SEC BOOKS

組込みソフトウェア開発における品質向上の勧め［設計モデリング編］

独立行政法人 情報処理推進機構　ソフトウェア・エンジニアリング・センター　編

ITmedia Inc.

目　次

1 はじめに

モデリングとは

設計書記述ルールの曖昧さがトラブルを生む

皆様はソフトウェアの設計書をどのように書いていますか？ 多くの場合、自由な図（箱と線がよく使われます）と日本語（いわゆる生活言語です）で書くことが多いのではないでしょうか？ こうした設計書は、ある意味直感的に理解しやすいですが、正確性に欠けます。

例えば図に描かれた箱がどういった意味を持っているのかを明確に定義していますか？ ある人は実行形式ファイルだと思っているのに、別の人は実行時のプロセスイメージと捉えているかも知れません。線に至ってはもっと曖昧なことが多いようです。漠然と書かれた関係線が、呼び出し関係なのか、変数の参照関係なのか、定義の依存関係なのか、それぞれ人によって違う意味合いで理解しているかも知れません。

また、日本語で記述された設計書の場合でも、各所に出てくる「本システム」「当該システム」「対象システム」等々といった用語がいったい何を指すのかはっきりしないことがありますし、書き間違い・読み間違いがあっても、生活言語で記述されている限りチェックしきれるものではありません。

このように設計書を書く場合にその記述ルールが曖昧だと、思わぬ誤解を引き起こし、設計の誤りや品質低下を誘発してしまいます。

モデリングは設計品質向上のスタートポイント

モデリングとは、こうした設計を「書き方・読み方の決まった方法で書く」ことを意味しています。即ち、モデリングを実践することにより、正確で曖昧性の少ない記述方法に基づいて設計を記述できるのです。

こうした設計の記述方法はたくさん用意されています。例えば状態遷移図やUML（Unified Modeling Language）などの図法はよく知られています。また、図だけでなく表やマトリクスもモデリングの際に使うことができます。

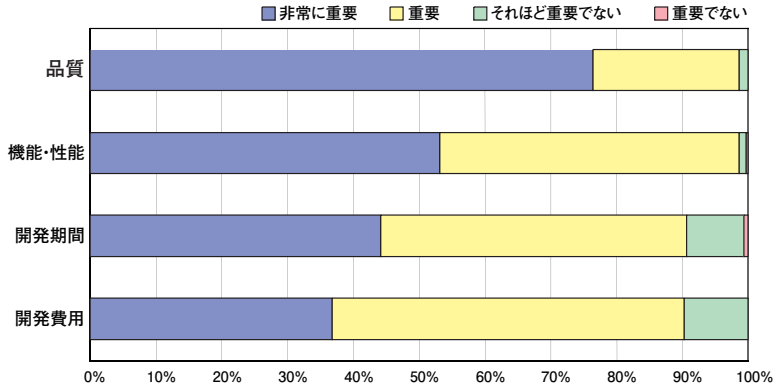
モデリングはソフトウェアの設計品質を向上させる上でのスタートポイントともいえます。本書は、こうしたモデリングの世界に関心を持っていただくために書きました。モデリングについて、その世界を少しでも垣間見て、設計品質向上の手段として興味を持っていただければ幸いです。

組込みソフトウェアとモデリング

組込みソフトウェアで問題となっている大きな課題の1つが「設計品質」です。

経済産業省の2005年版「組込みソフトウェア産業実態調査：開発プロジェ

図 1-1　プロジェクトで開発する製品・システムの重要度



経済産業省　2005年版「組込みソフトウェア産業実態調査：開発プロジェクト責任者向け調査」

クト責任者向け調査」において「プロジェクトで開発する製品・システムの重要度」（図1-1）は、『品質』がトップでした。組込みソフトウェアはその特性上、高度な信頼性や安全性が求められますが、ソフトウェアの大規模化・複雑化によって、従来からの熟練者の勘と経験に頼った手法では、その設計品質を維持できなくなってきているのです。

そうした中で、設計にモデリング技術を活用するということが、ソフトウェア開発における大きな流れになっています。そこで本書では、品質向上の解の１つであるモデリング技術に焦点をあて、設計品質との関係について解説します。

モデリングのメリット

では、モデリングには、どのようなメリットがあるのでしょうか。

●正確な設計が可能となる

書き方・読み方が定まった記述方法を用いることにより、記述もれ、曖昧さ、重複などの少ない、より正確な設計を行えるようになります。それに伴い、設計の正しさの確認もより厳密に行えるようになります。

●設計が見えるようになり、レビュー効率アップ

モデリング技術を使えば、ソフトウェアの設計がより把握しやすくなり、レビュー効率がアップします。また、品質管理のベースとして様々な計測を行うこともできるようになります。

●コミュニケーションが円滑になる

正確かつ捉えやすい設計書を記述することにより、開発に関わる様々な人の間でのコミュニケーションがスムースに行われるようになります。

●シミュレーションや検証をツール化できる

従来設計書の確認は人手でレビューするしかありませんでしたが、モデル化を行うことにより、ツールによるシミュレーションやチェック機能を活用する道が開けます。

●標準化が促進され再利用が容易になる

モデリングのための標準的な記述方法が各種提案されています。こうした標準記法を用いることにより、設計書の標準化が促進され、再利用が容易になります。

このように、モデリングによって期待されるメリットは様々なものがあります。

皆様もモデリング技術を活用して、新しい設計にチャレンジしてみませんか?

本書の目的

本書では設計モデリングについて、以下の目的を持って解説します。

- **モデリング技術の有効性の紹介**
  ――設計の役割、そこにおけるモデリング技術の重要性を提示
- **モデリング技術の全体像の解説**
  ――重要なモデリング技術の整理
- **検証技術とのかかわりの説明**
  ――モデリング技術とそれに対応した検証技術の整理

本書の構成

本書では、設計モデリングについて以下の構成で紹介します。また、モデリング技術導入編として各種モデリング記法や検証技術の概要なども合わせて紹介していきます。

- ●設計品質とモデリング技術
- ●モデリング技術のマップ
- ●モデリング技術概説
- ●モデリングと検証技術
- ●モデリングの組込みソフトウェアへの適用性
- ●組込みソフトウェアへのモデリング技術活用のポイント

本書の対象読者

本書では、モデリングの考え方や利用方法を確実に理解したい、組込みシステム開発のマネージャー・エンジニアを対象にしています。モデリングの重要性と基本的技法を理解していただき、その概要を知っている読者がそれを利用する読者へ移行することを期待しています。

2 設計品質とモデリング技術

2-1 設計品質とは

設計品質とは“ねらいの品質”

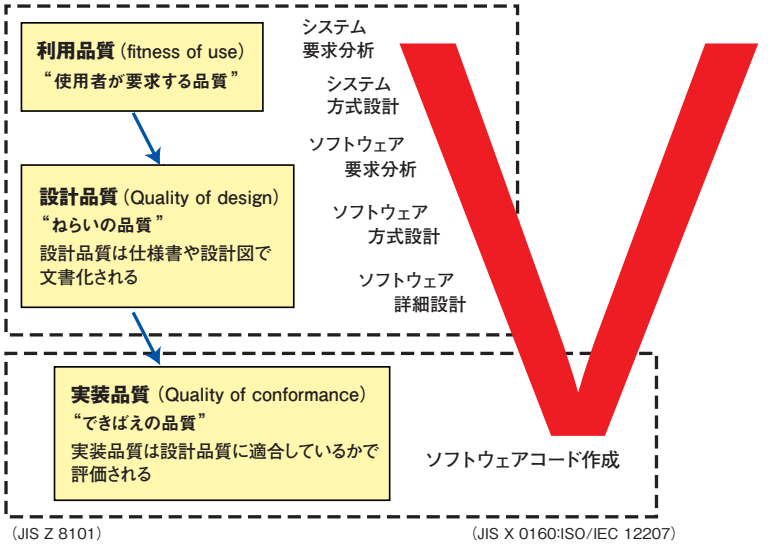
設計品質とは、“ねらいの品質”です。実装品質とは、“できばえの品質”です。設計品質は仕様書や設計図で文書化されます。実装品質は設計品質に適合しているかで評価されます。

設計図通りに施工しない手抜き工事は実装過程での品質の不良です。建物ができ上がってから、設計通りに施工しているかを検査することは、難しいことです。建物の下にある基礎、コンクリートに埋められた鉄筋などを、一部壊して検査しなければなりません。そこで、実装面での品質では、実装工程を検査することが重要となります。これはソフトウェア開発において、ソフトウェア開発プロセスが重要なことと共通しています。

耐震設計を偽装したような設計品質が低い建物は、どれだけ“できばえの品質”が良くとも、そもそもの“ねらいの品質”が低いわけですから、建物としての品質は不良です。設計を実装でカバーすることはできないのです。

設計品質はソフトウェアでも同様に重要です。しかしながら、現実には新しいハードウェアや新しいサービスに対応しようとした場合に、実装工程で設計を行ってしまうことが起こりがちです。この点についてモデリング技術がどのように役立つかを後ほど説明します。

図 2-1 設計品質とソフトウェアライフサイクルプロセス



設計品質には利用者の視点も必要

設計品質と実装品質だけでは、設計はいかに実装しやすいかといった開発者だけの視点になりがちです。そこで、利用者の眼から見た品質（利用品質）の視点を設計品質に加えることで、建物の場合であれば使い勝手や住み心地を考慮した設計ができることになります。どんなに耐震性が高いといわれても、窓が1つもない部屋には住みたくはありません。カーナビ、携帯電話やデジタル家電などの組込みソフトウェアで実現すべき使用品質は、パソコンとは異なるユーザ層を対象とした使用品質を設計しなければなりません。

本書において、設計品質はソフトウェアライフサイクルプロセス（SLCP）における「ソフトウェア要求分析」「ソフトウェア方式設計」「ソフトウェア詳細設計」に対応し、実装品質は「ソフトウェアコード作成」に対応するものとします。

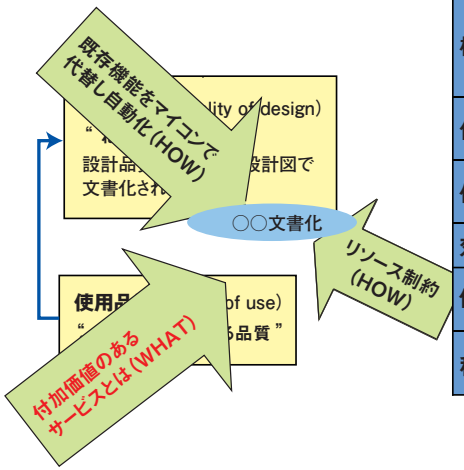
2-2　ソフトウェア品質特性

ソフトウェアの品質には信頼性以外にも様々な観点がある

ソフトウェア品質特性（JIS X 0129 : ISO/IEC 9126）は、ソフトウェアに求められる設計品質とは何かを知る上で参考になります。ソフトウェア品質特性ではソフトウェアの特性を以下の 6 つに分類し、それぞれに副特性を定義しています。

①機能性　②信頼性　③使用性　④効率性　⑤保守性　⑥移植性

図 2-2　ソフトウェア品質特性



特性	副特性
機能性	合目的性　正確性 相互運用性　標準適合性 セキュリティ
信頼性	成熟性　障害許容性 回復性
使用性	理解性　習得性 運用性
効率性	時間効率性　資源効率性
保守性	解析性　変更性 安定性　試験性
移植性	環境適応性　設置性 規格適合性　置換性

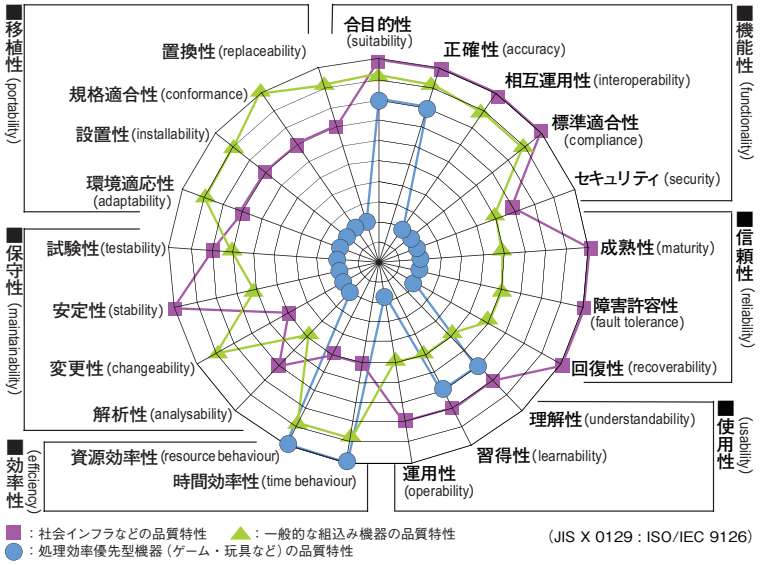
（JIS X 0129 : ISO/IEC 9126）

近年、電子技術や情報処理技術の進化により、従来ではあり得なかった新しいサービスを組込みソフトウェアで実現するようになってきました。こうした新規のサービス・機能の場合、過去の品質面での工夫や蓄積がない分、とりわけ設計段階で十分な品質確保が望まれます。

製品の特徴によって重視すべき品質特性は異なる

新しいサービスを搭載する製品では、これらの機能性、信頼性、使用性、効率性、保守性、移植性について、品質目標値を設定し、設計のアウトプットを検証することが必要になります。しかし、すべての項目で満点を取るなどは理想であって現実的ではありません。例えば、保守性を高めようとすれば、どうしても冗長な設計となり、効率性は犠牲になります。ですから、新しいサービスが必要とする品質特性がどこにあるかをはじめに設定し、設計にとりかからなければなりません。通信制御 LSI 向けデバイスドライバを開発する場合の品質特性、ユーザインタフェイスアプリケーションを開発する場合の品質特性、気温変化を予測解析し、ファンを制御するアプリケーションを開発する場合の品質特性など、それぞれ開発する対象によって品質特性のパターンが異なります。

図 2-3　ソフトウェア品質特性パターン



再利用する部分の品質レベルも考慮する

新しいサービスの全てを新規に開発することは有り得ません。既存機能

を再利用して新しいサービスを設計することもたくさんあります。そのため、再利用するソフトウェア部品がどの程度の品質水準にあるかといった情報があると、その部品を使えば、サービス全体の設計品質がどのようになるかを予測することができます。設計品質とは文書化されるものなので、ソースコードしかない既存機能を再利用することは、設計品質に悪い影響を与えることになります。

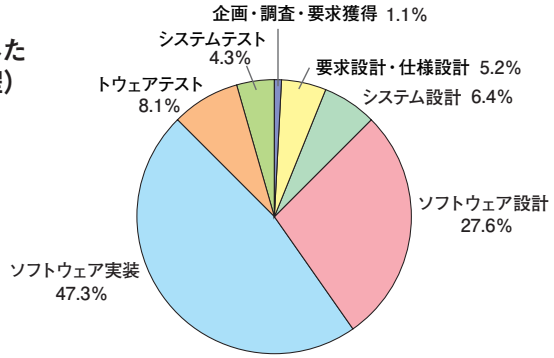
また、組込みソフトウェアは、証券システムのような1つのシステムではなく、多いものでは数百万台の製品に組み込まれるものです。部品を銭単位でコストダウンをはかるような組込み製品もあり、組込みソフトウェアが必要とするメモリサイズはシビアです。つまり、未知なる付加価値の高いサービスであっても、資源効率を良くして設計することが望まれます。

2-3　モデリング技術と設計品質

良い品質には設計段階の確認が重要

新しいサービスを提供しようとした場合、これまでは日本語などの自然言語で書かれた文書（設計書など）を参考にして、デザインレビューを行い、机上で新サービスの使い勝手、新サービスで起こり得るあらゆる事態とその対処方法を想定することが一般的でした。また、新サービスを動作させて検証することは、下流工程のコーディング終了後に、動作テストなどを行うことで初めて可能になります。動作テストでは様々な問題を発見することが可能ですが、ソフトウェア開発工程としては実際のソフトウェアをプログラムという形にしてしまった後にサービス仕様上の問題を検討していることになります。これは言い換えると、設計を製造工程で行っていることに等しいため、膨大なフィードバック工数がかかり、生産性や品質が大きく低下する原因になりかねません。

図 2-4　ソフトウェアの不具合を発生した工程（原因工程）



経済産業省　2005年版「組込みソフトウェア産業実態調査:開発プロジェクト責任者向け調査」

不具合の原因の大半は設計工程以前

経済産業省の調査では、「ソフトウェアの不具合を発生した工程」の半数近くは実装工程ではなく、設計工程となっています。

市場に出た後のリコールをなくすためには、設計段階であらゆる想定を行っておく必要があります。8～16ビットのプロセッサ上の組込みソフトウェアを開発する規模であれば、実装段階で設計をするようなことが許されたかも知れません。しかし、32ビット以上のプロセッサとなると、設計品質を設計工程で厳密に検査することが望まれるのです。

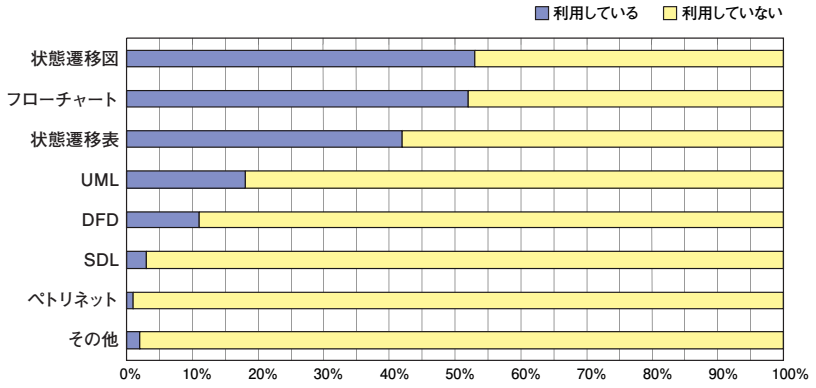
そこで、自然言語による設計から、コンピュータ上で動作可能なレベルのフォーマル（形式）言語による設計へ移行することが必要になります。このフォーマル言語による設計が、モデリングです。モデルを使うことにより、新しいサービスを設計段階で動作検証することができ、フィードバック工数を減少させることができます。

2-4　モデリング記述言語

設計を伝えるには“言葉”が必要

ソフトウェアの設計内容を表現する場合、何らかの“言葉”が必要とな

図 2-5　企画要求獲得・要求分析仕様設計・システム設計の工程で使用したフォーマル記述言語



経済産業省　2005 年版「組込みソフトウェア産業実態調査：開発プロジェクト責任者向け調査」

ります。組込みソフトウェアの世界では、フォーマルなモデルを記述する言語にはどのようなものが使われているでしょうか。

経済産業省の調査では、プロジェクトで「企画要求獲得・要求分析仕様設計・システム設計の工程で使用したフォーマル記述言語」は、状態遷移図とフローチャートが 5 割超、状態遷移表が 4 割超、UML が 2 割に満たない結果となっています。このように設計モデルを表現するための言語は様々ありますが、その中からどのような言語を選んで利用するかについては、それぞれの組織やプロジェクトの状況なども考慮し、適したものを選ぶようにしたほうがよいでしょう。

2-5　既存資産とモデリング

既存機能を再利用する際にも設計のモデルは重要

最近の組込みシステムでは、既に作成済みの機能（既存機能）を組み合わせることで、新しいシステムや装置を作り上げる場合も少なくありません。ところが、これらの既存機能のソフトウェアには、モデルはおろか、きちんとした文書すらなく、ソースコードだけしかないといった場合も考えられます。

こうした場合、既存機能に関する個々の部分については、新しいサービスを想定したテストケースを使い、“できばえの品質”（実装品質）を上げることは可能です。しかし、一方でこれらの機能を表現する設計モデルは必ずしも十分に整理されていないがために、結果的に、その部分に関する設計品質については確認がおろそかになってしまう場合があります。

こうした状況を回避するためには、再利用したい既存機能部分についてソースコードからモデルを作成し、それを利用することで設計品質の確認を進めるといった工夫が必要になります。すべての既存機能をモデル化するのが難しい場合、追加・変更を受けやすいホットスポットから行うとよいでしょう。

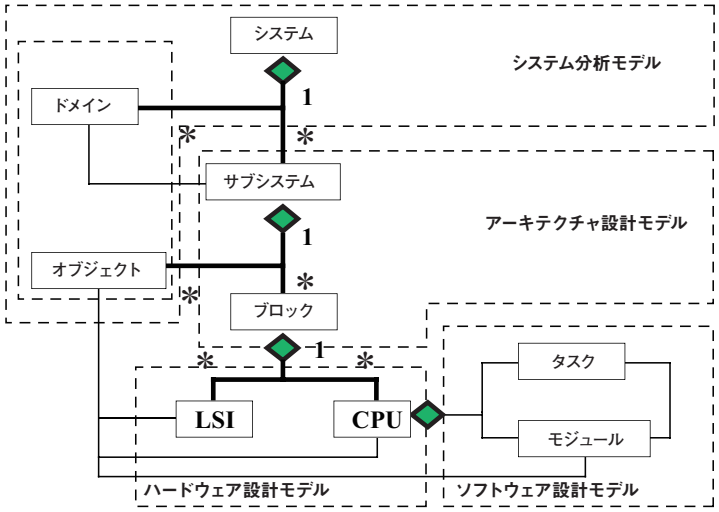
2-6　リソース制約とモデリング

設計モデリングではリソース制約の考慮も必要

組込みソフトウェアではメモリをはじめ様々なリソース面で厳しい制約があるシステムが少なくありません。こうしたリソース制約の厳しいシステムでは、アセンブラ言語を駆使して 1 バイトでも減らしたり、変数をレジスタ変数に割り当てるなど実装の段階で効率化を図るといった工夫が施されます。しかし、これらは“できばえの品質”の次元であり、リソース制約を実現する上で本質的な解にはなり得ません。本来、システムに求められるリソース制約の実現を意識するのであれば、実装以前の早い段階――即ち設計モデリングの時点からリソースの最適化などを意識することが求められます。

ここでいう設計モデリング段階でのリソース最適化とは、システム分析

図 2-6　組込みソフトウェア向け開発環境（静的モデル）



情報処理　2004.1（Vol.45 No.1）：渡辺政彦 「組込みソフトウェア」向け開発支援環境

モデルで与えられた制約を守れる最適なアーキテクチャモデルを設計することと考えても同じです。例えば自動車では、ECU を車載 LAN 上のどこに配置し、さらにその ECU にどの機能を配置するかを考え、最適なサブシステムを設計するためにアーキテクチャモデル化が行われ、効率化が検証されます。またデジタル家電では、HDD、DVC、ビデオのそれぞれのユニットに対する最適なメモリ構成、LSI 構成、バス構成といったアーキテクチャ設計の良し悪しが性能、コストに直結します。

このように、リソース制約が厳しい組込みシステムの場合、アーキテクチャ設計モデリングの段階からそれらを意識した設計が必須であり、これらを抜かして下流工程、例えば実装段階でリカバリーすることは、必ずしも良い作り方ではありません。

3　モデリング技術のマップ

3-1　マップの概要

設計対象によって利用するモデリング技術は異なる

モデリング技術には様々なものがあります。通常、設計対象とするシステムの種類によって使われるモデリング技術は異なります。ここではそれらの全体像を捉えるために、そのモデリング技術の持つ技術面の特性を示した後、ソフトウェア開発のフェーズに応じてその使われ方を示します。

なお色々あるモデリング技術ですが、本書では離散システム、典型的には状態マシンとして捉えられるシステムなどに対象を絞って解説していきます。組込みシステムでは、オープン／クローズドループ制御システムなどもありますが、本書では取り上げていません。

3-2　モデリング技術の特性

モデリングでは機能・構造と振る舞いを中心に考える

一般にソフトウェアをモデリングする際の対象は、以下の 3 種類に分類できます。

- **機能**：入力と出力の関係によって表すのが典型的です。即ち与えられた入力に対して、どのような出力を出すのかという関係によって表現します。

表 3-1　モデリング技術とそのモデル化対象

モデル化対象	モデリング技術
機　能	ユースケース図、データフロー図
構　造	クラス図、コンポーネント図、オブジェクト図、合成構造図、配置図、パッケージ図
振る舞い	アクティビティ図、シーケンス図、コミュニケーション図、インタラクション概観図 タイミング図、ステート機械図、状態遷移表、SDLプロセス図

- **構造**：主要な概念（あるいは情報、データ）、付属的な概念、概念間の関係によって表現します。
- **振る舞い**：与えられた入力イベントと、内部状態に対して、どのような出力イベントを出すのか、その関係によって表現します。

表3-1 は、それぞれの対象にとって主要なモデリング技術が何かを示したものです。

例示と一般化――利用する図を使い分ける

また、モデリングの際には、それが一般性を持った記述なのか、例示的な記述なのかを意識することが重要な場合があります。

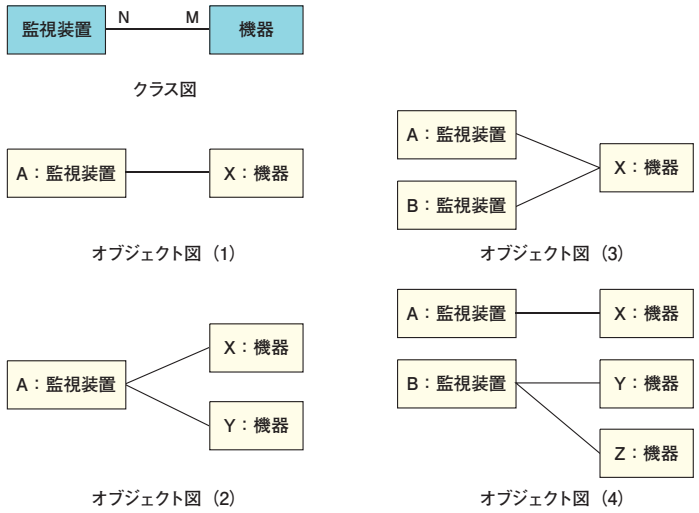
■一般的な記述

一般的な記述とは、クラス図や状態遷移図のようなモデルで表したものです。一般的な記述であれば多くの状況を 1 つのモデルにコンパクトに表現することができますが、その記述や理解はそれだけ難しくなります。

■例示的な記述

例示的な記述とは、オブジェクト図やシーケンス図のようなモデルで示すものです。例示的な記述では、その記述内容を理解してもらうことが相対的に容易となりますが、あくまで特定の状況の例示であること、そして一般に例示をいくつ集めてもすべての状況を書き尽くすことはできないことを理解しておく必要があります。

図 3-1　一般的な記述と例示的な記述



例えば、図3-1 に示すクラス図はその下に示す様々なオブジェクト図のすべての状況をコンパクトにまとめたものですが、このクラス図だけを見てそのような様々な状況が含まれていることを理解するのには、経験が必要です。逆にそれぞれのオブジェクト図は容易に理解できますが、こうしたオブジェクト図をいくつ並べてもクラス図が表せるすべてのオブジェクトの構造を書き尽くすことはできません。

3-3　利用されるフェーズによるマップ

フェーズに応じて利用するモデリング技術を選択する

ソフトウェア開発において、モデリング技術がどのように使われるかを概観したものが、P.20 の表 3-2 です。ここでは、ソフトウェア開発のフェーズの中から、分析、アーキテクチャ設計、実装設計の 3 つのフェーズを取り上げ、それらの作業とモデリング技術との対応を示しています。

表 3-2　フェーズとモデリング技術の対応

フェーズ	分析	アーキテクチャ設計	実装設計
作業の内容	概念整理 仕様定義	方式検討 基本構造定義 製品ファミリー定義	実装技術依存の詳細な設計（タスク設計等）
モデリングの目的	理解の基盤づくり 顧客とのコミュニケーション 仕様の明確化	基本構造の妥当性確認 再利用設計の妥当性確認	実装方式の妥当性確認
主に使われる図法	ユースケース		
	その他の図法		
モデリング上の特徴	ユースケースの活用 論理構造の作成	シナリオベース	厳密性・網羅性
典型的な検証手法	レビュー プロトタイプ	レビュー シミュレーション スコアリング 形式検証	レビュー シミュレーション 形式検証

この表からわかるように、一部を除き、ほとんどのモデリング技術は複数のフェーズに活用することができます。つまり、フェーズによるモデリング技術の違いは、図法（図の書き方）の違いではなく、何をモデル化するかという、表現する内容の違いなのです。

要求分析はモデリングの出発点

■このフェーズでのモデリングの主な役割

要求分析は設計ではありませんが、設計モデリングに繋がる重要なフェーズで、ここでのモデリングは設計に繋がる重要な役割を果たします。一般に分析においては、要求を整理し、それに基づき作るものを明確にしていきます。その過程で、対象ドメインの概念、システムに使われるコンテキスト、システムの仕様などをモデル化することになります。

ここで作られるモデルは、ソフトウェアの実現構造というよりはソフトウェアの扱う概念の論理的な構造を示すものといえます。こうしたモデルを作成することで、開発者相互の理解の基盤となるだけでなく、顧客とのコミュニケーションにも役立てることができます。従来こうした基本的な概念は、当然わかっていることとして明確に定義・整理されないまま開発が進められることも多かったのですが、実際には人によって異なった意味で理解していたり、その定義がずれていたりすることが多く、それが開発上の思わぬ間違いを引き起こすこともまれではありません。

■利用される代表的なモデリング技術

こうした分析フェーズで特徴的に使われるモデリング技術としてはユースケース図があります。他のモデリング技術も使われますが、全般的な傾向として、それらは網羅的・一般的なモデルより、重要なポイントを概略的に示したり例示したりするように利用されることが多いといえます。

分析フェーズでの検証は、主としてレビューやプロトタイピングが使われます。このフェーズでのモデルは、対象ドメインの概念などその検証には意味的な吟味が必要なことが多く、レビューがよく使われます。また、プロトタイピングは顧客とのコミュニケーションなどに有効です。

アーキテクチャ設計は設計モデリングの中心

■このフェーズでのモデリングの主な役割

組込みシステムの設計においては、実装の詳細や細部を考える前に、処理の方式や全体を支配する基本的な構造を明確にする必要があります。こうしたアーキテクチャ設計は、開発されるソフトウェアの様々な特性を決定づけるだけでなく、作業分担を含めるなど、ソフトウェア開発の方法も決定づけます。従って、こうした方式や構造をモデル化し、その妥当性を確認することは重要となります。

ここで作られるモデルはあくまで実現のための構造ですが、過度に詳細に立ち入った設計ではなく、骨格となる重要ポイントを描いたものとなります。必要であれば方式議論のために局所的に詳細な設計が含まれる場合もありますが、一様に詳細化されることはありません。ここでの目的は重要な判断の明示化であるため、すべてを詳細化するとそれが読み取れないからです。

この段階でのモデリングのもう１つの目的は、再利用性の設計です。様々なソフトウェアに使われるソフトウェア構造とするために、共通的に使われる部分や、再利用するソフトウェアごとで異なる部分を明示的にモデル化することにより、再利用性の高い構造を検討することができます。

■利用される代表的なモデリング技術

アーキテクチャ設計フェーズだけで使われるモデリング技術というものは特にありませんが、それらの使われ方の特徴の１つは重要なシナリオに照らした記述が活用されるということです。つまり、この段階ではすべての機能を詳細に検討しませんが、起動、終了、エラー発生時など、アーキテクチャ設計上重要な側面について十分に配慮する必要があり、そうした側面に照らしてモデル化するといった方法が使われます。

このフェーズでの検証手法としては、レビュー、シミュレーション、スコアリング、形式検証などが考えられます。アーキテクチャの妥当性の確認には経験者の発見的な視点が必要不可欠であり、レビューはきわめて重要です。また、シミュレーションは方式の妥当性などの確認に有効です。性能、信頼性、変更容易性などの品質特性の検討においてはスコアリングも有効になります。さらに、今後は形式検証を用いて、重要なポイントに絞って網羅的、厳密な確認を行う方法なども有効になると考えられます。

実装設計では実装を意識したより厳密なモデリングを

■このフェーズでのモデリングの主な役割

プログラムの実装を前にした実装設計では実装環境に依存した詳細な設計を行います。ここでは細部にわたって正確かつモレのない設計を行う必要があります。即ち、実装上の判断を間違いのない形で明確にモデル化することが重要です。

このフェーズでは実装する技術に依存したモデルを使うことが有効となります。例えばリアルタイムOSを使うなら、タスクやメールボックスなどの実現基盤上のメカニズムをステレオタイプでモデル要素と対応付けるなどします。

■利用される代表的なモデリング技術

このフェーズでのモデルの特徴は厳密的かつ網羅的であることです。そのためモデリング技術としては、例えば状態遷移図よりも状態遷移表のような図法の方が使いやすい傾向があります。

検証には、レビューやシミュレーション、さらに今後は形式検証なども有効になると考えられます。

4 モデリング技術概説

本章では、組込みソフトウェア開発の中で利用されている各種モデリング技術の概要を説明します。

組込みソフトウェアも含めたソフトウェア全般のモデリング技術としてUMLが広まっていますが、組込みソフトウェアではさらに状態遷移表及びSDLなどが使用されています。よって、ここではUML2.0の13種類の図、状態遷移表、SDLの2種類の図、そしてデータフロー図の計17の技術について取り上げることとします。特に、これらがどの品質特性に寄与するのかを述べていきます。

各モデリング技術の解説ではそれぞれの特徴をわかりやすくするために、すべて同一の飛行船制御システム（下記参照）をサンプルモデルとして利用しています。

サンプルモデルのシステム概要

このシステムは無人飛行船を制御するための地上施設で、情報処理学会ソフトウェア工学研究会主催の「組込みソフトウェアシンポジウム2005」で開催された「MDDロボットチャレンジ」などで取り上げているシステムの一部です。

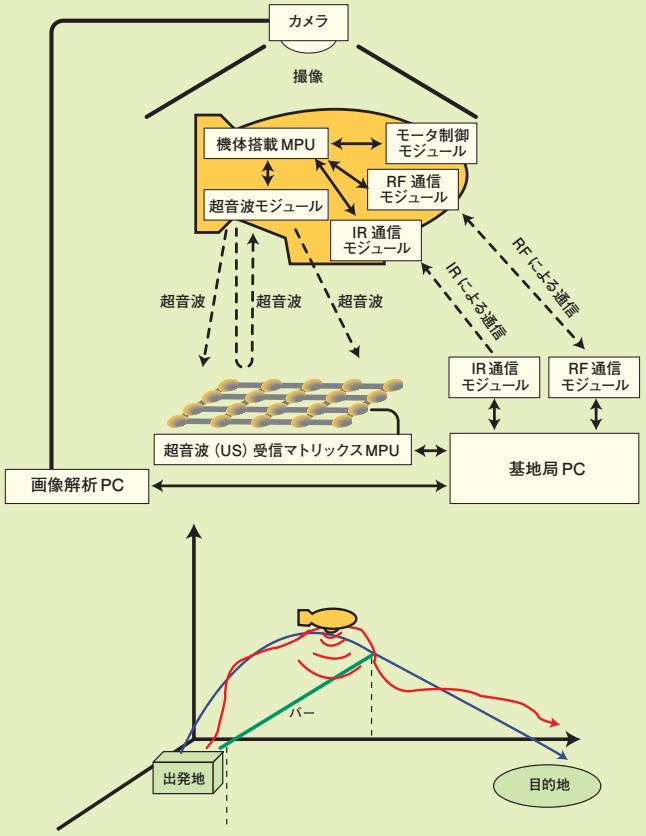
この地上施設には、メインの基地局PCと無人飛行船と無線、赤外線、超音波による通信モジュールが接続されています。基地局PC内のコースデータに従って無人飛行船を制御するシステムとなっています。

なお、大会では無人飛行船本体のシステムも対象としていましたが、ここでは簡略化のため地上施設を中心に取り上げます。また、このサンプルのモデルは、大会に出場した「ねこねこ専FU」チームのご協力を得て、同チームのモデルをベースにしました。同チームはモデリング大賞を受賞していますので、モデルの内容も適切なものといえるでしょう。ただし、ここでは各技術の説明が目的ですので、そのために多少の変更を加えていることはあらかじめご了承ください。

サンプルモデルのシナリオ

A地点からB地点を飛行する飛行船を、無線（RF）、赤外線（IR）、超音波（US）といった通信技術を利用して制御します。会場上空には飛行船を撮像するカメラが設置され、画像解析技術によって飛行状況を調べることができるようになっています。基地局PCは、各通信モジュールによって得られる情報と画像解析結果に基づき、飛行船に対して制御命令を発信します。参加者は、基地局PC、機体搭載MPU、画像解析PCの搭載プログラムを開発します。このプログラム開発のために、モデル設計を行うこととします。

図4-1　サンプルモデルのシナリオイメージ



※ UMLはOMG、SDLはITU-TのZ.100にて標準化されています。

4-1　クラス図　Class Diagram

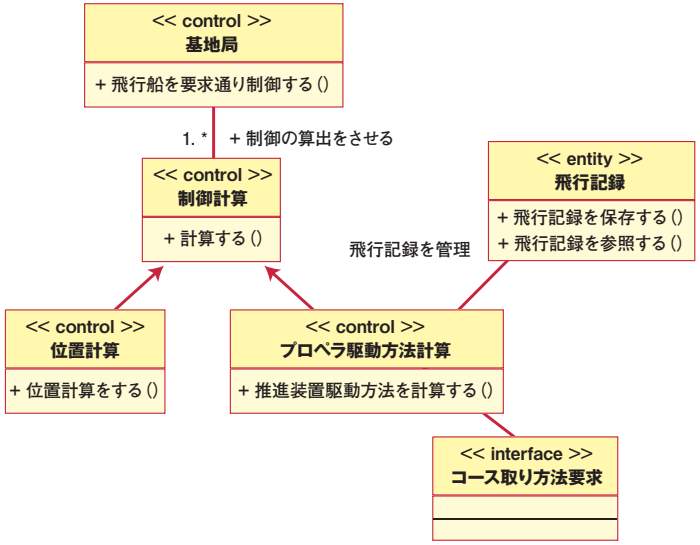
クラス図はシステムの構造整理に利用する

クラス図は、クラス、インタフェイス、パッケージ等のシステムの構成要素と、それら構成要素間の関係により、システムの構造を示す図です。クラス図は、分析から実装設計にわたる広範囲なフェーズにおいて用いることができます。上流工程では対象世界の大まかな構造や論理的な構造等を、下流工程では具体的なクラス構造等を示すために用いることができます。

クラス図のサンプルを図 4-2 に示します。

図 4-2 は、飛行船制御システムの基地局の設計フェーズにおけるクラス図の一部です。「基地局」、「位置計算」等の制御に関するクラス（ステレオタイプ <<control>>）は、「コース取り方法要求」等のインタフェイスとなるクラス（ステレオタイプ <<interface>>）、実体を示すクラス（ステレオタイプ <<entity>>）である「飛行記録」等から構成されることが示されています。このように、ステレオタイプを用いて、クラスの種別を示すことがあります。また、クラス図ではクラス間の関係についても示されます。例えば「プロペラ駆動方法計算」と「飛行記録」の間の関係において、「飛行記録」側は「飛行記録を管理する」という役割（ロール）を持っていることが示されます。なお、「位置計算」及び「プロペラ駆動方法計算」は、「制御計算」から継承したクラスです。

図 4-2　クラス図



クラス図は構造面の見通しをよくする

クラス図を用いてシステムの構造を明示化することにより、良いモジュール構造（典型的にはクラス構造）を導くことができるようになります。

モデリングにおいては、論理的に本来持つべき構造を明らかにすることは非常に重要です。そのような論理的な構造はクラス図を用いて表現することができます。こうした論理的な構造をもとに、例えば実行時のタスク分割をどうするか、開発時のファイル構造をどうするかといったアーキテクチャが検討されます。そうした側面のモデル化にもクラス図を用いることができます。なお、実行時の構造は性能等に、開発時の構造は再利用性や修正容易性に、それぞれ影響を与えます。

記述するクラスの粒度を揃えるのがコツ

このように、クラス図は開発の様々なフェーズで用いることができ、かつ様々な視点におけるモジュール構造を表現することができるものですが、その記述にあたっては、どの粒度のものを書くのか、どの視点からの構造を書くのかを明確にすることが大切です。1 つの図の中で、粒度や視点が揃っていないものが混在することは避けなければなりません。

4-2　オブジェクト図　Object Diagram

オブジェクト図はシステムの状態のスナップショットを示す

オブジェクト図は、クラスの実体であるオブジェクトと、それらのオブジェクト間のリンクを示す図です。オブジェクト図は、クラス図の１インスタンスであり、ある時点におけるシステム状態のスナップショットを示します。従って、クラス図を使う時に一緒に利用することができます。

オブジェクト図のサンプルを図 4-3 に示します。

図 4-3　オブジェクト図

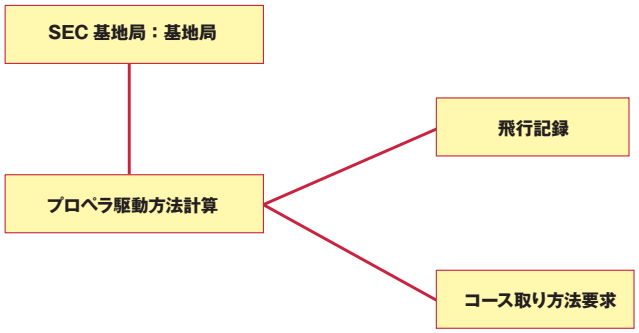


図 4-3 はオブジェクト図の一例で、「基地局」クラスのインスタンス「SEC 基地局」、「プロペラ駆動方法計算」クラスのインスタンス等で構成されることが示されています。図中のオブジェクト名は省略することもできます。図 4-3 では、「基地局」クラスのインスタンス以外、オブジェクト名をすべて省略しています。また、各オブジェクト間の線はリンクを示します。リンクとは、関連のインスタンスのことです。

オブジェクト図はクラス図の理解を助けるのに有効

オブジェクト図は、クラス図同様、システムの構造を明示化するために用いられるので、良いモジュール構造（典型的にはクラス構造）を導くことができます。具体的なインスタンスを例示することにより、クラス図が示す構造を理解しやすくするので、その構造を詳細に検討する際の一助となります。

また、あるスナップショットにおいて具体的に何個のオブジェクトが生成されているのか、どのオブジェクトとどのオブジェクトが関係を持つのかといったことを明らかにすることができるため、性能やスケーラビリティ等の検討にも役立ちます。

【コラム】 凝集度と結合度

凝集度とは、クラスやパッケージ内の機能要素と情報要素の関連性の強さを表す指標です。相互に関連する機能や情報が分散していると、仕様変更時の影響範囲が広くなり、保守性が低下します。そこで、これらの機能や情報を局所化するように設計し、相互に関連する機能や情報は閉じたモジュール内に収めるようにすると、凝集度は高くなり、関連性の高い機能が細分化されます。

結合度とは、呼び出し関係にあるメソッドの結び付きの強さを表す指標です。複数のクラスやパッケージ間の依存度が上がると、仕様変更やメンテナンスの対応が難しくなり、保守性が低下します。複数のクラスやパッケージ間の依存度を下げ、結合度を低くしていくと、プログラムはよりシンプルなものとなります。

凝集度・結合度の改善を行っていくと、理解しやすい構造→保守性の向上、シンプルなプログラム構造→信頼性の向上に繋がります。

4-3　コンポーネント図　Component Diagram

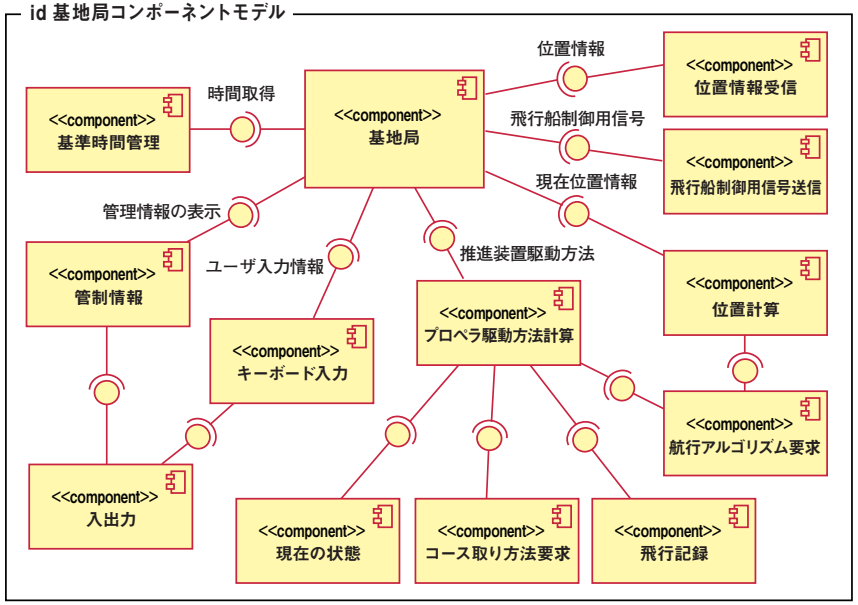
コンポーネント図はコンポーネント間の関係を整理する

コンポーネント図はシステムのモジュール化された部品であるコンポーネントの構造や依存関係を表すのに用います。コンポーネント図は設計、実装フェーズにおいてシステムのソフトウェア構成及びシステムの振る舞いを把握するのに適しています。また、コンポーネント図によりシステムに必要な機能がインプリメントされているかどうかの検証がしやすくなります。

コンポーネント図のサンプルを図4-4に示します。

図4-4は、「基地局」のコンポーネント図で、基地局におけるコンポーネントの構成とコンポーネント間のインタフェイスについて示しています。

図4-4　コンポーネント図



コンポーネントの振る舞いは、図4-5のようにコンポーネントが提供／供給するインタフェイスにより定義します。インタフェイスの提供は、そのコンポーネントが外部に振る舞いを公開することを、インタフェイスの要求は別のコンポーネントが提供するインタフェイスに依存することを、それぞれ示します。基地局コンポーネントと位置情報受信コンポーネントの関係では、基地局コンポーネントが要求する位置情報を位置情報受信コンポーネントが提供していることを示しています。

図4-5　コンポーネントのインタフェイス

コンポーネント図でシステムのパッチワークが進む

コンポーネントは設計上において複数のクラスやオブジェクトを大きな単位でカプセル化したものであり、外部に対する振る舞いはインタフェイスで明確に定義されて再利用や代替可能なモジュールとして作成されます。これによりシステムを分離可能なコンポーネントで構成し、システム上の振る舞いを局所化することで設計品質を高めることができます。

4-4　合成構造図　Composite Structure Diagram

合成構造図はサブシステムなどの内部構造を表すのに有効

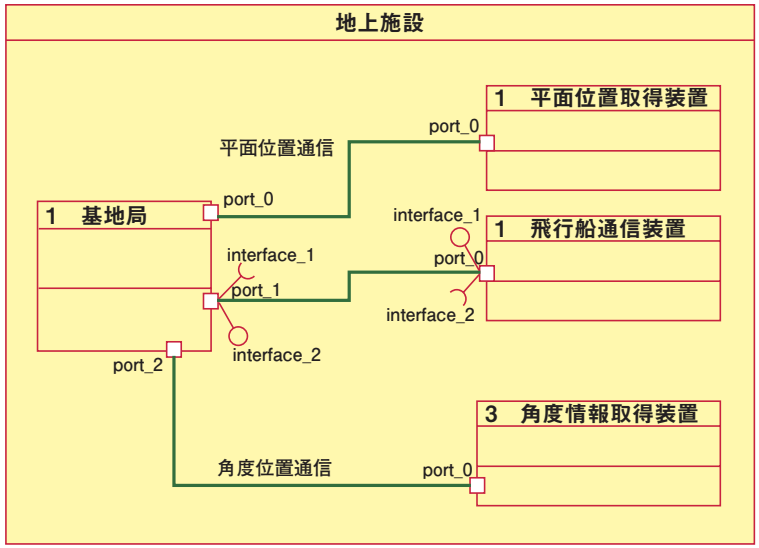
合成構造図は、あるモジュールの内部構造を表すために用いられます。上流工程ではシステム全体の大まかな構成を、下流工程ではあるサブシステム内部の構造を示すのに用いることができます。

あるサブシステムが外部とやり取りするための出入り口、及びサブシステム間でその出入り口同士をどう接続するのかを示すこともできます。また、その経路ごとでやり取りするイベントや公開される関数を、インタフェイスクラスを用いて指定することができます。

合成構造図のサンプルを以下の図 4-6 に示します。

図 4-6 では、「地上施設」というサブシステムを、「基地局」「平面位置取得装置」など 4 つのサブシステムに分割していることを示しています。

図 4-6　合成構造図



そして、「基地局」と「飛行船通信装置」の間には「飛行船通信」コネクタという経路があり、「基地局」から「Interface_1」、「飛行船通信装置」からは「Interface_2」という、それぞれインタフェイスクラスで決められたイベントがやり取りされることが示されています。

これらのサブシステムの分割は階層化することができます。例えば、図 4-6 中の「基地局」ブロックの中はさらにいくつかのサブシステムに分割できます。

各サブシステムは多重化することもできます。多重化とは同一の振る舞いを持つインスタンスを複数存在させるということです。図 4-6 中の「角度情報取得装置」の場合、3 つ存在することが示されています。

サブシステム間のインタフェイスが明確になる

合成構造図を利用すると、サブシステムをいくつ作るのか、そしてどのサブシステムとどのサブシステムがやり取りするのかが明確になります。

即ち、合成構造図では各サブシステムがやり取りできるイベントの種類がインタフェイスクラスで明示されるため、各サブシステムの使用性（Usability）が向上します。例えば、ブロック A では受けられないイベント E をブロック B から送ろうとしているなど、イベントの利用における矛盾をチェックできます。このように、合成構造図を使うとサブシステム間の境界が明確になり、イベントの種類までも特定できるため、変更による影響の範囲が把握でき、保守性も向上します。

特に P.50 の図 4-15（状態遷移図）と組み合わせることで、サブシステムに分解され、動作の状態遷移が明確になり、オブジェクト指向にとらわれずに開発することができます。

4-5　配置図　Deployment Diagram

配置図ではハード／ソフトの関係を明らかにする

配置図はシステムの物理的なソースにソフトウェアをどのように配置するかということを表した図です。PCやデバイス等の物理的なソースはノードで表し、ノードに配置されるソフトウェアは成果物として表現します。ノード間の通信等、ノード間の関係を表す場合はコミュニケーションパスとして実線でノード間を結んで表記します。配置図を使用することにより、ハードウェアとソフトウェアの構成及びノード間の通信を明確に把握することができ、ハードウェアとソフトウェアの組み合わせによるシステムの動作について検討することができます。

配置図のサンプルを図4-7に示します。

図 4-7　配置図

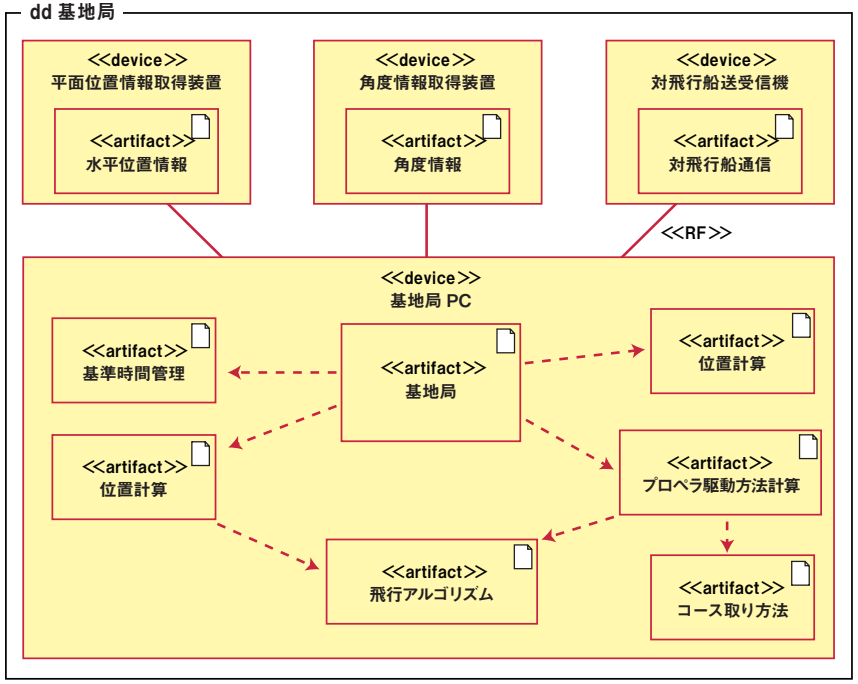


図4-7は、「基地局」の配置図を示したものです。基地局PC、平面位置情報取得装置、角度情報取得装置、対飛行船送受信機の4つのノードと、各ノードに配置されているソフトウェアが成果物として示されています。基地局PCのノードには7個の成果物（ソフトウェア）が配置されていることがわかります。また、基地局PCと平面位置情報取得装置、角度情報取得装置はコミュニケーションパスで接続され、基地局PCと対飛行船送受信のコミュニケーションパスにはステレオタイプ<<RF>>が付けられてRF通信が行われていることが示されています。

配置図をシステム構成ユニット間の関係を再整理できる

配置図により物理的なノードに配置されるソフトウェア及びノード間の通信を明確に表すことができます。これにより、設置性や置換性など保守性ならびに通信のセキュリティに関する検討を行うことができます。そして、システムを構成するハードウェアユニットやソフトウェアを構成するソフトウェアユニット（サブシステムなど）の関係を再整理して考えることができるようになります。

4-6　パッケージ図　Package Diagram

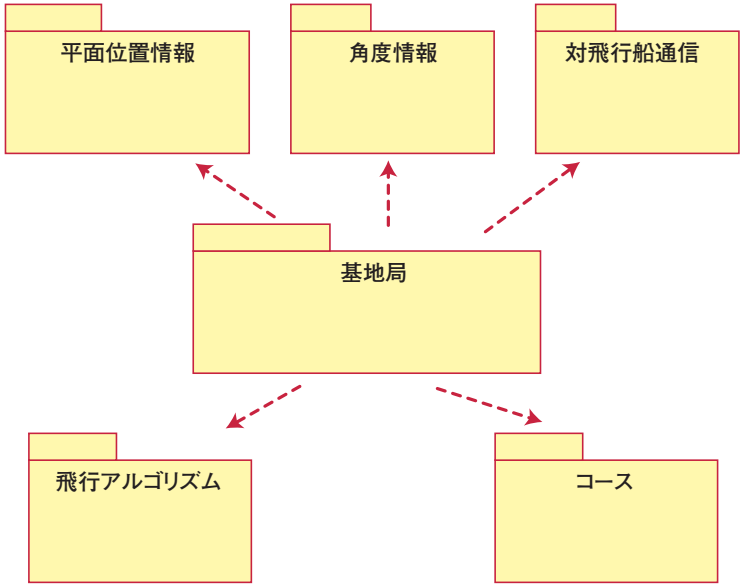
管理単位を意識し、パッケージとして考える

パッケージ図は、クラスなどのモデル要素をグループ分けした単位であるパッケージの相互関係を表します。パッケージは、モデル要素をグループ分けして管理する単位で、名前空間としての役割も持ちます。また、パッケージの内部に入れ子になったパッケージを定義することもできます。パッケージを用いてモデルを分割することによって、モデルを理解してもらうことや管理することが容易になります。

パッケージ図のサンプルを図 4-8 に示します。

図 4-8 は、「平面位置情報」「角度情報」「対飛行船通信」「基地局」「飛行アルゴリズム」「コース」の 6 つのパッケージがあることを表しています。

図 4-8　パッケージ図



パッケージ間を結ぶ点線の矢印は依存関係を表しています。従って、図 4-8 では「基地局」パッケージが「平面位置情報」「角度情報」「対飛行船通信」「飛行アルゴリズム」「コース」の各パッケージに依存していることが表されています。

パッケージ単位での管理や再利用などを考えやすくする

大きなモデルであっても、パッケージを用いて扱いやすい大きさに分割することによって、開発者が一度に把握しなければならない範囲を限定することができます。また、パッケージ図を用いることによって、詳細にとらわれずに全体像を理解することが容易になります。こうして保守性を向上させることができます。

パッケージを用いてモデル要素を意味のあるまとまりでグループ化することによって、パッケージ内のモデル要素間の結合は密になる一方で、パッケージをまたぐモデル要素間の結合はおろそかになります。その結果、設計されるソフトウェアの凝集度と結合度は改善され、機能の追加や変更が容易で不具合を起こしにくいソフトウェア構造を築けるようになります。

ある特定の機能を実現するために必要なモデル要素一式の場合、1 つのパッケージにまとめて管理することによってパッケージを単位としたソフトウェアを再利用することが可能になります。また、特定の言語処理系やハードウェアに依存した処理をソフトウェアの他の部分から分離してパッケージにまとめることにより、移植性の高いソフトウェア構造を築くことができます。パッケージは名前空間でもあるため、異なるパッケージ内に同じ名前を持つモデル要素があっても構いません。これにより、機能的な独立性と併せて、パッケージごとに分担して作業を進めるといったチーム開発が容易になります。

4-7　アクティビティ図　Activity Diagram

アクティビティ図でシステムの処理手順整理を

アクティビティ図とは、フローチャートに類似した表記法で、処理の実行手順を表すために用いられます。開発上流の要求把握・分析工程では、作業や操作、システムの動作などの手順をモデル化するために用いることができます。また、下流の設計工程ではソフトウェアの処理手順のモデリングに用いることもできます。

アクティビティ図のサンプルを以下の図 4-9 に示します。

図 4-9　アクティビティ図

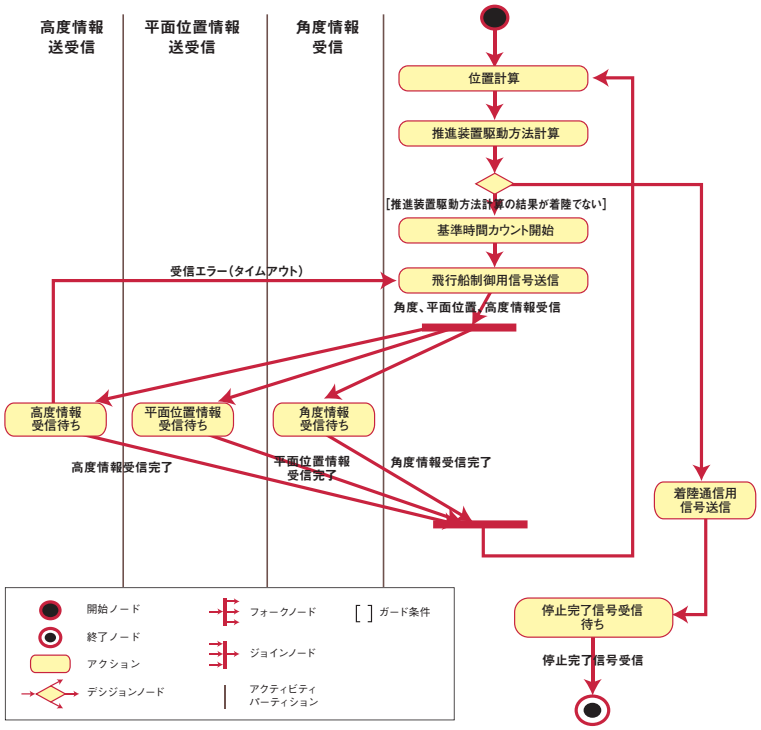


図 4-9 の例では、一連の処理は開始ノードから始まり、終了ノードで完了します。開始ノードから出発する一連の矢印をコントロールフローといい、制御の流れを表します。

アクションはひとまとまりの処理を表します。あるアクションが完了すると、コントロールフローに従って次のアクションの処理に移ります。条件によって異なる処理をする場合は、デシジョンノードを用いて 1 つのフローを複数のフローに分岐させます。分岐するそれぞれのコントロールフローにはガード条件を指定します。また、複数の処理が並行に進むときは、フォークノードを用いてフローを分割します。並行に進む処理の完了を待って 1 つのフローに集約するときは、ジョインノードを用います。

その他、アクティビティパーティションを用いて一連のアクションを実行する際の役割分担を表現できます。図 4-9 では「高度情報送受信」「平面位置情報送受信」「角度情報受信」の各パーティションにアクションを割り振っています。

ユーザ操作を意識して具体的なイメージ獲得が可能に

システムの振る舞いをアクティビティ図でモデル化すれば、機能が明確化され、その改良も容易となるため、機能性が向上します。形式的な図の表記を用いるので、システムの複雑な動作をわかりやすく、誤解しにくい形で整理することが可能になり、要求を正確に反映したシステムを開発することができます。

ユーザによる操作の手順など、HMI 仕様をアクティビティ図でモデル化すると、ユーザインタフェイスの機能が明確化され、使用性が向上します。分岐や並行性を持つような複雑な操作手順であっても、具体的なイメージがわかるように表現することによって、より使いやすく改良することができます。

4-8　シーケンス図　Sequence Diagram

システムの振る舞いを視覚的に整理して表現

シーケンス図は相互作用図の１つで、オブジェクト間のメッセージのやり取りを時系列に沿って並べて表現するために用いられます。シーケンス図ではメッセージのやり取りを時系列で表すため、シナリオに対応させて個々のトランザクションの変遷を示したり、イベント発生順序を視覚的に表現したりすることができます。

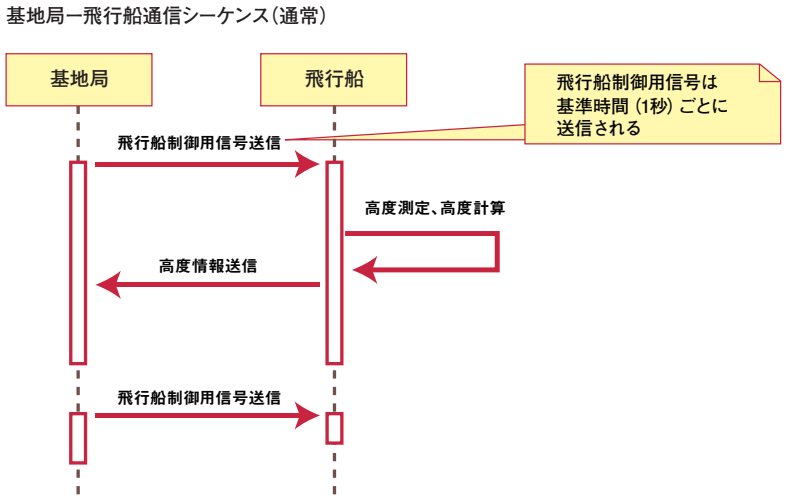
シーケンス図で使用する主なモデル要素は、メッセージ、活性化区間、生存線（ライフライン）です。

メッセージはオブジェクト（▭）間のやり取り（相互作用）で、生存線（┆）間の矢印（→）として表します。活性化区間はそのオブジェクトに制御が移っている期間を表し、細長い四角形（▯）で記述します。生存線はそのオブジェクトが生成されてから消滅するまでの期間を表します。生存線はオブジェクトの下に点線を引くことで記述します。ちなみに、オブジェクトは四角形の中に「オブジェクト名:クラス名」として記述します。なお、オブジェクト名とクラス名はどちらかを省略することが可能です。

シーケンス図のサンプルを図 4-10 に示します。

図 4-10 は、基地局と飛行船間の制御に関わる基本的な通信のシーケンスを示しています。クラスオブジェクトとしては、基地局クラスと飛行船クラスの２種類が記述されています。基地局から飛行船への航行制御は、図中のコメントシンボルに記述されているように、定期的（1 秒ごと）に、基地局から飛行船の方に「飛行船制御用信号送信」メッセージを送信して行われます。このモデルの飛行船は高度測定の機能を有しており、「飛行船制御用信号送信」メッセージを受信することにより、高度測定、高度計算の処理を実行し、その結果を「高度情報送信」メッセージとして基地局に返信します。

図 4-10　シーケンス図



システムの振る舞いを追うことで設計のモレを減らす

シーケンス図の大きな特徴は、誰でも容易に記述でき、かつ理解する側にとってわかりやすい図であることが挙げられます。従って、非常に使用性・保守性の高いダイヤグラムの１つといえます。

シーケンス図の用途の１つに、設計の初期段階でユースケースに従ったシステム全体の振る舞いを把握することがあります。この場合、アクターとシステム間、あるいはシステム内コンポーネント間との相互作用の概要を理解するのに非常に有効的と考えられます。

またシーケンス図では、システムを構成する個々のオブジェクトについて、メッセージによるオブジェクト間の同期的な呼び出しや非同期の呼び出し、生成や削除といった事象を視覚化できるので、設計の詳細化はもとより設計レビュー等においても、設計の把握、モレ・ヌケの確認等で効果的に使うことができます。

4-9　コミュニケーション図　Communication Diagram

システムを構成するオブジェクト間のやり取りに注目

コミュニケーション図は相互作用図の１つで、メッセージの送受信を行うオブジェクト間の構造的な接続関係に着目して、その間のメッセージやデータの流れを表す場合に用いられます。UML1.X ではコラボレーション図と呼ばれていました。

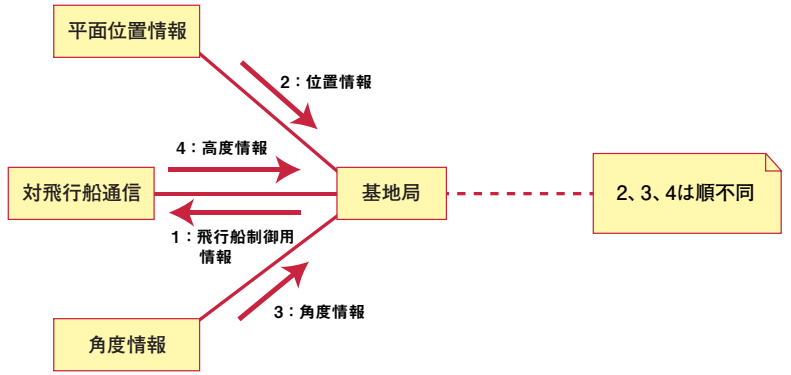
コミュニケーション図では、オブジェクト名にアンダーラインの付いたシンボルでオブジェクトを表現し、それらを関連リンクで結合して、各オブジェクトの参照・接続関係を示します。各リンク上を流れるメッセージは、リンクに付加されたラベル付きの矢印で表されます。各メッセージは、シーケンス番号、ガード条件、返り値の名前、メソッド名、パラメータのリストで構成されます。

シーケンス番号の表記は、番号または文字を小数点で区切って階層化した表記を用います。ただし、「1.1.1.2.1.1」のような階層が深くなったシーケンス番号になると、かえってわかりづらくなるので注意が必要です。また、番号の前に「メッセージ A5」と「B4」というように文字を用いて、それぞれが異なるスレッドにあることを示すことができます。この場合、同じスレッド内のすべてのメッセージは、逐次的に順序付けされます。異なるスレッド間のメッセージは、この順序付けについて明示的な依存関係がない限り、同時に発生し得ることとなります。

コミュニケーション図のサンプルを図 4-11 に示します。

図 4-11 は、地上施設を構成する個々のパッケージ間の相互作用を表しています。「基地局」オブジェクトから、「対飛行船通信」オブジェクトに対して、シーケンス番号1「飛行船制御用情報」メッセージを送信します。その後、「平面位置情報」オブジェクト、「角度情報」オブジェクト及び「対飛行船通信」オブジェクトから、非同期に、シーケンス番号２「位置情報」

図 4-11　コミュニケーション図



メッセージ、シーケンス番号３「角度情報」メッセージ及びシーケンス番号４「高度情報」メッセージが「基地局」オブジェクトに送信されます。その収集された情報を元にデータ解析を行います。シーケンス番号 2、3、4 が非同期であることは、コメントシンボルを用いて記述します。

実装に近いイメージでオブジェクト間の連携を確認できる

コミュニケーション図では、操作や機能配備に関してその実装方法をイメージしてオブジェクト間の相互作用を記述できるため、可読性が向上し、設計レビュー時等において設計品質の向上に役立てることができます。

4-10　インタラクション概観図　Interaction Overview Diagram

システム内の制御の流れを示す

インタラクション概観図は相互作用図の１つで、アクティビティ図のバリエーションとしてシステムやビジネスプロセス内の制御フローの概要を示す場合に用いられます。インタラクション概観図に記述されるインタラクション・フレーム内には、シーケンス図で記述される具体的なメッセージや生存線は表示されません。用途は、あくまでもインタラクション・フレームを使用して制御の流れの概観を示すことにあります。従って、最も簡単なインタラクション概観図は参照シンボルだけで記述できるかもしれません。

インタラクション概観図のサンプルを図4-12に示します。

図 4-12　インタラクション概観図

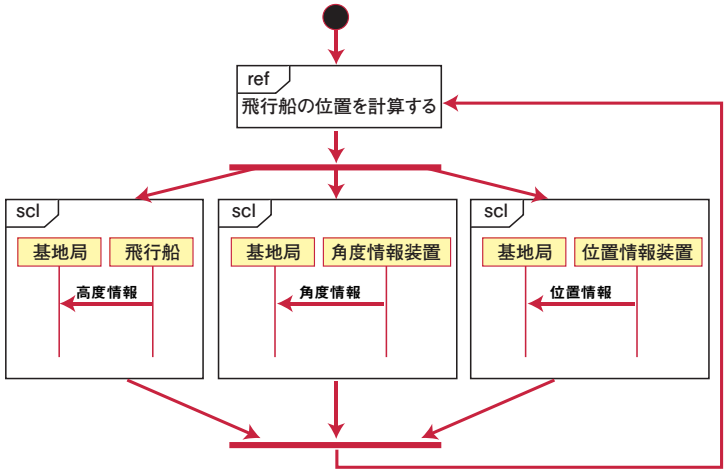


図4-12は、地上施設内で行われる位置情報の制御フローを記述しています。地上施設における位置情報は、飛行船から取得される高度情報、角度情報取得装置から得られる角度情報及び位置情報装置から得られる平面位置情報の３種類の情報を繰り返し取得して、処理されています。図中のインタラクション・フレームワーク内に示されているオブジェクトは、システムを構成する抽象的なオブジェクトとなっています。

複雑なシステムの概観を捉えるのに有効

インタラクション概観図は、表記の規則もシンプルで、その用途も制御フローの概要記述に用いると明確になりますので、その範囲において用いる分には、設計の使用性、保守性を高めるために有効と考えられます。システムが複雑で、個々の制御のフローの記述が煩雑な場合に、全体の概観を把握するための補足用に用いることが効果的と考えられます。

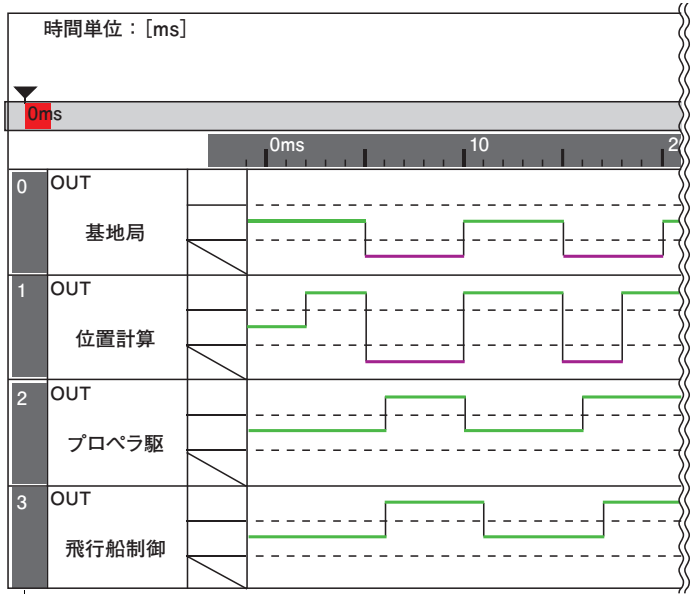
4-11　タイミング図　Timing Diagram

オブジェクト間の実行タイミングを意識する場合に利用

組込み系アプリケーションなどのリアルタイム処理システムでは、業務系アプリケーションなどよりも厳密なタイミングで制御することが要求されます。タイミング図ではタイミングルーラーという目盛りが用意され、細かい時間間隔での状態遷移や時間制約、メッセージの送受信などを表すモデルを描くことができ、時間軸を考慮した設計を図にすることができます。

タイミング図は横軸に時間、縦軸にオブジェクトの状態を書き、直線で経過を表します。状態が切り替わるポイントでは直線が垂直になることから、何らかのイベントが発生したものと考えることができます。タイミング図はシーケンス図と同様にイベント発生を意識した図です。

図 4-13　タイミング図



タイミング図のサンプルを図 4-13 に示します。

図 4-13 では、基地局、位置計算、プロペラ駆動、飛行船制御信号はそれぞれの状態を表しています。つまり、ある特定の時間に各状態が動作することを示しているのがタイミング図です。基地局内で現在の位置計算を行い、飛行船のプロペラを駆動し、駆動したタイミングで飛行船を制御する信号を送信し始めることがわかります。

組込みシステムではタイミングの検討は重要

タイミング図は、時間関係に注目して状態や状況の変化を描くための図です。表記法は視覚的に状態変化がわかりやすいという特徴があります。このため、プログラムの修正時には、時間的な変化の観点で他のどの部分に影響が及ぶかが見通しやすくなり、保守性が向上します。

航空宇宙・交通機関・燃焼機器・生命維持装置・安全装置等に代表されるクリティカルな組込みシステムでは、そのタイミング一つで重大な問題に直面することがあります。

厳密なタイミングでの制御が必要なリアルタイム処理では、所定時間内に処理が終わらないことは障害にも値するため、タイミング図で時間関係と状態や状況の変化を定義することは信頼性の向上に寄与します。

4-12　ユースケース　Use Case

システムの機能を考え、定義する際に利用する

本章ではユースケース図とユースケース記述の２つについて解説します。

ユースケース図

ユースケース図とは、システムの要求を分析し、システム自体がどのような機能を持つのか、システムの外部からみた振る舞いはどうなっているのかということを定義するために使用する図です。ユースケース図で使用するのは、基本的には「対象」といわれるシステム全体を指すものと、アクター、ユースケースになります。

ユースケース記述

一方、ユースケース記述（シナリオ、フローなど）とはユースケース図内の１つのユースケースにおけるアクターとシステムとの関係や振る舞いを、シナリオとして記述したものです。ユースケース名、概要、アクター、事前条件、基本フローなどを記述します。この記述は、誰が、どうすれば、何ができるのかということをシナリオ・フロー形式で明確に表記します。

ユースケース図のサンプルを図4-14に、ユースケース記述のサンプルを表4-1に示します。

図4-14及び表4-1は、要求分析で仕様書から要求を洗い出し、それらに基づいてシステム全体のユースケース図及びユースケース記述を作成したものです。開発対象をパッケージ分けし、パッケージごとにユースケースやその振る舞いをモデル化していますが、この段階では、要求分析より「システムは飛行船と地上施設に大別される」「地上施設を利用し、飛行船を飛行用コースの要求どおり航行させる」 ということを引用し、実際に使用されるハードウェア仕様には依存しないようになっています。

図 4-14　ユースケース図

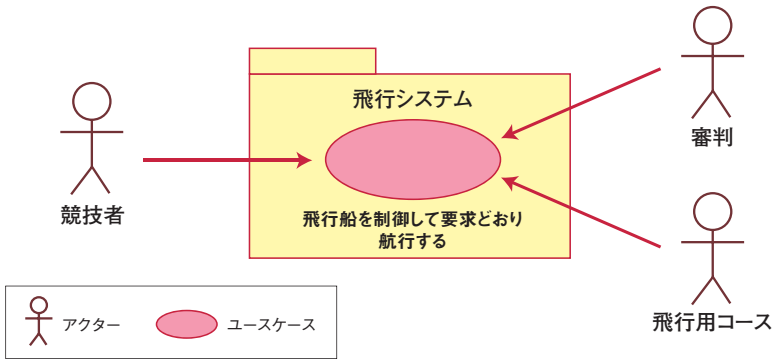


表 4-1　ユースケース記述

ユースケース名	飛行船を制御して要求どおり航行する
概要	飛行システムは地上施設及び飛行船から構成される 競技者は地上施設を利用し、飛行船を飛行用コースの要求どおり航行させる 飛行用コースは複数ある場合がある 審判は飛行船が要求どおり航行したか判断する
アクター	競技者、審判、飛行用コース
事前条件	飛行船を航行させるための地上施設、飛行船がある 飛行船、地上施設の電源が入っている
基本フロー	地上施設を利用し、飛行船を飛行用コースの要求どおり航行させる 審判に飛行船が要求どおり航行したか判断してもらう

ユースケースはシステム理解の出発点

ユースケース図とユースケース記述は、誰がシステムに対して何をするか、あるいは何ができるかをシナリオ的に記述したものです。従って、開発者が利用者側の機能要求を明確に捉えて内容を整理することにより、機能全体を把握することが可能です。そうすれば、顧客要求を明確にビジュアル化し、設計の早い段階でシステム全体の振る舞いを理解することに繋がります。

4-13　状態遷移図　State Transition Diagram

状態遷移図はシステム内部状態の変化を整理する

状態遷移図はUMLではステート機械図（State Machine Diagram）とも呼ばれています。そこには、システム、サブシステム、タスク、モジュール、オブジェクトといったあらゆる粒度の振る舞いを記述します。

振る舞いをシーケンス図やシナリオで記述すると無限になります。この無限の振る舞いを有限状態機械（Finite State Machine）として表現するのに用いられるのが、状態遷移図です。

有限状態機械は、事象、アクション、状態及び状態遷移により構成されます。状態遷移図は、特に状態遷移の流れを明示するもので、状態を□、状態の移り変わりを→で表現します。遷移を示す矢印には、遷移するトリガとなる事象を記述します。事象を受けた際の動作は、事象の後にアクションとして記述します。

状態遷移図のサンプルを図4-15に示します。

図 4-15　状態遷移図

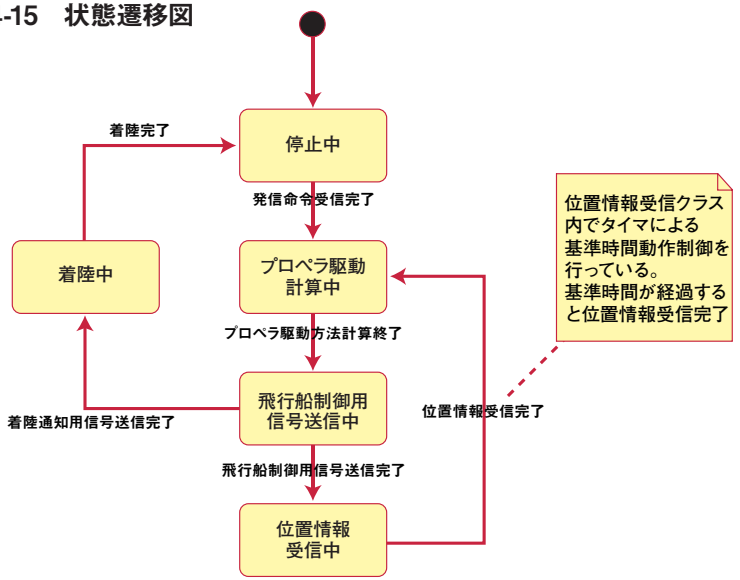


図4-15は無人飛行船システムの基地局クラスの振る舞いを記述したものです。初期状態である「停止中」状態で「発信命令受信完了」事象を受けると、「プロペラ駆動計算中」状態に遷移します。「飛行船制御用信号送信中」状態では、「着陸通知用信号送信完了」事象を受けると「着陸中」状態に、「飛行船制御用信号送信完了」事象を受けると「位置情報受信中」状態に、それぞれ遷移します。

システム構造のスパゲッティ化を防ぐ

システムを状態（State）、事象（Event）、アクション（Action）の構造（ここではSEA構造と呼ぶ）にすることで、ソフトウェア品質特性の「保守性」及び「移植性」が向上します。従来のフローチャート設計はフラグ変数が多用されたスパゲッティプログラム構造になりやすいことがあります。そこで、状態遷移図を用いることで、状態、事象、アクションが分類されます。SEA構造にすれば、システム固有の影響を受ける、いわゆるホットスポットは「状態」「事象」となり、アクションは派生するシステムでも再利用できる部品として設計することができます。

無限の動作を有限の振る舞いにすることで、ソフトウェア品質特性の「機能性」及び「使用性」は向上します。組込みソフトウェアの特徴は外界に対する制御で、これには複雑な振る舞いが存在します。この振る舞いの文書化は、読む人によって異なる解釈を導いてしまう曖昧さがあります。従って、その振る舞いを状態遷移図にすることで、システム（タスクでもオブジェクトでもよい）がどのような動作をするかが明示できるようになります。従来のフローチャート設計では、どのように動かすかという情報をモデル化したのに対し、状態遷移図設計では、「いつ」「どこで」「何をするか」という情報をモデル化します。

4-14　状態遷移表　State Transition Matrix

基本的に状態遷移図と同じ役割を持つ

状態遷移表の目的は状態遷移図と同様のものとなります。図 4-15 状態遷移図を参照して下さい。

状態遷移図では有限状態機械を、事象、アクション、状態、及び状態遷移により構成します。一方、状態遷移表は、特に事象と状態の組み合わせを明示します。

状態遷移表のサンプルを表 4-2 に示します。

表 4-2 は無人飛行船システムの基地局クラスの振る舞いを記述したものです。図 4-15 の状態遷移図で表現された振る舞いと同じ意味を表しています。

状態などを網羅的に確認する際に有効

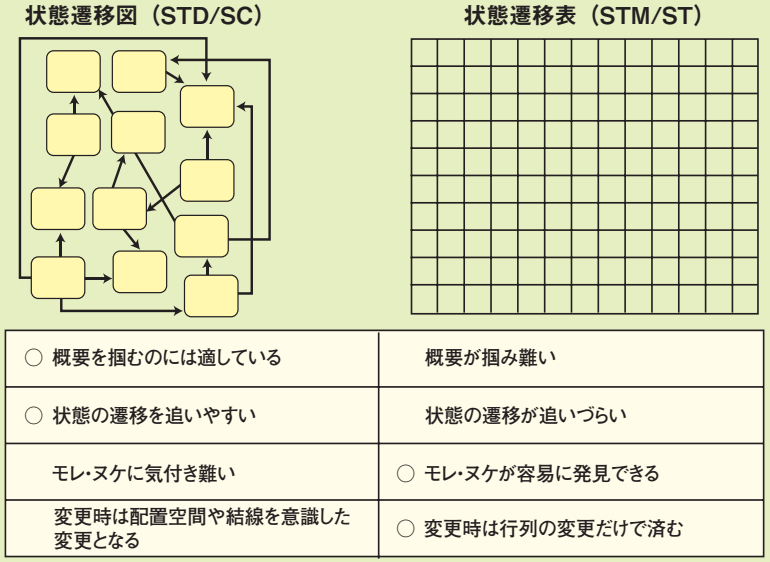
表を作ることによるメリットも状態遷移図と同じです。

表 4-2　状態遷移表

	S	停止中	プロペラ駆動計算中	飛行船制御用信号中	位置情報受信中	着陸中
0		0	1	2	3	4
発信命令受信完了	0	→ プロペラ駆動計算中	／	／	／	／
プロペラ駆動方法計算終了	1	×	→飛行船制御用信号送信中	×	×	×
飛行船制御用信号送信完了	2	×	×	→ 位置情報受信中	×	×
位置情報受信完了	3	×	×	×	→ プロペラ駆動計算中	×
着陸通知用信号送信完了	4	×	×	→ 着陸中	×	×
着陸完了	5	×	×	×	×	→ 停止中

信頼性を必要とする組込みソフトウェアでは、全状態で全事象との組み合わせを網羅的に設計することが重要です。状態遷移表を使えば、全事象と全状態の組み合わせを網羅的にモデル化することができます。

図 4-16　状態遷移図（STD/SC）と状態遷移表（STM/ST）の比較



○ 概要を掴むのには適している	概要が掴み難い
○ 状態の遷移を追いやすい	状態の遷移が追いづらい
モレ･ヌケに気付き難い	○ モレ･ヌケが容易に発見できる
変更時は配置空間や結線を意識した変更となる	○ 変更時は行列の変更だけで済む

補足　STD と STM の比較

作成の目的は同じ状態遷移図（STD/SC）と状態遷移表（STM/ST）ですが、視覚的な違いが大きいように、その特色にも大きな違いがあります。図 4-16 に状態遷移図（STD/SC）と状態遷移表（STM/ST）の比較を示します。

4-15　SDLブロック図　SDL Block Diagram

SDLブロック図はモジュール内部構造の整理に利用

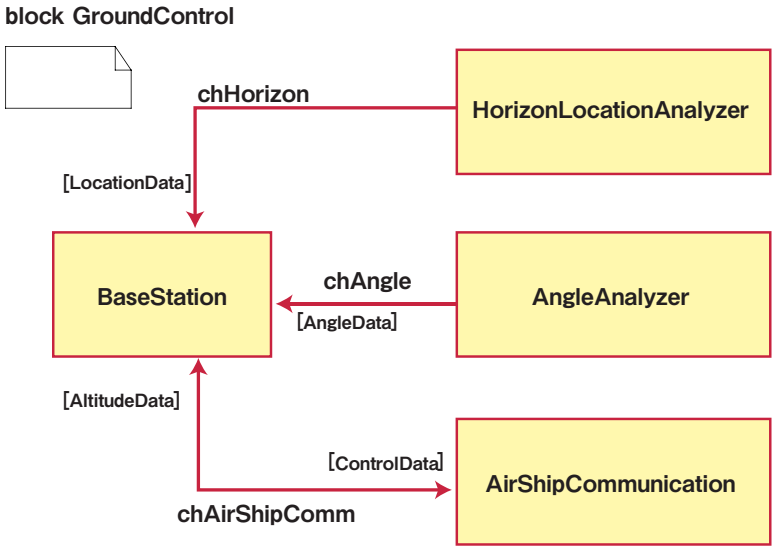
SDLブロック図は、あるモジュールの内部構造を表すために用いられます。上流工程ではシステム全体の大まかな構成を、下流工程ではあるサブシステム内部の構造を示すのに用いることができます。

また、各サブシステム間でやり取りする経路と、そこでやり取りされるイベントの種類を明示します。

SDLブロック図のサンプルを図4-17に示します。

図4-17では、「GroundControl」というサブシステムを、「BaseStation」、「AngleAnalyzer」など4つのサブシステムに分割していることを示します。また「BaseStation」と「AirShipCommunication」の間には「chAirShipComm」チャネルという経路があり、「BaseStation」から「AirShipCommunication」

図4-17　SDLブロック図



の方向には「ControlData」というイベントが、逆の方向には「AltitudeData」というイベントがやり取りされていることが図示されています。ちなみに、SDLでは日本語は使用できません。

これらのサブシステムの分割は階層化することができます。例えば、「BaseStation」ブロックの中をさらにいくつかのサブシステムに分割できます。このような内部構造は、システムをサブシステムに分割するというようなトップダウンでも、既存のモジュールを組み合わせていくというようなボトムアップでも、どちらの方法でも作ることができます。

サブシステムレベルの振る舞いが理解しやすくなる

SDLブロック図を使えば、サブシステムをいくつ作り、またどのサブシステムとどのサブシステムがやり取りするのかが明確になります。また、各サブシステムがやり取りできるイベントの種類が明示されるため、各サブシステムのモジュール性や使用性（Usability）が向上します。例えば、ブロックAでは受けられないイベントEをブロックBから送ろうとしているといった、イベントの利用における矛盾をチェックすることができます。

さらに、サブシステム間の境界が明確になり、イベントの種類も特定できるため、変更による影響の範囲が把握でき、保守性を向上させることが可能になります。特に、SDLではあるサブシステム内の変数に他のサブシステムから直接アクセスすることができないため、オブジェクト指向でのカプセル化と同様の効果が期待できます。

SDLそのものはターゲット環境には依存しません。従って、別の環境に移す場合にも記述の変更は原則的に不要になるため、移植性は高いといえます。また、P.56の図4-18　SDLプロセス図と組み合わせることで、システム全体で使用されていないイベントの発見も容易になるので、効率性の維持にも有効といえます。

4-16　SDLプロセス図　SDL Process Diagram

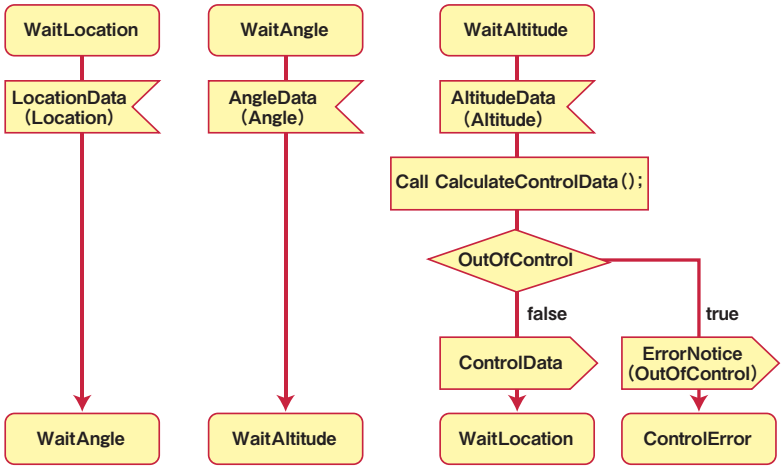
SDL プロセス図はサブシステム内の動作を詳細に定義

SDLプロセス図は、あるサブシステム内の状態遷移の動作をフローチャートで示します。上流工程ではシステム全体の振る舞いを表すこともできます。また下流工程では、あるモジュールの詳細な動作を表すのに使用されます。

SDLプロセス図のサンプルを図4-18に示します。

が状態を示し、がイベントの受け取り、がイベントの送信、が条件分岐、が動作を示すシンボルです。図4-18では「WaitLocation」、「WaitAngle」、「WaitAltitude」、「ControlError」という4つの状態が定義されています。「WaitLocation」状態で「LocationData」イベントを受けると「WaitAngle」状態に、「WaitAngle」状態で「AngleData」イベントを受けると「WaitAltitude」状態にそれぞれ遷移します。そして、

図 4-18　SDL プロセス図



「WaitAltitude」状態で「AltitudeData」イベントを受けると遷移を実行します。これらのイベントには、それぞれ括弧書きでパラメータも受け取ることが示されています。

「WaitAltitude」状態で「AltitudeData」イベントを受けると、「CalculationControlData」というサブルーチンをコールして、「OutOfControl」かどうかをチェックします。そして、「false」なら「ControlData」イベントを送信して「WaitLocation」状態に、「true」なら「ErrorNotice」イベントを発行して「ControlError」状態に移ります。このようにSDLプロセス図では、状態遷移の詳細な動作までモデル化することができます。ただ、SDLでは日本語は許されていないので、すべて英語で記述する必要があります。

サブシステム内の処理の理解性が向上

SDLプロセス図では、状態、イベント、その動作の組み合わせが明確になり、かつ動作の内容がフローチャート形式でわかりやすく示されます。このため、処理内容の理解性が向上し、処理中でのイベント出力のタイミングなどレビューでバグを発見することが容易になります。これは変更時にもあてはまり、保守性の向上にも大きく寄与します。

また、詳細な動作まで記述できるので、機能の正確性や合目的の検証がモデルレベルで実現できます。詳細な動作も追えることから、解析性も高く、保守性の向上も得られます。そして、SDLはターゲットコードに依存しない動作の記述が可能ですので、移植性を確保することもできます。

特に図4-17 SDLブロック図と組み合わせれば、ソフトウェアの実装まで行うことができます。SDLブロック図で示される各モジュールの動作をSDLプロセス図で作成することにより、C言語などのプログラミング言語を用いることなく、モデルだけでソフトウェアを開発できます。

4-17　データフロー図　Data Flow Diagram

データの処理・加工などの流れを追う場合に有効

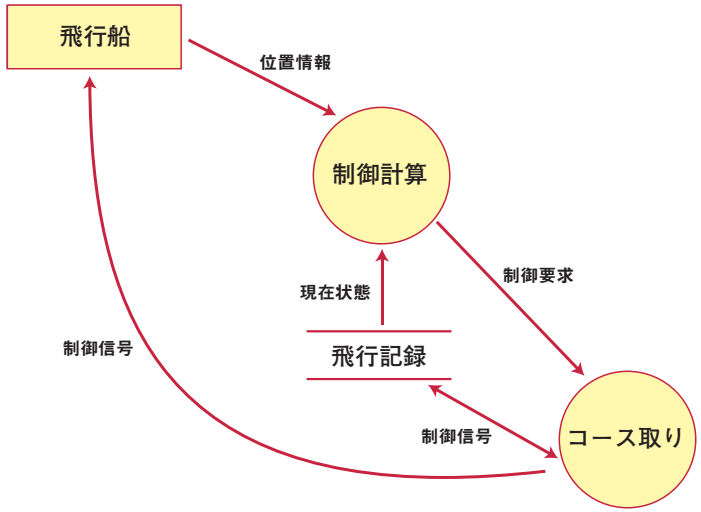
データフロー図（DFD：Data Flow Diagram）は主に分析フェーズに用いられます。データフロー図は「プログラム＝データの変換」という考え方に沿って、流れるデータをキーとし、データと処理の流れを視覚的に図式化した図です。

データフロー図では大きな対象を分割して階層化することにより、データの流れを容易に明確にできるとともに、分析によりモデル化することで全体像も容易に把握できるようになります。こうして、本来あるべき姿や効率化しやすい箇所を簡単に発見できるといったメリットが生まれます。

データフロー図のサンプルを図4-19に示します。

図4-19は飛行船からの「位置情報」を受けて、「制御計算」「コース取り」を実施する基地局の振る舞いを分析したデータフロー図です。□の中にデータの発生源と行き先を表し、○の中にデータの処理・プロセス、＝の中に保存するデータのファイル名などのストア箇所を示しています。

図4-19　データフロー図（DFD）



利用するデータの視点からのレビューが可能に

データフロー図を用いた分析フェーズのモデリングでは、まず開発対象とするシステムのスコープを明確にし、データの入出力と変換のプロセスを定義していきます。次に、開発対象とするシステムの機能をブレークダウンします。その際、各機能の具体的な動作仕様が見えるまで分割していきます。この過程では関係者間で要求仕様の確認やヌケ・モレの発見、問題の解決を行っていきます。

この方法の利点は、要求・要求分析段階といった早期のフェーズで処理内容が明確になることです。特に、要求・要求分析段階のモデリングに用いられるため、プログラムの構造など作り手の事情に左右されにくく、理解性・習得性・運用性・標準適合性といったソフトウェアの使用性に関する要素の向上に寄与します。

また、工程が進んだ段階で分析フェーズにおけるデータフロー図と照合することで、システムが利用するデータの視点に立ち返り、当初の要求仕様との間に起きたズレも容易に確認できます。

5　モデリングと検証技術

5-1　テストと検証

システムの信頼性／安全性はシステム開発の必須要件

あらゆる機器にソフトウェアが組み込まれるようになり、組込みソフトウェアの信頼性は、利用者にとっても開発者にとってもますます重要な課題となっています。特にソフトウェアの規模と複雑度の増大、開発期間の短縮などに伴い、組込みソフトウェアの信頼性に関する問題は組込みシステム開発におけるアキレス腱になるのではないかと懸念されています。

こうした中、設計手法や管理手法といったソフトウェア工学分野の成果が組込みソフトウェア分野にも導入され、状況を改善させる取り組みが進められています。システムの信頼性や安全性を向上するために、より直接的な方法としては、検証やテストの高度化といった方向性を考えることも必要です。

モデリング技術をベースにした検証手法の必要性

特に組込みシステムの動作検証の分野については、きちんと表現された設計モデリングの情報をベースにした形式的手法のように、科学的な手法に対する期待が高まっています。

ソフトウェアの信頼性を高めるための手法については現在、テスト手法

の開発が中心となっています。しかしながら、ソフトウェアの規模の拡大により、コードに対するテストだけでは品質向上には不十分で、設計に対する検証を行うことが重要となってきます。

本章ではモデリング技術を使った設計検証について解説します。テスト、あるいはここで紹介する設計検証の手法はそれぞれに得手、不得手がありますので、その特徴を正しく理解し、目的に応じて使い分けていくことが品質向上にとって不可欠となります。

5-2　静的検証と動的検証

検証手法には様々なものがある

検証手法の代表的なものには、レビュー、プロトタイピング、シミュレーション、形式検証等があります。検証手法には、限られた工程で使われるものも存在しますが、レビューのように複数の工程にわたって適用できる手法も存在します。また、ソフトウェアの検証対象はプログラムだけではなく、仕様書や設計書、本書の４章で解説したクラス図やオブジェクト図などの設計記述言語など、多岐にわたります。一般に、テストがプログラムのように動作するものだけを対象とするのに対し、これらの検証手法は分析、設計（基本、詳細）、実装、テスト（単体、機能）、システムテストといったソフトウェアの各開発工程で実施されるもので、ソフトウェアの信頼性向上にとって効果的に機能します。

静的検証と動的検証――検証できることが違う

検証を別の観点から見ると、静的検証と動的検証という捉え方もできます。

静的検証

静的検証は、開発した成果物（プログラムなど）を動作させることなく、目視あるいはツールを使って静的に解析する検証のことです。ソ

フトウェアの複雑な動きを時系列で観察し、検証することはできません。従って、静的検証は、あるイベントが起きたときに状態遷移先が変数で示されている場合など、プログラムを実際に動かしてみないとその挙動がわからないような事項には向いていません。

動的検証

動的検証は、作成物を動かして検証することです。そのため、一般には下流フェーズで使われます。代表的なものは、プログラムを実行可能な形式にして行う動作確認テストになります。

モデリング技術を使った検証は開発上流での検証を可能にする

一般にテストは開発フェーズの下流で行われるため、不具合発見時に手戻りが大きくなるといった問題があります。開発全体の効率を考えた場合、問題点の発見は極力上流であるに越したことはありません。

こうしたことが背景となり、近年着目されているモデリング技術を使った検証手法は、分析や設計段階で作成したモデルを用いて、開発における早期の段階（上流段階）で検証を実施するものとなっています。

5-3　モデリング技術を使った検証手法

本章では、開発におけるモデリング技術を使った代表的な検証手法について解説します。もちろん、これまでのプログラム開発における検証手法の多くは、モデリング技術に対しても同様に適用できます。

(1) レビュー：検証の王道

本書の読者には、レビューという言葉を聞いたことのある方が少なくないと思います。レビューは、開発の上流から下流までの多くの工程で採用することができる検証方法の中で、最も手軽なものです。例えば、実装の段階では、作成したソースコードを対象としてコードレビューという形でレビューを行うことができます。

■レビューの形式

レビューの形式については、第三者を交えて実施するフォーマルレビューや、開発グループの仲間同士で行うピアレビューなど様々な形態が知られています。それぞれ長所短所があります。

■レビューで利用する手法

また、実際のレビューで成果物をチェックする方法としては、インスペクションやウォークスルーといった手法も知られています。これは多くの場合、基本的にツールによる（自動）検証が難しいフェーズや検証項目について、複数の人手で行うことを指します。

■モデルを利用したレビュー

モデルによるレビューは上流工程である分析や設計フェーズで行われます。この場合のレビュー対象は分析・設計フェーズで作成したモデルが対象になります。レビューを行う前に、モデルに対するレビューのルールやチェックポイントを明確にすることで、大きな効果をあげることも可能です。

ただしこの場合、レビューに参加する人のスキルに依存する面があるため、注意が必要です。モデルを使ったレビュー時のポイントとメリットは次のようになります。

モデル使用時のレビューのチェックポイント

- 仕様が過不足なく実現されているか？
- モデルの表現方法は正しいか？
- より良いモデル（対案）は考えられないか？
- 分析のモレやヌケがないか？

モデル使用時のレビューのメリット

- 自然言語に比べて曖昧さが解消され、分析結果を正確に共有できる
- 暗黙の了解事項と思われていることも、明示的にモデル化することにより、思わぬ間違いを発見できる
- レビューの勘所・ノウハウが蓄積される
- デザインパターンなど過去の解決策を適用できる

(2) プロトタイピング：動きや振る舞いの早期確認に有効

完全な製品やシステムを作り上げてしまってから問題が発見されたのでは、手戻りが大きく、開発工程にも大きな影響を及ぼす可能性が高くなります。そこで、評価に使用できる製品や、システムの全体もしくは一部を表しているものを使用して評価を行う方法があります。これをプロトタイピングといいます。

プロトタイピングは動的な検証手法で、上流から採用できます。ユーザにとっては、文字で書かれた文章よりも直感的に理解しやすく、お互いのコミュニケーションを円滑に進める上で助けとなる有力な手法となります。そのため、特にシステムの動きや振る舞い、ユーザインタフェイスの確認などを目的として利用される場合が多いようです。

一方、分析フェーズのような上流工程においては、特定のユースケースの表面上の振る舞いだけを示すような構成になりがちです。従って、工程の進捗に合わせてプロトタイピングの開発を実施していくことが有効になります。

(3) モデル解析：モデルの静的チェックも有効

モデル解析は設計フェーズでの静的検証手法です。モデル言語でのアーキテクチャ設計や基本設計の段階で採用します。ソースコード解析の考え方をモデルに適用したものと考えれば理解しやすいでしょう。作成したプログラムコードをコーディング規約でチェックするツールと、適用対象がモデルに変わるだけで、考え方は同じと解釈しても構いません。

モデル解析は、モデルの構文や整合性を静的にチェックします。市販のツールも数多く存在します。例えば、UMLエディタを利用すれば基本的な記法のチェックは行えます。

チェック項目としては、ネーミングルール、記法や複雑度の上限値（例えば1クラス内の状態数、状態のネスト数、継承の階層数）などがあります。これらを測定する指標はメトリックスと呼びます。的確な解析を行うためには、それぞれの性質を正確に示すメトリックスを規定することが大切になる。モデル解析は定量化できるため、客観的な評価として活用できます。

(4) 形式検証：モデルの正しさを論理的に確認

形式検証は、検査対象であるソフトウェアの仕様や振る舞いを数式や論理式で表現し、誤りがないかどうかを数学的な証明手段により保証する静的検証手法です。アーキテクチャ設計や基本設計を厳密に検証することが

できます。

形式検証にはモデル検査手法と定理証明手法があります。近年、計算機の性能向上や検証技術の進化によりモデル検証に注目が集まっています。

テストデータを用いたシミュレーションでは、そのテストデータにおける動きの検証しか行えませんが、モデル検査では、検証対象を有限状態モデルで記述することで、すべての状態でチェックしたい性質が成り立つかどうかを自動的に検査することができます。

形式検証については、P.82 コラムを参照してください。

(5) スコアリング：設計の良し悪しなどを点数付けして評価

スコアリングは、大きく捉えればレビューに含まれる静的検証手法となります。ただ、複数のアーキテクチャや設計案が存在する際の比較手法であることから、検証手法ではなく評価手法といった方が適切な場合もあります。

スコアリングでは、評価対象に複数の評価項目とその重要度を設け、評価基準に従って点数（スコア）をつけていきます。特定の記述方法と結び付いた手法ではないため、導入も簡単です。ただし、評価項目やその重要度などは人に大きく依存するため、客観的になり難く、明確性に乏しいという面があります。繰り返し行い、改善を施していかないと、アドホックになってしまう危険性が存在しています。

(6) モデルシミュレーション：設計モデルを動かして確認

モデルシミュレーションは分析モデルや設計モデルをツールにより動作させ、システムテスト・機能テストを実施する手法です。プログラミング言語を使ったプログラミングでのテストはソースコードができるまで実行できませんが、モデルシミュレーションでは、分析モデルや設計モデルを用いることにより、上流時点からテストが行えます。近年は多くのツールが開発され、ツール上でモデルを実行し、そこでテスト結果を確認したり、モデル自体を修正することも行えなるようになっています。

モデルシミュレーションはホワイトボックステストで、一般に状態遷移パステストやデータフローパステストとして使われます。ソースコードのテストと同様にテストスクリプトを作成して実行し、結果を比較します。ただし、テストスクリプト作成の作業の際、大きな負荷量を必要とする場合があるので注意が必要です。

最近では、これらの作業を自動化する仕掛けも進んでいます。パステストのカバレッジを高めるために状態図を網羅的にテストできるテストケース（テストデータ）を自動生成し、テストを実行するというものです。その他、テストケースをシーケンス図で表現してテスト結果と比較するといったものもあります。自動化は、プログラムを変更したときにその変更によって予想外の影響が現れていないかどうかを確認するテストで、特に大きな効果を発揮します。

6 モデリングの組込みソフトウェアへの適用性

6-1 組込みソフトウェアへのモデリング技術の適用

オブジェクト指向でなくてもモデリング技術は利用できる

モデリング技術は、組込みソフトウェアに対しても有効な技術です。多くの組込みソフトウェアはC言語などの非オブジェクト指向言語を用いて実装されるため、UMLなどのモデリング技術は使えないと考える人もいるようですが、それは間違いです。設計段階でのモデリング技術としてUMLを用い、それを非オブジェクト指向の言語で実装するという開発の流れでも、適切に用いれば十分な効果が得られます。

一方、組込みソフトウェアは、エンタプライズ系のソフトウェアと比べてソフトウェアそのものの特性、利用するOSなどのプラットフォーム、さらに開発の形態などで異なった特徴を持っています。モデリング技術を適用する際には、こうした組込みソフトウェアの特徴を踏まえて適用すれば、より効果的に活用することができます。

以下、モデリング技術を組込みソフトウェアに適用する際の留意点やヒントなどについて解説します。

6-2 モデル化の方法

組込みソフトウェアをモデル化する際に、どのような工夫が必要かを概観します。

リアクティブ性と実時間制約を考慮する

組込みシステムの1つの典型的な特徴として、リアクティブな振る舞い（環境との恒常的なインタラクションを維持すること）を持つこと、実時間の制約を持っていることなどが指摘できます。

リアクティブ性への対応

リアクティブな振る舞いについては、状態モデルを活用することで適切にそれを表現することができます。またステート図等で表現されたモデルを解析したりシミュレーションすることで、その振る舞いの妥当性を確認するといったことも可能です。

実時間制約への対応

実時間の制約はシーケンス図上での時間制約を活用したり、タイミング図を利用したりすることで表現することができます。しかしながら、一般にこうした実時間の制約記述は人間が読むための情報であり、これに基づいて解析やシミュレーションをすることはまだ一般的ではありません。これは、ソフトウェアの振る舞いはミクロな実行時間やクロックの積み上げだけでは判断できず、タスクのスケジューリング、同時に動作する各種の処理、様々な外部環境などに左右されるため、その取り扱いが困難だからです。

組込みシステムの処理の特性をモデルに反映する

組込みソフトウェアの設計では、割込み、タスク、セマフォ、メッセージキューなど、ソフトウェアの動作環境が提供する比較的プリミティブな

メカニズムを意識する必要があります。開発するソフトウェアの処理の特性にあわせ、これらのメカニズムを適切に利用しないと、期待する応答性などが得られない可能性があるからです。こうした点は、基本的に対象ドメインの問題構造のモデル化に注力するエンタープライズ系のソフトウェアのモデル化とは大きく異なる点です。

そうしたメカニズムを考慮したモデル化においては、問題構造のモデルとメカニズムの構造のモデルを明示的に意識して使い分けることが重要です。例えばステレオタイプを上手に活用することで、そのモデルが問題構造を示しているのか、あるいは具体的な実現メカニズムを示しているのかを正しく理解することができます。利用するプラットフォームに応じた、ちょっとした工夫や取り決めによって、モデル化がしやすく、かつ明確に行えるようになります。

モデルにおけるハードウェアの扱いや位置づけに注意する

ソフトウェアから見てハードウェアをモデル化する際には、ハードウェアを抽象化してモデル化することが一般的です。それは、ハードウェアとの直接的・具体的なやり取り部分を隠蔽し、一種のソフトウェアコンポーネントとみなしてモデルに組み入れる方法です。こうした抽象化は実装のテクニックとしても使われますので、それによって実装との対応付けも容易になることが期待されます。

一方、システム全体を捉える際にはブロック図等を用いることが有効です。そうすれば、開発するソフトウェアと外界との関わりを理解することができます。なお、それによるシステム全体の振る舞いの解析やシミュレーションもある範囲までは可能ですが、本書では詳しくは触れません。

6-3　モデル化と実装の関係

UML に代表されるモデリング技術の多くが、オブジェクト指向の考え方を取り入れています。そのため、C 言語等、非オブジェクト指向言語で実装を行うことが多い組込みソフトウェアにおいては、モデリングが使えないという誤解もあるようです。以下にモデルと実装との関係について述べます。

モデリングとC言語の実装は共存する

オブジェクト指向の概念でモデル化を行っても、それを C 言語で実装することは可能です。以下に、考え方について 2 つのヒントを示します。

ヒント -1：

立場の 1 つは、オブジェクト指向の概念はあくまでモデル化の便法として利用し、それをそのままの形では実装には反映させないという立場です。この場合、モデルの中でクラスとして定義しているものが、実装上は必ずしもまとまった単位として現れない可能性もあります。すると、モデルと実装のトレーサビリティはやや損なわれますが、対応関係のルールや原則を決めることで、ある程度は回避できます。

ヒント -2：

もう 1 つの立場は、非オブジェクト指向言語を使ってオブジェクト指向的な実装を模擬する方法です。例えば、クラス単位にファイルを作り、その中で属性や操作に対応する変数や関数を定義するといった手法です。あまり複雑なメカニズムを模擬することは避ける方が無難ですが、トレーサビリティも確保しやすく、わかりやすい方法です。また、組込みソフトウェアでは動的なメモリ管理は避けておきたいため、クラスに対してインスタンスを 1 つというルールでモデルを作成することも妥当な方法です。

コード自動生成技術への期待

モデルと実装方法とのマッピングがある程度整理できるなら、その部分をコード生成することは理論的には可能です。しかし、こうした環境を作ることはそれなりにコストがかかりますので、同種のマッピングが利用できるソフトウェアを複数開発する状況などにおいて有効となります。組込みソフトウェアの場合、リソース制約や実時間制約、リアクティブ性など多くの制約や特性があるため、技術的な課題は残りますが、現在様々な検討が進められており、今後期待される方法であるといえます。

7 組込みソフトウェアへのモデリング活用のポイント

この章では、モデル化を行うことにより期待できることを、設計者やマネージャーの視点で解説します。

7-1 モデリングへの期待

複雑なものをわかりやすく、再利用をもっと楽に

一般に、組込みソフトウェアは複雑な構造と処理体系を持っています。

モデリングには、何を作るかということに注目した分析モデリングと、どうやって実現するかということに注目した設計モデリングの２つの方法があります。この２つの方法は、新しいソフトウェアを初めて見る設計者にとって、ソフトウェアが何をしようとしているのか、またどうやって実現しようとしているのかといったことをわかりやすくします。分析モデルは、要求仕様と設計モデルの橋渡しをします。一方、抽象表現で記述されている設計モデルは、全体像をわかりやすくし、様々な問題や事情を発見しやすくします。これらのモデルは、ソフトウェアを再利用するときや複数の設計者で設計内容を共有したいときに非常に役に立つものになります。

設計を可視化できれば、レビューで設計品質を改善

モデルは、ソフトウェアを動作させるために、設計者が考えたことをわ

かりやすく表現することが可能です。複雑な部分は図を用いて、全体を構造的・視覚的に説明し、理解しやすくできるのです。図だけで伝えることが難しいと判断された重要な説明には、文章の記述があるため、正しく理解するための助けとなります。

設計者側でも、プロジェクトに関わるメンバー同士で、お互いに考えていることを相互に正しく理解できるようになります。共通に理解できる方法があれば、デザインレビューを効率的に進めていくことで、設計の本質（設計品質）を議論でき、上流設計で設計品質を改善することができるようになります。つまり、モデリングはソフトウェア設計の非常に大きなパラダイムです。

モデルを利用した検証により実装前に設計内容を確認

モデルがソフトウェア全体の動作を表現できる場合、モデルシミュレーションツールを活用すれば、実装前にソフトウェアの動作検証をすることができます。

一般に組込みソフトウェアの世界では、ソフトウェアの機能は拡張化し、ハードウェアは短期間で新しいものに変えられていきます。あるいは、1つのソフトウェアを色々な機器に組み込んだりします。そんな中で組込みソフトウェアは、機能追加部分とハードウェア制御部分に変更を加えるだけで、できるだけ長く使おうとされます。従って、機能拡張とハードウェア変更によるソフトウェアの変更部分にしぼって開発が速やかにできるかということが、常に問題視されています。

モデリングは、変更部分をいかにうまく設計し実装するか、そして、シミュレーションツールは変更した部分をいかにうまく動作検証するかといったことを補助します。これらは、ソフトウェア再利用における差分開発と部分検証をサポートし、ライフサイクルを長く維持することに寄与します。

設計、実装、ハードウェアの問題の切り分け効率を改善

モデルシミュレーションツールによる設計段階での動作検証は、設計で発生する問題を除去します。そのため、その後の実装では実装の問題を解決することに集中することができます。また、機能実現がシンプルで無駄のない、良いソフトウェアを開発するためには、設計者が設計と実装の問題を意識することが大切です。

モデリングとモデルシミュレーションツールは、設計品質を作りこむことを助け、実装での問題が何であるかをわかりやすくします。ソフトウェアだけの世界で動作検証ができれば、ハードウェアと結合させたときに問題が起きた際、それがハードウェアの問題かソフトウェアの問題かということを比較的楽に判断することができます。モデリングとシミュレーションツールの活用は、設計 ⇔ 実装 ⇔ ハードウェアの問題を特定することを容易にします。

7-2　モデリング導入時の課題

新しいことを始めるには悩み事はつきもの

モデリングの経験がないときに、モデリング技術を新たに導入することは大変なことです。初めて導入するときは色々な課題にぶつかるようです。そんな課題には、次のようなものがあります。

①プロジェクトメンバーがモデリングで何が解決できるのかを理解できず、従来の方法に頼ることから抜け出せないでいることがあります。なぜ従来の方法ではいけないのかと、疑問を感じるのです。結局、現状の問題とモデリングで解決できることをはっきりと示せないことが多いようです。

②プロジェクトリーダーやマネージャーにとっては、プロジェクトの達成（QCD）が重要です。プロジェクトリーダーやマネージャーは、モデリ

ングの開始がプロジェクトにもたらすかもしれないリスクを危惧します。また、モデリングの利点を理解していなかったり、管理方法が違うことに戸惑ってプロジェクトに踏み切れなかったりします。

③モデリングを導入するにあたって、ツールや教育など比較的大きな投資が必要となります。しかし、モデリングによる効果や利点が示せない（当然、経験がなく利点を明確にできないため、結局、再利用ができるとか品質が良くなるといった通り一遍の説明になってしまう）と、投資できる提案まで至らないことが多いようです。

問題の観察と相互理解――技術導入の悩みの特効薬

モデリング技術は本来、従来から行われてきた設計のやり方に問題があり、それらを改善するために有効な方法であるはずです。しかし、従来の問題のうち、何をモデリングで解決できるのかということを不明確にしたまま進めてしまうことが多いようです。ここで示したような悩み事で困らないために、モデリングを導入したいと思ったときには、何が組込みソフトウェア開発の問題であり、何を解決したいのかということを、関係者間でじっくり話し合って進めることが重要です。そして、多くの組込みソフトウェア開発者が長年かかっていろいろなやり方を学んできたように、モデリング技術を習得するにも、体験を積み重ね、理解を深め、改善しながら進めていくことが必要です。そのためには導入の際、身の丈にあった取り組み方を考え、実施中においても常に問題を解決しながら進めていくことが重要です。

最初に取り組むときは、企業の中で経営者、マネージャー、プロジェクトリーダー、設計者、全社員の目的や狙っているレベル、達成できる効果などを相互に理解し、納得して進めていくことが重要なことでしょう。

7-3　設計が見えるとは

モデルを使わなくても設計は見えるか

従来の設計方法（モデルを使わない方法）においても、設計が見えないわけではありません。機能の列挙、モジュールの定義、動作に関する説明など、基本的なものはすべて明確にできます。しかし、こういったものを活用してデザインレビューで良い設計に導く議論を行うには豊富な経験とスキルを要します。また、経験の豊富な設計者でも、別の設計者が考えた提案内容を理解して最良の方法をアドバイスするには、多大な時間と労力を要します。最適な設計とは何かを決定すること、実現するにあたって多くのモレを見つけること、そのためにはレビューメンバーが共通の理解を持ち、発展的な議論をできることが必要となります。

モデリングは設計の見える化と理解のための良薬

モデリング手法により記述するクラス図、シーケンス図、コミュニケーション図、ステートマシン図等は、ソフトウェアの構造（モジュール階層化とデータ、処理）、動き（イベントに対する挙動、責任を負うものの存在等）、あらゆる関連性（独立した存在、関連するもの同士の存在、時系列に動きあうもの同士の存在等）を明確にすることができます。そして、組込みソフトウェア開発においては、実際に存在するもの（見えるもの、例えばモーター、センサー、走行コース等）が豊富でオブジェクトとして定義でき、ビジュアルに描くことが、他の人の理解を容易にします。

「設計が見える」とは、設計者が考えていること、あるいは考えたことが見えることにあります。これらがよいかどうかをレビューで議論できることが、良い設計品質に繋がります。設計をよく理解した設計者がソースコードだけを一生懸命開発すれば、その開発プロジェクトは良い設計品質を得ることができるかもしれません。しかし、組込みソフトウェア開発では、

そのソースコードを使って次の開発を行うことが当然のようになっています。資産として設計情報を共有することなしに、設計とソースコードを理解した設計者に属人的に頼るといったことは、期待できない状況になっているのです。

「設計が見える」ことにより、多くの開発メンバーが設計の本質（設計者が考えていること、あるいは考えたこと）を共有して設計に参加し、レビューすることがあたりまえになったとき、いつも改善された設計品質が得られるようになります。モデルは、設計と設計者の考えを見えるようにする役割を果たすのです。

7-4　ソフトウェアの設計検証の効率化

組込みソフトウェアの設計は動作の設計がポイント

組込み開発の大きな特徴の１つとして、イベント処理駆動が多いことがあげられます。どこでどんなイベントが発生し、それをどこが受け取るかということを正しく理解し、設計者たちの間で共有することが重要です。イベントの責務と発生条件、タイミングを正しく理解することは、どんなタイミングでイベントが発生しても正しく処理できることに繋がります。組込み開発におけるモデリングの重要性は、上記のような動作設計をいかにうまくできるか、かつ表現できるかということにあります。そのために、クラス間のイベント発生・処理をコミュニケーション図で表現したり、各クラスがイベントによりどのような状態遷移を起こすかを状態遷移図で表現します。

モデルを用いない設計では動作設計の表現が難しい

従来の設計方法は、ソフトウェアの動作を表現できる方法をあまり持っていません。そのため、この部分の多くは、設計担当者の頭の中に封印されてしまいます。特に、組込みソフトウェア開発においては、ハードウェアからのイベントが全体の動作を複雑にします。設計者が問題解決にぶつかってしまえば、プロジェクトの進行は危機に立たされます。

モデルを利用して動作の表現を確実に見える化する

一方、何らかの方法で全体の動作を表現できれば、多くの設計者にとって、システム全体の動きや個々の担当部分の振る舞いを理解することが比較的容易になります。またソフトウェアが正しく動作するかどうかをデザインレビューで追っていくことも楽になります。モデリングの手法の中には、システムの振る舞い、動きなどに焦点をあて、それらをダイヤグラムとして表現するものもあります。これらを利用することで、システム動作の把握や確認を確実なものにすることができます。さらにモデルシミュレーションツールを使えば、修正後の設計検証も容易になることがあります。

モデルを利用するとハードウェアとの擦り合せも楽になる

機能拡張を常に伴い、ハードウェアより長いライフサイクルで再利用され続けるソフトウェアは、設計の構造を常に変化させながら、そのときのハードウェアに最適化させていかなくてはなりません。ハードウェアに組み込まれたときは、すぐに動作することがいつも期待されているのです。モデリングとシミュレーションツール環境は、設計の動作を見えるようにした結果、ハードウェアに組み込む前に、設計の動作を早く、確実に検証できるようにします。

7-5　問題の切り分けへの対処

不具合原因の切り分けは難しい

従来の設計方法では、ソースコードを記述して動作させることにより動作検証を行います。ですから、問題が発生したとき、不適切な設計による問題と不適切なソースコード実装による問題を同時に検討していくことになります。QCD に追われる多くの設計者は、この２つの問題を区別することなく解決しがちです。本当は設計の問題として解決しなければならないのに、それに気付かず、ソースコード実装のテクニックで解決してしまうこともあるのです。こうして、結果的に複雑なコードを生み出してしまいます。

設計／実装、ソフト／ハードの切り分けが明確に見えてくる

モデルは、全体の動作を描き、設計動作の検証をできるようにします。一方、ソースコードができ上がった後に発生する多くの問題は、ソースコード記述に関する問題（実装の問題）であることが多くなります。

組込みソフトウェア開発では、ソフトウェアとハードウェアの問題を切り分けることが容易ではありません。非常に時間のかかるものです。従って、あらかじめソフトウェアの動作を確認し、それから実機上で動作させれば、ソフトウェアとハードウェアのミスマッチで起こる色々な問題が特定しやすくなります。

モデリングはツールの力を借り、設計検証（モデルシミュレーション）を比較的楽にしてくれます。そして、ソフトウェアだけの世界での設計検証手段を提供してくれます。

設計の問題 ⇔ 実装の問題 ⇔ ハードウェアとの関連の問題の切り分けは、組込みソフトウェア開発の中で大きな問題の１つです。モデリングとモデルシミュレーションで得られた品質は問題の切り分けを楽にすることにより、トータル開発効率の改善の一助となります。

モデリングの導入は長期視野を持って計画的に

モデリングとモデルシミュレーションにかかる負担だけを思えば、簡単に導入を進められないこともあります。そのため、組込みソフトウェア開発全体のトータルコストに着眼して、その改善を視野に入れながらモデリング技術とモデルシミュレーションを活用することをお勧めします。ただし、そのアプローチは決して最初の段階で画期的な開発効率を生み出すわけではなく、$(改善)^n$ を地道に進めていくことになることを理解しなくてはなりません。一般的な技術導入と同様、導入のためのコストと効果、導入にあたっての様々な制約条件なども考慮し、長期的な視野の下に計画的に導入していくことが近道といえるでしょう。

【コラム】 形式検証への期待

自分の書いたプログラムの正しさが代数や幾何の定理証明のように証明できれば、こんな嬉しいことはありません。日夜現場で忙しさに追われている状態の我々には夢のような話です。こうした技術はまだ大学などの研究室の世界の話と考えていましたが、多くの研究者の皆さんの努力によって世の中は少しずつ進んでいると感じます。

その１つが「形式検証」技術でしょう。形式検証は、大きくは(1)定理証明 と(2)モデル検査 に分けられるようです。

(1) 定理証明

まず対象システムの処理や性質の定義を形式論理（述語論理など）で表現します。そして、推論規則を用いて性質を導出し、証明を行います。中学や高校で習った数学の定理証明に似ています。無限状態を持つものは扱えますが、証明戦略や使用する定理を決める必要があり、対話的な推論が必要になります。

(2) モデル検査

対象システムを有限状態モデルで表現し、状態を網羅的に遷移させ、ある状態で検査したい「論理的な性質が成立するかどうか」を自動的に検査するものです。従って、状態数並びに変数の数や大きさによって、計算量が莫大（状態爆発）になることが容易に想像できます。また、変数の値によって状態遷移が定まるケースでは、変数の値を無限に取れる問題は扱うことができなくなります。たとえ有限でも、状態数が爆発的に増えると現実的ではなくなります。

近年、モデル検査の分野においては、計算機の速度や取り扱えるメモリ容量の進歩と共に、いくつかの工夫から多くの研究や実用に向けた取り組みが見られるようになりました。設計工程の比較的上流である「分析」「アーキテクチャ設計」や「基本設計」フェーズでの利用がそれにあたります。

【コラム】 SPIN と Promela

モデル検査のために利用できる道具の１つとして SPIN があります。

SPIN は AT&T のベル研で開発されたモデル検査ツールです。SPIN でシステムの検証をするためには、まず Promela という言語で対象システムをモデル化し、証明したい論理的な性質を LTL（線形時相論理）で記述する必要があります。それを SPIN に入力し、検証させます。SPIN は状態を自動的に遷移させ、すべての状態について指定された論理的な性質を満たしているかどうかをチェックします。満足していない状態（反例）が見つかると、出力を行います。

この反例を解析することで、デッドロックが発生するかどうか等の問題点を発見することができます。このように形式検証は、テストにおけるテスト抜けをなくして網羅的に検証できるのですが、下記のようないくつかの課題が存在します。

・UML などで記述された設計を Promela でどのように記述するか
・検査したい設計上の性質を LTL でどのように表現するか
・状態爆発などを起こさずに実際に検証できるか

また、Promela 上で検証された性質が、実際の設計やコード上でも成り立つかどうかも、重要な問題となります。現在、これらの課題を解決し、SPIN や Promela などを実際のシステム開発の中で利用できるようにするために、いくつかの研究やツールの開発が進んでいるようです。

8 おわりに

ソフトウェア開発の世界では“銀の弾丸”はないというのが定説になっています。

その一方で、開発されるソフトウェアの品質、コスト、納期などを見ると、多くのプロジェクトや組織で様々な問題が山積しています。ですから、多くの開発者、管理者、経営者が“銀の弾丸”の出現を待ち望んでいるのです。本書で紹介した“設計モデリング”は、果たしてこうした方々にとって“銀の弾丸”になり得るでしょうか。

世の中には、その使い方などによって、光り輝くことも色あせることもある道具があふれています。ソフトウェア開発の世界において設計モデリングは、ソフトウェアの品質――特に設計の段階での品質を向上させるための道具の1つになります。

この道具を光り輝く“銀の弾丸”にすることができるかどうかは、ひとえにこれを利用する皆さんにかかっているのかもしれません。本書を参考に、設計品質向上に向けた“銀の弾丸”を手に入れる旅の最初の第一歩を踏み出していただければと思っています。

なお本書は、経済産業省の組込みソフトウェア開発力強化推進委員会、及び独立行政法人 情報処理推進機構のソフトウェア・エンジニアリング・センター(IPA／SEC)の下に組織したタスクフォースが作成したものです。

このタスクフォースでは「現場で役立つ組込みソフトウェア・エンジニアリングの導入」を目的に、様々な手法の開発や施策検討・展開を進めています。組込みソフトウェア開発に適したモデリング技術の整理・普及となる本書も、その活動の一環として整理・執筆したものです。この成果を積極的に利用していただき、皆さんの組織内における組込みソフトウェア開発力の強化につなげていただければと思います。

参考文献

■ソフトウェア工学大事典／片山卓也、土居範久、鳥居宏次 監訳／朝倉書店 ISBN：4254121237

■リアルタイム・システムの構造化分析――リアルタイム・システムの仕様書作成手法／ Derek J.Hatlay、Imtiaz A.Pirbhai 著／立田種宏 監訳／日経BP社 ISBN：4822270750

■ Object Oriented SELECTION　UMLモデリングのエッセンス第３版 ／マーチン・ファウラー 著／羽生田栄一 監訳／翔泳社　ISBN：4798107956

■シュレイアー・メラー法によるオブジェクト・モデリング――リアルタイムシステムの静的解析法／レオン・スター 著／ Shlaer-Mellor 研究会 訳／ピアソン・エデュケーション　ISBN：4894710366

■組み込み UML―― eUML によるオブジェクト指向組み込みシステム開発／渡辺博之、渡辺政彦、堀松和人、渡守武和記 著／翔泳社　ISBN：4798102148

■思考系 UML モデリング即効エクササイズ――モデ力を鍛える 13 の自主トレメニュー／渡辺博之、芳村美紀、桑本茂樹、敷山喜与彦 著／翔泳社　ISBN：4798107123

■UML 動的モデルによる組み込み開発／渡辺政彦、飯田周作、石田哲史、山本修二、浅利康二 著／オーム社　ISBN：4274065138

■拡張階層化状態遷移表設計手法 Ver2.0 ―― Embedded SE のための設計手法／渡辺政彦 著／キャッツ　ISBN：4894690047

■リアルタイム UML　第２版――オブジェクト指向による組込みシステム開発入門／ブルース・ダグラス 著／渡辺博之 監訳／オージス総研 訳／翔泳社　ISBN：4881359797

■ソフトウェアテクノロジー シリーズ４ ソフトウェアアーキテクチャ――アーキテクチャとドメイン指向トラック／岸知二、野田夏子、深澤良彰 著／共立出版 ISBN：4-320-02777-9

■ UML2.0 Superstructure Specification Version2.0 ／ The Object Management Group（OMG）／ http://www.omg.org/docs/ptc/03-08-02.pdf

●執筆者

経済産業省 組込みソフトウェア開発力強化推進委員会 設計品質技術部会（2005 年度）

主査	岸　知二	北陸先端科学技術大学院大学
副主査	渡辺　政彦	キャッツ株式会社
	青木　奈央	IPA SEC ／キャッツ株式会社
	大野　克巳	IPA SEC ／トヨタテクニカルディベロップメント株式会社
	鈴木　弘一	日本テレロジック株式会社
	関本　康久	日本テレロジック株式会社
	竹垣　盛一	三菱電機株式会社
	中田　和美	オリンパスシステムズ株式会社
	橋倉　英樹	キヤノン株式会社
	松瀬　健司	株式会社リコー
領域幹事	平山　雅之	IPA SEC ／株式会社東芝
オブザーバ	野田　夏子	日本電気株式会社

●監修

組込みソフトウェア開発力強化推進委員会

組込みソフトウェア開発における品質向上の勧め［設計モデリング編］

2006 年 6 月 20 日　初版第 1 刷発行

編　著　者	独立行政法人 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
発　行　人	大槻利樹
発　行　所	アイティメディア株式会社 〒 100-0005 東京都千代田区丸の内 3-1-1 国際ビル 8F 電話 03-4500-4351　FAX 03-3284-0105 http://www.itmedia.co.jp/
印刷・製本	図書印刷株式会社

ISBN4-86147-011-0 Printed in Japan