストを2冊程度精読した。
 - その他
 - ◇ NII 等からモデリングにおいて助言を得たことが効果的であった。
- ・ 情報源
 - 本ガイダンス第 11 章ケーススタディ2: 入退室管理システム
 - 入退室管理システムに関するモデル検証の一例、赤井健一郎 (株式会社三菱総合研究所) 石黒 正揮 (株式会社三菱総合研究所) 中島 震 (国立情報学研究所) 田中 一之 (株式会社日立ソリューションズ)

12.2.11. Lockheed Martin、Praxis - C130J ミッションコンピュータ

ドメイン	航空運輸
開発対象	航空制御システム
国	イギリス
開発組織	Lockheed Martin、Praxis
形式手法(言語、ツール)	CoRE、SPARK (SPARK Examiner, SPADE Automatic Simplifier, SPADE Proof Checker)
適用範囲・規模(形式手法)	ソフトウェアのコアとなる部分(全体の 80%)
適用対象のソフト種別	リアルタイム制御系
適用目的・工程	要求仕様(CoRE)、コード記述・検証 (SPARK)
実装言語	SPARK
実装規模	200K 行程度
効果	人手で行う場合は 30 日かかるようなバグの発見が、SPARK Examiner を利用することにより 1 時間で発見できた。

- ・ 詳細情報
 - 検証内容
 - ◇ CoRE で記述された仕様とコードから、SPARK で Proof Obligation を生成し、検証を行った。
 - ◇ 多くの証明責務は SPADE Automatic Simplifier を用いて証明できた。残りは SPADE Proof Checker を用いて対話的に証明した。
- ・ 判断
 - 形式手法を利用した動機
 - ◇ DO178B のレベル A への準拠が要求されていた。ここでは、MC/DC テストカバレッジが要求されている。このテストは多大な時間を要するため、出来るだけ 1 回限りの実行で終わらせたいという動機があった。そのため、MC/DC テストを実行するまえに、全てのバグを発見しておく必要があった。
 - 手法・ツール選択理由
 - ◇ 本ドメインにおいて、SPARK は多くの導入事例があったため。

- ◇ ソフトウェアに多くの入力と出力がある場合は、CoRE が適合しているため。
- 障害と工夫
 - ◇ 効率よく検証を行うため、コードの構造をできるだけ単純化した。たとえば、事前定義のチェックなどをコードから削除した。
 - ◇ プログラムユニットの実装及び検証を別々に実施した。
- ・ 情報源
 - Roderick Chapman, Industrial Experience with SPARK, ACM SIGAda Ada Letters - special issue on presentations from SIGAda 2000 Martin Croxford and James Sutton, Breaking Through the V and V Bottleneck, Lecture Notes in Computer Science, Vol. 1031, 1996

12.2.12. イギリス国防省 - SHOLIS(ヘリコプタ着陸システム)

ドメイン	航空運輸
開発対象	艦載ヘリコプタ運行システム
国	イギリス
開発組織	PMES (Power Magnetics and Electronic Systems)、Praxis Critical Systems
形式手法(言語、ツール)	Z (CADiZ tool)、SPARK (Examiner, Simplifier, Proof Checker)
適用範囲・規模(形式手法)	要求ドキュメントは 4000 行以上。ソフトウェア要求仕様は Z、英語、数学的定義を含めて 300 ページ。
適用対象のソフト種別	リアルタイム制御系
適用目的・工程	要求仕様(Z)、設計(Z、SPARK)、コード記述(SPARK)
実装言語	SPARK
実装規模	133000 行(Ada 宣言 13000 行、Ada 文 14000 行、SPARK flow アノテーション 54000 行、SPARK proof アノテーション 20000 行、空白行やコメント 32000 行)
効果	UK Interim Defense Standard (IDS) 00-55 及び 00-56 への準拠

- ・ 詳細情報
 - 検証内容
 - ◇ 最も重要な Safety プロパティは「あるセンサの値が現在の SHOL (Ship Helicopter Operating Limits, 操作を行う際の許される値) の範囲を逸脱していた場合は警告が出され、センサの値が SHOL の範囲に収まっていた場合は警告が出されない」ということであり、これは Z を用いて証明された。
 - ◇ グローバル変数や定数の一貫性、初期状態の存在、事前条件の検査について Z を用いて実施した。
 - ◇ 障害発見比率は以下のとおりである。(障害発見比率/工数比率)を計算すると、Z による証明の工程が、最も効率的に障害を発見していることがわかる。

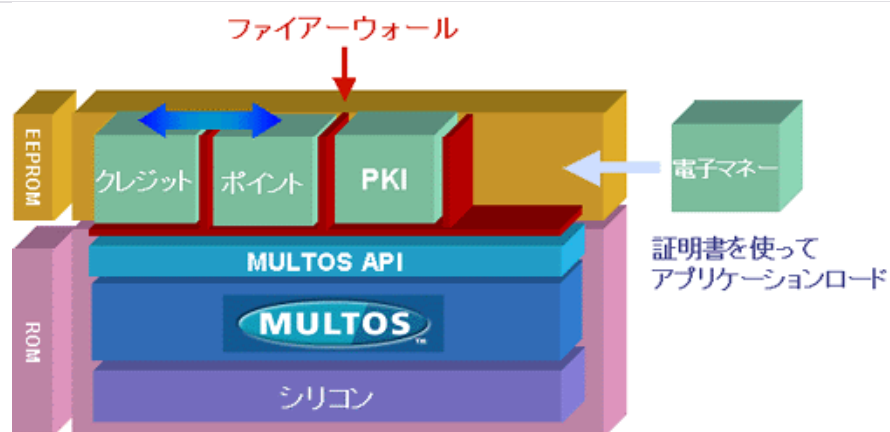
工程	障害発見比率 [%]	工数比率 [%]
仕様	3.25	5
Zによる証明	16	2.5
概念設計	1.5	2
詳細設計、実装、簡易テスト	26.25	17
単体テスト	15.75	25
統合テスト	1.25	1
コードの証明	5.25	4.5
システム妥当性テスト	21.5	9.5
受け入れテスト	1.25	1.5
その他(スタッフの教育、プロジェクト管理と計画等)	8	32

- 期間
 - ◇ 1996年 - 1997年
- 判断
 - 形式手法を利用した動機
 - ◇ UK Interim Defense Standard (IDS) 00-55 及び 00-56 に準拠する必要があった。これは、形式的な安全管理及び品質システム、システムの振る舞いの形式仕様、仕様及びコードレベルの両方における形式検証、プログラムプロパティ(インフォメーションフロー、タイミングやメモリ使用量等)の静的解析を要求する。
 - ◇ テストを実施するよりも、形式手法による検証を実施するほうがコスト面で有利であった。
 - 手法・ツール選択理由
 - ◇ SPARK を利用した理由は、論理的な健全性、簡潔な形式記述、表現力、セキュリティ、検証性、実行時の時間と使用リソース量である。
- 組織
 - 体制
 - ◇ 検証は 4 人の技術者によって実施された。二人は Z の検証を担当した。このうちの一人と残り二人は、Z の仕様に基づいて SPARK 検証アノテーションを生成し、SPARK の検証を実施した。
 - ◇ 全ての技術者は、Z や CSP 等の形式手法に数年間関わっていた。しかし、SPARK Simplifier や Proof Checker を使ったことがあるのは一人だけであった。
- 情報源
 - Steve King, Jonathan Hammond, Rod Chapman, and Andy Pryor, Is Proof More Cost Effective than Testing?, IEEE Transactions on Software Engineering, Vol. 26, No 8, August 2000

12.2.13. Mondex International、Altran Praxis - MULTOS 認証システム(Global Key Center)

ドメイン	通信
開発対象	MULTOS (IC カードのプラットフォーム) の認証局 (IC カードの活性化等を行う)
国	イギリス
開発組織	Mondex International、Altran Praxis
形式手法(言語、ツール)	Z、CSP、SPARK (Examiner)
適用範囲・規模(形式手法)	重要な部分のみ形式手法を適用。28 のセキュリティポリシモデル

	があった
適用対象のソフト種別	エンタプライズ系
適用目的・工程	要求仕様(Z)、設計(Z->CSP)、自動コード生成(CSP->Ada)、コード検証(Ada に対して Spark Examiner の利用)
実装言語	SPARK (30%)、Ada95 (30%)、C++ (30%)、C (5%)、SQL (5%)
実装規模	10 万行程度
効果	実装前に問題点を理解することができる 各工程をより厳密に進めることができる 可用性 99.999%を達成 形式手法を用いないで開発したシステムよりも、バグの数を少なく抑えることができた。(2002 年時点でバグは 4 つのみ発見されている)



(出典:「MULTOS について:IC カードシステムソリューションズ:日立」,
http://www.hitachi.co.jp/Prod/comp/ic-card/about_ic/multos.html)

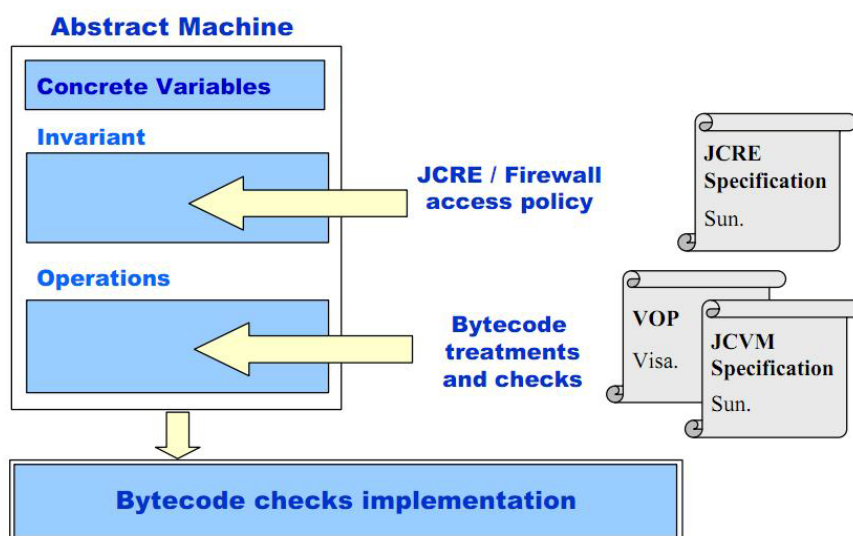
図 12-7: MULTOS の概要

- ・ 詳細情報
 - 検証内容
 - ◇ CSP モデルに対して、デッドロックがないこと、セキュリティが特に重要な機能において並行プロセスが存在しないことを検証した。
 - ◇ SPARK のインフォメーションフロー解析を実施した。
 - ◇ Communications-Electronics Security Group のマニュアルである「F」を簡潔にして実践した。
- ・ 判断
 - 形式手法を利用した動機
 - ◇ UK ITSEC スキームの最も高いレベル(E6)に準拠する必要があった。E6 では、初期段階から形式手法を使うことを求めている。
 - 手法・ツール選択理由
 - ◇ SPARK は、ITSEC E6 の要求(プログラミング言語は、ソースコード中の全ての文に対し不明瞭でない意味を定義することが求められる)を満たし、実際に産業界で利用可能なほぼ唯一の言語である。
 - ◇ 実装すべきセキュリティモデルの多くが、Z を利用することで直接モデル化可能であった

- 障害と工夫
 - ◇ 既存製品を利用するとセキュリティ検証が煩雑になるため、できるだけ自前で実装した。
 - ◇ システムを、セキュリティが特に重要な部分、セキュリティに関連する部分、セキュリティとは無関係な部分に分割した。セキュリティが特に重要な部分は SPARK を用いて開発を行った。
 - ◇ CSP のモデルをコードに変換するルールを作成し、適用した。
- ・ 情報源
 - Anthony Hall and Roderick Chapman, Correctness by Construction: Developing a Commercial Secure System, IEEE Software, Jan/Feb 2002, pp18-25

12.2.14. Gemplus - スマートカード

ドメイン	電子部品・デバイス																															
開発対象	スマートカードの OS																															
国	フランス																															
開発組織	Gemplus																															
形式手法(言語、ツール)	B (Atelier B)																															
適用範囲・規模(形式手法)	一部のコンポーネントにのみ適用																															
適用対象のソフト種別	制御系																															
適用目的・工程	設計、コード生成																															
実装言語	不明																															
実装規模	不明																															
効果	<p>最終的な開発工数を削減することができた。</p> <table border="1"> <thead> <tr> <th></th> <th>形式手法を利用した開発</th> <th>従来型の開発</th> </tr> </thead> <tbody> <tr> <td>開発</td> <td>12 週間</td> <td>12週間</td> </tr> <tr> <td>検証</td> <td>6 週間</td> <td>NA</td> </tr> <tr> <td>テスト</td> <td>1 週間</td> <td>3 週間</td> </tr> <tr> <td>統合</td> <td>1 週間</td> <td>2 週間</td> </tr> <tr> <td>合計</td> <td>20 週間</td> <td>17 週間</td> </tr> <tr> <td>レビューによるバグの発見数</td> <td>13</td> <td>24</td> </tr> <tr> <td>証明によるバグの発見数</td> <td>29</td> <td>NA</td> </tr> <tr> <td>テストによるバグの発見数</td> <td>32</td> <td>71</td> </tr> <tr> <td>合計</td> <td>74</td> <td>95</td> </tr> </tbody> </table>			形式手法を利用した開発	従来型の開発	開発	12 週間	12週間	検証	6 週間	NA	テスト	1 週間	3 週間	統合	1 週間	2 週間	合計	20 週間	17 週間	レビューによるバグの発見数	13	24	証明によるバグの発見数	29	NA	テストによるバグの発見数	32	71	合計	74	95
	形式手法を利用した開発	従来型の開発																														
開発	12 週間	12週間																														
検証	6 週間	NA																														
テスト	1 週間	3 週間																														
統合	1 週間	2 週間																														
合計	20 週間	17 週間																														
レビューによるバグの発見数	13	24																														
証明によるバグの発見数	29	NA																														
テストによるバグの発見数	32	71																														
合計	74	95																														



(出典: Industrial Use of Formal Methods at Gemplus, Gemplus Research Lab, <http://www.gemplus.com/smart/rd/publications/pdf/Lan99fmg.pdf>)

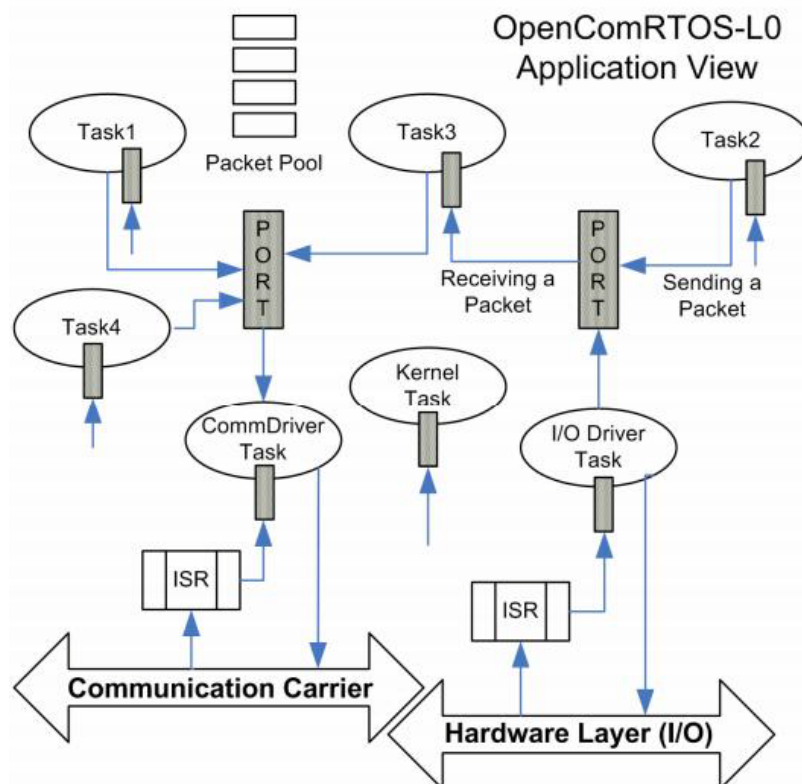
図 12-8: 実践した形式モデリングの流れ

- ・ 詳細情報
 - 検証内容
 - ◇ ある Java アプレットが他の Java アプレットのデータにアクセスできないこと
 - ◇ Java アプレットが OS のコードにアクセスできないこと
 - ◇ 正しいモデルを作成し、Atelier B で自動証明を行った
 - ◇ 残った証明に関しては、対話式証明を行った。対話式証明により、モデルやループにおける不変条件の不備やそれらをどう構築すれば良いかを理解することができた
- ・ 判断
 - 形式手法を利用した動機
 - ◇ スマートカードの高いレベルでの認証が必要だった。また、検証コストを削減する必要があった。
 - 障害と工夫
 - ◇ 最初の B モデル仕様のレビューをしっかりと行った
 - ◇ Atelier B の自動証明に適合するようにモデルをチューニングした
- ・ 情報源
 - Jean-Louis LANET, The use of B for Smart Card, In Forum on Design Languages, 2002

12.2.15. Altreonic - 分散 RTOS (OpenComRTOS)

ドメイン	電子部品・デバイス
開発対象	組込み機器用リアルタイム OS
国	ベルギー
開発組織	Altreonic
形式手法(言語、ツール)	TLA+/TLC
適用範囲・規模(形式手法)	一部分のみに適用
適用対象のソフト種別	リアルタイム制御系
適用目的・工程	アーキテクチャ設計

実装言語	不明
実装規模	不明
効果	RTOS の性質である安全性とリアルタイム性を実現できた コードサイズが従来と比較して1/10 程度になった



(出典: Eric Verhulst, Gjalt de Jong, OpenComRTOS - Distributed RTOS development using formal modeling methods)

図 12-9: OpenComRTOS のアプリケーション概要

- 詳細情報
 - 検証内容
 - ◇ 形式モデルに対し、形式モデルエンジニアとソフトウェアエンジニアがミーティングを行い、より詳細なモデルを作成していった。また、モデルが正しいかどうかのチェックを行った。
 - ◇ モデルが十分実装に近くなったら、PC 上の仮想的なターゲット上でシミュレーションモデルを構築した。
 - ◇ 次にこのコードを実際の 16 ビットのマイクロプロセッサに移行し、最適化した。
- 判断
 - 形式手法を利用した動機
 - ◇ RTOS は高い安全性と信頼性を要求されるにも関わらず、2004 年時点では、ほとんどの RTOS が検証されていなかった。信頼性の高い RTOS を構築するために形式手法の適用を決めた。
 - 手法・ツール選択理由
 - ◇ C 言語に近い SPIN 等よりも、より抽象的なタームの重要性を理解することができる。

- 障害と工夫
 - ◇ 完全な証明のためにはターゲットとなるハードウェアもモデル化して検証する必要があるが、それは複雑であり状態爆発を起こすため、実施しなかった。
 - ◇ TLA モデルはパラメータ化ができないため、特定の部分だけをモデル化した。
- 組織
 - 体制
 - ◇ 初期のアーキテクチャを定義した後、2 チーム作成し、1チームはアーキテクチャモデルを作成し、もう1チームは、TLA+/TLC を利用して初期の形式モデルを作成した。
- 情報源
 - Bernhard H.C. Sputh, et.al, OpenComRTOS: Reliable performance with a small code size
 - Eric Verhulst, Gjalt de Jong, OpenComRTOS - Distributed RTOS development using formal modeling methods

12.2.16. National Air Traffic Services (NATS) Praxis - iFACTS、航空管制システム

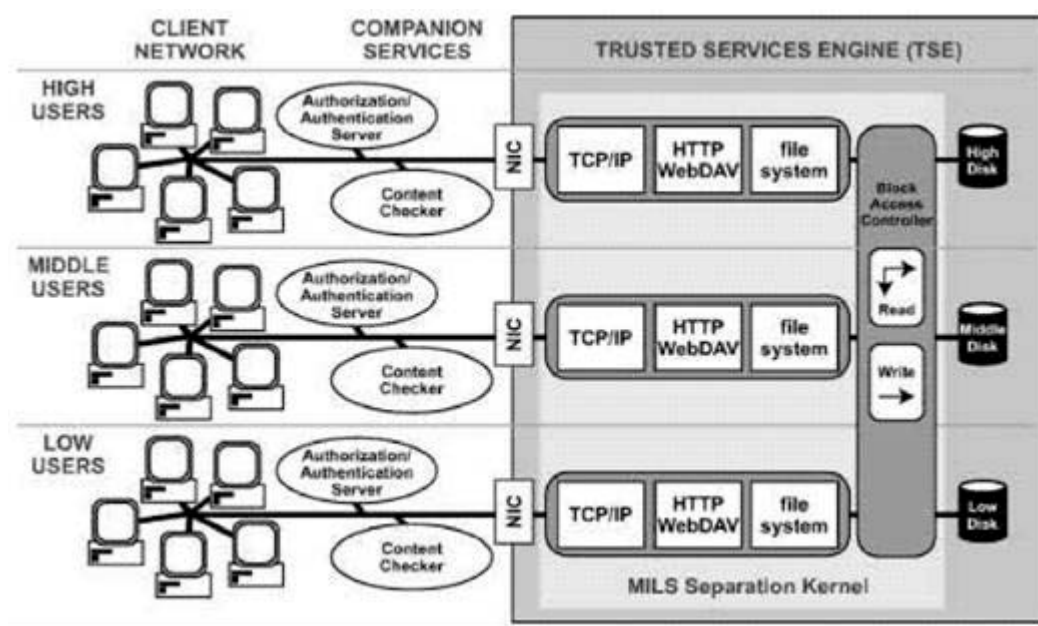
ドメイン	航空運輸
開発対象	航空管制システムの拡張機能。iFACTS (Interim Future Area Control Tools Support)は、NATS に従来からある航空管制システム NERC (New En-Route Centre)の拡張であり、管制官を支援するツールを提供する。
国	イギリス
開発組織	Praxis
形式手法(言語、ツール)	Z (Microsoft Word Add-ins, Fuzz: 型チェッカー)
適用範囲・規模(形式手法)	機能仕様記述 ATS システムソフトウェアの開発 既存の NERC システムに付加し、仕様記述と実装 4250 ページ
適用対象のソフト種別	エンタプライズ系
適用目的・工程	仕様記述、コーディング、テスト、検証
実装言語	SPARK Ada
実装規模	150,000SLOC (Ada SPARK)
効果	<p>[バグ排除] ソフトウェアに重大な欠陥は発見されず、ソフトウェア品質に問題がないことが確認された。</p> <p>[形式手法の経験] Z を用いたモデリングを経験： テスト実施者やレビュー実施者はテストケース作成やコード・設計のレビューを効率的に行えるようになった。 各工程間のコミュニケーション促進・曖昧さを解消に役立った。 (設計者、プログラマ、テスト実施者・コードレビュー実施者、保守作業における人的要素を過小評価しないこと)</p> <p>[規格、認証準拠] IEC61508、SIL4 対応に対応できた。</p>

- 判断

- 形式手法を利用した動機
 - ◇ 英国政府は、空港の年間フライト数の急増に備えて、信頼性の高い航空管制システムを導入する必要があった。
- 障害と工夫
 - ◇ Z の記述には Word を用いた。言語とツールと同時に新しいことを教えるのは困難という判断があった。Z のフォントと FuZZ タイプチェッカーの起動ボタン、仕様と構造の対応を確認する GUI ツールへのリンクなどを備える。
- ・ 組織
 - 教育
 - ◇ Z の読み
 - 3 日間のコースを実施。1 週間の業務ですらすらと読めるようになった。75 名をトレーニングした。
 - ◇ Z の書き
 - 3 日間のコースを実施。3 ヶ月の業務ですらすらと書けるようになった。11 名をトレーニングした。
 - ◇ SPARK
 - 57 名をトレーニングした。
- ・ 情報源
 - Open-DO, The Use of Formal Methods on the iFACTS ATC Project (Neil White), <http://www.open-do.org/2010/04/20/the-use-of-formal-methods-on-the-ifacts-air-traffic-control-project/>

12.2.17. Galois - Trusted Services Engine (TSE)

ドメイン	通信
開発対象	Bell-LaPadula モデルに対応した分散ファイルシステム
国	米国
開発組織	Galois (SPAWAR: Space and Naval Warfare Systems Command, NAS: National Security Agency のファンドを受ける)
形式手法(言語、ツール)	Isabelle
適用範囲・規模(形式手法)	一部のソフトウェアコンポーネントである BAC (Block Access Controller)
適用対象のソフト種別	エンタプライズ系
適用目的・工程	設計
実装言語	C
実装規模	780 行 (C コード)
効果	EAL 6 の認証基準への準拠



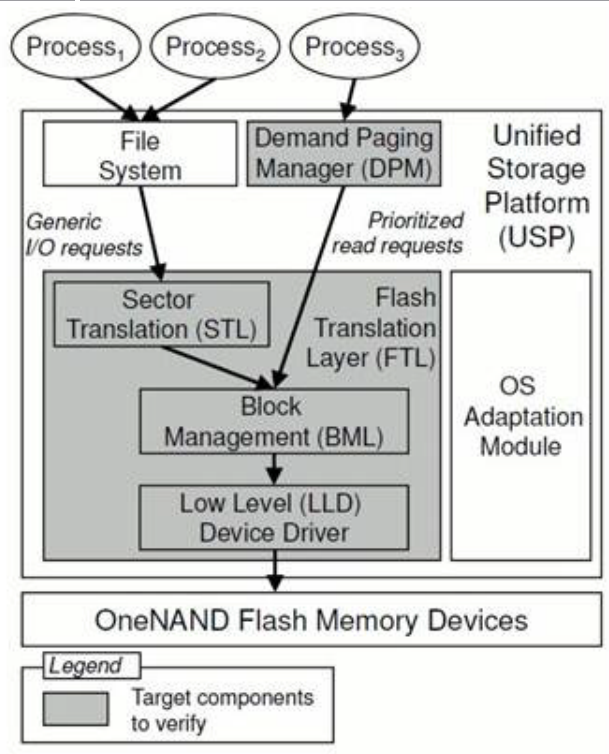
(出典: Christopher Gunderson, Worldwide Consortium for the Grid (W2COG) Research Initiative Phase 1 Final Report, 2006)

図 12-10: TSE アーキテクチャの概要

- ・ 詳細情報
 - 検証内容
 - ◇ Isabelle による検証
 - TSE のセキュリティポリシーとモデルが正しく対応していること
 - エラー状態へ到達しないこと
 - 高いセキュリティレベルのコンポーネントによるアクションが、低いセキュリティレベルのコンポーネントから不可視であること
 - ◇ QuickCheck tool による検証
 - HOL コードと C の実装が正しく対応していること
 - QuickCheck tool により、HOL モデルからテストケースを生成し、C で記述されたコードに対してテストを実施した。
 - 同一レベルで書かれたデータを読めること
 - 正しい読み取りができること
 - 高いセキュリティレベルの読み取りができないこと
 - 高いセキュリティレベルのコンポーネントによるアクションが、低いセキュリティレベルのコンポーネントから不可視であること
 - ◇ 人による検証
 - 各ラインについて最低 2 人が HOL コードと C コードを見比べてレビューを行った。
- ・ 判断
 - 形式手法を利用した動機
 - ◇ EAL 6 の認証基準に基づいて開発を行う必要があった。
- ・ 情報源
 - Christopher Gunderson, Worldwide Consortium for the Grid (W2COG) Research Initiative Phase 1 Final Report, 2006

12.2.18. Samsung Electronics - フラッシュメモリデバイスドライバ

ドメイン	電子部品・デバイス
開発対象	OneNAND フラッシュメモリ
国	韓国
開発組織	Samsung, KAIST
形式手法(言語、ツール)	BLAST, CBMC
適用範囲・規模(形式手法)	150~2450 (LOC)
適用対象のソフト種別	制御系
適用目的・工程	組み込みシステムの実機テスト前
実装言語	C
実装規模	不明
効果	Samsung が過去に見つけれなかったバグを検出した。 セマフォ例外の不十分なハンドリング 消去時の状態を保持しない論理的なバグ



(出典: Moonzoo Kim, Yunho Kim, Hotae Kim, "A Comparative Study of Software Model Checkers as Unit Testing Tools: An Industrial Case Study," IEEE Transactions on Software Engineering, vol. 99, 2010)

図 12-11: OpenNAND フラッシュメモリのためのストレージプラットフォーム

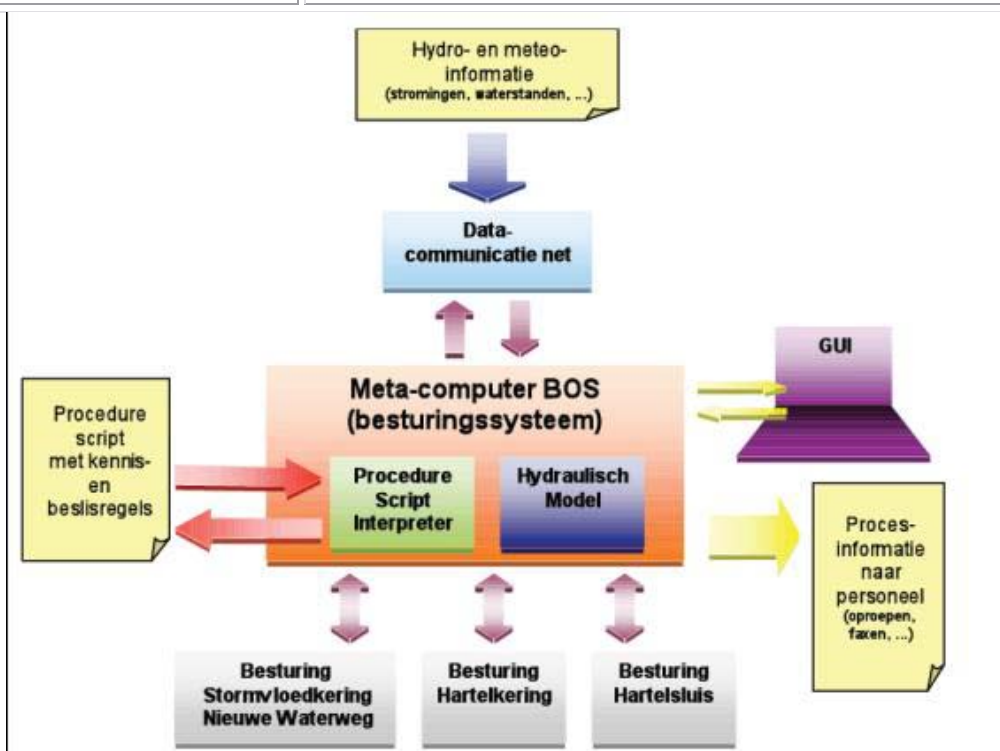
- ・ 詳細情報
 - 検証内容
 - ✧ OneNAND フラッシュメモリのプラットフォームソフトウェアの MSR (マルチセクターリード) の動作。

- ◇ レースコンディションが発生しないこと
- ◇ デッドロックが発生しないこと
- ◇ バイナリセマフォにおいて、セマフォの数がどのステップにおいても 0 以上 1 以下であること
- ◇ セマフォを利用している関数の動作が終了したとき、セマフォの数が 1 になっていること
- 期間
 - ◇ 6 か月
- 判断
 - 形式手法を利用した動機
 - ◇ バグ検出
 - 手法・ツール選択理由
 - ◇ Blast と CBMC はソフトウェアモデル検査ツールの中では安定している。(10 年近くメンテナンスされている)
 - ◇ ユーザコミュニティが存在する。
 - ◇ Blast と CBMC はオープンソースソフトウェアである。検証のパフォーマンス向上等、ユーザに改善の余地が残されている。
 - ◇ 数年にわたる利用経験
 - 障害と工夫
 - ◇ フラッシュストレージプラットフォームは、物理的なユニットと論理的なユニットを結びつける複雑なデータ構造を多用するため、動作検証が難しい。
- 情報源
 - Moonzoo Kim, Yunho Kim, Hotae Kim, "A Comparative Study of Software Model Checkers as Unit Testing Tools: An Industrial Case Study," IEEE Transactions on Software Engineering, vol. 99, no. PrePrints, , 2010
 - Moonzoo Kim, Yunho Kim, and Hotae Kim. 2008. Unit Testing of Flash Memory Device Driver through a SAT-Based Model Checker. In Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE '08). IEEE Computer Society, Washington, DC, USA, 198-207. DOI=10.1109/ASE.2008.30 <http://dx.doi.org/10.1109/ASE.2008.30>
 - Moonzoo Kim, Yunja Choi, Yunho Kim, and Hotae Kim. 2008. Formal Verification of a Flash Memory Device Driver --- An Experience Report. In Proceedings of the 15th international workshop on Model Checking Software (SPIN '08), Klaus Havelund, Rupak Majumdar, and Jens Palsberg (Eds.). Springer-Verlag, Berlin, Heidelberg, 144-159. DOI=10.1007/978-3-540-85114-1_12 http://dx.doi.org/10.1007/978-3-540-85114-1_12

12.2.19. Logica Nederland B.V. - オランダ運河のマエスラント防潮可動橋（水門）

ドメイン	その他(水門)
開発対象	水門開閉の意思決定システム
国	オランダ
開発組織	Logica Nederland B.V. (旧 CMC Den Haag B.V.) - Rijkswaterstaat (Dutch Ministry of Transport, Public Works and Water management の一部門) の委託を受けて開発
形式手法(言語、ツール)	Z (ZTC type checking tool)、SPIN、PVS
適用範囲・規模(形式手法)	コアとなる部分のみ。Z の仕様記述 - 29 プログラム/20KLOC
適用対象のソフト種別	制御系

適用目的・工程	アーキテクチャ設計 (SPIN)、詳細設計 (Z)
実装言語	C++
実装規模	C の運用システム - 250KLOC
効果	費用対効果の面で有利であった。 モデル検査により、プロトコルレベルで大きな変更を実施する必要性が判明した Z の仕様記述により、テストケース作成やコード/設計レビューを有効に行うことができた。また、設計者、プログラマ、テスト、コードレビュー者間の意思疎通を曖昧さを排除して行うことができた



(出典: Klaas Wijbrans, et al., Software Engineering with Formal Methods: The storm surge barrier revisited, ACISION, 2008)

図 12-12: 意思決定システム (BOS) の全体像

- ・ 詳細情報
 - 検証内容
 - ◇ Promela/SPIN による検証
 - コアアーキテクチャにおいてライブロック及びデッドロックが存在しないこと
 - ◇ Z による検証
 - 仕様の妥当性を検証した
 - Ward & Mellor 手法に基づいてモデル化を行った
 - 各プロセス、store、flow における機能やデータをモデル化した
 - 期間
 - ◇ 1期: 1995-1996 年、2 期: 2007-2009 年
- ・ 判断
 - 形式手法を利用した動機
 - ◇ IEC 61508 では safety-critical システムの開発には形式手法の利用を推奨しており、

本プロジェクトでは IEC 61508 の遵守を決定した

- ・ 組織
 - 体制
 - ◇ 1期の開発
 - 3名の形式手法の専門家の支援を受け、5名が中心となって開発を行った
 - 7名が主にコーディングを行った
 - ◇ 2期の開発
 - 7名が開発を行った
 - 教育
 - ◇ Promela/SPIN によるモデリングは習得が容易であった
 - ◇ Zによるモデリングは難しく、習得に時間がかかった
- ・ 情報源
 - Ken Madlener, et al., A Formal Verification Study on the Rotterdam Storm Surge Barrier, ICFEM 2010
 - Klaas Wijbrans, et al., Software Engineering with Formal Methods: The storm surge barrier revisited, ACISION, 2008
 - 形式手法適用調査 調査報告書、情報処理推進機構、2010

12.2.20. Rodin Project - 航空情報表示システム

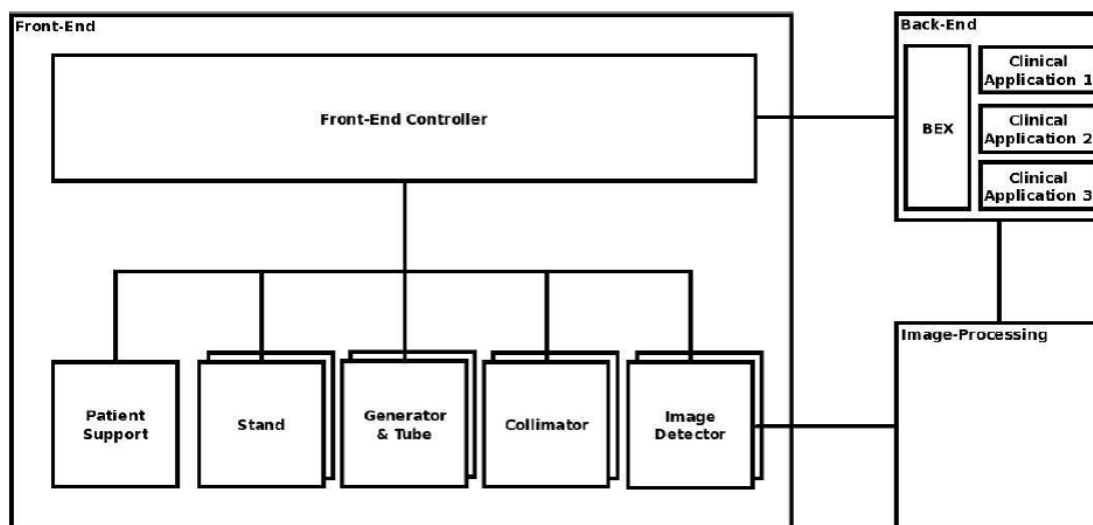
ドメイン	航空運輸
開発対象	CCF 表示及び情報システム (CDIS)
国	イギリス
開発組織	Rodin Project
形式手法(言語、ツール)	Event-B (Rodin)
適用範囲・規模(形式手法)	既存の CDIS システム (1992 年作成) の一部の再構築。 既存の CDIS システムは、VVSL (VDM の一種) で記述され、 1200 ページの仕様書、3000 ページの設計ドキュメントが作成された。 再構築された部分における、Event-B の抽象仕様は 4 ページ。
適用対象のソフト種別	エンタプライズ系
適用目的・工程	要求仕様、設計
実装言語	不明
実装規模	不明
効果	<p>[仕様の理解] 抽象的な仕様記述から段階的に詳細化することにより、巨大なシステムの全容の理解を容易にした</p> <p>[バグ排除] 仕様と実際の設計との形式的なリンクを対応させることができた</p>

- ・ 判断
 - 形式手法を利用した動機
 - ◇ 既存の CDIS システムは VVSL を用いて作成されたが、形式的推論は行われていなかった
 - 手法・ツール選択理由
 - ◇ 分散システムモデリングに適しているという理由で Event-B を採用した

- 障害と工夫
 - ◇ 理想的な仕様と現実的な設計との間の形式的なリンクの欠如
 - 理想的な仕様の拡張としてまず抽象仕様を作成した。理想的な仕様では、異なるユーザが異なる場所である変化するデータを見たとき、同じ時刻で見た場合は全く同じデータが見えるはずであるが、実際には遅延等によって異なるデータが見えることがある。これを現実的な抽象仕様として記述するために、履歴トラッキングシステムを導入した。システムは全てのデータ変更の履歴を保持し、ユーザは履歴のいずれかのデータを閲覧することとした。この抽象仕様を徐々に詳細化した。
 - まず基本的なディスプレイシステムの仕様記述を行い、航空に特化した部分を全て無視することによって、全体をうまく概観できるようにした。Rodin ツールを利用してそれを徐々に詳細化していった。各ステップの証明責務はおおよそ 20 以下で済んだ。
 - ◇ Event-B においてレコード型を SETS、CONSTANTS、PROPERTIES を利用して表現した
- ・ 情報源
 - Rezazadeh, A. and Evans, N. and Butler, M., Redevelopment of an Industrial Case Study Using Event-B and Rodin, BCS-FACS Christmas 2007 Meeting-Formal Methods In Industry.(December 2007)

12.2.21. Philips Healthcare 社 - X 線 CT スキャン

ドメイン	医療
開発対象	X 線 CT スキャン
国	オランダ
開発組織	Philips Healthcare
形式手法(言語、ツール)	Verum ASD Suite
適用範囲・規模(形式手法)	患者をサポートするハードウェアを制御するソフトウェア
適用対象のソフト種別	制御系
適用目的・工程	設計、コード生成
実装言語	C++, C#
実装規模	<ul style="list-style-type: none"> ● C++ 7697 行、C# 19684 行 ● 79 人月
効果	<p>[バグ排除] 検知したモデルエラー 423 個</p> <p>[コスト削減] ASD Suite を利用した開発に要したコストは 541,161 ユーロであり、従来の手法であれば 848,811 ユーロ必要であったと考えられる(実行コード行数より算出)。したがって、36%のコスト削減が可能となった。</p>



(出典: Schuts, M. , Radboud University, Improving Software Development, 2010)

図 12-13:X 線 CT スキャンのアーキテクチャの概要

- 判断
 - 形式手法を利用した動機
 - ◇ 開発コストの削減
 - ◇ テストおよび統合フェーズにおける工数削減
 - 手法・ツール選択理由
 - ◇ 現場で利用することが容易な形式手法ツールであったため。
- 情報源
 - Robert C. Howe, ASD SaaS Business Case for Philips Healthcare

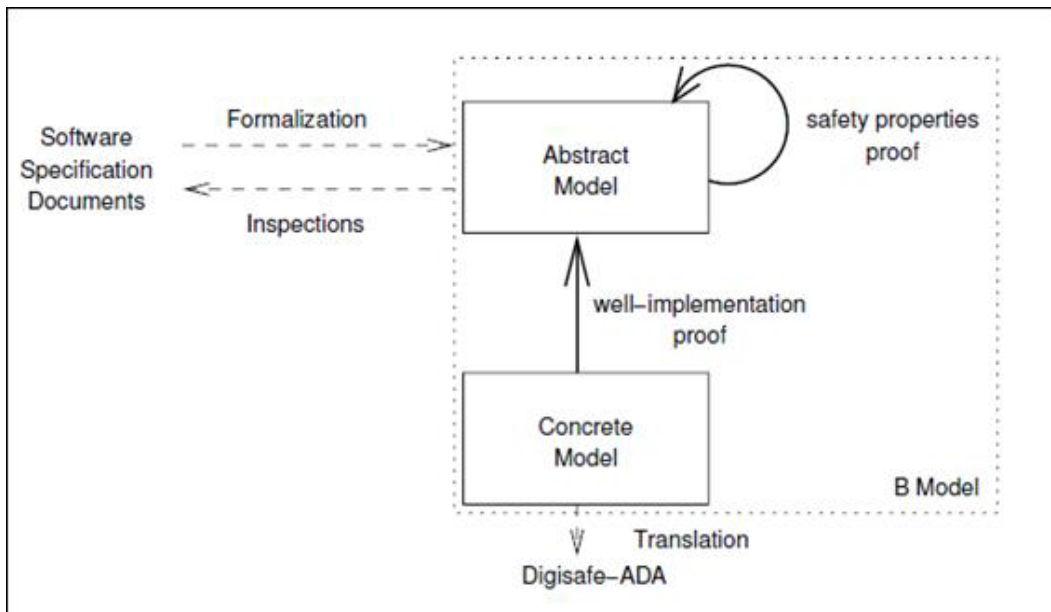
12.2.22. ClearSy - コンポーネント仕様のモデル化

ドメイン	自動車
開発対象	車載コンポーネントの機能
国	フランス
開発組織	ClearSy
形式手法(言語、ツール)	Event B (Atelier B)
適用範囲・規模(形式手法)	<ul style="list-style-type: none"> • 一部の機能 • 50B モデル • 7000 イベント • 2000 抽象変数 • 1 台目は 14 人年、2 台目は 5.6 人年
適用対象のソフト種別	制御系
適用目的・工程	実機テスト
実装言語	不明
実装規模	不明
効果	自動車の各コンポーネントが、仕様に基づいて正しく作成されているかどうかを確認された

- ・ 詳細情報
 - 検証内容
 - ◇ 自動車の全てのコンポーネントについて、正しくて完全な **Event-B** モデルを作成した
 - ◇ 実際の自動車のコンポーネントとその挙動と、**Event-B** モデルとの明確な対応付けを行った
 - ◇ コンポーネントの挙動を記録し、**Event-B** モデルと整合しているかどうかの確認を行った
- ・ 判断
 - 形式手法を利用した動機
 - ◇ 従来の不具合発見手法では、複雑化した自動車の不具合を見つけることが困難になってきた
 - 手法・ツール選択理由
 - ◇ 仕様の正確な記述を行う必要があった
 - 障害と工夫
 - ◇ 自動車の全てのコンポーネントについて実機の検証を行うのはコストが大きいため、重要なシナリオのみを対象とした
- ・ 情報源
 - Guilhem Pouzancre, How to diagnose a modern car with a formal B model, 2003

12.2.23. ClearSy - シャルルドゴール空港の無人シャトル制御

ドメイン	鉄道
開発対象	無人シャトル制御システム
国	フランス
開発組織	ADP (Paris Airport)、Siemens Transportation Systems (STS)、ClearSy
形式手法(言語、ツール)	B (EDiTh B, Bertille, Atelier B)
適用範囲・規模(形式手法)	ソフトウェア仕様ドキュメント: 228 ページ(84 の機能モジュール) B モデル: 183,897 行
適用対象のソフト種別	リアルタイム制御系
適用目的・工程	要求仕様、設計、コード生成
実装言語	Ada
実装規模	Ada: 158,612 行
効果	限られた予算、納期内で、安全要求を全て満たし、バグがほとんど無いシステムの開発を行えた 開発されたソフトウェアは IEC 61508: EN 50126, EN 50128, EN 50129 に準拠。また、SIL4 に分類された。



(出典: ClearSy and Siemens Transportation Systems, Using B as a High Level Programming Language in an Industrial Project: Roissy VAL, 2005)

図 12-14: 無人シャトル制御システムにおける抽象モデルと具体モデルとの関係

- ・ 詳細情報
 - 検証内容
 - ◇ 制御ユニットが、コントロールセンターから無人シャトルの経路についての命令を受け取り、それに基づいて制御ユニットが無人シャトルを制御するシステムにおいて、全ての安全要求を満たしていることを検証した
 - 検証規模
 - ◇ B モデルで記述した全ての要素間に矛盾が無いことを検証した
 - ◇ リファインメントした際に矛盾が生じていないことを検証した
- ・ 判断
 - 形式手法を利用した動機
 - ◇ 限られた予算内で、初期リリースからほとんどバグの無い巨大なソフトウェア構築が求められた。特に安全要求への対応が求められた。
 - 手法・ツール選択理由
 - ◇ ソフトウェア仕様書がとても精度が低いものであり、システムを理解するためには何らかの形式化が必要であった。
 - 障害と工夫
 - ◇ 自動リファインメントを行うには問題の規模が大きすぎたため、手動で B モデルを分割した
 - ◇ 自動リファインメントを実施すると、実行速度が遅いアルゴリズムを採用してしまうことがあったため(本来であれば線形時間で解けるアルゴリズムを指数時間かかるアルゴリズムにしてしまった)、その場合は手動で新しいルールを作成し、適用した
 - ◇ 仕様書の理解を助けるために、質問/回答データベースを作成し、ClearSy の質問と Siemens の回答の対応付けを行った
 - ◇ 記述した B モデルが、自然言語で記述された仕様書を正しく表現していることを確認するために、全ての B モデルの要素について、B モデルを記述していないメンバーがチェックした
 - ◇ B モデルと自然言語のトレーサビリティを確保するために、B モデルの各オペレーシ

ョンにおいてコメントを記述した

- ・ 組織
 - 教育
 - ◇ 仕様書を理解するためのドメイン知識及び B の読解能力、記述能力の教育を行った。詳細化のモデルに関してはツールの導入により負荷の軽減を図った。
- ・ 情報源
 - ClearSy and Siemens Transportation Systems, Using B as a High Level Programming Language in an Industrial Project: Roissy VAL, 2005

12.2.24. POSCON - 地下鉄のプラットフォームドア

ドメイン	鉄道
開発対象	地下鉄のプラットフォームドア
国	ブラジル
開発組織	POSCON 社 (韓国 POSCO 社のグループ企業)
形式手法(言語、ツール)	SCADE
適用範囲・規模(形式手法)	不明
適用対象のソフト種別	リアルタイム制御系
適用目的・工程	設計、コード生成
実装言語	C
実装規模	10000 行
効果	<ul style="list-style-type: none"> ● RAMS SIL-3 の達成 ● 信頼性 99.95%、可用性 99.998% ● システムの信頼性及び生産性の改善 ● 製品価値の向上及び運用コストの削減 ● プラットフォームドアシステムの安全性の保証 ● ソースコードにおける V&V 活動の工数を大幅に短縮することができ、厳しい納期に間に合った



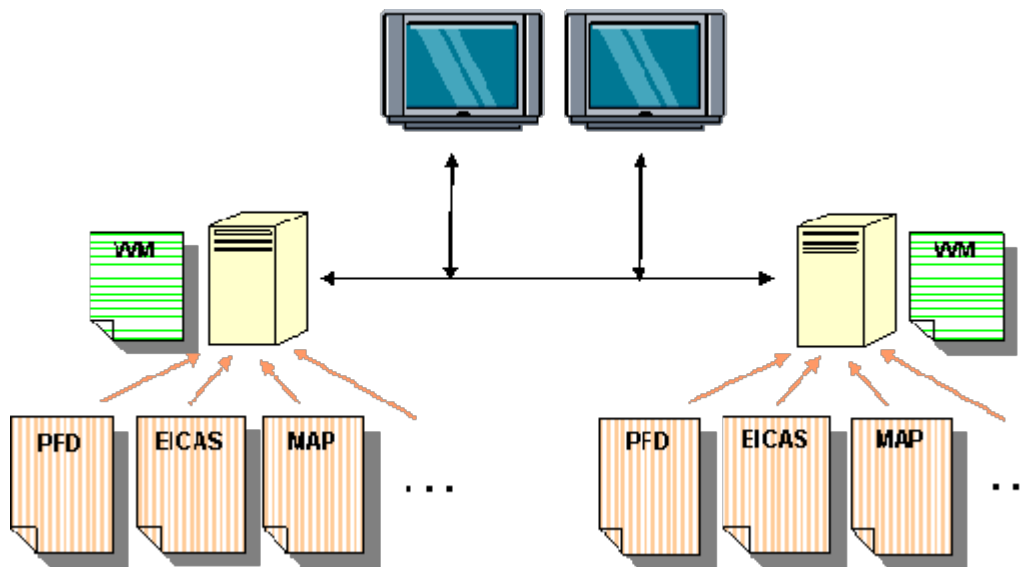
(出典: Esterel Technologies, 「Poscon :: Success Stories」,
<http://www.esterel-technologies.com/technology/success-stories/poscon>)

図 12-15: 開発されたプラットフォームドア

- 詳細情報
 - 検証内容
 - ◇ EN 50126 (IEC 62278)、EN 50128 (IEC 62279)、EN 50129 (IEC 61508)に基づく安全性の検証
- 判断
 - 形式手法を利用した動機
 - ◇ プロジェクト期間がとても短かった
 - ◇ SIL3のためには、ソフトウェアのソースコードレベルを含む様々な V&V 活動が要求された
 - 手法・ツール選択理由
 - ◇ SCADE Suite は、SIL3-4 に対応した唯一の C 言語生成ツールであった
- 組織
 - 体制
 - ◇ POSCON 社の 3 名が開発を行った。開発者はソフトウェア工学の学位を持つが、形式手法については知識や経験は無かった。
 - 教育
 - ◇ SCADE Suite はとても学習が容易であり、1 週間のトレーニングによって設計者はモデリングを開始できた。
- 情報源
 - POSCON 社プレゼンテーション資料(SCADE User Group Conference 2009)

12.2.25. Rockwell Collins - パイロット用のディスプレイを管理するシステム

ドメイン	航空運輸
開発対象	パイロット用ディスプレイを管理するソフトウェア (ADGS-2100 Adaptive Display and Guidance System Windows manager)
国	米国
開発組織	Rockwell Collins
形式手法(言語、ツール)	NuSMV, Simulink
適用範囲・規模(形式手法)	Simulink : 16117 ブロック 4295 サブシステム NuSMV :9.8x10 ⁹ ~1.5x10 ³⁷
適用対象のソフト種別	リアルタイム制御系
適用目的・工程	設計
実装言語	不明
実装規模	不明
効果	563 性質に対してエラー98 件を発見した



(出典: Michael W. Whalen, John D. Innis, Steven P. Miller, and Lucas G. Wagner, ADGS-2100 Adaptive Display & Guidance System Window Manager Analysis, NASA Contractor Report CR-2006-213952, 2006)

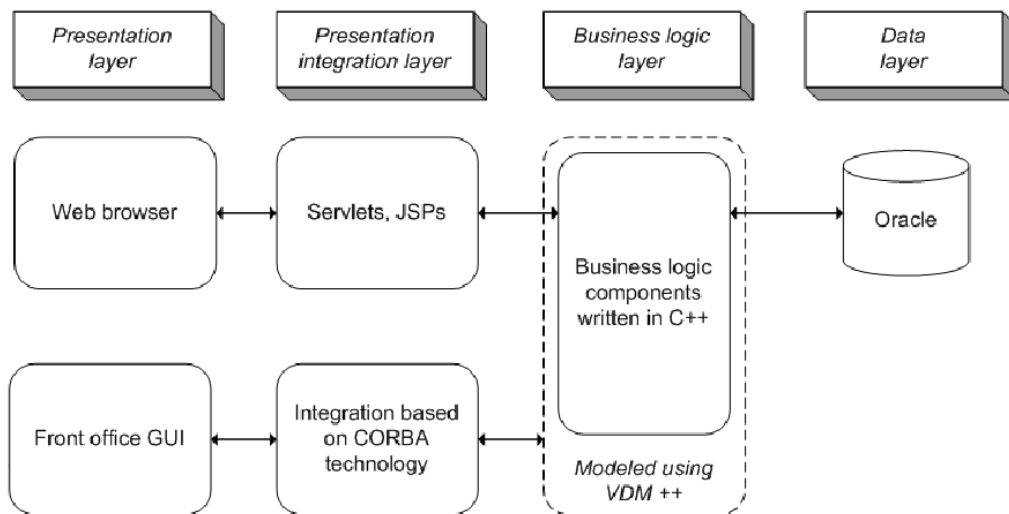
図 12-16: ディスプレイアーキテクチャ(オレンジ色はディスプレイアプリケーション、緑色はウィンドウマネージャ)

- ・ 詳細情報
 - 検証内容
 - ◇ 異なるデータ表示アプリケーションから、適切なディスプレイにデータがルーティングする、ウィンドウマネージャの性質を検証する。
- ・ 判断
 - 障害と工夫
 - ◇ 要求仕様を検証する性質に落とし込むこと
 - ◇ **Simulink** などの商用モデリングツールのデータを異なる種類の形式手法を用いた分析ツールの入力とする、変換ツールを作ること
 - ◇ 分析結果を分析者と製品エンジニアから分かりやすくするツールを作ること
 - ◇ アプリケーションを個々に分析できるようなサブシステムに分解する手法を策定すること
 - ◇ プロセスを改善できるように、検証プロセスを反復していくこと
 - ◇ **Rockwell Collins** 社とミネソタ大が共同開発した変換フレームワークを利用している。このフレームワークにより、商用モデリングツールの言語から、モデル検査器や定理証明器への変換ができるようになっている。
 - ◇ 次の3つの要件を満たす検証プロセスを策定すること
 - 分析結果の妥当性
 - ソフトウェア要求仕様からすべての形式的な検証性質が追跡可能であること
 - すべての要求項目が少なくとも一つの形式的な検証性質により検証されること
- ・ 組織
 - 教育
 - ◇ 開発者は1日程度の訓練と作業中の助言だけでモデル検査ツールの実践的な導入が可能

- ・ 情報源
 - Steven P. Miller, Michael W. Whalen, Darren D. Cofer, Software Model Checking Takes Off, Communications of the ACM, February 1, 2010, pp.58-84.
 - NASA LaRC Formal Methods Program Research

12.2.26. CSK - TradeOne

ドメイン	バックオフィスシステム
開発対象	証券バックオフィスシステム TradeOne
国	日本
開発組織	CSK
形式手法(言語、ツール)	VDM++ (VDMTools)
適用範囲・規模(形式手法)	TradeOne 全体のうち、マル優サブシステムとオプションサブシステムに適用した(ソースコード行数の割合では約 6 割)。 マル優サブシステムに関して VDM++の行数は、ビジネスロジックが 8,102 行、制約が 1,539 行、TradeONE システムのユーティリティが 1,342 行、汎用的な目的のためのユーティリティが 774 行 合計 11,757 行。 オプションサブシステムに関しては、ビジネスロジックが 10,846 行、テストシナリオが 13,771 行、テストケースが 31,641 行等、合計 68,170 行
適用対象のソフト種別	エンタプライズ系
適用目的・工程	仕様記述、設計
実装言語	C++, Java
実装規模	TradeOne 全体では、1,342,858 DSI (DSI はソースコード行数とほぼ同義。COCOMO で利用される。) マル優サブシステムは 18,431DSI オプションサブシステムは 60,206DSI
効果	エラー発生率が、 <ul style="list-style-type: none"> • マル優サブシステムは 0 件 • オプションサブシステムは 0.05/KDSI • TradeOne 非適用部分では、0.67/KDSI であった。 生産性が、見積りよりも 50-60%程度向上した。



(出典: John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, Marcel Verhoef, Validated Designs for Object-oriented Systems, version 1.2, 2004)

図 12-17: TradeOne システムの構成

- 詳細情報
 - 検証規模
 - ◇ テストの達成率は、C0(命令網羅)レベルで 98%であった。
 - 期間
 - ◇ 2000-2001
- 判断
 - 形式手法を利用した動機
 - ◇ 上流工程での検証が可能であるため、高信頼性、高生産性を実現できる。
- 組織
 - 体制
 - ◇ マル優サブシステム
 - 6人チーム(ひと月あたり平均4人、3.5か月間)で開発された。まずシステムアーキテクトが VDM++モデルの全体的な基本的なフレームワークを設計した。次に、4人の VDM++経験者が、並行してモデルを作成した。全ての VDM++モデルはチーム全てのメンバーによってレビューされた。最後にプログラマが VDM++モデルから手動で実装した。
 - 全ての開発者は40歳を越えており、ソフトウェア開発経験は20年以上あった。
 - 4人は、形式手法について基本的な事前知識があった。
 - ◇ オプションサブシステム:
 - 10人チームで開発された。
 - まずドメインのエキスパートが日本語でオプションサブシステムの機能要求を記述した。
 - 並行して、マル優サブシステム開発者の VDM++エキスパートの一人が、1週間以内で VDM++について指導した。
 - 次に、ドメインエキスパートがオブジェクト指向のエキスパートの協力を得て、UMLでユースケースを書いた。
 - 並行して、マル優サブシステム開発者であったシステムアーキテクトが、VDM++モデルの全体的なフレームワークを記述した。
 - 次に、二人の VDM++エキスパートと、新しく指導された3人のプログラマが

VDM++モデルを完成させた。

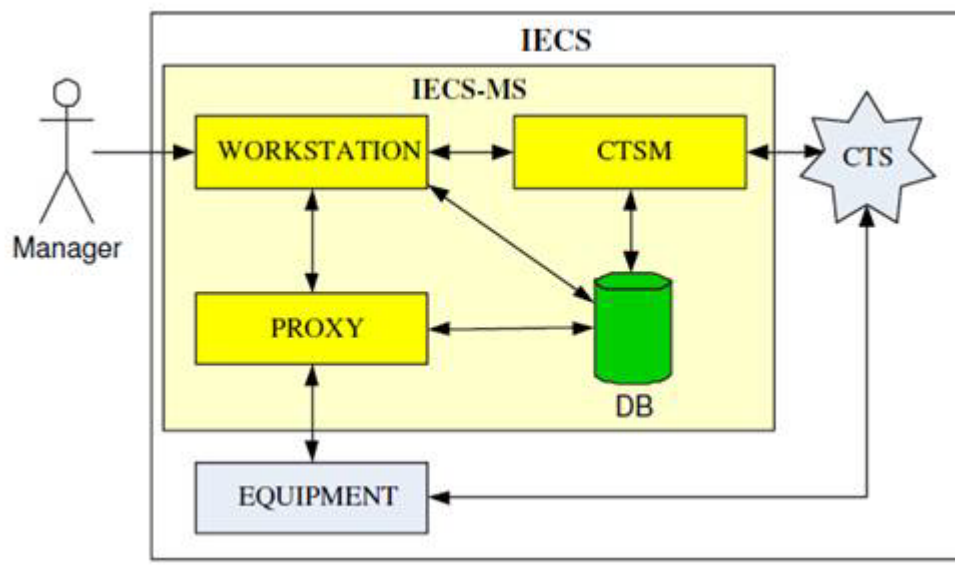
- 教育
 - ◇ オプションサブシステムの開発工数は合計で 60.1 人月であり、教育に要した時間はそのうちの 2.8 人月であった。

・ 情報源

- ソフトウェアコンポーネント技術に関する調査研究報告書,
<http://it.jeita.or.jp/eltech/report/2004/04-jou-4.html>, JEITA
- John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, Marcel Verhoef, Validated Designs for Object-oriented Systems, version 1.2, 2004,
<http://overtureeditor.googlecode.com/svn/trunk/Documentation/literature/VDM%20Book/bookfinal.pdf>
- VDM information web site - 導入事例,
<http://www.vdmtools.jp/modules/tinyd1/index.php?id=5>

12.2.27. Selex Communications - 船舶通信システム

ドメイン	船舶
開発対象	船舶通信システム IECS
国	イタリア
開発組織	Selex Communications
形式手法(言語、ツール)	SPIN (ステート図や、PSC: Property Sequence Charts に基づく検証プロパティを CHARMY で記述し、SPIN 用に変換。)
適用範囲・規模(形式手法)	不明
適用対象のソフト種別	制御
適用目的・工程	仕様記述、設計
実装言語	不明
実装規模	不明
効果	各プロパティの検証および、デッドロックや到達不能パスや正しくない最終状態が存在しないことを検証できた。



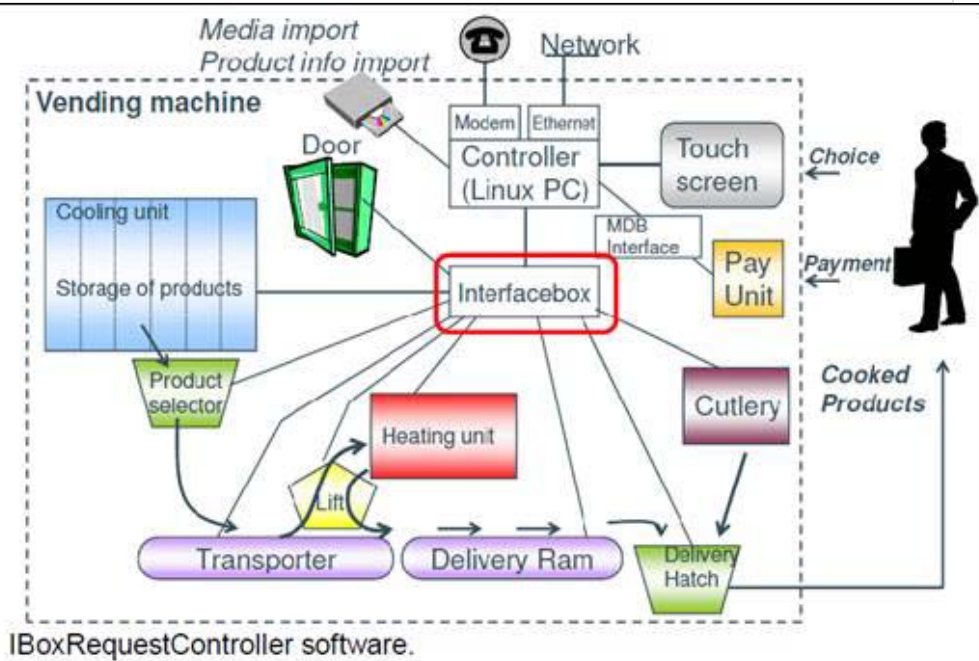
(出典: Daniela Colangelo, Daniele Compare, Paola Inverardi, and Patrizio Pelliccione, Reducing Software Architecture Models Complexity: a Slicing and Abstraction Approach, Formal Techniques for Networked and Distributed Systems-FORTE 2006)

図 12-18:IECS ソフトウェアの構成図

- 詳細情報
 - 検証内容
 - ◇ デッドロックが無いこと、正しくない最終状態が存在しないこと、到達不能なパスが存在しないこと。その他、**Activate Service** (ユーザがサービスを非活性化させていないとき、**ActivateService** リクエストの送信後、サービスがアクティベートされる)、**Deactivate Service**、**Reconfiguration Service**、**Modify Equipment**、**Modify Equipment by Service** の各プロパティ。
 - 検証規模
 - ◇ SPIN でのデッドロックフリー等の検証時のモデルのサイズは以下のとおりであった。
 - 状態数: 1.3e+08
 - 遷移数: 6.2e+08
 - メモリ使用量: 2.0Gb
 - ◇ 各プロパティの検証については、状態爆発を起こしてしまったため、スライシング技術を利用してモデルサイズを削減し、検証を実施した。
- 判断
 - 障害と工夫
 - ◇ モデルのサイズを小さくするため、CARMY で記述されたソフトウェアアーキテクチャモデルから、TESTOR を拡張して構築した DEPCOL アルゴリズムやスライシング技術によって検証すべき部分を抽出した。
- 情報源
 - Daniela Colangelo, Daniele Compare, Paola Inverardi, and Patrizio Pelliccione, Reducing Software Architecture Models Complexity: a Slicing and Abstraction Approach, Formal Techniques for Networked and Distributed Systems-FORTE 2006

12.2.28. Sioux - ホットフード自動販売機

ドメイン	その他
開発対象	ホットフード自動販売機
国	オランダ
開発組織	Sioux Embedded Systems
形式手法(言語、ツール)	Verum ASD Suite
適用範囲・規模(形式手法)	自動販売機を中心とするコンポーネント
適用対象のソフト種別	制御系
適用目的・工程	設計、自動コード生成
実装言語	C#
実装規模	2,889LOC
効果	品質および生産性が向上した。(ASDの教育時間を除くと30%生産性が向上した。教育時間を含めると1%生産性は低下した。)



(出典: Leon Bouwmeester and Arjen Klomp, Improving productivity and quality using the ASD:Suite, Sioux evaluates Verum's ASD for embedded software, 2009)

図 12-19:開発された自動販売機の構成図

- 詳細情報
 - 検証内容
 - ◇ デッドロック、ライブロック、レースコンディションの有無を検証した。
 - 期間
 - ◇ 2 か月間
- 判断
 - 形式手法を利用した動機
 - ◇ 形式手法を利用することで、従来の開発よりも生産性や品質に良い影響を与えるか

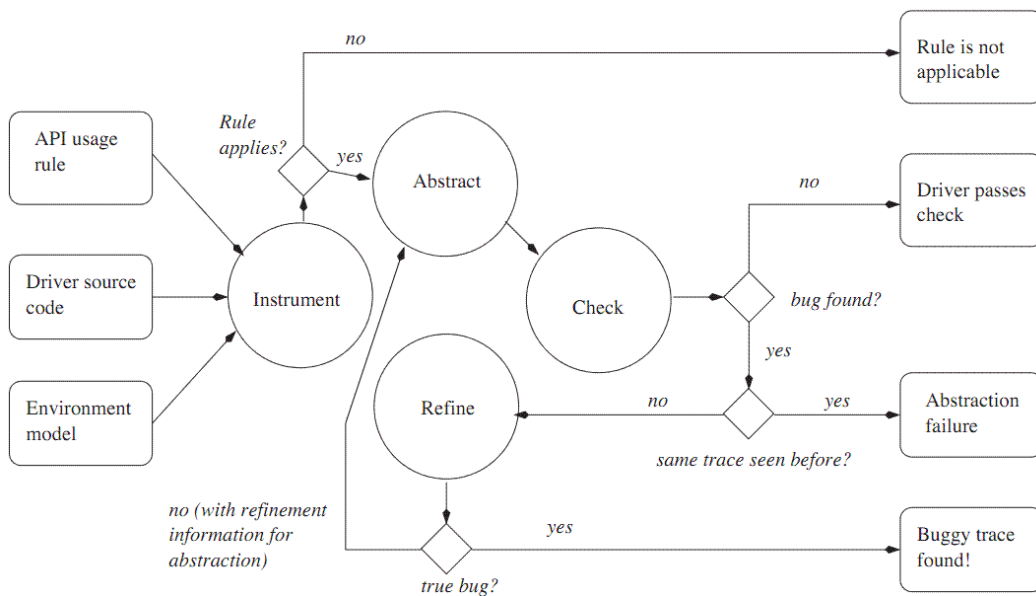
どうかの評価を行うためであった。(該当ソフトウェアは従来手法で既に開発済みであり、今回新たに ASD を用いて再開発を行い、比較した)

- ・ 組織
 - 体制
 - ◇ 熟練のソフトウェアエンジニアがフルタイム、熟練のソフトウェアアーキテクトが 20% のエフォートで開発に携わった。
 - 教育
 - ◇ ASD の教育に 2 人合計で 140 時間を費やした。
 - その他
 - ◇ Verum のアシスタントが必要に応じてサポートした。

- ・ 情報源
 - Leon Bouwmeester and Arjen Klomp, Improving productivity and quality using the ASD:Suite, Sioux evaluates Verum's ASD for embedded software, 2009

12.2.29. Microsoft - Windows デバイスドライバ

ドメイン	その他
開発対象	126 の WDM (Windows Driver Model) ドライバと 20 の KMDF (Kernel Mode Driver Framework) ドライバの検証
国	米国
開発組織	Microsoft
形式手法(言語、ツール)	SLAM (Static Driver Verifier tool)
適用範囲・規模(形式手法)	WDM ドライバについては 60 のルール(検証項目)を設定し、KMDF ドライバについては 40 のルールを設定した。
適用対象のソフト種別	リアルタイム制御系
適用目的・工程	コード検証
実装言語	C
実装規模	48-130,000LOC であり、平均 12,000LOC
効果	長年利用されてきたドライバの不具合を多数発見することができた。



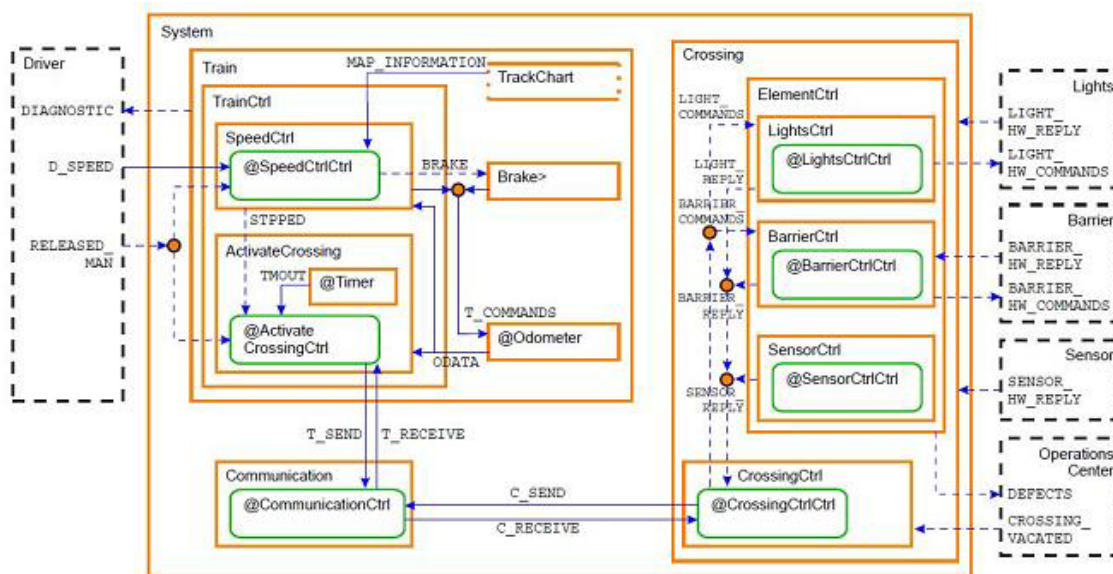
(出典: Ball, T. and Bounimova, E. and Cook, B. and Levin, V. and Lichtenberg, J. and McGarvey, C. and Ondrusek, B. and Rajamani, S.K. and Ustuner, A., Thorough static analysis of device drivers, Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006)

図 12-20: Static Driver Verifier tool の解析エンジンアーキテクチャ

- ・ 詳細情報
 - 検証内容
 - ◇ `IoCompleteRequest` が呼び出されているときはドライバは `STATUS_PENDING` を返さない
 - ◇ もし存在するならプラグアンドプレイ I/O リクエストパッケージはより低レイヤのドライバに渡される
 - ◇ `IoAttachDeviceToDeviceStack` は適切なデバイスオブジェクトと共に呼び出される等。
 - 検証規模
 - ◇ 約 93%について自動的に検証することができた
- ・ 判断
 - 形式手法を利用した動機
 - ◇ コードが取りうる全ての振舞いを検証するため
 - 手法・ツールの選択理由
 - ◇ Windows のデバイスドライバを検証するため
 - ◇ 検証のための入力を必要としないため
 - 障害と工夫
 - ◇ SDV における Windows カーネルモデルやルール(検証項目)の不具合等により、デバイスドライバに不具合が無くても不具合があると報告されることがあったため、これらの修正を並行して行った。
- ・ 情報源
 - Ball, T. and Bounimova, E. and Cook, B. and Levin, V. and Lichtenberg, J. and McGarvey, C. and Ondrusek, B. and Rajamani, S.K. and Ustuner, A., Thorough static analysis of device drivers, Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006

12.2.30. ボンバルディア - 鉄道制御システム

ドメイン	鉄道
開発対象	Deutsche Bahn 社の無線鉄道制御システム
国	ドイツ
開発組織	ボンバルディア
形式手法(言語、ツール)	Rational StateMate/LSC (Live Sequence Chart)
適用範囲・規模(形式手法)	不明
適用対象のソフト種別	リアルタイム制御系
適用目的・工程	要件定義、システム設計、ソフトウェア設計
実装言語	不明
実装規模	不明
効果	形式的に要求を記述することにより、仕様の厳密さが高まり、再利用性も向上した。



(出典: Brill, M. and Buschermohle, R. and Damm, W. and Klose, J. and Westphal, B. and Wittke, H., Formal verification of LSCs in the development process, Integration of Software Specification Techniques for Applications in Engineering, pp.494--516, 2004)

図 12-21: 鉄道制御システムの主要な StateMate モデル

- ・ 詳細情報
 - 検証内容
 - ◇ 列車が地上子などのアクティベーションポイントに近づいたら必ず通信チャンネルがセットアップされる。
- ・ 判断
 - 形式手法を利用した動機
 - ◇ 開発するコンポーネントが安全要求を満たしていることを保証するため。
 - ◇ モデル検査はテストと異なり全ての入力および入力の組合せについて検査すること

ができるため。

- 手法・ツール選択理由
 - ◇ **Statemate** に組み込まれている **Live Sequence Charts** はよく知られている **MSC2000** の **Message Sequence Chart** や **UML** のシーケンス図を基にしており、クリティカルな通信プロトコルの記述および検証に向いているため。
- 障害と工夫
 - ◇ モデル検査はモデルを解析するのみであり、ハードウェアやソフトウェアの実装に関して検証するものではない。そのため、作成したモデルから自動的にテストを作成し、実装に適用した。

- 情報源

- Bohn, J. and Damm, W. and Klose, J. and Moik, A. and Wittke, H. and Ehrig, H. and Kramer, B. and Ertas, A., Modeling and validating train system applications using statemate and live sequence charts, IDPT, 2002
- Brill, M. and Buschermohle, R. and Damm, W. and Klose, J. and Westphal, B. and Wittke, H., Formal verification of LSCs in the development process, Integration of Software Specification Techniques for Applications in Engineering, pp.494--516, 2004

13. 手法の概要および選択法に関する情報

想定読者	(1)プロジェクト管理者 (2)開発者
目的	7 章「手法の選択方法」で対象としなかった手法以外の代表的な形式手法に関する概要を整理し、目的に即した形式手法を特定するための情報を提供する。また、モデル検査の性質検証に利用することの多い、LTLとCTLについて整理する。
想定知識	ソフトウェア開発に関する基礎
得られる事	● 主な形式手法の概要

13.1. その他の代表的な形式手法

本ガイダンス本編の「第 7 章 手法の選択方法」では、実システムへの応用実績の多い形式手法について具体的な特徴や実績を示した。本節では、それらに準じる主要な形式手法について概要をまとめる。

手法名	年代	開発組織	説明	関係サイト
PVS	1980年代	米国・スタンフォード研究所 (SRI)	<ul style="list-style-type: none"> ● PVS(Prototype Verification System)は形式仕様記述と検証のためのシステムである。 ● PVS は、 <ul style="list-style-type: none"> * 仕様記述言語(関数、集合、リストなど) * 事前定義された定理 * 型チェッカー * 対話的定理証明器 * シンボリックモデル検査器(BDD) ● などからなる。 ● 仕様記述言語は、高階型付き言語に基づいている。表現力のある言語と強力な自動推論により、大規模な形式化と証明に対応できる。 ● PVS は SRI における 25 年間の形式手法ツール開発の知見に基づいている。 	PVS Specification and Verification System http://pvs.csl.sri.com/
FDR2	1990年代	Formal Systems Europe Ltd. (Oxford University)	<ul style="list-style-type: none"> ● FDR2(Failures/Divergence Refinement 2) は CSP(Hoare's Communicating Sequential Processes)で記述されたモデルのプロパティに対するモデル検査ツールである。 ● 状態遷移のリファインメント関係、デッドロックやライブロックを検証できる。 	http://www.fsel.com/index.html
Alloy	1990年代	MIT (Daniel Jackson)	<ul style="list-style-type: none"> ● Alloy は形式仕様記述言語と自動解析ツールを含む形式手法である。言語は、ソフトウェア設計のための軽量なモデリング言語である。記述した仕様に対して、Alloy Analyzerを用いて完全に自動的な分析ができる。また、可視化機能によって、解法と判例を把握しやすい。 ● Oxford 大の Z の表記法と、Pittsburgh 大の SMV の解析に啓発されて作られたとされる。 ● Alloy は関係代数に基づいており、ドット演算子による join 演算は RDB の join と類似している。 ● SAT に基づく自動解析を特徴とする。 	Alloy Community http://alloy.mit.edu/community/

SAL	2000年代	米国・スタンフォード研究所 (SRI)	<ul style="list-style-type: none"> ● SAL(Symbolic Analysis Laboratory)は、抽象化、プログラム解析、定理証明、シンボリックモデル検査などのツールを組み合わせたフレームワークである。 ● SAL の重要な部分は、遷移システムを記述する中間言語である。この中間言語は、他のモデリング言語やプログラミング言語への変換や、他の解析ツールへの入力となることを意図している。もともとは、状態遷移図と $Mur\phi$ や SMV といったモデル検査ツール、さらには不変条件の生成ツールとの間の中間言語を意図していた。実際のところ、SMV への変換の負担から独自のモデル検査器を作っている。 ● モデル検査ツールは、BDD を用いたシンボリックモデル検査と、SAT を用いた有界モデル検査、および試験的な "Witness"モデル検査、さらには SMT ベースの "Infinite"有界モデル検査ツールである。検証性質は LTL で表現する。 ● シミュレータ、デッドロック検査ツール、自動テスト生成ツールも具備している。 	http://sal.csl.sri.com/
CBMC	2000年代	Carnegie Mellon University	<ul style="list-style-type: none"> ● CBMC は ANSI-C と C++プログラムの有界モデル検査ツールである。 ● 配列の境界 (バッファオーバーフロー)、ポインタの安全性チェック、例外とユーザの指定したアサーションについて検証できる。検証は、プログラム中のループ展開と論理式への変換によりなされる。 	http://www.cprover.org/cbmc/
JavaPathfinder	2000年代	NASA Ames Research Center	<ul style="list-style-type: none"> ● JPF(Java Pathfinder)は Java のバイトコードを対象としたモデル検査ツールである。 ● デッドロック、アサーション違反、処理されない例外などを検出する。検査は JPF 独自の JVM により解釈、実行される。 ● NASA で開発されたオープンソースソフトウェアであり、火星探査機の制御システムの検証などにも使われた実績を持つ。 	http://babelfish.arc.nasa.gov/trac/jpf/wiki/intro/start
ESC/JAVA2	1990年代	School of Computer Science and Informatics at University College Dublin	<ul style="list-style-type: none"> ● ESC/Java2(Extended Static Checker for Java version 2) は JML(Java Modeling Language)のアノテーションを含む Java プログラムの静的コード解析を行い、ランタイムエラーを検出する。 ● JMLは契約に基づく設計(DbC: Design by Contract¹⁶³)を念頭に置いた言語であり、事前条件、事後条件などを記述する。 ● ESC/Java2 は、JML 付きのプログラムを論理式で表現し、プログラムのアノテーションで記述された仕様に対する妥当性を検証する。証明器は、Simplify と呼ばれる。 ● 基になった DEC/SRC ESC/Java は DEC/Compaq/HP で開発された。 	http://secure.ucd.ie/products/opensource/ESCJava2/
Coq	1980年代	INRIA-Rocquencourt	<ul style="list-style-type: none"> ● Coq は型理論に基づく、定理証明系である。Coq は、数学的証明、形式仕様、プログラム、プログラムの仕様に対する正しさの検証用途に設計されている。 ● Coq は Gallina という名の形式仕様記述言語を提供する。Gallina の用語は、プログラムと、そのプログラムの性質、およびそれらの性質の証明を表現できる。 	http://coq.inria.fr/whatis-coq

¹⁶³プログラムコードの中にプログラムが満たすべき仕様についての記述を埋め込むことで設計の安全性を高める方法。

Bandera	2000年代	Kansas State University, University of Nebraska (Lincoln).	<ul style="list-style-type: none"> ● Bandera は Java のソースコードに対するモデル検査するためのツールセットである。 ● プログラムのスライシングと抽象解釈を行うことで、モデル検査を行う。ソフトウェアモデル検査であり、モデルを作る難しさと性質を記述する難しさを緩和することを目標としている。性質の記述は LTL で行う。 	http://bandera.projects.cis.ksu.edu/
CafeOBJ	2000年代	北陸先端大学院大学	<ul style="list-style-type: none"> ● CafeOBJ 言語は OBJ 言語を拡張した代数仕様言語である。振舞仕様、書き換え仕様、パラメータ化仕様などが記述できる。 ● CafeOBJ 言語システムは、等式を書き換え規則として実行することで等式推論を健全にシミュレートすることができ、対話型検証システムとして利用できる。 ● CafeOBJ はメタ言語 (Meta Language) としての機能も備えており、形式言語設計にも使える。 	http://www.idl.jaist.ac.jp/cafeobj/
LOTS	1980年代	University of Twente など	<ul style="list-style-type: none"> ● LOTS は (Language Of Temporal Ordering Specification) 分散システム、OSI (Open Systems Interconnection) サービス・プロトコルの設計に利用する形式仕様記述言語である。 ● CCS と CSP に基づくプロセス代数と、抽象データ型言語 (ACT ONE) に基づくデータ代数とからなる。並行、非決定的、同期的、非同期的なコミュニケーションの記述に適している。ISO 標準にもなっている。 	http://www.cs.stir.ac.uk/~kjt/research/well/
Agda		Chalmers 工科大学、産業技術総合研究所システム検証研究センター	<ul style="list-style-type: none"> ● Agda は依存型関数プログラミング言語である。値に依存して変化するデータ型を持つことで、ML や Haskell よりも複雑な型を扱うことができる。 ● また、構成的型理論に基づく対話型定理証明支援系でもある。依存型に基づく定理証明型であり、Coq にも類似する。 	http://wiki.portal.chalmers.se/agda/pmwiki.php
Z	1970年代	Programming Research Group (PRG) at the Oxford University Computing Laboratory (OUCL)	<ul style="list-style-type: none"> ● Z 記法は、計算機システムを表現するための形式仕様記述言語である。抽象的な高水準で複雑な振る舞いを正確かつ簡潔に書くことを目的としている。 ● Z の意味は Zermelo-Fraenkel 集合論と一階述語論理に基づいており、Z による数式表現は、代数的、論理的に扱うことができる。 ● Z の特徴的な考え方には「スキーマ」がある。スキーマは、いくつかの公理 (axiom) により仕様記述された名前付きオブジェクトの集合であり、Z はスキーマをまとめて大きな仕様を後段で作成できるような仕組みを提供する。 ● Z は ISO 標準にもなっている。 	Z User Group http://www.zuser.org/

13.2. LTL と CTL

モデル検査における性質の検証には、時相論理が一般に利用される。SPIN(7.3.1.1)では LTL、NuSMV(7.3.1.2)では LTL と CTL、UPPAAL(7.3.1.3)では CTL のサブセットを利用している。

以下では、LTL と CTL により表現される性質について整理し、モデル検査による性質記述を把握するための情報を提示する。

図 13-1 は状態遷移モデルを LTL と CTL がどのようにとらえるかを概念的に示している。前者は実行パスによる直線的な解釈をし、後者は木構造による解釈をする。13.2.1.1 では LTL による表現について、13.2.1.2 では CTL による表現について、13.2.1.3 では LTL と CTL の表現パターンライブラリについて記載する。

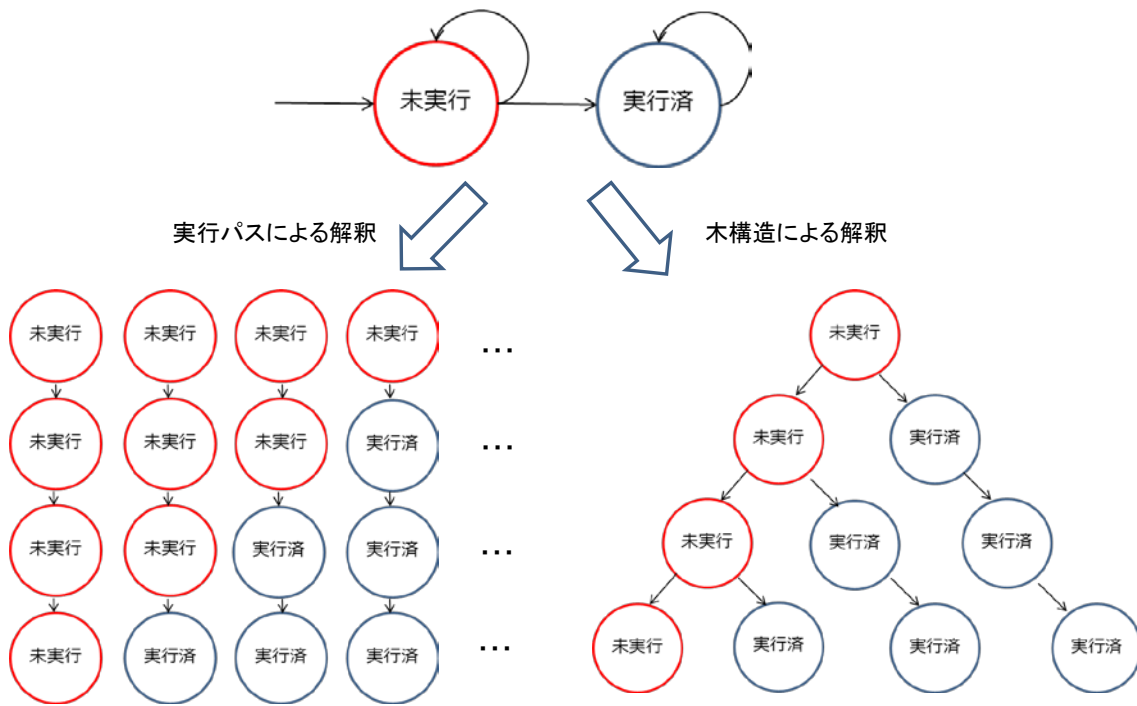


図 13-1: LTLとCTLの簡約な表現¹⁶⁴

13.2.1.1. LTLによる表現

LTL では、無限長の実行パスによる表現により、性質が成り立つかどうかを記述する。LTL では、図 13-2 に示す性質を記述できる。

- ① 「いつか」は、現在の状態か、それ以降の状態で P が成り立つことを意味する。
- ② 「つねに」は、現在の状態と、それ以降のすべての状態で P が成り立つことを意味する。
- ③ 「つぎに」は、現在の状態の次の状態に P が成り立つことを意味する。
- ④ 「～まで」は、現在の状態か、それ以降の状態まで、 P が成り立つことを意味する。

¹⁶⁴出典: "Model Checking A Hands-On Introduction", A. Cimatti, M. Pistore, and M. Roveri, 2003 に基づき MRI が作成

13.2.1.3. 検証性質の記述パターンライブラリ

検証性質は時相論理式で書かれるので、専門知識が必要となる。しかし、並行・リアクティブシステムの仕様において有限状態の検証で利用される検証性質にはいくつかのパターンがあると考えられ、そのパターンを整理してリポジトリ化する動きがある¹⁶⁷(以下、パターンライブラリと記す)。

パターンライブラリには LTL や CTL の性質記述パターンも含まれており、代表的な検証性質が表 13-1 に示す階層で分類されている。また、パターンライブラリとは別に、8.2.4 小節にも典型的な記述例を示している。

表 13-1: 性質記述パターンライブラリの概要¹⁶⁸

Occurrence パターン	
absence	ある状態やイベントが、指定された対象範囲内で決して生じないことを表す。
universality	ある状態やイベントが、指定された対象範囲内で常に生じることを表す。
existence	ある状態やイベントが、指定された対象範囲内で生じる(ことがある)ことを表す。
bounded existence	ある状態やイベントが、指定された対象範囲内で k 回生じる(ことがある)ことを表す。バリエーションとして、少なくとも k 回や高々 k 回などがある。
Order パターン	
precedence	状態またはイベント P, S に対して、指定された対象範囲内で、P の前に必ず S が先行することを表す。
response	状態またはイベント P, S に対して、指定された対象範囲内で、P の前に必ず S が応答することを表す。
precedence chain	状態またはイベントの列 P1, ..., Pn, および S1, ..., Sm に対して、指定された対象範囲内で、P1, ..., Pn の前に必ず S1, ..., Sm が先行することを表す。
response chain	状態またはイベントの列 P1, ..., Pn, および S1, ..., Sm に対して、指定された対象範囲内で、P1, ..., Pn の前に必ず S1, ..., Sm が応答することを表す。

¹⁶⁷ 出典: Spec Patterns, SAnToS laboratory, <http://patterns.projects.cis.ksu.edu/>

¹⁶⁸ 出典: "Spec Patterns", SAnToS laboratory, <http://patterns.projects.cis.ksu.edu/>に基づき MRI が作成

14. ソフトウェア再利用とフォーマルメソッド

ここではソフトウェア再利用における検証一般の問題について説明した後、そこでフォーマルメソッドがどのように利用できるか、いくつかの利用局面について紹介する。それを踏まえ、ソフトウェア再利用においてフォーマルメソッドを活用する際に必要となると考えられるいくつかの留意点について触れる。

14.1. ソフトウェア再利用

他のソフトウェアの構成要素や構造を、ソフトウェアの開発に利用する技術のことをソフトウェア再利用という。ソースコードやバイナリレベルでの再利用から、設計やアーキテクチャの再利用、要求仕様の再利用など、およそソフトウェアの様々な成果物が再利用の対象となる。

図 14-1 は、再利用の基本的な構造を模式的に示したものである。ここで再利用資産は再利用されるソースコードや設計などを示し、それが複数の製品の開発に使われることを示している。単一の再利用資産を利用するというものもあるが、例えばクラスライブラリ、フレームワーク、あるいはプロダクトライン開発におけるコア資産のように、複数の再利用資産を体系的に管理し、各製品の開発において必要なものを取捨選択して再利用することも多く行われている。こうした際には再利用資産の組み合わせ方などに条件や制約が発生することがあるので、再利用資産の妥当な構成方法について考慮しながら再利用することになる。

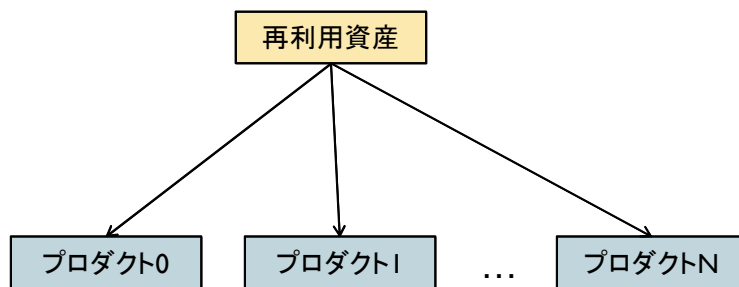


図 14-1: 再利用の基本構造

再利用が行われる理由は、それによって様々な利点が期待されるからである。表 14-1 は、ソフトウェア再利用に対する様々な期待を示したものである。ここに示されるように、新規に作るためには人的なリソースがかかり、また開発上のリスクも大きく、さらに利用実績のないものは信頼性の確保が大変であり、開発期間がかかるため、再利用によってより効率的かつ低リスクで信頼性の高いソフトウェアを作ることが期待感としてあることが分かる。

表 14-1: 再利用に対する期待¹⁶⁹

信頼性の向上	実績があり安定したソフトウェアを使うことで、新たに作るよりも信頼性が高まる
開発上のリスクの軽減	これから新規に開発するよりも、検証済みの既存のものを使うほうがリスクが小さい
開発リソースの有効利用	専門家の知識や労力の結果である既存資産を活用 (毎回新規開発すれば専門家を毎回投入する必要がある)
標準化への適合	標準に適合化のために標準的な部品を再利用する (標準的な GUI 部品を使うなど)

¹⁶⁹ Pressmann, Roger S.: Software Engineering, McGraw-Hill, 邦訳: 西他訳、実践ソフトウェアエンジニアリング - ソフトウェアプロフェッショナルのための基本知識、日科技連出版社。

開発期間の短縮	すべてを作らずに再利用することで開発期間を短縮し、製品化までの時間(time to market)を短縮する
---------	--

一方、再利用には様々な難しさもある。表 14-2 は再利用を行う際の困難を示したものである。ここに示されるように、再利用資産を使った開発上の問題だけでなく、保守を含めた運用全般の問題が困難の原因となっていることが分かる。

表 14-2: 再利用を行う際の困難

保守コストの増加	再利用部分のソースコードなどが利用できない場合には、保守のコストが増加する
ツール支援の欠如	多くの開発ツールは新規開発を支援するが、再利用の支援が不十分である
NIH シンドローム	ソフトウェア技術者は自分で新たに開発することを好む (保守がやりやすい、新たに作る方がチャレンジング) NIH: Not Invented Here
再利用資産の維持	部品を広め、使わせ、維持することは困難
検索、理解、適用	利用できる部品を探し、その部品を理解し、自分のソフトウェア開発に適用することはいずれも困難な作業

このように、ソフトウェア再利用に対する期待感はあるものの、実際に行うためには様々な困難さがあることも事実である。しかしながらソフトウェアは大規模化、複雑化する一方、開発期間は短くなっており、困難さにも関わらず再利用をしない開発はほとんど考えられない状況になっている。また上述したように、フレームワークやプロダクトライン開発のコア資産のような大規模で体系だった再利用資産の活用も進められており、ソフトウェア再利用の効果的な活用はソフトウェア開発における重要なテーマのひとつとなっている。

本章では、こうしたソフトウェア再利用において、フォーマルメソッドがどう使えるのか、ということについていくつかの考え方や使い方について紹介する。ここではフォーマルメソッドのうち、モデル検査技術に限定する。なおソフトウェア再利用に対するフォーマルメソッドの適用についてはいろいろな検討がなされているが、特定の使用方法が広く認められているわけではなく、また技術的な課題も多いのが現状である。ここで紹介するものは、そうした適用の考え方の一部であることを注意しておく。

14.2. 再利用における検証の課題

ソフトウェア再利用における検証上の課題について考えてみる。ソフトウェア再利用は、再利用されるソフトウェア(再利用資産)を作るという活動と、その再利用資産を利用してソフトウェア(プロダクト)を作るという活動によって構成される。ここで再利用資産は一般に複数回利用されることが通常である。このとき、再利用における検証の課題は、模式的に
図 14-2 のように捉えることができる。

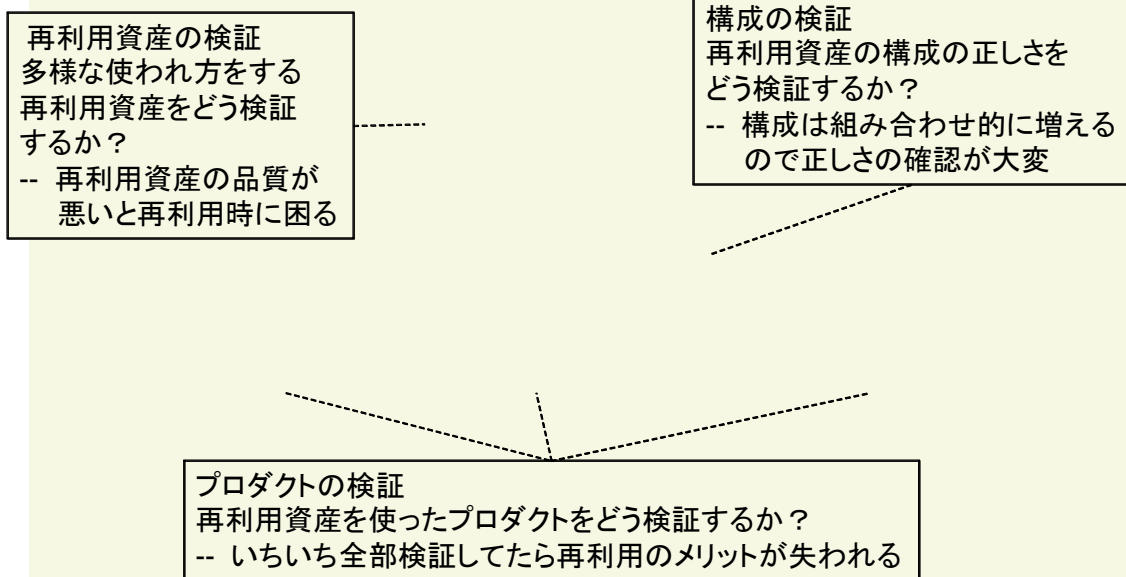


図 14-2: 再利用における検証一般の課題

14.2.1. 再利用資産の検証

一般にソフトウェアを検証する際には、そのソフトウェアがどのように使われるかということに照らして検証する。しかしながら再利用資産は様々なソフトウェアの開発に使われるため、その使われ方も多様である。したがって再利用資産に対しては、そうした多様な使われ方に照らして検証を行うことが求められる。もしも再利用資産に対して十分な検証がなされておらずその品質が不十分だと、仮に再利用資産を利用して少ないコストでプロダクトを構築することができたとしても、そのプロダクトの検証においては利用した再利用資産部分を含めて検証を行う必要がでてくる。一般に開発全体に占める検証のコストは非常に大きいため、これでは十分な開発の効率化が期待できない。

しかしながら、再利用資産を開発する時点では、それが将来どのようなプロダクトによってどのような使われ方をするかを決定することができないことが多く、多様な使い方に即した検証をあらかじめ行うことは困難である。仮にある程度の使い方が想定されたとしても、それらすべてをあらかじめ検証することは検証量からできないことも多く、再利用資産開発時の検証を困難にしている。このように再利用資産をどこまでどのように検証するかは重要な課題である。

14.2.2. プロダクトの検証

再利用資産を利用してプロダクトを開発した場合は、利用せずに開発した場合より、再利用資産に関わる部分の検証が楽になることが期待される。上述したように検証のコストが削減できないと再利用によるコスト削減効果は小さくなるからである。

しかしながら現実には再利用資産を開発した時点で十分な検証を行うことは難しいため、そのプロダクトの使い方に即した検証はなされていない事もあり得る。既に実績のある再利用資産であっても、使い方が異なれば不具合が出てくることは十分にあり得る。また再利用の困難さに示されているように、再利用資産はそのソースコードが入手できない場合などもあり、利用者が十分な検証を行ったり、それに基づいて修正を行ったりすることが難しい場合も多々ある。

現実には、プロダクトの開発において過去に実績のない方法で再利用資産を使う場合には、プロダクトの検証時に再利用資産に関わる検証を行わざるを得ないため、そうした検証をどう効果的・効率的に行うかは重要な課題である。

14.2.3. 構成の検証

体系だった再利用を行うためには、必要と考えられる再利用資産を複数整備し、それらを体系的

に管理、活用することが求められる。例えばアプリケーションフレームワークは、特定業務やドメインの製品の開発に必要となるクラス定義などをあらかじめ用意することで、個々の製品を効率的に作ることを狙ったものである。また製品ライン開発においては、想定する製品群の共通性と可変性を分析し、そうした製品群の開発に必要な再利用資産をコア資産として整備する。

こうした再利用開発においては、複数の再利用資産を活用して行われるが、そうした再利用資産は、例えばある再利用資産を使う際には別の再利用資産も利用することを想定しているとか、ある再利用資産と別の再利用資産は組み合わせて使うことは意図していないとかいった様々な構成上の条件や制約が課せられることが一般的である。したがって、複数の再利用資産を用いて製品を開発する人は、自分の利用している再利用資産群の構成が正しいものであるかどうかを確認することが必要となる。

一般に再利用資産の構成方法は組合せ的に考えられるため、組み合わせる再利用資産の規模が多くなればなるほど、その正しさの確認は難しく煩雑になるため、そうした構成の正しさをどう検証するかも課題となる。

14.3. 再利用における検証手法の例

こうした再利用における検証の課題に対してどのような対応方法が考えられるのか、テストなどの一般的な検証における例を簡単に紹介する。

14.3.1. 再利用資産の検証

再利用資産の検証に対しては、考えられる利用方法をできるだけ網羅することが求められる。しかしながらあらゆる利用方法を網羅することは困難である。例えばひとつの関数がパラメータを複数持つとき、それらのパラメータの値や組み合わせをすべて網羅した検証を行うことは現実には不可能なことが多い。そのため、一般的なテストにおいては、パラメータの値を同値分割や境界値分割していくつかの代表値で検証したり、直交表などを用いてパラメータの対の組合せをできるだけ網羅するように検証したりすることで、より効果的な検証を行うことなどが行われる。

もちろんこうした手法は一定の有効性を示すが、再利用資産の組み合わせに関わる検証においては、その組合せを網羅することはさらに困難となる。ひとつの素朴な考え方としては、製品毎に使い方が異なる部分に関しては、再利用資産開発時には検証を行わず、製品によらず同じような使われ方をする部分に注力して検証を行うというアプローチもあり得る。しかしながら製品を開発する立場にとれば、製品毎に利用方法が違う部分に対しても、一定の検証がなされていることが期待される。

製品ライン開発などにおいてはコア資産が膨大となるため、例えばサンプルアプリケーション戦略(SAS: Sample Application Strategy)などが提案されている。SASは、コア資産検証の段階で、いくつかのサンプルアプリケーションを想定し、それらの使われ方の中でコア資産を検証する方法である。この方法は、実際のアプリケーションの利用方法の中で再利用資産を検証することができるため、現実的な検証が期待される。実際にアプリケーションを作って検証するためコストがかかり、また取り上げたサンプルアプリケーション以外のコンテキストでの検証はできないという問題はあるが、再利用資産開発の段階で早期の確認ができること、再利用資産開発時に想定されている製品をサンプルアプリケーションとして選定するなどすることで、現実的な戦略として活用できる利点がある。

14.3.2. 製品の検証

再利用資産を使った製品の検証においては、例えばその製品の使い方を踏まえ、十分な網羅性はなくても一通りの検証を行うことで再利用資産の品質を査定し、その程度や状況に応じて、再利用資産に関してどの程度の検証が必要か判断して検証を行うなど、再利用資産に関わる検証のコストを減少させるように運用することが多いと考えられる。しかしながら製品に求められる品質の程度や、再利用資産の状況によっては、再利用資産に関わる検証も新規部分と同

程度に行うこともあり得る。

そうした製品の検証を効果的に行うために、例えば共通性と再利用戦略 CRS (Commonality and Reuse Strategy)などが提唱されている¹⁷⁰。これは再利用資産開発時には製品によらず同じように使われる部分の検証に注力し、製品毎に利用方法が変わりうる部分については、その部分を検証するための検証資産を作成する方法である。ここで検証資産とは、例えばテストであればテストデータ、スタブ、ドライバ、期待値など、テストを行う際に必要となる各種の資産を指す。個々の製品の検証においては、共通部分の検証についてはコア資産の検証時に使われた検証資産を利用し、製品固有の使われ方をする部分の検証については、用意された検証資産をそのアプリケーションの使い方に適合させてからテストを行う。この戦略では、検証そのものを減らすわけではないが、検証資産の再利用によって、製品の検証のコストを削減することが期待される。

14.3.3. 構成の検証

再利用資産の構成については、再利用資産設計時に再利用資産間にどのような依存関係や排他関係があるかを定義しておき、再利用資産の特定の組み合わせがその関係を満たしているか、あるいはその関係を満たす再利用資産の組合せとしてどのようなものがあるかを求めることなどがなされる。

こうした構成の検証は、構成管理の分野で一定の技術が培われている。また製品ライン開発においては、フィーチャモデルを構成の検証に応用することも検討されている。フィーチャモデルは、本来は要求項目としてのフィーチャの構造を記述するためのモデリング手法だが、それを応用することで再利用資産の構造を導出したり表現したりする提案が多くなされている。図 14-3 はフィーチャモデルの記述例である。ここでは、製品に対するフィーチャが必須、選択(製品によって使われたり使われなかったりする)、代替(製品によって選択肢中のいずれかが使われる)が階層的に示されており、あわせてフィーチャ間の依存関係も示されている。フィーチャモデルはこうした条件や制約を満たしたフィーチャの構成をコンパクトに示していると考えられ、その構成を満たす具体的なフィーチャの組合せを導出する手法の検討などがなされている。

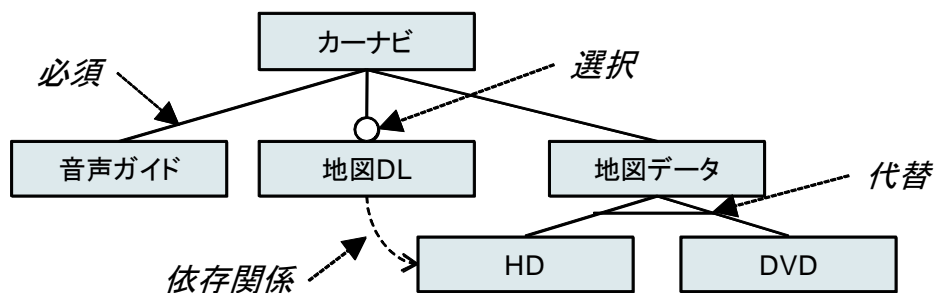


図 14-3: フィーチャモデルの例

14.4. フォーマルメソッドの適用

以上説明してきたように、ソフトウェア再利用はソフトウェア開発にとって重要で不可欠な技術であるが、その活用や運用には様々な困難さがあり、検証についてもいろいろと課題がある。こうした再利用における検証に、フォーマルメソッドを活用するという取り組みはいろいろとある。なおここでフォーマルメソッドとしてはモデル検査技術に限定する。

ソフトウェア再利用における検証にフォーマルメソッドを適用することの素朴な期待感としては、以下のようなものが挙げられる。

¹⁷⁰ Pohl, K. 他: Software Product Line Engineering, Foundation, Principles, and Techniques, Springer, 2005.

- 再利用資産は複数のプロダクトから繰り返し使われるものであるから、品質が高いことが求められる。したがって、フォーマルメソッドのように従来の検証手法より厳密で信頼の高い検証手法の活用が期待される。
- 再利用資産は様々なプロダクトから様々な使われ方をするために、その検証が困難である。モデル検査技術のような手法を用いることで、こうした使われ方に照らしてより網羅的な検証が可能になることが期待される。
- 再利用資産は繰り返し使われるため、単独のプロダクトに比較してより検証コストが高くついたとしても全体としてはメリットが得られることがあり得る。フォーマルメソッドは一般に高価な検証手法と考えられるが、再利用資産の検証という局面で、相対的に有利なコストで活用されることが期待される。

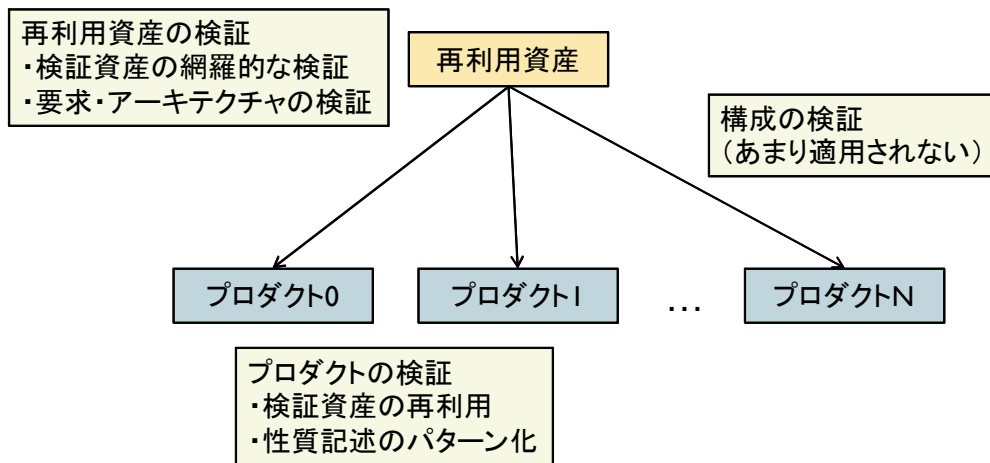


図 14-4: 再利用におけるモデル検査技術の適用の概観

図 14-4 は、再利用におけるモデル検査技術の適用について概観したものである。

再利用資産の検証においては、検証資産を様々な使われ方に照らして網羅的に検証するためにモデル検査を利用することが考えられる。この段階では詳細なプロダクトの機能などに関わる部分を検証するのではなく、要求やアーキテクチャに関わる重要なポイントについて検証することが多い。

一方、プロダクトの検証においては、モデル検査によるプロダクトの検証を効率的に行うために、検証資産を再利用することが考えられる。また検証資産の一部である性質記述については、様々なパターン化の試みなどがなされている。

構成の検証に対するフォーマルメソッドの適用例もいろいろとあるが、これらはモデル検査技術以外の技術を利用するものがほとんどであり、ここでは触れない。

以下、再利用資産の検証およびプロダクトの検証に対するフォーマルメソッドの適用例を紹介すると同時に、そうした適用を行うとしたときに、考慮しておくといと考えられる事項について簡単に説明する。

14.5. 再利用資産の検証とフォーマルメソッド

14.5.1. 検証方法の例

再利用資産の検証では、様々なプロダクトの使い方に応じた検証が必要であることを指摘した。モデル検査を利用することによって、そうした多様性を考慮した検証を効果的に行う提案がある。例えば、再利用資産の設定によってその振る舞いを変えることができ、プロダクトごとに設定が変わりうる場合、設定によって成り立つ性質が変わってしまう。そのような場合、どのような設定があり、その設定に応じてどのような性質を持つということを定義し、その性質を検証する必要がある。プロダクトライン開発などでは、こうした利用するプロダクトごとの違いを可変性と呼ぶが、再利用資産の検

証において、この可変性を利用することで、プロダクトに応じた多様な性質を一度にモデル検査することが可能となる。

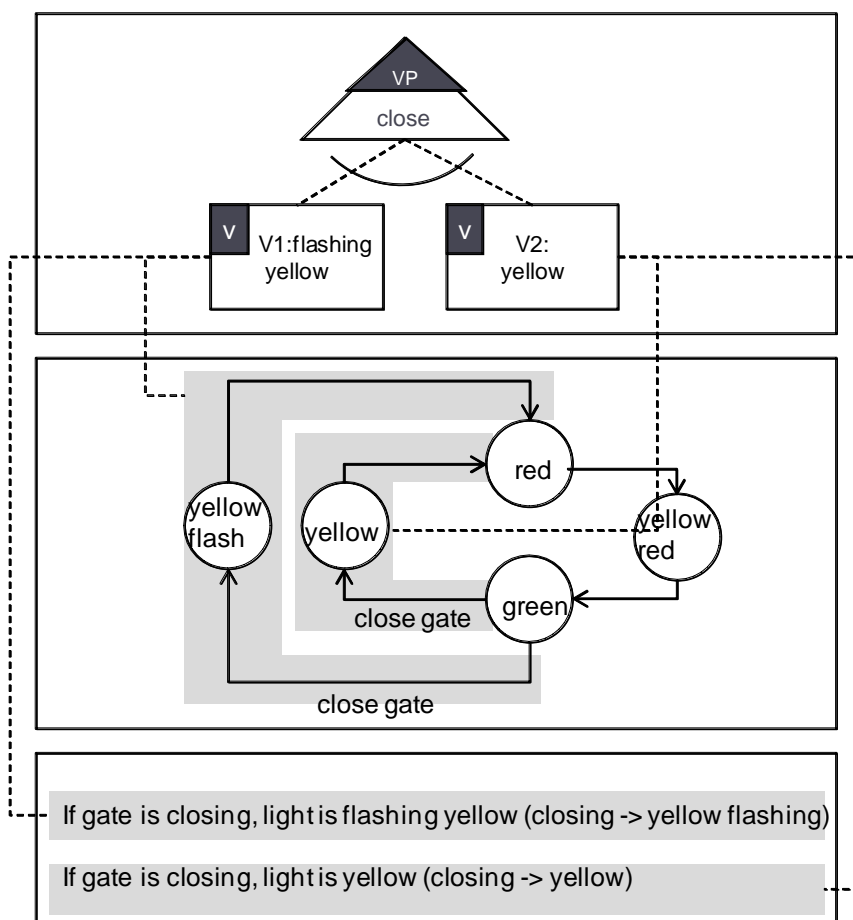


図 14-5: 再利用資産のモデル検査の例¹⁷¹

図 14-5 は、Lauenroth らによる再利用資産のモデル検査の概略を示すものである。ここでは信号システムの例が示されている。図の上部は、OVM(Orthogonal Variability Model)と呼ばれる可変性のモデルである。ここでは遮断されたときに黄色が点滅するプロダクトと、黄色が点灯するプロダクトがあることが示されている。図の中部は、対象とするシステムのふるまいを示す状態遷移図の一部である。ここで黄色が点滅(yellow flash)する状態と遷移と、点灯する(yellow)する状態と遷移が含まれている。ここで可変性の V1 が選ばれたときは点滅の状態や遷移が、V2 が選ばれた時は点灯の状態や遷移が選ばれる。また図の下部は性質記述で、V1,V2 に応じて異なった性質記述がある。モデル検査を行う際には、どの可変性が選ばれたならどのような性質が成り立つ、というように可変性の選択状態を含めた性質記述を用いてモデル検査を行う。一般に可変性は複数あるので、可変性の選択状態はベクトルで表すことができ、意味のあるベクトル(可変性の組み合わせ)を用いて性質記述をすることになる。なおこの研究では、検証対象は要求記述となっている。

14.5.2. 考慮点

再利用資産の検証を行う際に、プロダクトごとの使い方の多様性を一度に検証しようとする、上

¹⁷¹ Lauenroth, Kim, 他: Model Checking of Domain Artifacts in Product Line Engineering, Proc. of Automated Software Engineering, 2009.

述の例のように利用のされ方の多様性を整理する必要がある。プロダクトライン開発においては、こうしたプロダクトごとの違いを可変性として捉え、上記の例のような OVM や、前述したフィーチャモデルなどを使ってモデル化することができる。ただし、多様な性質を検証することで検証が複雑化し、単一のプロダクトの利用方法のみに関わる性質であれば検証が可能であったものが、検証できなくなるということもあり得る。上記の例で検証対象が要求記述となっているのは、設計記述になるとより対象が複雑化するためであるとも考えられる。

しかし仮に多様な性質を一度に検証することをしなくても、可変性モデルなどを使って多様性を捉えることは有効である。可変性モデルを使うことで、多様性の広がり認識することができ、その一部の状況に応じた性質しか検証していないとしても、それが本来の多様性のうちのどれだけの部分であるかを理解することができるからである。可変性モデルに照らして、どういう状況に関しては検証がなされているかを明確にすれば、プロダクトの開発を行う人は、自分の使い方が検証済の使い方なのかどうかを理解することができ、再利用資産に関わる検証をどれだけ行うかの判断に利用することもできる。

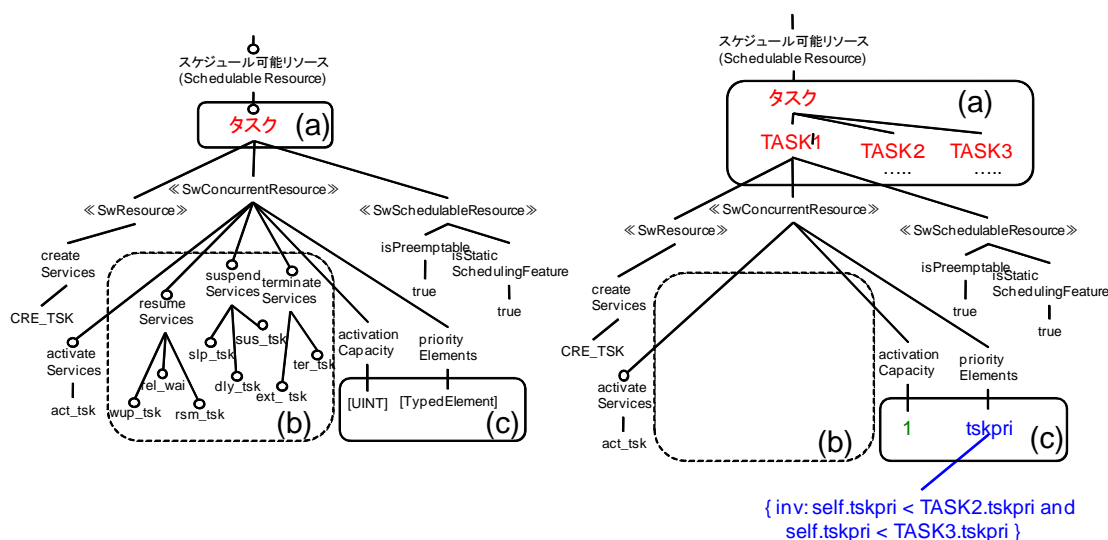


図 14-6: フィーチャモデルを用いた検証状況の明確化例¹⁷²

図 14-6 は、再利用資産を利用する際のリアルタイム OS の設定等の多様性をフィーチャモデルによって表現した例である。詳細は省くが、図の左は考えられる多様性をあらかじめテンプレートとして定義したもの、右はそのテンプレートを踏まえ具体的なプロダクトにおける設定等を明確にしたものである。こうした記述を利用することで多様な設定方法の中で、どういう設定について検証をしたのかが明確になる。

14.6. プロダクトの検証とフォーマルメソッド

14.6.1. 検証方法の例

再利用資産が完璧に検証しきれない以上、どれだけの検証コストを費やすかという程度の議論はあっても、プロダクトの検証において再利用資産の検証をゼロにすることはできない。少なくとも再利用資産の使い方が正しいかどうかはプロダクト側の問題であるから、再利用資産開発時にそれを検証することは不可能である。したがってテストなどの検証で提案されている CRS のような戦略

¹⁷² 朝倉功太, 他: 想定モデリングに基づくソフトウェアプロダクトラインのコア資産検証手法, 情報処理学会 組込みシステムシンポジウム 2009 (ESS2009).

が、モデル検査においても有効になる。すなわち、モデル検査に必要な検証資産を再利用可能な形で用意し、製品の検証時にはその検証資産を再利用することで、検証を効果的、効率的に行おうというものである。

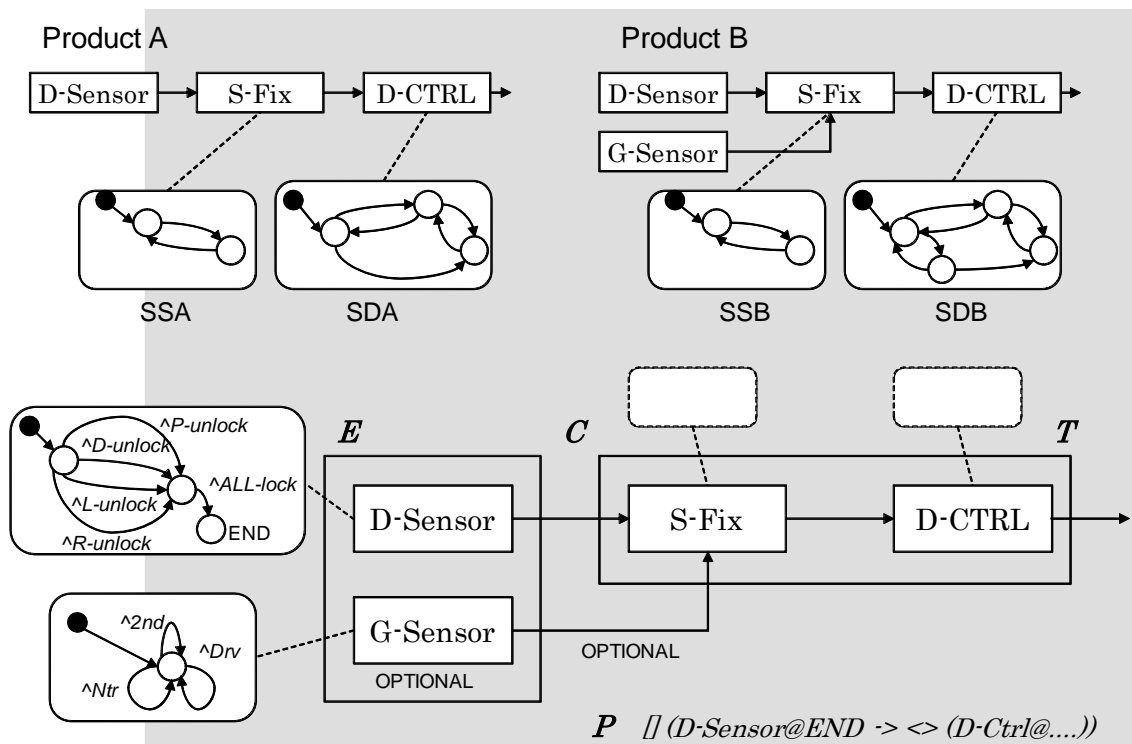


図 14-7: 再利用可能な検証資産の例¹⁷³

図 14-7 は Kishi らの提案する再利用可能な検証資産の概略を示すものである。上部には二つの異なる製品の構造が示されている。これらは同じ再利用資産群を利用しているが、使い方や構成が異なる。こうした製品の多様性を踏まえ、それらの再利用資産の使い方や構成を包括的に含んだ検証モデルをあらかじめ用意しておく。また性質記述も製品ごとに変更される部分は変更可能な記述として用意しておく。図の下部がそれらに対応する。製品の検証においては、こうした再利用可能な検証モデルや性質記述から、その製品に適した検証モデルや性質記述を導出し、モデル検査を行う。

また性質記述に関してはよく利用する性質記述をパターン化するという提案もなされている。例えば Dwyer らは、応答(response)や不在(absence)などといったカテゴライズを行い、例えば S が P に応答するということが常に成り立つなら、CTL では " $AG(P \rightarrow AF(S))$ "、LTL では " $\square(P \rightarrow \diamond S)$ " と記述するといったように、よく使われる時相論理式のパターンを整理している¹⁷⁴。こうした提案では、応答性など汎用性の高い性質のパターンを集めているが、特定のプロジェクトにおいてよく使われる性質記述を再利用できるようにパターン化しておくとう用とえられる。

14.6.2. 考慮点

検証資産を再利用することは、製品の検証において有用性があると考えられる。モデル検査は検証モデルや性質の定義に特有のスキルが必要であるし、注意深く定義しないとささいな間違

¹⁷³ Kishi, T. 他: Formal Verification and Software Product Lines, Communications of the ACM, Vol. 49, No12, 2006.

¹⁷⁴ Dwyer, M.B. 他: Patterns in Property Specifications for Finite-State Verification, Proc. of International Conference on Software Engineering, 1999.

いによって、本来の検証目的が達成できない危険性もある。したがって、実績のある検証モデルや性質を再利用することは検証の信頼性を上げる上でも効果があると期待される。また、テストやレビューは、そのプロジェクトで作るべき成果物そのものが対象となるが、モデル検査技術は多くの場合、そうした成果物とは別に、対象を抽象化したモデルを構築するといったコストが無視できない。したがって一旦構築した検証モデルや性質を繰り返し利用できるとすれば、複数のプロダクトに利用できるという意味で単位あたりのコストを下げるのが期待される。

もちろん、検証資産の再利用はソフトウェアの再利用と同様の困難さが伴う。そもそもプロダクトの開発ごとにモデル検査技術によって繰り返し検証すべき重要な性質とは何かという点を明確にすることが本質である。既存の研究では、本質的な要求モデルや、アーキテクチャ設計上の性質に焦点をあてた研究が多い。また、検証モデルは検証したい性質に強く依存しているので、検証性質が異なれば検証モデルそのものを大きく変更しなければならない事もある。したがってソフトウェアそのもの以上に、どの部分をどのように検証するかということを明確にすることが重要となる。特にモデル検査技術の場合、内部状態に応じた性質記述をするので、性質記述に使われる状態が、その検証モデルにおいて把握しやすいようにモデルが作られていることが重要である。こうした点は、検証資産を再利用するという局面に限ったことではないが、再利用においては一層注意を払うべき点と考えられる。

なお検証性質については、いろいろな図式の方法が提案されている。パターン化された性質をこうした図式表現で表すことがそのプロジェクトメンバにとって理解性を向上すると判断される場合は、そうした表現の活用も有効と考えられる。

14.7. まとめ

以上、ソフトウェア再利用とフォーマルメソッドの関わりについて、モデル検査技術による再利用資産の検証、プロダクトの検証という二つの局面から、いくつかの提案を紹介した。冒頭で述べたように、再利用におけるフォーマルメソッドの活用について、広く認められている提案があるというわけではない。ここでは既存の提案の中からいくつか参考になると判断されるものを紹介した。

なおモデル検査技術による検証においては、状態爆発の問題を上手に回避する必要がある。上述したように再利用資産の検証で様々な利用方法に照らして性質を検証したり、プロダクトの検証においてモデルにプロダクト毎の切り替えを含むような仕掛けをいれたりすると、単一プロダクトの検証と比較して状態数が増える可能性が大きくなる。モデルを「上手に」作ることが本質ではあるが、それは一定のスキルが必要であり、工学的にはいくつかの割り切りが必要となると考えられる。

表 14-3: 検証量を減らすための工学的な手法例¹⁷⁵

設計の記述レベル (アーキテクチャへの注目)	設計の抽象度が上がると表現される情報量が減少するために検証量が減る。但しモデルの解像度は下がるので、検証性質は粗くなる。
特性の機能に限定 (範囲の限定)	検証対象を限定化することで検証量が減る。但しその限定した範囲がどういうもので、その範囲を検証するというのがどういう意味を持つのかについて十分な検討が必要。
基本的な処理に注目 (パターン化)	パターン化して複数の処理を代表する処理に抽象化することで検証量が減る。但し、パターンで成り立つ性質が、そのパターンに帰属する具体的な処理で成り立つかどうかということに関しては、十分な検討が必要。
シーケンス長を制限 (検証の深さ)	探索の深さを限定することで検証の量が下がる。但し、より長いシーケンスにおける性質は分からないので、検証の質は下が

¹⁷⁵ 岸知二, 他: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, SEC Journal 12号, 2007.

	る。
--	----

表 14-3 は、状態量を減らすために考えられるソフトウェア工学視点からの方策例である。特段目新しい手法でも、再利用に特化した手法でもないが、こうした工学的な観点からの状態量削減を行うことも有効であり状態数が多くなりがちな再利用に関わる検証を行う際には参考になると考えられる。

15. ソフトウェアの品質に関する計測の重要性

本章では、ソフトウェアの品質に関する計測法について整理する。フォーマルメソッドの適用前後で、ソフトウェアの品質がどのように向上したか、比較評価するための参考となる。

15.1. 計測の重要性

ソフトウェア開発に於ける計測と分析の主たる目的は、プロジェクトが目指している方向に向かっている事を監視し、問題があれば是正し、目標としている到達地点へと導く事である。

一般的なソフトウェア開発プロジェクトが目指すべき方向とは高品質なソフトウェアを短期間に低コストで開発する事である場合が多く、これらの目標を達成するためには、常にプロジェクトの動向を監視し、問題の前兆を早期に察知し是正する事が不可欠となる。

ある程度ソフトウェア開発を経験している企業であれば、ソフトウェア開発に関する何らかの数値は計測しているケースは多い。しかしながら、集めたデータをどのように使っているか？効果的な使い方ができているか？となると集めているだけで効果的な使い方ができていない場合も多いのではないだろうか。

そもそも計測とは(ソフトウェア開発に限らず)、本来あるべき姿や向かうべき先とそこにたどり着くまでの進捗を把握するために必要な情報が明確になった上で、具体的に何を計測すべきかが決められるべきである。このように目的を明確にした上で、計測を行うフレームワークとしてはメーランド大学の Basili 教授らによって提唱された GQM(Goal/Question/Metric)パラダイムが有名である。

GQM パラダイムをモデルとして計測を行う例を以下に示す。

Goal	• dd 日までに実装を完了したい
Question	• 全体の実装量の内、どの程度実装完了しているのか？ • このままで間に合うのか？
Metric	• 残実装量 • 現在日付 • 1 日当たりの生産性 • 担当者の負荷状況

「dd 日までに実装を完了したい」という Goal に向けて、途中で進捗を把握するためには、「全体の実装量の内、どの程度実装完了しているのか？」「このままで間に合うのか？」といった事を確認する必要がある。これらの Question に回答するためには、「残実装量が nn キロステップで 1 日当たり n キロステップ実装しているので、あと何日で実装完了できる見込み」といった分析が必要になる。また、「このままだと間に合わないため、担当者を増やす必要がある」「担当者の負荷が集中しているため、他の作業を別の担当者に割り当てる」などといった是正処置が必要な場合もあるだろう。これらの分析や是正処置を行うためには、「残実装量」「現在日付」「1 日当たりの生産性」「担当者の負荷状況」といったメトリクスの必要性が導出される。

仮に、「残実装量」が計測されていないとすると、いくら「現在日付」や「1 日当たりの生産性」を計測していても Goal に向かっていることの分析は困難になってしまう。また、「担当者のコミット数」が計測されていたとしても「dd 日までに実装を完了したい」という Goal に対しては役に立たないデータであり、計測の手間が無駄になってしまう。

無駄なく必要最低限の計測を行う為には、的確な「Question」の設定がポイントとなる。上記の例で、Question が「全体の実装量の内、どの程度実装完了しているのか？」しか設定されていなければメトリクスとしては「残実装量」しか計測されていないだろう。この場合、「dd 日までに実装を完了したい」という Goal を達成するための分析や是正処置が行えなくなってしまう。

計測における「Question」を設定する際、Goal を満たすために必要な項目として Question を導き出すだけでなく、どのような事が起きると Goal が満たせなくなるのか？にも着目して Question を導き出すと良い。

15.2. 計測データの信頼性

ソフトウェア開発を計測し分析する上で、計測データの信頼性が非常に重要である。例えば「現在日付」として計測された値が 3 日遅れていればその後の分析や是正処置は誤ったものになってしまい、折角行った計測と分析全体が無意味なものになってしまう。計測データの信頼性確保に当たっては、以下のポイントが重要となる。

- ・ 計測方法(計測対象、計測タイミング、計測手順、分類など)
- ・ 計測期間(計測の開始時期と終了時期)

ソフトウェア開発プロジェクトでデータを収集する場合、各担当者などの複数人がデータを収集する機会が多い。このようにして集められたデータを同じデータとして分析を行うのであれば、計測を行う担当者間で計測対象や計測するタイミング、計測の方法などが統一されている必要がある。例えば、ソースコードの量を計る場合、コメントを含めるか否かやヘッダファイルを含めるか否かの統一がされていない状況で各担当者が集めたソースコードの量を分析しても正確な結果を得る事はできないし、レビューの効果を分析するためにレビューの指摘件数を計測した場合、インスペクション形式で実施したレビューのデータとパスマラウンド方式で実施したレビューのデータを同一に扱うのにも無理がある。

また、テスト工程で発見した不具合件数を計測する場合、テスト工程の開始時期や終了時期がいつなのか？といった計測の開始時期や終了時期についても明確になっている必要がある。

効果的な計測を行う為には、計測対象のメトリクスに対して「誰が、いつ、何を、どのように計測して、どこに格納するのか」といった計測に関するルールを厳格に定義し、計測に携わる全てのメンバー間で共有する事が必須となる。

15.3. 計測情報と品質の関係について

ソフトウェア開発に於ける品質とは、大きく以下の 2 種類が考えられる。

- ・ 要件定義／設計／実装工程における品質
- ・ テスト工程における品質

前者は主にレビューによる品質確保を行い、後者は主にテストによって品質が確保される。だが、ここで「本当に十分な品質が確保されたのか？」という疑問が湧いてくる。この疑問に答える為には、レビューとテストのデータを計測し、定量的／定性的な分析を行う必要がある。

また、多くのソフトウェア開発現場では軽視されがちだが、テスト項目を設定する為の活動であるテスト設計も、前者と後者に関わる。要件定義や設計に曖昧性や欠陥があり品質が悪い場合、テスト設計の項目作成時に発見されることがよくある。一方、テスト設計の品質が悪く、テスト項目が必要以上に多い場合や、テスト項目が荒く十分に確認しきれない場合、テスト工程での進捗の滞りや、欠陥摘出漏れを発生させることとなる。そのため、「テスト設計における品質」も、定量的／定性的な分析を行う必要がある。

15.3.1. レビューによる品質確保の計測と分析

レビューによる品質確保が十分に行われたか？を分析するためには、個々のレビューが一定以上のパフォーマンスで問題を十分に抽出できたのか？といった定量的な分析と十分に問題が抽出できなかったレビューは特別な理由があるのか？その理由は許容可能な理由なのか？といった定性的な分析を実施する事が多い。

レビュー品質の計測における典型的な例は以下がある。

Goal	•レビューによって十分な品質確保を行う事
Question	•レビューによって不具合をどの程度抽出しているか？ •レビューに掛けた時間は適当か？
Metric	•レビュー対象 •レビュー時間(事前の準備時間も含む) •レビュー参加人数 •指摘数 •レビュー対象規模 •レビューの種類

レビュー品質の分析を行う場合、「レビュー速度」や「レビュー指摘密度」といった評価尺度を用いる事が多い。レビュー速度とは $\text{レビュー規模} / \text{レビュー時間}$ で求められ、レビューの密度が適切かを分析する際に用いられる。組織の閾値と比べてレビュー速度が速い場合は、欠陥抽出が不十分である可能性があり、逆に遅い場合はレビュー効率が悪かった事が予想される。一方レビュー指摘密度とは、 $\text{指摘件数} / \text{レビュー規模}$ で求められ、レビューでの指摘件数が適切かを分析する際に用いられる。組織の閾値と比べて密度が高い場合は、前工程での品質確保が不十分であったなど、品質が悪い事が予想され、逆に密度が低い場合は、残存してしまっている欠陥がある事が予想される。このように定量的に組織の閾値と比較して標準から逸脱しているレビューを探しだし、それらに対して定性的な分析を行い、必要に応じて再レビューなどの是正処置を行う事が重要である。

組織の閾値は過去の実績から定義されることが理想的であるが、過去実績のデータが無い場合は直近のデータなどから設定する。閾値は定期的に見直しを行い、徐々に精度の高いものにしていく事が重要である。また、レビューのパフォーマンスはインスペクション／パスアラウンドなどのレビュー方法や初回レビュー／再レビューなどによって変わってくる為、閾値を設定する際にはこれらの要素も勘案すると良い。

15.3.2. テストによる品質確保の計測と分析

テストによる品質確保が十分に行われたか？を分析するための手法はゾーン分析、トレンド分析、信頼度成長曲線による分析など多岐に渡るがここでは信頼度成長曲線を活用した計測／分析の例を示す。

Goal	•テストによって欠陥を抽出し、品質を保証する事
Question	•テスト完了予定までに不具合が収束するか？
Metric	•テスト件数 •消化件数 •欠陥抽出数

テストで品質を確保するためには、そもそもテスト設計自体が妥当であるか？という検証が必要と

なるが、これについては「テスト設計自体の妥当性の検証の重要性」にて後述する。

信頼度成長曲線による分析では、以下の 4 種類の線を用いて分析を行う。(図 15-1 信頼度成長曲線 参照)

- ・ テストケースの消化予定線
- ・ テストケースの消化実績線
- ・ 欠陥摘出累積予定線
- ・ 欠陥摘出累積実績線

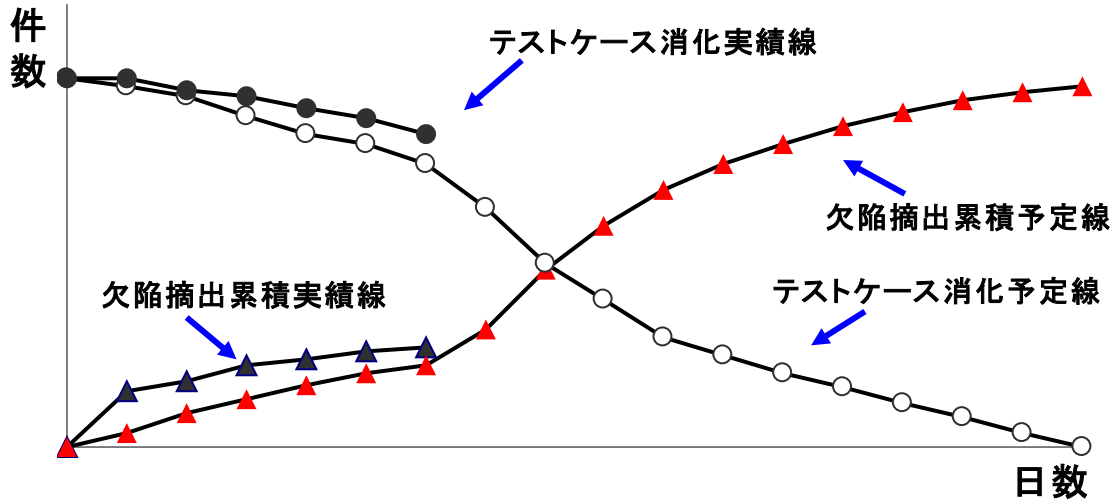


図 15-1: 信頼度成長曲線

本分析手法では、テストの消化状況と欠陥発生状況のバランスを見ることによって、テスト進捗だけでなく品質の良し悪しが判断できる。テストの終盤段階では、欠陥摘出累積が収束傾向(グラフの線が水平になる)になると、品質が安定してきたと言える。信頼度成長曲線では、図 15-2 に示す傾向にある場合にテスト自体に発生している問題を早急に察知可能である。

#	組み合わせ		信頼度成長曲線	考えられる問題
	消化実績	摘出実績		
1	正常	正常		特になし
2	消化 停滞	欠陥 多発		前工程における欠陥摘出不足 仕様変更、途中追加による欠陥の 作りこみ
3	消化 停滞	摘出 不足		デバッグ／テスト環境の不備 テスト要員不足
4	急激 消化	摘出 不足		テスト項目の質が低い デバッグ実施優先度が単純機能 に偏っている

図 15-2: 信頼度成長曲線の傾きとテストに発生している問題の例

信頼度成長曲線による計測／分析は結合テスト、システムテストなどのテストレベル毎に行う。テスト全体の有効性や品質の安定度合いは各テストレベルで摘出した欠陥数の分布を見ることで評価する。図 15-3 は、各テストレベルで摘出した欠陥数と、そこから導き出される品質評価の例である。

工程別摘出欠陥数	品質の評価
<p>抽出欠陥数</p> <p>単体 組合せ システム テスト工程</p>	<p>①抽出欠陥数が単調減少(指数型)</p> <p>— 品質は安定傾向</p>
<p>抽出欠陥数</p> <p>単体 組合せ システム テスト工程</p>	<p>②抽出欠陥数が単調非減少(一様)</p> <p>— 品質は非常に悪いことが多い</p> <p>— 前工程で摘出すべき結果を次工程に持ち越している。</p>
<p>抽出欠陥数</p> <p>単体 組合せ システム テスト工程</p> <p>増大 減少</p>	<p>③抽出欠陥数が増大から減少に</p> <p>— 組合せテスト工程において結果を大部分摘出</p> <p>— ただし、各工程の欠陥数の目標に占める割合の再評価が必要。</p>

図 15-3: 欠陥の抽出工程分布と品質評価

これらの分析を行い、テストにおける問題が発見された場合は、テスト項目の追加や前工程に戻

って品質を再確保するなどの是正措置を早急に行う事が重要となる。

15.3.3. テスト設計の品質の検証

一般的にテストとは、ほぼ無限にあるテスト項目に対して限られた時間とコストで臨む事になる。従って、いかに効率よく欠陥を抽出できるかが肝であり、より少ないテスト項目でより多くの欠陥を抽出するために行うのがテスト設計である。通常テスト設計では何に対してどのようにテストを行い、どのような結果を期待するのか？を設定する。つまり、テスト設計の品質がテストの成否を左右する。

また、テストとは設計された通りにソフトウェアが実装されていること、または顧客が要求したとおりにシステムが動作する事を確認する行為であり、テスト設計とはこれらを確認するためのテスト項目を設定する為の活動であるため、設計内容を要求仕様やシステム要件の視点で見直す事にもなる。この為、テスト設計を行うと仕様や設計の欠陥や曖昧である部分が発見されることが良くある。

以上よりテスト設計工程では、テスト設計そのものの品質と、要件定義／設計の品質の 2 つの測定／分析を行う。

テスト設計の計測における例を以下に示す。

Goal	<ul style="list-style-type: none">・より少ない項目でより多くの欠陥を抽出するテスト項目を作成する事・要件定義／設計の品質を検証する事
Question	<ul style="list-style-type: none">・テスト項目は妥当か？・要件定義／設計の不具合をどの程度抽出しているか？
Metric	<ul style="list-style-type: none">・要件定義／設計の仕様書の規模・テスト項目数・要件定義／設計の欠陥／曖昧性抽出数

仕様書の規模に対してテスト項目数が少ない場合、テスト設計が荒く、テスト工程で十分に不良を抽出しきれない事が想定される。一方、仕様書の規模に対してテスト項目数が多い場合、必要以上のテスト項目が作成され、テスト工程で必要以上の時間を浪費してしまうことが想定される。

欠陥／曖昧性抽出数が多い場合、要件定義／設計の品質が悪いことが想定される。このように、テスト設計工程で要件定義／設計の品質が悪いことがわかると、前工程への手戻りが発生する事になってしまう。この点に注目し、設計が終了したタイミングでテスト設計を行うW字モデル開発(図15-4)という考えがある。

W字モデルでは、要件定義／設計工程が終了した段階で、その工程に対応するテスト設計を行う。このため、テスト設計工程で発見された要件定義／設計工程での欠陥や曖昧性を、実装終了後にテスト設計を行う場合に比べて早い段階で抽出可能となる。したがって、W字モデルを適用することで、前工程への手戻りにかかる工数を削減できる。

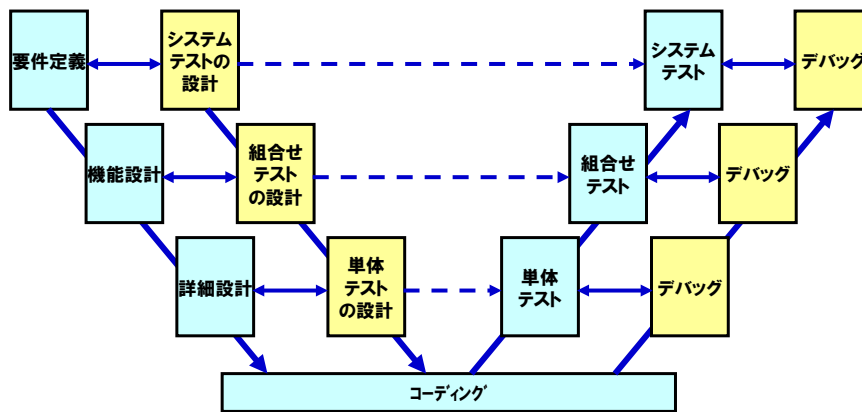


図 15-4:W 字モデル開発

15.4. プロセス管理での利用上の効果

計測や分析は品質に寄与するのみではなく、ソフトウェア開発プロセスに従った開発を推進する上でも非常に重要な役割を担う。組織で定められたプロセスがある場合、実際に行われている開発作業の計測／監視を行う事で、どの程度プロセスに沿った開発が行われているかを評価する事ができる。ここで計測対象となるのは、先に述べた数値的な部分のみではなく開発の進め方そのものも計測対象とする必要がある。このようにプロセスがどれだけ遵守されているかを監視する考え方は「プロセス QA」と呼ばれる。プロセス QA については、CMMI のプロセスエリア「PPQA」を参照の事。

プロセス QA の目的は、プロセスの不遵守を発見し是正させる事のみではなく、プロセス自体の問題点を発見する上でも役立つ。「不遵守＝守らない側が悪い」とするのではなく、なぜ不遵守なのかを分析した上で必要に応じてプロセス自体を改善する事も重要である。

15.5. 簡便な計測方法

そもそも計測とはソフトウェア開発を成功に導くために行う活動であって、計測活動そのものが目的となってしまうように注意する必要がある。過度な計測やあまりにも高度な分析はそれ自体に多大なコストがかかるため、計測そのものがソフトウェア開発の足枷となってしまうリスクを孕んでいる。計測活動はあくまでもソフトウェア開発のサポートに位置する活動である事を認識し、計測や分析に掛かるコストを算出した上で活動の範囲が決定されるべきである。

効果的に計測活動を行う為のポイントとしては以下がある。

- ・ 計測対象の絞込み
- ・ データ収集の自動化
- ・ 分析基準の数値化

15.5.1. 計測対象の絞込み

事業目標や業界動向などからコアコンピタンスとなる部分を定め、計測を行う対象を絞り込む。また、既存のメトリクスを複数掛け合わせる事によって関数的に導出可能なメトリクスは新たなメトリクスとして定義しないなどの工夫も重要である。

15.5.2. データ収集の自動化

ツールの活用によって、データ収集の自動化を行うようにする。例えばソースコードの量であれば、構成管理ツールにコミットしたタイミングで自動的にステップカウンタに掛ける事で自動的に収

集したり、作業工数であれば出退勤のシステムから情報を自動的に取得する事もできるかもしれない。データ収集の自動化は、計測コストの削減のみならず人間系の計測による作業ミス発生を回避する効果もある。

15.5.3. 分析基準の数値化

標準からの逸脱を判断するための基準を数値化し、誰でも同じ判断が瞬時に行えるようにする。

16. モデル検査ツール(SPIN)の使い方等のヒント

モデル検査の目的に応じて、3つのステップ(1)モデル検査 C コード生成、(2)コンパイル、(3)モデル検査実行、において用いられるツール・オプションの依存関係を整理し、オプションの組み合わせに関する利用パターンを整理する。

対象読者	開発技術者
目的	モデル検査ツール SPIN の使い方に関して、分かりにくいオプションの組合せについて基本的な使い方を整理する。
想定知識	SPIN の基礎知識
得られる知見等	<ul style="list-style-type: none">● モデル検査の基本的な実施順序● SPIN によるツール利用の3ステップ(1)モデル検査系 C コード生成、(2)コンパイル、(3)モデル検査実行、において用いられるツール・オプションの依存関係と組み合わせの制約● Windows 版 SPIN のインストール注意点

16.1. SPIN モデル検査の基本的な実施手順例

SPIN によるモデル検査の実施手順は、目的やスキルによって様々な流れが考えられるが、一般的な場合で効率的にモデル検査を行うためには、以下のような実施手順を参考にするとよい。

(1) モデル自体の基本的な検査(到達性解析)

モデル(Promela コード)が、備えるべき基本的な条件で、ツールにより自動チェック可能な到達性解析を最初に行うとよい。モデルが LTL 式により定義される検証性質を満たすか検査したり、シミュレーション実行でモデルの振舞いを確認する前に、自動チェック可能な到達性解析を先に行うとよい。

到達性解析により、対象システムのモデル(Promela コード)のデッドロック、ライブロックなどの好ましくない記述の検出を行うことができる。また、Promela コードの特定の位置で、特定の条件を満たすかどうか確認するための `assert` 文の検査を行う。

(2) 進行性解析(ループ解析)

Promelaコードのある並行プロセスの特定のコード上の位置(ステートメント)を無限に繰り返し訪れる場合、他の並行プロセスの進行性を保証できなくなる。これらは進行性解析により検出することができる。また、不具合とは見なさない想定されるループなどが存在する場合、それらはエラーと見なされないように、ループを含むステートメント上に `progress` ラベルを指定する方法がある。これらの使い方に関しては注意が必要で、文献¹⁷⁶の 3.1.3 節に具体的に書かれているため参考にするとよい。

(3) LTL 検証性質のモデル検査

システムが提供すべき機能(ライブネス)とシステムが守るべき条件(安全性)を LTL 式で記述して、モデル検査を行う。

上記の検証の補助的な手段として、反例(不具合)が発見された場合のガイド付きシミュレーションやランダムシミュレーションの実行により挙動を確認する。

16.2. SPIN モデル検査のオプション組合せヒント

SPIN モデル検査では、3ステップ (1)モデル検査系 C コード生成、(2)コンパイル、(3)モデル検査

¹⁷⁶ 中島震: SPIN モデル検査 -検証モデリング技法, 近代科学社, 2008 .

査実行、により実行される。前節で示した検査の種類ごとに、これらの3ステップで用いられるオプションには依存関係があり、前のステップで指定したオプションはその後のステップで指定するオプションに制約を与える。それらの制約やオプションの組合せについて以下にまとめる。

16.2.1. モデル自体の基本検査（到達性解析）の場合

モデル検査実行3ステップにおけるオプションの組合せは以下の通りである。

(1)モデル検査系 C コード生成

```
% spin -a <モデルファイル>
```

(2)コンパイル

```
% gcc pan.c
```

(3)モデル検査実行

```
% ./pan
```

16.2.2. 進行性解析（ループ解析）の場合

Progress ラベルを設定した後、モデル検査実行3ステップにおけるオプションの組合せは以下の通りである。

(1)モデル検査系 C コード生成

```
% spin -a <モデルファイル>
```

(2)コンパイル

```
% gcc -DNP pan.c
```

(3)モデル検査実行

```
% ./a.out -l
```

16.2.3. LTL 検証性質に関するモデル検査の場合

安全性の検査、ライブネスの検査によってオプションの組合せがことなることに注意が必要であるモデル検査実行3ステップにおけるオプションの組合せは以下の通りである。

16.2.3.1. 安全性検査の場合

(1)モデル検査系 C コード生成

```
% spin -a -f "LTL 式" <モデルファイル>
```

(2)コンパイル

```
% gcc -DSAFETY pan.c
```

(3)モデル検査実行

```
% ./pan
```

16.2.3.2. ライブネス検査の場合

spin コマンドの -DNP オプションなしで実行可能である。

(1)モデル検査系 C コード生成

```
% spin -a -f "LTL 式" <モデルファイル>
```

(2)コンパイル

```
% gcc pan.c
```

(3)モデル検査実行

```
% pan -a -f
```

Fairness を仮定したモデル検査の場合、以下のようにする。

(1)モデル検査系 C コード生成

```
% spin -a -f "LTL 式" <モデルファイル>
```

(2)コンパイル

```
% gcc -DNFAIR=3 pan.c
```

(3)モデル検査実行

```
%.pan -a -f
```

LTL 式をファイルで入力する場合以下のようにする。

(1)モデル検査系 C コード生成

```
% spin -a -F <LTL 式ファイル> <モデルファイル>
```

(2)コンパイル

```
% gcc pan.c
```

(3)モデル検査実行

```
% pan -a -f
```

16.2.3.3. Never claim生成と利用

LTL 式をコマンドラインオプションとして与え、Never Claim を生成するためには以下のようなオプションを使う。

(1) Never Claim の生成

```
% spin -a -f "<LTL 式>" > <never claim ファイル>
```

(2) Never Claim をコマンドラインとしてモデル検査系 C コード生成

```
% spin -a -N <never claim ファイル> <モデルファイル>
```

以下は、16.2.3 節と同様である。

16.2.4. SPIN シミュレーション実行と反例解析

Promela コードを入力として、ランダムシミュレーションを実行する場合以下のようにする。

```
% spin <モデルファイル>
```

また、反例が見つかった場合に、その反例に至るガイド付きシミュレーションを実行するのは以下の通りである。

```
% spin -t -g <モデルファイル>
```

-g は、大域変数の表示

また、反例に至る最短パスを探索するためには以下のように実行する。

```
% pan -i
```

16.2.5. モデル検査の計算効率化のオプション

内部で生成するハッシュ表サイズ(デフォルト $w=18, 2^{18}$)を設定する場合以下のようにする。

```
% pan -w20
```

ハッシュ表の圧縮を強化する場合、以下のように実行する。

```
% gcc -DCOLLAPSE pan.c
```

ただし、この場合、メモリ使用量は減らせるが、圧縮の計算時間が増える。

状態遷移システムの状態を表す状態ベクトルをハッシュ表の代わりに minimal automaton で保持するためには、以下のように実行する。

```
% gcc -DMA=10 pan.c
```

この場合、メモリ使用量は劇的に減らせるが、計算時間が増える。

物理メモリがあふれスラッシングが発生する場合、それを回避するための方法として、モデル検査に利用するメモリのサイズ上限を設定し、物理メモリを超えないようにする。

```
% gcc -DMEMLIM=n pan.c
```

Promela コードに対してスラッシングを実施する場合、以下のように実行する。

```
spin -A <モデルファイル>
```

16.3. Windows 版インストールの注意点

本節は、モデル検査ツール SPIN (Windows 版)のインストール手順を示すものである。必要に応じて参照すれば良い。

16.3.1. ツールのダウンロードおよびインストール

SPIN のインストーラーは、<http://spinroot.com/spin/Bin/index.html>の「Windows PC Spin executable」と記載されている箇所からダウンロードができる。2011 年 3 月時点での最新バージョンは 6.0.1 であり、spin601.exeをダウンロードできる。

名前を spin.exe に変更し、任意のフォルダに保存する。ここでは、「C:¥spin」フォルダを作成し、そこに保存する。

gccを利用可能にするため、CygwinまたはMinGW等をインストールする必要がある。ここではMinGWのインストール方法を述べる。MinGWは、<http://sourceforge.net/projects/mingw/>の「Download」と記載されている箇所からダウンロードできる。ダウンロードしたファイルを実行するとインストーラが立ち上がる。基本的には全てNextを選択して良い。インストール先は任意の場所で良いが、ここではデフォルトである「C:¥MinGW」とする。

また、必須ではないがGUIとしてispin、jSpin、xpinが用意されている。ここではispinの利用方法を示す。ispinは、<http://spinroot.com/spin/Src/index.html>からダウンロードすることができる。特別なインストール作業は必要ない。さらに、Windows環境でtclを実行可能にするため、ActiveTcl等をインストールする必要がある。ActiveTclは、<http://www.activestate.com/activetcl>からダウンロードすることができる。ActiveTclはデフォルトの設定でインストールすれば良い。

16.3.2. 環境設定

マイコンピュータを右クリックして、プロパティを選択。詳細設定タブを選択し、環境変数ボタンをクリックする。次にユーザ環境変数内にある **PATH** を選択し、編集をクリックする。変数値の欄に「**C:¥spin;C:¥MinGW¥bin**」を追記し、**OK** をクリックして全て閉じる。**SPIN** や **MinGW** の保存先を異なる場所に行っている場合はそれに合わせる。

16.3.3. 実行

スタートメニューから「ファイル名を指定して実行」を選択し、名前に「**cmd**」と入力して **OK** をクリックするとコマンドプロンプトが立ち上がる。コマンドプロンプト上で、「**spin**」と入力したときに、

```
Spin Version 6.0.1 -- 16 December 2010
spin: error no filename specified
```

のような表示が出力されれば **SPIN** のインストールは成功である。また、コマンドプロンプト上で、「**gcc**」と入力したときに、

```
gcc: no input files
```

のような表示が出力されれば **MinGW** のインストールは成功である。

また、**ispin.tcl** をダブルクリックし、**ispin** のウィンドウが立ち上がれば、**ActiveTcl** のインストールは成功である。

17. 関連文献、リンク集

17.1. 形式手法に関する解説文献のガイド

本節では、対象とした形式手法について、個々の手法を詳しく学習する前に、各手法の概要、特徴、応用事例等を知るために参考にできる文献を紹介する。

17.1.1. 形式手法の解説文献の概要

ここで対象とする解説文献は、実応用レベルの形式手法について、あまり前提知識を必要とせず、形式手法の概要および特徴を理解できる文献を対象とする^{177,178}。各文献について、主に、以下の項目について情報をまとめる。

- 対象手法
- 記述内容概要
- 応用事例情報の有無
- 想定読者、記述レベル
- 文献の入手元情報

17.1.2. 形式手法の解説文献の個別紹介

前節に示した形式手法の解説文献を中心に、個々の文献についてまとめる。以下の節では、モデル検査、定理証明、総合(モデル検査および定理証明を含む文献)に分類し、順に概要をまとめる。

17.1.2.1. モデル検査

[1] モデル検査法のソフトウェアデザイン検証への応用

文献情報	モデル検査法のソフトウェアデザイン検証への応用, 中島 震, コンピュータ ソフトウェア, Vol. 23 (2006), No. 2, 2_72-2_86
入手先	http://www.jstage.jst.go.jp/article/jssst/23/2/2_72/pdf/char/ja/
対象手法	SPIN を中心とする。SMV, NuSMV など他のモデル検査手法については応用動向についてまとめている。
想定読者	モデル検査など形式手法の前提知識の無い技術者
概要:	モデル検査法およびそのツールなどに関して、技術の適用方法を伝えることに主眼をおいて分かりやすく解説している。また、ソフトウェア開発のいろいろな局面での利用例を紹介することで実用性について示している。簡単な例を用いてソフトウェアなどを状態遷移システムによるモデリングする方法と検証の基本原則について説明している。また、各種モデル検査ツールによる設計検証の応用事例を概観している。幅広くモデル検査ツールについて比較一覧をまとめている。また、モデル検査の書籍に関する概要紹介を行っている。 本文献により、始めて学習する人にとってもわかりやすくモデル検査の基本概念が理解でき、また、具体的な目的に応じてツールの選択を行う際の参考情報が得られる。

[2] ソフトウェア工学からみたモデル検査法, 中島 震

文献情報	中島震. ソフトウェア工学からみたモデル検査法. 第 22 回回路とシステム軽井沢シンポジウム, 2009.
入手先	
対象手法	SPIN 中心, SMV などを含む。

¹⁷⁷ 本ガイダンスの公的な性質から、有償ツールは対象外とする。

¹⁷⁸ http://www.jstage.jst.go.jp/browse/jssst/_vols/char/ja

想定読者	モデル検査など形式手法の専門知識の無い技術者
概要:	<p>モデル検査に関して、記号モデル検査、状態探索法などのアルゴリズムの基本原理を分かり易く説明している。原理の説明において、一部、前提知識を必要とするため、抽象的に説明することで、一般の人にある程度理解できるように書かれている。また、モデル検査におけるシステムのモデリングにおいて必要とされる、モデルの抽象化について、幅広く概観している。さらに、モデリングのアプローチとして、入力情報に応じた分類を行い、その特徴比較を行っている。</p> <p>具体的な実用事例は記載されていない。</p>

[3] ソフトウェア科学基礎(第 10 章 モデル検査ツール)

文献情報	ソフトウェア科学基礎(第 10 章 モデル検査ツール)、磯部 祥尚, 櫻庭 健年, 田口 研治, 田原 康之, 糸野 文洋, 田中 謙、近代科学社
入手先	ISBN-10: 4764903555 (近代科学社)
対象手法	SPIN, SMV, LTSA, UPPAAL
想定読者	モデル検査の基礎を少し理解している人
概要:	<p>モデル検査 SPIN, SMV, LTSA, UPPAAL を対象に、各手法の概要、記述言語の概要、と構文に関する簡単な説明、例題によるツールの簡単な使い方についてツールの画面等を示しながらまとめている。</p> <p>各手法の概要のところに、手法の用途や歴史について簡単に触れられている。</p> <p>本書のモデル検査に関する章の基礎的な部分を読むなどして、モデル検査の基礎を少し理解した人を対象とした記述となっている。</p>

[4] モデルに基づく開発方法論の全て(第 3 章 検証)

文献情報	組み込みソフトウェア 2007 モデルに基づく開発方法論の全て, 第 3 章 検証, 日経 BP 社
入手先	ISBN-10: 4822202585 (日経 BP 社)
対象手法	NuSMV, VDM, LTSA, Garakabu, SCADE について具体的に記載。Z, CafeOBJ, SPIN, ESC/Java, SLAM, BLAST, UPPAAL については概略に触れている。
想定読者	モデル検査など形式手法の前提知識の無い技術者
概要:	<p>最初の章で、形式手法の概観を行い、次章以降で、NuSMV, LTSA, Garakabu, SCADE 等のツールに関する具体的な特徴説明などを行っている。モデル検査を現場に導入する際の課題としてモデル検査の状態爆発の問題とそれを回避するための方法などについても説明している。</p> <p>概観の章で、実応用例が数件挙げられている。また、各ツールの章でも、実応用を事例として手法の特徴や機能の概略を説明している。</p>

17.1.2.2. 定理証明

[6] B メソッドと支援ツール 来間 啓伸

文献情報	来間 啓伸, B メソッドと支援ツール, コンピュータ ソフトウェア, Vol. 24 (2007), No. 2, 2_8-2_13
入手先	http://www.jstage.jst.go.jp/article/jssst/24/2/2_8/pdf-char/ja/
対象手法	B メソッド, B4free
想定読者	仕様検証について知識を持った人
概要:	

B メソッドと、支援ツールである B4free について解説している資料である。仕様検証について知識を持った人が抽象機械記法を用いた仕様記述を行い、コードを生成するまでの過程について解説されている。具体例を挙げ、そのシステムの仕様記述を例示することで具体的な記法や制約条件について記述している。また、実際の B メソッドを用いたシステムでの検証結果について参照しながらその長所についても説明されている。

本文献からは B メソッドを用いた仕様記述の概観が得られ、実際にツールを使って仕様検証をする際の参考情報が得られる。

[7] オブジェクト指向形式仕様記述言語 VDM++ 支援ツール VDMTools, 佐原 伸, 荒木 啓二郎

文献情報	佐原 伸, 荒木 啓二郎, オブジェクト指向形式仕様記述言語 VDM++ 支援ツール VDMTools, コンピュータ ソフトウェア, Vol. 24 (2007), No. 2, 2_14-2_20
入手先	http://www.jstage.jst.go.jp/article/jssst/24/2/2_14/pdf-char/ja/
対象手法	VDM, VDM++, VICE
想定読者	仕様記述を理解しており、VDM についての概要を把握している人
<p>概要:</p> <p>VDM とその拡張 VDM++, VICE によるモデル作成、仕様記述を支援するツールである VDMTools について解説が行われている。VDM ツールの機能概要や使い方、オブジェクト指向と同期並行処理記述を可能とする VDM++ の記述例が掲載されており、ツールを用いた仕様記述の概観が得られる。また、VDM++ を用いた成功例について紹介されており、適用例や適用方法の参考となる。VDM++ を利用したときの筆者の経験や注意点についての情報がある。</p> <p>本文献からは VDMTools を用いた具体例と適用方法についての情報、さらに VDM を用いた仕様記述についての参考情報が得られる。</p>	

[8] 仕様記述言語 Z と証明環境 Isabelle/HOL-Z, 来間 啓伸, Burkhart WOLFF, David BASIN, 中島 震,

文献情報	仕様記述言語 Z と証明環境 Isabelle/HOL-Z, 来間 啓伸, Burkhart WOLFF, David BASIN, 中島 震, コンピュータ ソフトウェア, Vol. 24 (2007), No. 2,2_21-2_26
入手先	http://www.inf.ethz.ch/personal/basin/pubs/z-env.pdf
対象手法	Z, Isabelle/HOL-Z
想定読者	Z 以外の仕様記述言語を用いたことがある人
<p>概要:</p> <p>仕様記述言語 Z の概要と証明環境 Isabelle/HOL-Z を用いた仕様記述と検証過程が解説されている。Z の概要の紹介では、Spivey による誕生日帳の例題を用いて、Z の基本的な使い方を解説している。具体的には誕生日システムを遷移システムとして定式化し、記述例した後にリファインメント、リファインメントの検証を行っている。</p> <p>また、Z で書かれた仕様のための証明環境である HOL-Z を用いた検証の概要についても記述されており、誕生日帳の検証の様子を示している。</p> <p>本文献からは Z の概要と証明環境 HOL-Z の使い方についての情報が得られる。Z の周辺情報についても整理されている。</p>	

[9] Alloy: 自動解析可能なモデル規範形式仕様言語, 中島 震, 鶴林 尚靖,

文献情報	仕様記述言語 Z と証明環境 Isabelle/HOL-Z, 来間 啓伸, Burkhart WOLFF, David BASIN, 中島 震, コンピュータ ソフトウェア, Vol. 24 (2007), No. 2,2_21-2_26
入手先	http://www.inf.ethz.ch/personal/basin/pubs/z-env.pdf
対象手法	Alloy
想定読者	Z 記法について概略的な知識を有し、Alloy の概観を理解したい人

概要:
仕様記述言語 Z の自動解析可能なサブセットである Alloy について、その概要が示されている。Alloy の歴史的経緯やその特徴について解説されており、Alloy についての文献についての紹介もある。具体的な応用例として、モデル規範形式仕様の標準的な例題である「誕生日帳」の Alloy 記述例が紹介され、関係論理の考え方、解析の方法が示されている。筆者による利用経験や、Alloy の周辺についての研究動向と関連事情、その位置づけについての記述がある。Z 記法についての知識を持った人を対象にしており、そのサブセットである Alloy の情報を得ることができる。

[10] CafeOBJ 入門(1) 形式手法と CafeOBJ 二木 厚吉, 緒方 和博, 中村 正樹, コンピュータ ソフトウェア Vol. 25 (2008), No. 2

文献情報	CafeOBJ を用いたシステムの振舞の仕様記述・検証, コンピュータ ソフトウェア, Vol. 24 (2007), No. 2, 21-2_26
入手先	http://www.jstage.jst.go.jp/browse/jssst/25/2/_contents/-char/ja/
対象手法	CafeOBJ
想定読者	プログラミング(関数プログラミングであればなおよい、ただし、高階関数は必要ない)の経験があり、数学的帰納法による証明に触れたことのある方
概要:	<p>本文献は、6編から構成される CafeOBJ(代数仕様言語・処理系)入門の第1編である。CafeOBJ 入門は、読み進めると同時に処理系と対話することで、CafeOBJ をシステムの振舞の仕様記述・検証へ応用できる力を養うことを目的としている。CafeOBJ のウェブサイト (http://www.idl.jaist.ac.jp/cafeobj/) から、処理系および CafeOBJ 入門で使われている例題(仕様書等)を入手可能である。第1編では、簡単な例題を用いて、システムの振舞のモデル化(状態機械の作成)、状態機械の CafeOBJ による記述方法、および、状態機械が望みの性質を有すことの検証方法について簡潔に説明してある。第1編を読むだけで、CafeOBJ を用いたシステムの振舞の仕様記述・検証の概略を把握できる。さらに、処理系と対話しながら、第6編までおとして読むことで、背景にある基礎技術から応用力まで身につけることができる。</p>

17.1.2.3. 総合

[5] 中島震, ソフトウェア工学の道具としての形式手法

文献情報	ソフトウェア工学の道具としての形式手法, 中島震, コンピュータ ソフトウェア, NII Technical Report ISSN 1346-5597
入手先	http://research.nii.ac.jp/TechReports/07-007J.pdf
対象手法	形式手法全般(VDM, Z 記法, B メソッド, OCL と Alloy 等)
想定読者	形式手法について興味を持つ人、前提知識不要
概要:	<p>形式手法全般について、その歴史と現在の動向について筆者の経験を元に記述されている文書。形式手法全般に関する知識と代表的な形式手法についての知識、実際の利用に関する筆者のアドバイスが掲載されている。</p> <p>ソフトウェア開発における、形式手法によって可能となる設計記述の正しさを証明する手段、概略を説明している。また、代表的な形式手法をその特徴に従って分類し、VDM, Z 記法, B メソッド, OCL と Alloy といった例について解説を行っている。形式手法の組織・個人への導入・習得について整理がされている。上記に挙げた形式手法に関する参考文献や、実際の利用に関する注意点などについても言及されており、形式手法について興味をもつ初心者向けの文書である。</p>

17.1.3. 学習者向け参考書

学習参考書として代表的なものを以下に示す。これらは網羅的ではないが、参考図書をカタログ

グ化する第一歩として、いくつかの図書を選んでみる。

17.1.3.1. モデル検査

文献 1	中島震, "SPIN モデル検査," 近代科学社, 2008.
概要	さまざまな例題を交えて SPIN の使い方やモデリングの考え方を分かりやすく説明している。

文献 2	M. Ben-Ari, "Principles of the SPIN model checker," 邦訳 オーム社, 2010 年出版予定
概要	プログラムの設計検証に関して具体的な SPIN のコードを使いながら分かりやすく説明している。

文献 3	萩谷昌巳監修, "SPIN による設計モデル検査", 近代科学社, 2008
概要	SPIN による設計モデル検査について分かりやすく体系的に整理している。

文献 4	米田友洋, 梶原誠司, 土屋達弘, "ディペンダブルシステム," 共立出版, 2005.
概要	モデル検査の理論を簡潔にまとめ、具体例として SPIN および CTL に基づくモデル検査として SMV を解説している。

17.1.3.2. その他

文献 5	荒木啓二郎, 張漢明, "プログラム仕様記述論," オーム社, 2002.
概要	プログラム検証の入門、プログラム検証に特化した論理、VDM、Z などについて分かりやすく説明している。

文献 6	荒木啓二郎, 張漢明, 萩野隆彦, 佐原伸, 染谷誠(訳), "ソフトウェア開発のモデル化技法," 岩波書店, 2003.
概要	モデリングの基本的な概念や考え方について詳しく説明している。

文献 7	佐原伸, "形式手法の技術講座," SRC, 2008.
概要	ソフトウェア開発現場を想定読者とし、VDM++ のためのツール VDMTools を中心として、形式手法の概要および適用の第一歩となる形式仕様記述言語について解説している。

文献 8	玉井哲雄, "ソフトウェア工学の基礎," 岩波書店, 2004.
概要	「モデル化」の基本的な考え方とその個別技術に重点をおき「ソフトウェア工学」全般を解説している。

文献 9	栗田 太郎, 荒木啓二郎, モデル規範型形式手法 VDM と仕様記述言語 VDM++ - 高信頼性システムの開発に向けて -, 日本信頼性学会誌「信頼性」2009 年 9 月
概要	VDM++ をベースとして、その言語仕様の概略およびモデリングの考え方について解説している。

17.2. フォーマルメソッドの研修や導入支援サービス

4.1 章、4.2 章で示したようにフォーマルメソッドの導入には、外部有識者の協力が非常に大きな推進力となる。また、フォーマルメソッドの技術習得コストを一定程度に抑え、実践を通じて必要な技術を効果的に習得する上でも、外部有識者との協力は有用である。ここでは、公的な組織を

対象に、フォーマルメソッドの研修や導入支援サービスとその参照先(Web サイトの URL)を以下にまとめる。

○組込み適塾 システムアーキテクト実践コース

<http://www.kansai-kumikomi.net/ptraining/kumikomi.html>

組込みソフト開発のプロジェクトにおいて技術リーダとして活躍できるシステムアーキテクトの育成を目標としたコースが開設されている。本コースのマネジメント&アドバンス科目の1つとしてモデル検査の講義が提供されている。また、「組込み適塾」の修了生を主な対象者とし、システムアーキテクトとして必要な知識の習熟度を高めるための講座として「実践演習編 実践的モデル検査」も開催されている。

○産業技術総合研究所 関西産学官連携センター 組込みシステム技術連携研究体 検証サービス

組込みシステム技術連携研究体 <http://cfv.jp/cvs/>

組込みシステム産業振興機構が提供している「さつき」による検証サービス

http://www.kansai-kumikomi.net/activities/development_support/satsuki.html

企業に対し、要求仕様や設計仕様、ソースコード、ヒアリング等を元に、モデル検査など数理的技法に基づく自動検査を産総研で実施する検証サービスを提供している。あわせて、連携検証施設「さつき」にあるクラスターシステムを大規模モデル検査等で利用できるサービスも提供している。利用形態として、組込みシステム産業振興機構が提供している「さつき」による検証サービスと産総研との共同研究が提供されている。

○国立情報学研究所 Grace センター トップエスイー

トップエスイー ホームページ <http://www.topse.jp/>

NPO 法人トップエスイー教育センター <http://topse.or.jp/>

次世代のソフトウェア産業を牽引するスーパーアーキテクトを養成する講座「トップエスイー」が開設されている。本教育プログラムでは、SPIN、SMV、UPPAAL、JavaPathFinder 等のモデル検査ツールの実践や VDM、B method などの形式仕様記述言語の実践などフォーマルメソッド実践教育の科目が充実している。トップエスイー教育センター経由で必要な科目のみを受講することも可能である。

以上のような公的な機関以外に、民間においても実践に即したフォーマルメソッド導入支援サービスを実施している企業はある。それらの具体的な情報については、本導入ガイダンスを作成した下記の組織から情報提供できる。

フォーマルメソッド導入支援サービスに関する情報の問合せ先:

株式会社三菱総合研究所

クラウドセキュリティグループ

TEL:03-6705-6047

FAX:03-5157-2148

E-mail: fm-inquiry@mri.co.jp

17.3. フォーマルメソッドの普及活動等

フォーマルメソッドの普及展開を目指した組織・コミュニティのレポートやセミナーも有用な情報源となる。以下に国内の主なものをまとめる。この他にもフォーマルメソッドの研究が盛んな大学との共同研究や技術指導を受けることも外部有識者の協力を得る手段の一つである。

○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

<http://sec.ipa.go.jp/>

高信頼性システムを構築する技術としてフォーマルメソッドに着目し、適用動向調査やフォーマルメソッドを実践できる人材の育成に関する施策検討等を実施している。フォーマルメソッドの応用動向に関する報告書として以下を公表している。

高信頼ソフトウェア構築技術に関する動向調査(2008年)

<http://sec.ipa.go.jp/reports/20080606.html>

フォーマルメソッド適用調査(2010年)

<http://sec.ipa.go.jp/reports/20100729.html>

○VDM 研究会

<http://www.vdmtools.jp/modules/tinyd1/index.php?id=2>

仕様記述言語とそれを支援するツール群を平易に利用可能なソフトウェアとして普及させることを目的としたコンソーシアムである。仕様記述法、仕様記述言語、それらを支援する開発・保守支援環境等、仕様記述を行なう上での VDM をベースとした基盤環境を構築し、VDM や仕様記述に関する社会的認識と技術水準の向上を図るための啓蒙・普及活動を行っている。証券システムへの VDM 適用事例の紹介(定量評価結果を含む)や汎用ライブラリやテスト支援ツール、例題集等を公開している。また、VDMTools を提供している CSK では VDM の研修や導入支援を行うサービスも提供している。

○モデル検査によるソフトウェア・テストの実践研究会

<http://www.modelcheck.jp/>

モデル検査によるソフトウェア・テストを実務導入・活用するための実践研究を行い、普及啓発活動を行っている。モデル検査によるソフトウェア・テストの適用事例・ノウハウ収集と体系化、モデル検査の実践ノウハウを取り入れた実務者向け教材の開発、モデル検査支援ソフトウェアの開発、大規模システム検証用 64bit 版モデル検査器の開発などの活動を行っている。モデル検査の導入を容易にするための支援ソフトウェアの公開や Web システムによるモデル検査サービス(オンラインサービス)を試行提供している。

○CSP コンソーシアム

<http://www.csp-consortium.org/action/action3.html>

CSP モデルに基づく技術者教育とアプリケーション開発を推進し、検証ツールに基づく開発手法を確立する事を目指したコンソーシアムである。CSP モデルの普及のために CSP 研究会を開催しており、その発表内容を公開している。

○形式手法の実践ポータル

<http://formal.mri.co.jp/>

経済産業省「新世代情報セキュリティ研究開発事業」において実施された成果を踏まえて、形式手法をソフトウェアシステム開発現場に普及させることを目的として、形式手法の導入方法、プロジェクト管理、適用技術、手法の比較、ケーススタディ、実践応用事例、ニュース情報の形式手法に関する情報やノウハウを幅広く公開するポータルサイトである。三菱総合研究所、国立情報学研究所によって運用されている。

○Dependable Software Forum (DSF)

ソフトウェアの信頼性向上のために2010年に企業6社と国立情報学研究所が共同で設立した研究活動を行う団体である。エンタプライズ系ソフトウェアを対象として、フォーマルメソッドの適法方法などについて検討している。図書館システムを例題とした試行実験に基づき、Event-B、Spin、VDM++などについてフォーマルメソッド活用ガイドを作成している。

17.4. 海外におけるフォーマルメソッド適用に関する情報

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が公開している報告書や学会のサー

ベイ論文¹⁷⁹にもあるとおり、海外では数々の適用事例があることが知られている。また、適用のためのガイドラインを公開している組織やプロジェクトもある。こうした情報を得るための主な情報源を以下にまとめる。

17.4.1. 国際学会

フォーマルメソッドの国際的な研究コミュニティである**Formal Method Europe(FME)**が運営しているウェブサイト¹⁸⁰から多くの国際会議を知ることができる。たとえば、以下の国際会議がある。これらの国際会議の中で産業応用のセッションやワークショップから応用動向の情報を得ることができる。

- INTERNATIONAL SYMPOSIUM ON FORMAL METHODS (FM)
- International Conference on Computer Safety, Reliability and Security (SAFECOMP)
- International Conference on Verified Software: Theories, Tools and Experiments (VSTTE)
- International Symposium on Automated Technology for Verification and Analysis (ATVA)

この他にも以下の国際会議があり、適用事例についても発表されている。

- International Conference on Computer Aided Verification
- International Conference on Formal Engineering Methods
- International Conference on integrated Formal Methods
- International Conference on Software Engineering and Formal Methods
- International Workshop on Formal Methods for Industrial Critical Systems

また、特定のフォーマルメソッド・ツールや応用分野を対象とした国際会議や研究機関が主催する会議もある。

- The International B Conference
- Spin Workshop
- Formal Methods for Automation and Safety in Railway and Automotive Systems
- Annual IEEE/NASA Software Engineering Workshop
- NASA Formal Methods Symposium
- Brazilian Symposium on Formal Methods

ソフトウェアエンジニアリング全般を対象とした国際会議(または併設ワークショップ)や学会誌や論文誌等でも紹介される場合がある。

- International Conference on Software Engineering
- The joint meeting of the European Software Engineering Conference and the ACM
- International Conference on Automated Software Engineering
- SIGSOFT Symposium on the Foundations of Software Engineering
- IEEE Computer, IEEE Software, Communications of the ACM 等

17.4.2. 研究所

下記の例にあるように、フォーマルメソッドの適用事例を発表している民間の研究所や公的機関の研究所もある(ただし、多くは学会等で公開されたものである)

- Microsoft Research Rigorous Software Engineering
<http://research.microsoft.com/en-us/groups/rse/>
- Intel Corporation
<http://www.intel.com/technology/itj/index.htm>
- Nokia Research Center
http://research.nokia.com/people/ian_oliver
- Google Research Publications by Googlers in Algorithms and Theory

¹⁷⁹ JIM WOODCOCK 他, "Formal Methods: Practice and Experience", ACM Computing Surveys, Vol. 41, No. 4, Article 19, Publication date: October 2009.

¹⁸⁰ <http://www.fmeurope.org/>

- <http://research.google.com/pubs/AlgorithmsandTheory.html>
- ClearSy System Engineering
<http://www.clearsy.com/index-en.php>
- Altran Praxis Formal Computing
<http://www.altran-praxis.com/formalComputing.aspx>
- Escher Technologies Limited.
<http://www.eschertech.com/>
- NASA Langley Formal Methods Team
<http://shemesh.larc.nasa.gov/fm/index.html>

17.4.3. 実証プロジェクト

以下にあるように欧州を中心にフォーマルメソッドの実証プロジェクトが続いており、適用事例や適用研究に関する情報を得ることができる。

- Automatic Verification and Analysis of Complex Systems (AVACS)
<http://www.avacs.org/>
- Deploy
<http://www.deploy-project.eu/>
- AVANTSSAR
<http://www.avantssar.eu/>
- COCONUT
http://www.iaik.tugraz.at/content/research/design_verification/coconut/
- HATS
<http://softtech.informatik.uni-kl.de/Homepage/HATS>
- MOGENTES
<https://www.mogentes.eu/>
- CONNECT
<http://connect-forever.eu/index.html>

17.4.4. 適用ガイド

NASA や過去の実証プロジェクトではフォーマルメソッド適用のためのガイドも公開している。

- NASA Formal Methods Guidebook, Vol. I, Release 2.0, 1998

NASA Formal Methods Guidebook, Vol. II 1997

http://eis.jpl.nasa.gov/quality/Formal_Methods/

本ガイドの補足するものとして有用であると考えられるが情報が古く、英語である点が難点である。フォーマルメソッドに関するプロジェクトマネジメントや費用対効果などについては殆ど書かれていない。

- EASIS Guidelines for verification and validation of dependability requirements

Formal Verification Techniques

自動車分野の高安全なシステム開発技術の研究開発プロジェクトである EASIS project(Electronic Architecture and System Engineering for Integrated Safety Systems) の成果報告書の一部である。自動車分野を対象としているものの、モデル検査の適用に関する有用なノウハウをまとめている。公開 Web サイトがクローズされたため、その入手は難しいが、一部は IPA 報告書「高信頼ソフトウェア構築技術に関する動向調査」で紹介されている。フォーマルメソッドに関するプロジェクトマネジメントや費用対効果などについては殆ど書かれていない。

18. フォーマルメソッド導入ガイダンス検討委員会

本ガイダンスは、以下の検討委員会においてガイダンスの方向性を議論し、レビューを行った。

フォーマルメソッド導入ガイダンス検討委員会

2008年10月1日～2010年3月31日

(委員長)

荒木 啓二郎 大学院システム情報科学研究院 情報工学部門 計算機科学講座 教授

(委員)

岸 知二 北陸先端科学技術大学院大学 情報科学研究科 教授
木下 佳樹 独立行政法人産業技術総合研究所 システム検証研究センター長
玉井 哲雄 東京大学大学院総合文化研究科 広域システム科学系 教授
中島 震 国立情報学研究所アーキテクチャ科学研究系 教授
萩谷 昌巳 東京大学大学院理学系研究科情報科学専攻 教授
二木 厚吉 北陸先端科学技術大学院大学 情報科学研究科 教授
本位田 真一 国立情報学研究所アーキテクチャ科学研究系
来間 啓伸 株式会社日立製作所 システム開発研究所
田中 一之 日立ソフトウェアエンジニアリング
梅村 晃広 株式会社 NTT データ

(事務局)

経済産業省商務情報政策局情報セキュリティ政策室
株式会社三菱総合研究所

フォーマルメソッド導入ガイダンス検討委員会

2010年4月1日～2011年3月31日

(委員長)

荒木 啓二郎 大学院システム情報科学研究院 情報工学部門 計算機科学講座 教授

(委員)

岸 知二 早稲田大学 理工学術院 創造理工学部 経営システム工学科 教授
木下 佳樹 独立行政法人産業技術総合研究所 システム検証研究センター長
玉井 哲雄 東京大学大学院総合文化研究科 広域システム科学系 教授
中島 震 国立情報学研究所アーキテクチャ科学研究系 教授
萩谷 昌巳 東京大学大学院理学系研究科情報科学専攻 教授

二木 厚吉	北陸先端科学技術大学院大学 情報科学研究科 教授
来間 啓伸	株式会社日立製作所 システム開発研究所
田中 一之	日立ソフトウェアエンジニアリング

(事務局)

経済産業省商務情報政策局情報セキュリティ政策室
株式会社三菱総合研究所

表目次

表 2-1: フォーマルメソッドの適用レベル	7
表 2-2: 形式検証のアプローチ	8
表 4-1: 導入プロセスの参考としたフォーマルメソッド適用事例	13
表 4-2: 導入プロセスのパターン分類	14
表 5-1: フォーマルメソッド導入の主な効果	26
表 5-2: フォーマルメソッドの主な投資コスト	27
表 5-3: ソフトウェアの不具合に起因する損害リスクの要素	29
表 5-4: 障害発生時の影響度を把握するための主な検討項目(参考例)	33
表 5-5: フォーマルメソッドによる品質の向上に関する主な効果	34
表 5-6: 効果の具体例	35
表 5-7: SPINの技術習得コストの例	42
表 5-8: フォーマルメソッドを用いた設計レベル	42
表 5-9: ケーススタディにおけるフォーマルメソッドの設計付加コスト(時間)	43
表 5-10: 鉄道システムに対するBメソッドの適用プロセスの工数比率	43
表 6-1: ソフトウェアに対する要求	46
表 6-2: ISO/IEC 9126 (ソフトウェアの品質特性)	46
表 6-3: 非機能要求の主な要素とその内容	47
表 6-4: ディペンダビリティの構成要素	48
表 6-5: ディペンダビリティに対する脅威の分類	49
表 6-6: ディペンダビリティに関する対策の分類	49
表 6-7: ディペンダビリティ対策の手法例	49
表 6-8: 検証のレベル	54
表 7-1: 主な形式手法の概要	64
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	83
表 8-2: モデリングパターンの選択基準	85
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)	93
表 9-1: 抽象モデルと具体モデルの関係と用途	102
表 9-2: 型と変域の大きさ	107
表 11-1: モード制御ミドルウェアの要求仕様	119
表 11-2: Blu-ray系インタフェースの内部状態	121
表 11-3: Blu-rayミドルウェアモジュールの内部状態	121
表 11-4: 内部状態の対応関係	122
表 11-5: Blu-ray系インタフェースのAPI	123
表 11-6: における各部分のモデリングの方針とその理由	125
表 12-1: 検証対象システムの構成要素	135
表 12-2: ボックス管理システムの要求仕様	137
表 12-3: 待機者リストのデータ型	137
表 12-4: 待合室管理プロセスの受信メッセージと対応するアクション	138
表 12-5: 閲覧室X(X=A or B)管理プロセスが受信するメッセージと対応するアクション	138
表 12-6: 閲覧室Xの扉の解錠条件	140
表 12-7: 検証対象システムのモデリングの前提	143
表 12-8: 待合室管理プロセスの状態遷移表	148
表 12-9: 閲覧室Xプロセスの状態遷移表	150
表 12-10: 各ID端末のID番号割り振り	151
表 12-11: ID端末から送信可能なメッセージ	152
表 12-12: 各管理プロセスの処理と処理対象	154

表 12-13: 本ケーススタディにおける要求仕様と検証性質.....	158
表 13-1: 応用事例情報の整理項目	166
表 14-1: 性質記述パターンライブラリの概要	215
表 15-1: 再利用に対する期待	216
表 15-2: 再利用を行う際の困難 ¹⁵⁸	217
表 15-3: 検証量を減らすための工学的な手法例.....	225

目次

図 2-1: フォーマルメソッドの位置付け(設計検証の場合)	6
図 2-2: フォーマルメソッドの具体的イメージ(モデル検査SPINの場合)	7
図 4-1: 導入プロセスの概略(共通部分)	14
図 5-1: 発生可能性と障害発生時の影響度とリスクの関係	30
図 5-2: 不具合が発生したことによる対策費の総合計(2008 会計年度)	32
図 5-3: 開発現場が抱える問題点とフォーマルメソッドの適用で期待される効果の例	37
図 5-4: 欠陥の混入工程、発見工程、除去コスト	38
図 5-5: 開発プロセスのフロントローディングと許容負荷(文献EASISを元に作成)	40
図 6-1: ディペンダビリティ確保の対策分類と利用される技術の関係	51
図 6-2: 信頼性・安全性・セキュリティの関係	52
図 6-3: フォーマルメソッドの主な適用箇所と用途	54
図 6-4: 開発V字モデルにおける各種手法の適用箇所	55
図 6-5: 障害の区別と評価対策アプローチ	57
図 7-1: 主な形式手法の用途例の俯瞰	62
図 7-2: 成功事例に基づく目的と対象(レベル)による適用手法の典型例	63
図 7-3: SPINの開発プロセスにおける位置づけ	69
図 7-4: NuSMVの開発プロセスにおける位置づけ	71
図 7-5: UPPAALの開発プロセスにおける位置づけ	73
図 7-6: SCADEの開発プロセスにおける位置づけ	74
図 7-7: 過去 25 年の形式手法の変遷	75
図 7-8: Bの開発プロセスにおける位置づけ	77
図 7-9: Event-Bの開発プロセスにおける位置づけ	78
図 7-10: VDM++の開発プロセスにおける位置づけ	80
図 8-1: コンポーネント駆動パターンの概要	86
図 8-2: システムの構成要素と相互作用の例	88
図 8-3: 状態遷移表に基づくPromela記述テンプレート(例)	92
図 8-4: アルゴリズム駆動プロセスの概要	94
図 8-5: 要求性質駆動プロセスの概要	95
図 9-1: 2つの並行プロセスの状態遷移図(例)	98
図 9-2: 非同期並行実行による状態遷移図	98
図 9-3: 交通信号モデルの抽象化(例)	100
図 9-4: 抽象化の健全性の条件(言葉による説明)	101
図 9-5: モデルの抽象化と状態回避策に関する全体像	103
図 9-6: 実質的に逐次的な処理のシーケンス図	105
図 9-7: 冗長な中間状態s1 がある場合	111
図 9-8: 状態s1 を省いて、a;b を 1 ステップにした場合	111
図 9-9: スタックの利用前部分の不定値	113
図 11-1: 検証対象システムの構成	116
図 11-2: 検証対象システムの構成	118
図 11-3: プロセスの構成図	120
図 11-4: 実際のシステムのプロセス構成とモデリングを行う単位	125
図 11-5: 実際のシステム構成とモデルのプロセス構成	126
図 11-6: モードモジュールの状態遷移表	128
図 11-7: ミドルモジュールの状態遷移表	130
図 12-1: ボックス管理システム全体像	135

図 12-2:表示装置の更新タイミング	142
図 12-3:実際に近いモデル(全体像)	145
図 12-4:単純化した検証対象システムのモデル(全体像)	146
図 12-5:閲覧者の動線(システム設計時)	147
図 12-6:parsonData型の定義	148
図 12-7:BoothRoomParsonData型の定義	150
図 12-8:チャンネルpciの定義	151
図 12-9:扉脇ID端末認証装置のモデル化.....	152
図 12-10:在室センサのモデリング	153
図 12-11:データベース(DB)と表示装置のモデル化	154
図 12-12:表示装置のモデリング	156
図 12-13:閲覧者のオートマトン(大枠)	157
図 12-14:待合室における閲覧者のオートマトン	157
図 12-15:検証内容S01 のLTL式.....	159
図 12-16:検証内容S02 の基本となるLTL式.....	160
図 12-17:検証内容S02 のLTL式.....	160
図 12-18:待合室のPromela記述.....	161
図 12-19:assert文を追記した待合室のPromela記述.....	162
図 12-20:S03 の検証結果.....	163
図 12-21:対処を行った待合室のPromela記述	165
図 13-1:ソフトウェアアーキテクチャ設計	167
図 13-2:SCADEにより作成されたスケジューリングとコミュニケーションモデル	169
図 13-3:UPPAALによるシーケンス制御モデル図.....	170
図 13-4:入退室管理システムの全体像.....	172
図 13-5:プラットフォームドアの構成.....	173
図 13-6:列車制御システムの構成	176
図 13-7:MULTOSの概要	183
図 13-8:実践した形式モデリングの流れ	185
図 13-9:OpenComRTOSのアプリケーション概要	186
図 13-10:TSEアーキテクチャの概要.....	189
図 13-11:OpenNANDフラッシュメモリのためのストレージプラットフォーム.....	190
図 13-12:意思決定システム(BOS)の全体像	192
図 13-13:X線CTスキャンのアーキテクチャの概要	195
図 13-14:無人シャトル制御システムにおける抽象モデルと具体モデルとの関係.....	197
図 13-15:開発されたプラットフォームドア	198
図 13-16:ディスプレイアーキテクチャ(オレンジ色はディスプレイアプリケーション、緑色はウインドウマネージャ)	200
図 13-17:TradeOneシステムの構成	202
図 13-18:IECSソフトウェアの構成図	204
図 13-19:開発された自動販売機の構成図.....	205
図 13-20:Static Driver Verifier toolの解析エンジンアーキテクチャ.....	207
図 13-21:鉄道制御システムの主要なStatemateモデル.....	208
図 14-1:LTLとCTLの簡約な表現	213
図 14-2:LTLによる表現.....	214
図 14-3:CTLによる表現.....	214
図 15-1:再利用の基本構造	216
図 15-2:再利用における検証一般の課題	218
図 15-3:フィーチャモデルの例.....	220

図 15-4: 再利用におけるモデル検査技術の適用の概観	221
図 15-5: 再利用資産のモデル検査の例	222
図 15-6: フィーチャモデルを用いた検証状況の明確化例	223
図 15-7: 再利用可能な検証資産の例	224
図 16-1: 信頼度成長曲線	230
図 16-2: 信頼度成長曲線の傾きとテストに発生している問題の例	231
図 16-3: 欠陥の摘出工程分布と品質評価	231
図 16-4: W字モデル開発	233

索引

Agda	234	障害の発生可能性.....	29
Alloy	232	障害発生時の損害規模	29
Bandera.....	234	情報セキュリティ経済学.....	32
CafeOBJ.....	234	信頼性.....	52
CBMC.....	233	SCADE.....	65
Coq	233	SPIN.....	64
CTL	235	設計仕様	54
ESC/JAVA2	233	絶対安全	31
FDR2	232	潜在化損失.....	31
FMEA.....	50	ソフトウェアの信頼性	47
FTA.....	50	ソフトウェアの品質	47
HAZOP.....	50	損害リスク.....	28
JavaPathfinder.....	233	ディペンダビリティ	48
LOTOS	234	ディペンダビリティ対策.....	50
LTL.....	235	テスト中心プロセス	39
PVS.....	231	NuSMV	64
SAL.....	232	B	66
Z	235	非機能要求.....	46
安全性	52	VMM	25
Event-B.....	66	VDM++.....	67
UPPAAL.....	65	フォーマルメソッド	5
機能要求	46	フォーマルメソッドの適用レベル	7
脅威.....	49	フロントローディング	11
形式手法	5	無形資産	33
形式仕様記述言語.....	5	モデルベース開発.....	50
欠陥予防開発プロセス.....	39	要求.....	54
顕在化損失.....	31	要求仕様	54
CIO	ii	要件定義書.....	54
CTO	ii		
仕様.....	54		