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	● 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適用することにより、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。

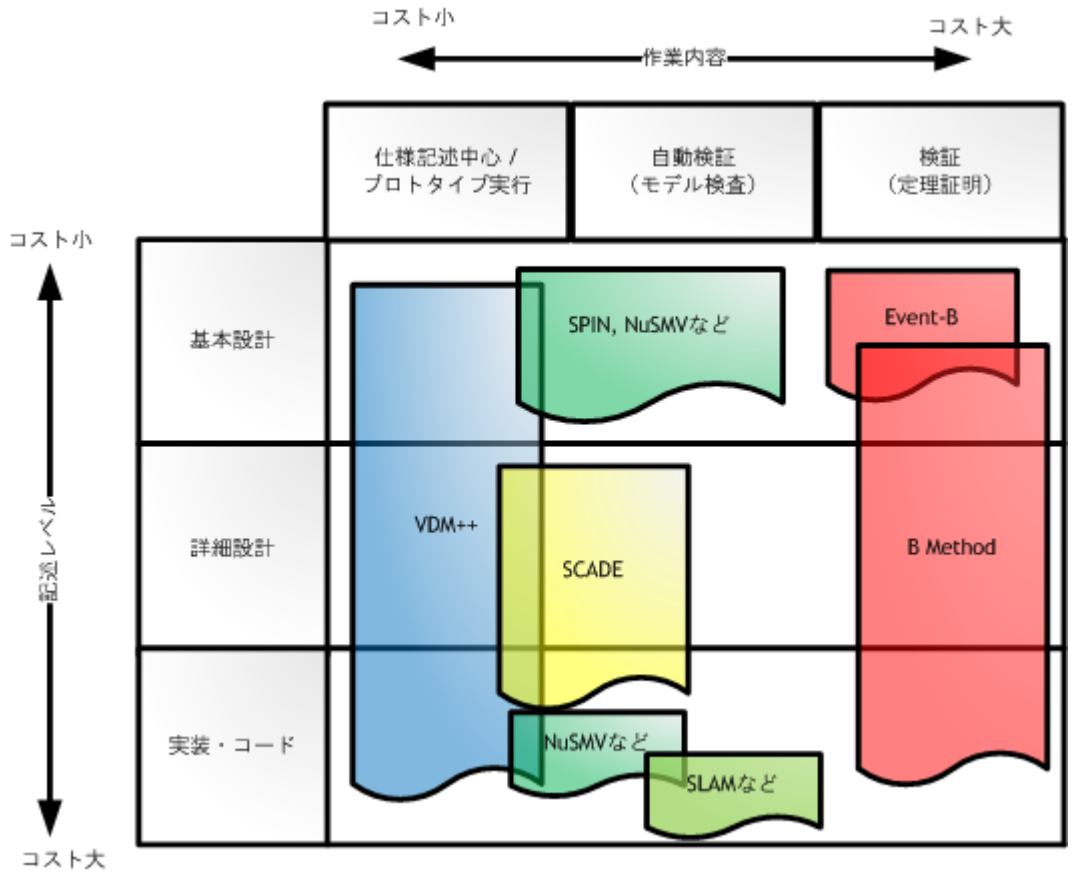

図 7-1:主な形式手法の用途例の俯瞰

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。

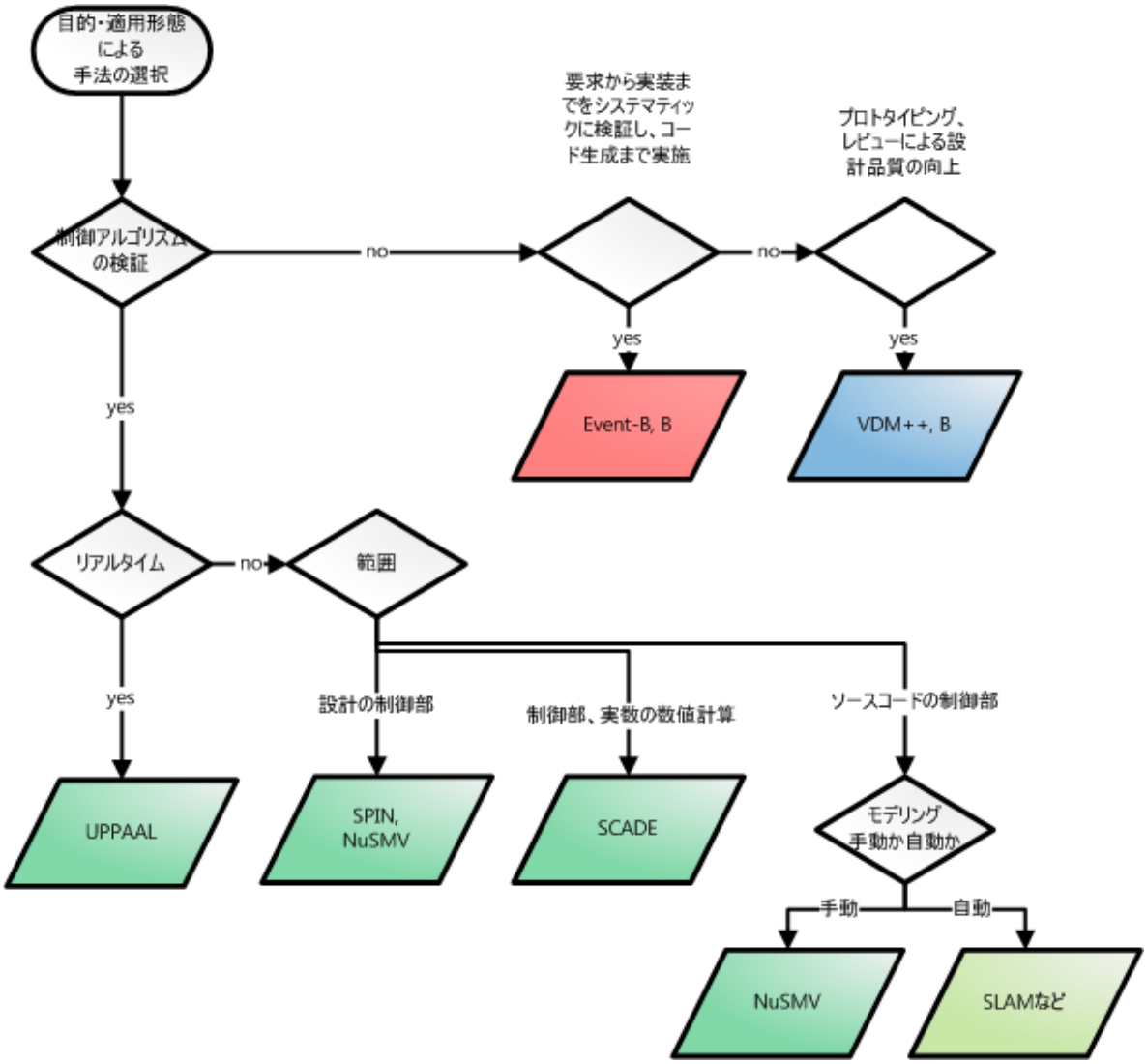

図 7-2:成功事例に基づく目的と対象(レベル)による適用手法の典型例

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++はプロトタイピングなど、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	● 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	● 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)	● 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。 - 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ックス - 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	● 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	● 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、 * 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	● 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．各段階で仕様の盤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 言語という言語を用いて、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	● 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、 - 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 である。 ● 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.org 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

			● 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	● 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/twiki/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さ

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 式を組み合わせて検証する。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（∀）、存在記号（∃）に加え、G([],Always, つねに)、F（<>, 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。

  []($p \rightarrow$(<>q)))

■ システムのある特定の状況が無限回発生する（公平性）

([]<>p)→φ

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。

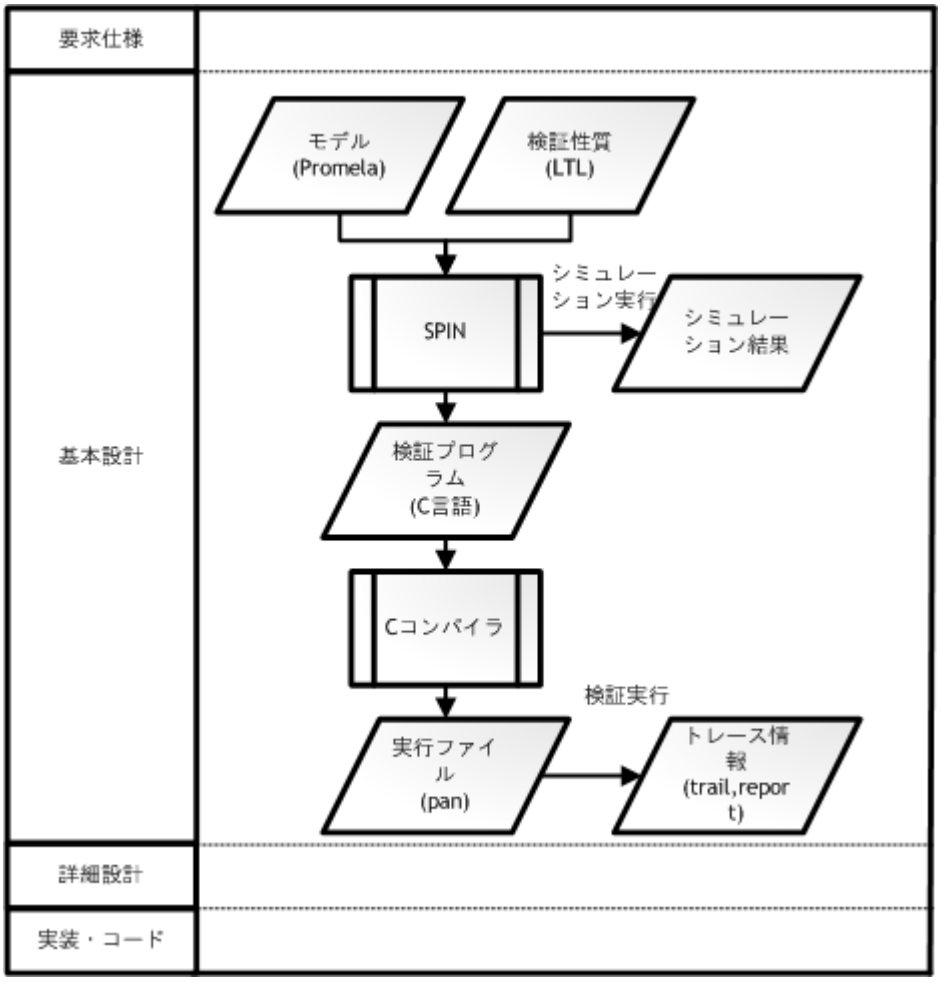

図 7-3：SPIN の開発プロセスにおける位置づけ

[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PSL[136]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

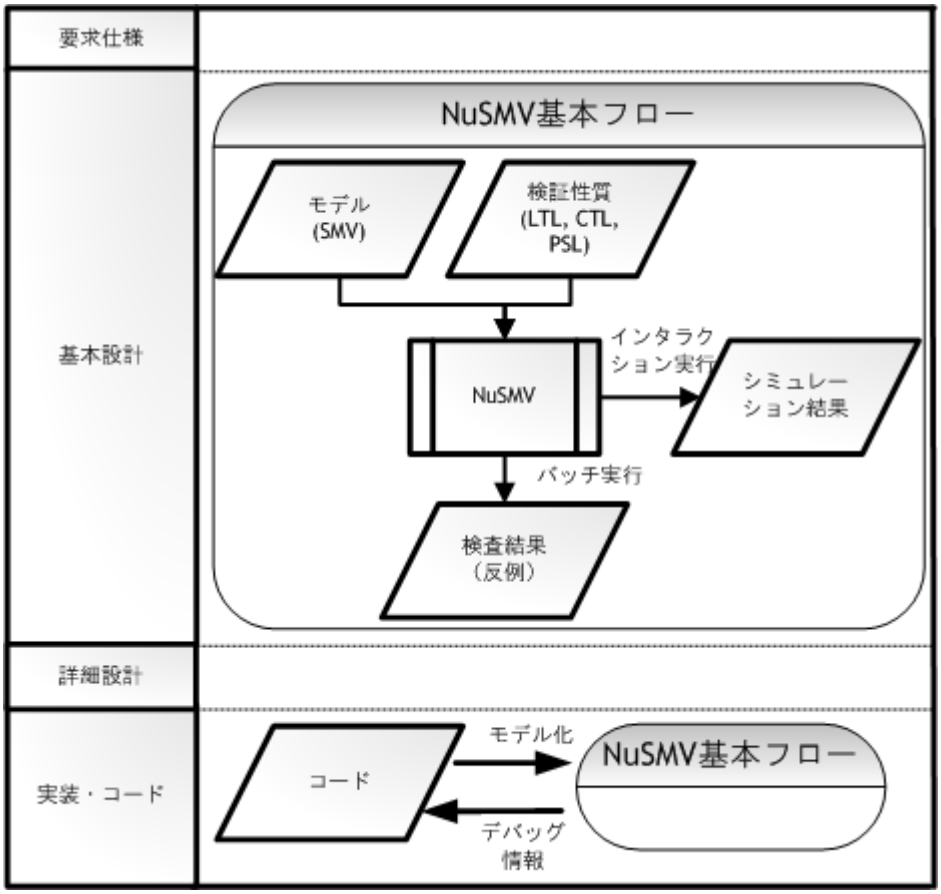

図 7-4：NuSMV の開発プロセスにおける位置づけ

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

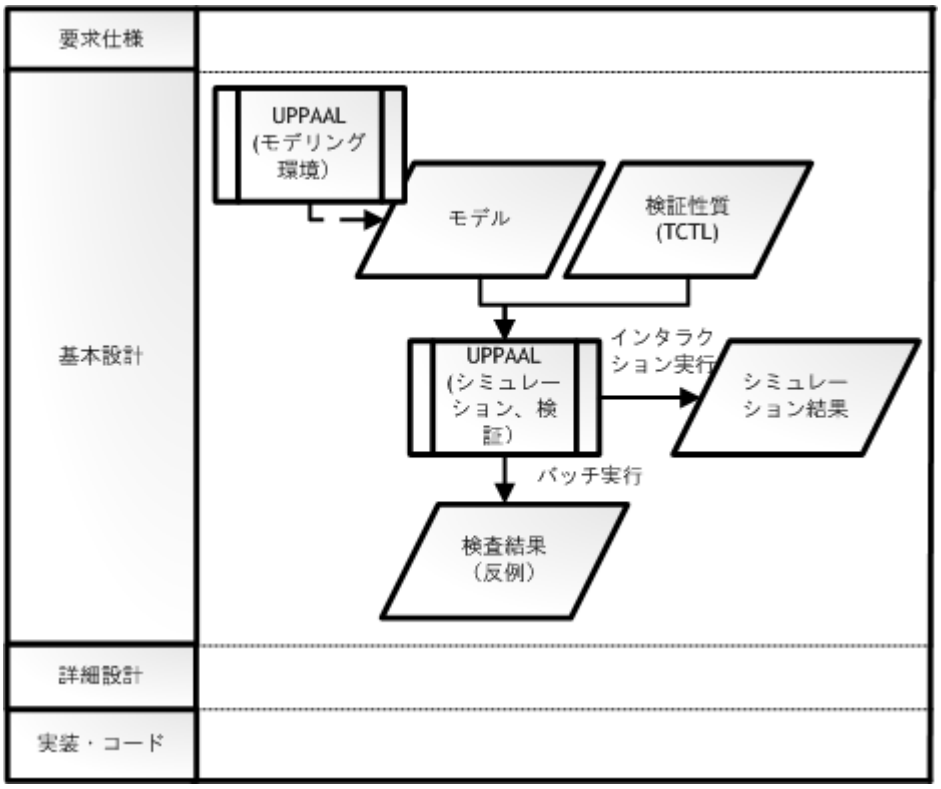

図 7-5：UPPAAL の開発プロセスにおける位置づけ

7.3.1.4. SCADE

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 SCADE 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。

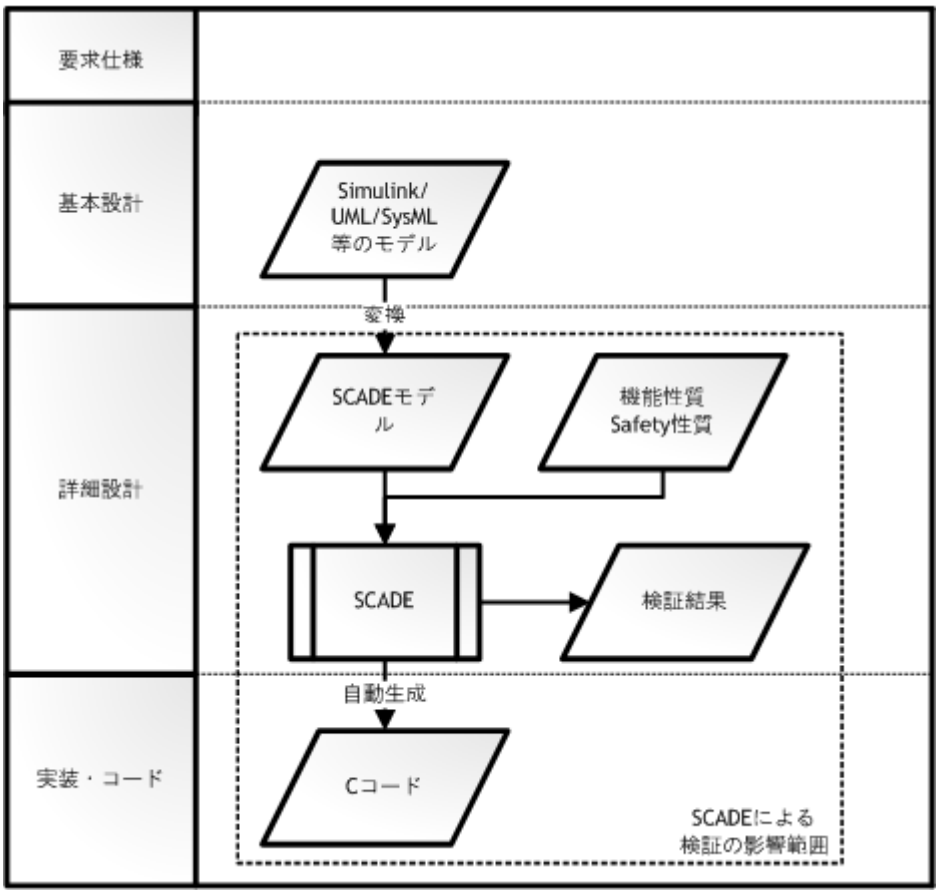

図 7-6：SCADE の開発プロセスにおける位置づけ

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。

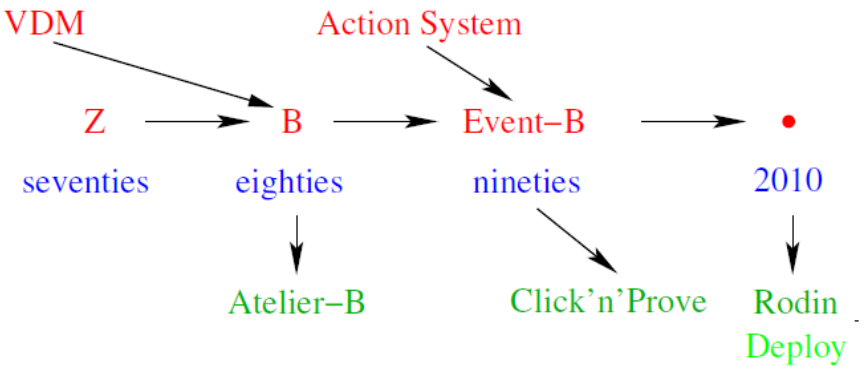

図 7-7:過去 25 年の形式手法の変遷[138]

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。

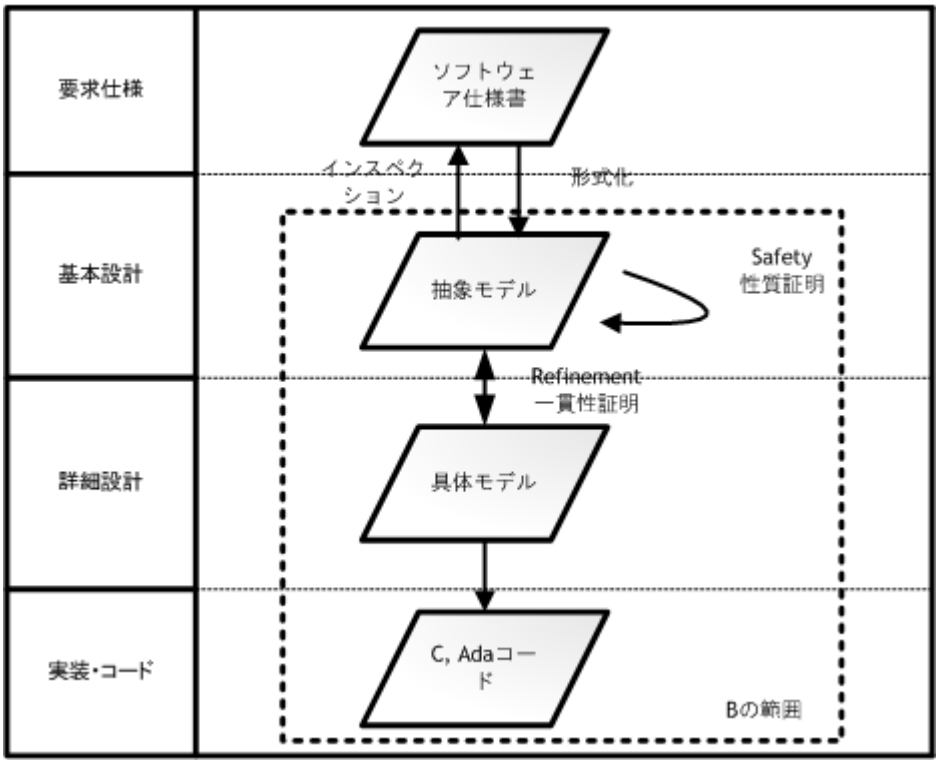

図 7-8：B の開発プロセスにおける位置づけ

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 Platform[139]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。

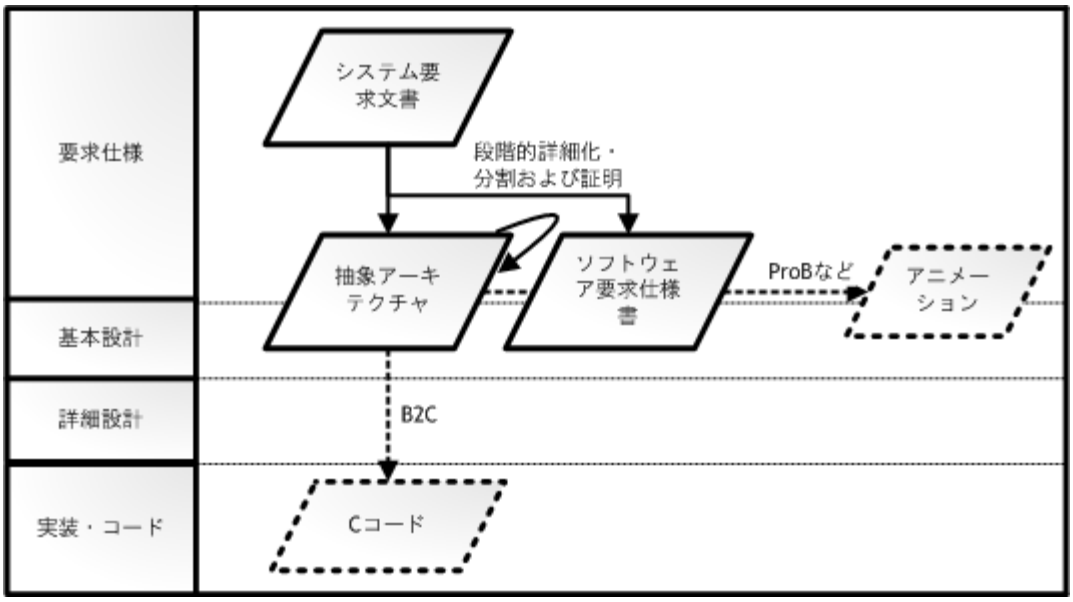

図 7-9:Event-B の開発プロセスにおける位置づけ

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ープリタとデバッグ機能
- 実行可能仕様のコードカバレージ計測機能
- Rational Rose との連動機能

[141] 出典：VDM とは， VDMTools.jp， http://www.vdmtools.jp/modules/tinyd1/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。

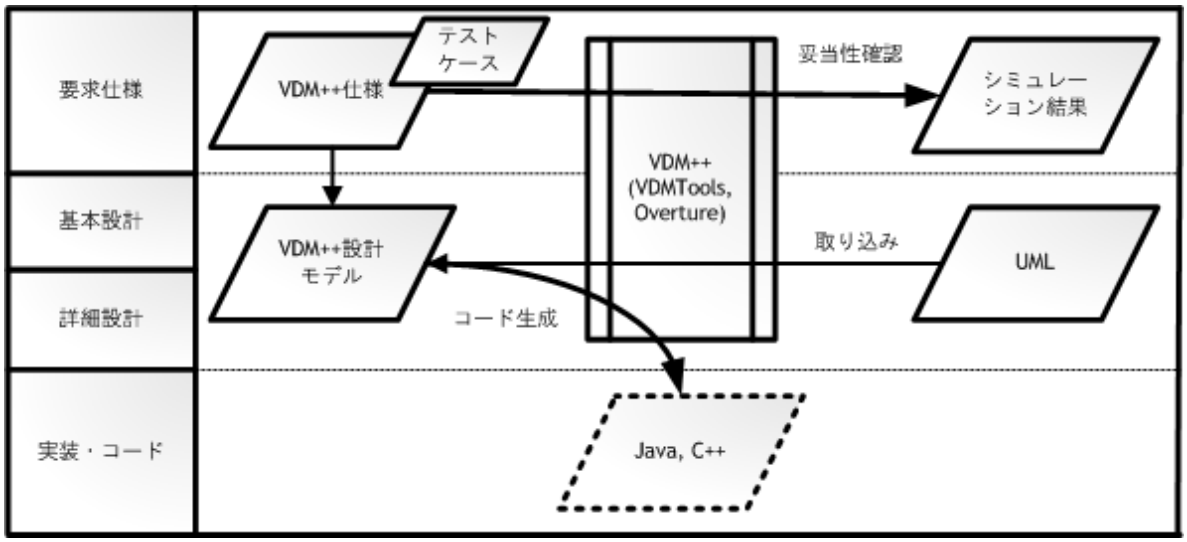

図 7-10：VDM++の開発プロセスにおける位置づけ