伊藤忠テクノソリューションズ㈱　小麦田 哲也　KOMUGIDA Tetsuya
インダストリアル・エンジニアリング営業部インテリジェントシステム課に所属．医療機器，コンシューマー製品の組込み系開発経験を経て，MDDの最先端の状況を把握・習得するために2004年より，Telelogic製品の販売支援・サポート業務・ユーザの開発支援を行っている．

伊藤忠テクノソリューションズ㈱　徳江 愛　TOKUE Ai
インダストリアル・エンジニアリング営業部 インテリジェントシステム課に所属．1997年伊藤忠テクノサイエンス㈱（現 伊藤忠テクノソリューションズ）入社．2000年より，Telelogic製品の販売・サポートに従事し，国内のユーザに対して開発支援を行っている．

監修：HASHIMOTO SOFTWARE CONSULTING Inc.　橋本 隆成　HASHIMOTO Takanari
SEEI認定CMMIインストラクター．護衛艦の艦船搭載用弾道計算プログラム，新型戦闘指揮システム，新型射撃制御システムなどの大規模ハードリアルタイムシステムのソフトウェア開発に10年間従事．以後，日本ヒューレット・パッカード，オージス総研，ソニーにて製品開発業務，開発手法・方法論のコンサルティングおよびCMMIによるプロセス改善業務に従事．

オブジェクト指向によるMDD（Model Driven Development；モデル駆動開発）では，開発および管理プロセスが重要になってきます．近年は開発・管理プロセスが非常に重要視されており，効果的な開発を期待するには，適切な開発・管理プロセスを用いることが重要です．今回の企画にあたり，詳細に開発・管理プロセスを１度に解説していくことは誌面の都合上，困難ですので，オブジェクト指向によるMDDを３回に分けて解説していくことにします．

MDD

1 トリロジー企画

オブジェクト指向技術による 実践! モデル駆動開発

シーズンI ～要件定義編～

MDD Chapter 1

MDDが求められている理由

本企画の狙い

組込み開発の現場に今，必要なアプローチ

現在のソフトウェア開発は，多くの問題に直面しています．特に組込みソフトウェア開発は，その性質や特徴上，多くの制約や技術的課題の中で開発を行います．技術的課題が多く，性能面でも高い完成度が要求されます．

ご存知のように，日本は世界的な製造業大国です．ソフトウェアの開発も，ITシステムが注目されがちですが実のところ，組込みシステムやリアルタイムシステムと呼ばれる開発が主流でもあります．

しかし，これまで日本の組込みソフトウェア開発の世界は，十分と言えるような開発手法・方法論や開発プロセス，開発環境が利用されている状況にはありませんでした．

理由は，日本の開発現場で紹介されてきた多くの開発手法・方法論や開発プロセス，開発環境が，組込みシステムやリアルタイムシステムに特化したものではなかったこともあります．また，組込みソフトウェア開発が比較的小規模であり，開発現場も従来のやり方で対応が可能であったことも，こうした開発手法，開発プロセスの導入が加速しなかった理由のひとつだと思います．

ところがここ数年，これまでのような長年の状況が大きく変化しつつあります．組込みシステムやリアルタイムシステムの規模，機能の複雑性が急激に大きくなり，その上，製品の市場投入の期間は短縮化の一途を辿っているからです．また，インドやアジア諸外国との価格競争などにもさらされています．

このような状況の変化から，日本の開発現場は従来の開発スタイルでは行き詰まりを感じており，新しい開発アプローチへの期待が急速に高まっています．また世界的に見ると，このような状況に合わせて，ソフトウェア開発を取り巻くさまざまな問題を解決する効果的なアプローチも数多く登場しつつあります．代表的なものとしては「MDD（MDA）」，「プロダクトラインエンジニアリング」，「SEI-CMMI」，「アスペクト指向技術」などがあります．

これらの技術や標準化は，最新というわけではありませんが，それぞれ時間をかけて発展してきたものですし，実製品への実績もあります．ポイントは，上記のアプローチはいずれも，どれか1つを採用することでソフトウェア開発が直面している問題が解決するわけではなく，複合的な取り組みが重要となることです．事実，「オブジェクト指向技術」，「ユニファイドプロセスなどの開発プロセス」，「MDD（MDA）」，「プロダクトラインエンジニアリング」，「SEI-CMMI」，「アスペクト指向技術」などの各手法・技術・アプローチ自身，相互に影響を与えて発展しています．

組込みソフトウェアの開発現場で直面している問題を解決するには，単純な取り組みでは難しくなっているのが現状です．開発現場で問題となっている課題を明らかにして，解決策を真剣に検討することが必要です．多くの場合は，技術的な取り組みに加えてSEI-CMMIなどのプロセス改善などの活動も必要となるなど，複合的な活動を継続的に実施する場合があると思います．

本企画の狙い

組込みシステムやリアルタイムシステムの開発現場に，新しい技術が積極的に導入されてきてはいますが，まだ十分，情報や事例があるとは言えない状況です．組込みシステムやリアルタイムシステムの事例紹介は，開発製品の技術情報や企業の開発戦略を表に出すことになり，なかなか難しいことも事実です．

そこで本企画は，近年，特に現場での注目と導入の期待が高まっている，組込みシステムやリアルタイムシステムの開発環境を用いた，オブジェクト指向開発によるMDD（モデル駆動開発）を紹介することを試みました．実際の開発では，先に上げたアスペクト指向技術，プロダクトラインエンジニアリング技術，SEI-CMMIなどの他の技術の利用も重要であり，関連してきますが，いきなり多くを説明し，学習を進めていただくのは難しいと考えています．今回はオブジェクト指向開発によるMDDに焦点を絞って解説することにしました．

従来型組込み開発からMDDへの移行のための例題

本企画では上述のように，最近，組込みソフトウェア開発現場で取り組みが盛んになりつつあるオブジェクト指向開発を用いたMDDを紹介したいと思います．オブジェクト指向技術，MDDは，これまで組込みソフトウェア開発ではそれほど主流になっていませんでしたが，現在，多くの企業が積極的に導入を競っています．

一方で，これまで組込みソフトウェア開発で実施されてきたソフトウェア開発の技法やプロセスからオブジェクト指向技術やMDDに移行する際に，具体的なイメージがわからない，どのような設計書の成果物を作成することになるのか知りたい，組込みシステムを題材とした例題が欲しいなどの声があるのも事実です．

そこで本企画では，反復型の開発プロセスとモデリングツールであるCASEを用いたオブジェクト指向による開発を，MDDによって具体的にどのように進めていくのかを，ケーススタディを使って紹介していくことにします．

●MDDに対する取り組みが盛んな理由

近年，組込み分野では，UMLを用いたモデル駆動型開発に対する取り組みが盛んに行われていますが，その大きな背景としては，

- ソフトウェアの規模が大規模化・複雑化している
- Time to Marketの短縮
- 要件仕様の変更に対応する必要性

などの課題があります．これらの課題に対応するためには，より効率よく設計・開発を行わなければならず，その解決方法の1つとして，MDDが取り入れられています．

要件定義からソースコード生成までを解説

オブジェクト指向によるMDD開発では，開発プロセス・管理プロセスが重要になってきます．効果的な開発を期待するには，適切な開発・管理プロセスを用いることが重要です．しかしここで詳細に開発・管理プロセスを解説していくことは，誌面の都合上困難ですので，オブジェクト指向によるMDDを紹介するために必要な範囲で解説したいと思います．なお，本誌Vol.3特別企画1「ユビキタスコンピューティング時代の組込みリアルタイムソフトウェア開発のアプローチ［基礎］」で，組込みリアルタイムシステム開発と管理プロセスについて解説していますので，必要に応じて参照してください．

さて，開発・管理プロセスですが，本来，企業の開発現場の製品の特徴，企業文化などを考慮して決定されますので，読者のすべての方に対して最適なプロセスを紹介するのは限界があります．そこでここでは，近年のソフトウェア開発で主流になりつつある，反復型のプロセスを利用する想定のもと，話を進めます．ただし，誌面の都合上，反復ごとの技術的な作業手順と成果物を詳細紹介することは難しいことも事実です．そこで，反復の記述を省略し，要件定義からソースコード生成までの開発作業過程を，ケーススタディを使って示していくことにします．

「エレベータ制御システム」のケーススタディ

本企画では，MDDを用いた要件仕様の開発からモデルによるコード生成までを，「エレベータ制御システム」

の開発の流れというケーススタディを用いて紹介します．

●本企画の予定

このエレベータ制御システムは，実際の製品開発と比べて単純な仕様となっていますが，それでも要件仕様の開発からモデルによるコード生成までを紹介するのは多くのページを必要としますので，今号以降，次の3回に分けて解説する予定です．

●第1回（今号）

①開発環境「Rhapsody」の紹介（Chapter2）
②ケーススタディ「エレベータ制御システム」の説明（Chapter3）
③ハードウェアブロック図の作成と解説（Chapter4）
④ユースケースによる要件仕様記述の作成と解説（Chapter4）
⑤イベント分析（Chapter4）

●第2回

①オブジェクト指向によるシステム分析と分析モデルの解説
②アーキテクチャ分析の解説
③ユースケース駆動開発の解説

●第3回

①設計モデルの解説
②シミュレーション技術と紹介
③コード生成と解説

なお，3回に渡ってオブジェクト指向によるMDD開発を紹介するあたって，いくつかの制約があります．たとえば，実際の開発では戦略的な反復開発を含む開発プロセス，プロジェクト管理プロセスの話題，詳細な開発作業の話題など数多く取り扱うテーマを扱わなくてはなりません．本来ならば可能な限り，実開発で取り扱う内容を取り上げ，具体的な作業の実例を解説することが理想ですが，限られた誌面ですべてを紹介するのは困難ですので，本企画では焦点を当てて解説するテーマを絞ることにしています．組

MDD Chapter 2

Rhapsodyとは

組込み・RTシステム向けモデル駆動開発CASE

分析・設計開発環境CASE「Rhapsody」

組込みシステムやリアルタイムシステムにおける，オブジェクト指向開発によるMDDアプローチを紹介するに当たって，何らかの手法，開発環境を利用することが必要となります．

今回は，典型的なMDD開発環境であり，書籍[注1]も何冊か出版されている開発方法論「Harmony」をサポートする開発環境「Rhapsody」を利用することにしました．

開発環境Rhapsodyは，日本でも購入，サポート支援を受けることが可能なリアルタイムシステムのオブジェクト指向開発環境であり，MDD開発主導のアプローチを支援するツールです．詳しくはこの後で紹介しますが，現代的な開発プロセス，オブジェクト指向技術，開発環境上でのシミュレーションおよびモデルからターゲットの実コード生成までをサポートしています．さらに開発環境のリポジトリで管理されているモデル情報から簡単にドキュメント生成が可能となるなどの機能を備えています．

導入事例

Rhapsodyは，オブジェクト指向/UMLをサポートしたソフトウェア開発環境として，1999年にTelelogic社からリリースされた製品です[注2]．OA（Office Automation），消費者家電（Consumer Electronics），FA（Factory Automation；工場の自動化），自動車（Automotive）など，幅広い分野で採用されています．実際に，戦闘機（米国），プリンタ（日本）など，さまざまな規模のシステムの開発事例があります．さらに1部署ごとの採用規模としては，50～100ライセンス[注3]というような大規模での適用実績がここ数年飛躍的に伸びています．

Rhapsodyの特徴

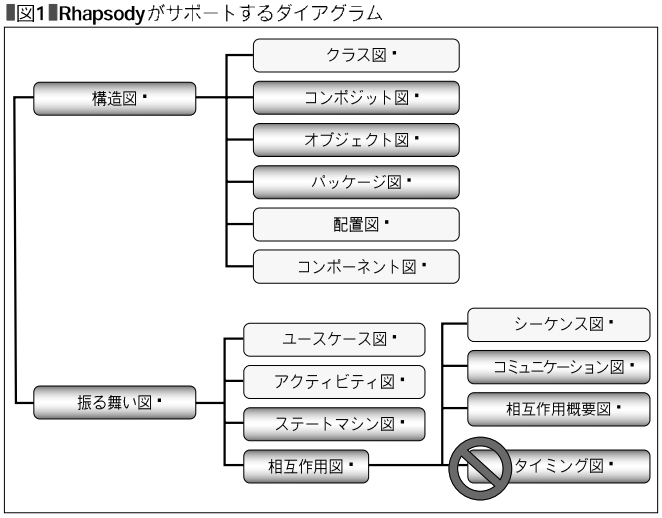
MDDは，モデル駆動型開発という名の示すとおり，モデルを中心に開発を進める方法で，最終的にはC，C++などの開発言語に落とし込むことができます．このMDDを実現する上で重要なポイントとなるのが，UMLのモデリングからコード生成までをサポートしているCASEツールです．本稿では，数あるCASEツールの中でも組込み開発に特化したRhapsodyを使って，MDDをベースとしたソフトウェア開発の進め方を紹介します．

Rhapsodyが持つ機能としては，

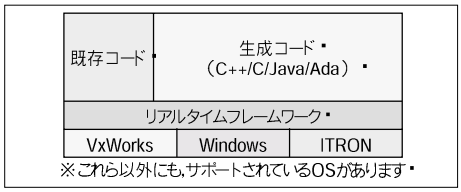
●UMLによるモデリング

注1●『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』／Bruce Douglass著／（株）オージス総研訳／渡辺博之監訳／翔泳社／ISBN4-8813-5979-7
『Real Time UML Workshop for Embedded Systems』／Bruce Douglass 著／Newnes／ISBN0-7506-7906-9
『Doing Hard Time : Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns』／Bruce Douglass 著
注2●当時の開発ベンダであるI-Logix社は，組込み分野向けの設計・検証環境を提供する会社として起業し，「ハレルの状態遷移図」をベースとした組込みシステム向けの設計環境StatemateMAGNUM（現Statemate）を販売していました．また，UMLの設立メンバ8社のうちの1つとして，UMLの標準化に大きく貢献した企業でもあります．日本国内では，伊藤忠テクノソリューションズ㈱（10月に会社名変更：旧伊藤忠テクノサイエンス㈱）が販売・サポートを実施しています．
注3●スタンドアロンライセンス，もしくは，ネットワークライセンス．ライセンス管理はFlexLMツールを使用．

■図1■Rhapsodyがサポートするダイアグラム



■図2■リアルタイムフレームワーク



- **コードの自動生成**
- **リアルタイムフレームワーク**
- **アニメーション機能**
- **リバースエンジニアリング**
- **ラウンドトリップ機能**
- **周辺ツール（要件管理ツール，構成管理ツールなど）との連携**
- **ドキュメント生成**

などが挙げられます．以下，具体的に説明していきます．

●UMLによるモデリング

Rhapsodyは，MDDのベースとなるモデルをUMLで記述し，コードジェネレータによってUMLのモデル情報を変換し，C/C++/Java/Adaの実装コードを生成させることができます．Rhapsodyは，UML 2.0の13あるダイアグラムのうち，タイミング図を除く12のダイアグラムをサポートしています（**図1**）．

各ダイアグラム上で記述されたUML要素は，モデルデータとして登録され，ツリー状に表示されるブラウザウインドウで確認することができます．

●コードの自動生成

Rhapsodyは，作成したモデルから，スケルトンではないC，C++，Java，Adaのソースコードを生成します．このとき，コード生成のベースとなるのが，システムの構造を表したクラス図やオブジェクト図，そして，システムの振る舞いを表現したステートマシン図やアクティビティ図です．ステートマシン図をコード生成の対象とすることで，組込みシステムの開発において重要なポイントとなる非同期イベントの処理を，モデルとして表現し，実装コードに落とし込むことができます．

●リアルタイムフレームワーク

組込みソフトウェア開発のさまざまなターゲットプラットフォームに簡単にポーティングするためのしくみとして，リアルタイムフレームワークが提供されています（**図2**）．このリアルタイムフレームワークを適用することで，Rhapsodyでは，設定を変えるだけで同じモデルからさまざまなプラットフォーム（RTOS）用のバイナリを生成させることができます．このリアルタイムフレームワークのソースコードは公開されているため，カスタマイズも可能です．

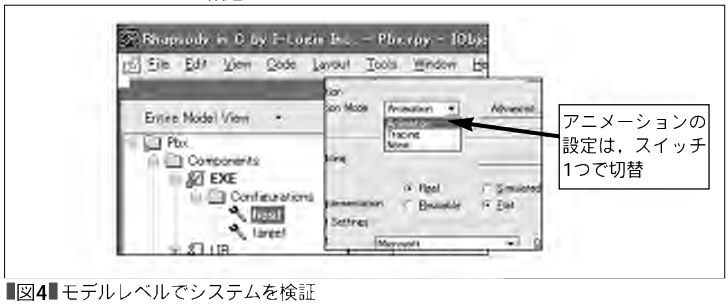
●アニメーション機能

モデリングしたものが正しく動作するか，検証するためには，モデルを実行してみる必要があります．Rhapsodyで記述したUMLモデルは，モデルの振る舞いの部分もコード生成の対象であるため，モデルからソースコードを生成・コンパイルして，それを動かしてみることができます．さらに，モデル上でのデバッグ機能として，アニメーション機能を提供しています（**図3**）．このアニメーション機能を利用すると，

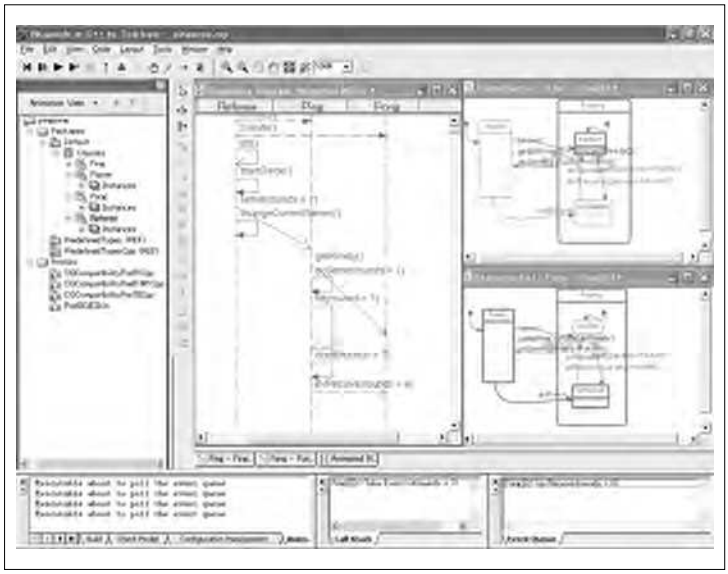
①ステートマシン図上で状態遷移の様子を確認する
②オブジェクト間のメッセージのやり取りをシーケンス図にロギングする

など，コードレベルではなくモデルレベルでシステムを検証することができます（図4）．

■図3■アニメーションの設定



■図4■モデルレベルでシステムを検証



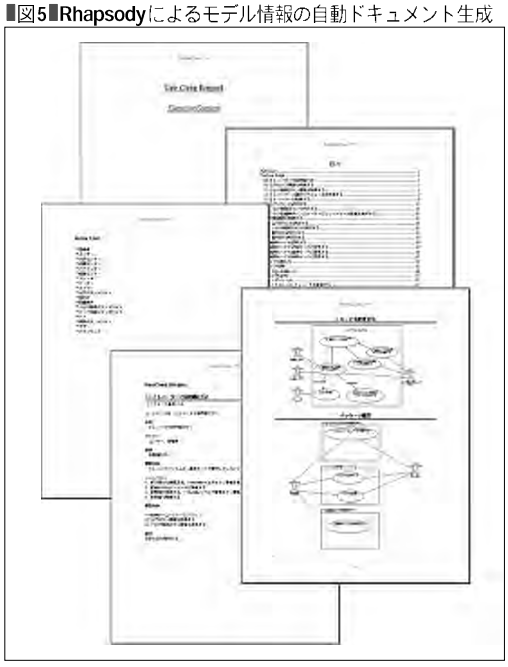
●ドキュメント生成

Rhapsodyには，作成したモデルデータから，さまざまなフォーマットの仕様書を作成するためのレポート生成機能が用意されています．各ダイアグラムやモデル要素の情報をどのように仕様書に記述していくかを指定したテンプレートが20ほど用意されており，それらのテンプレートを元に，WordやHTMLなどのドキュメントを生成させることができます（図5）．レポート生成用のテンプレートは，用意されている以外にも，専用のエディタを使って，ユーザ自身が簡単にカスタマイズしたり，あるいは新しく作成することもできます．

●リバースエンジニアリング

組込み機器開発は“レガシー（既存資産）”を使っての開発がほとんどです．MDDへのシフトを効率よく行うために，Rhapsodyは既存のソースコードをモデルとして変換するリバースエンジニアリング機能を持っています．これにより，既存のシステムのアーキテクチャ分析やシステムの再構築を強力にサポートします．

■図5■Rhapsodyによるモデル情報の自動ドキュメント生成



●ラウンドトリップ

モデルをベースとした開発を行っていくとはいえ，製品開発における実装は実装コード（ソースコード）をベースにして行います．ときとして，開発者はモデルではなくコードの変更を直接行い，モデルとコードの整合性が取れなくなるということが開発現場ではよくあります．Rhapsodyでは，コードで変更された内容を，モデル情報に反映させるラウンドトリップ機能によって，モデルでの開発とソースの修正を組み合わせた柔軟な開発が可能になります．組

MDD Chapter 3

ケーススタディの紹介

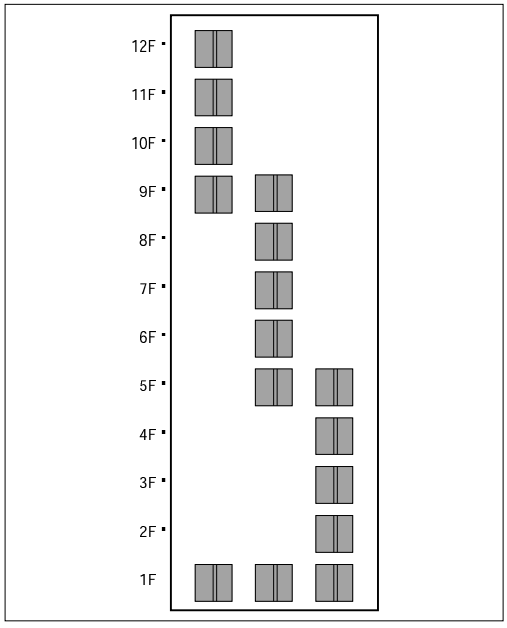
エレベータ制御システムの仕様

「エレベータ制御システム」をとおして紹介したいこと

今回のケーススタディは「エレベータ制御システム」にしました．エレベータ制御システムは，読者の皆さんも毎日のように利用されているエレベータを例題とすることで理解の助けになると考えたからです．

またエレベータ制御システムは，多くの書籍でもケーススタディとして取り上げられており，例題としては適している題材でもあります．読者の皆さんは，他の書籍でエレベータ制御システムの例題を見つけたときは，比較してみるとおもしろいと思います．

図1 3基のエレベータ停止フロア概略図



すでにお話しましたが，本企画ではエレベータ制御システムのケーススタディをとおして，組込みシステム向けソフトウェアのオブジェクト指向によるMDD開発，およびCASEツールである開発環境Rhapsodyによる自動生成技術・シミュレーション・ドキュメント生成などを紹介します．3回に渡る解説でMDDの特徴であるモデル検証による「シミュレーション」，「アニメーション」，「モデルからの自動ソースコード生成」，「モデルからの自動ドキュメント生成」をRhapsodyを使って紹介していきます．さらに，誌面の都合上，モデルをすべて紹介することはできないので，ケーススタディ「エレベータ制御システム」のモデル情報をとWebからダウンロードが可能にすることも検討中です．

エレベータ制御システムの仕様

ここでは，エレベータ制御システムの仕様として，システムの概要，フロア，エレベータ，その他の要件事項，エレベータの概念図，運行ルールとスケジューリング，OS・開発言語などの環境を説明します．

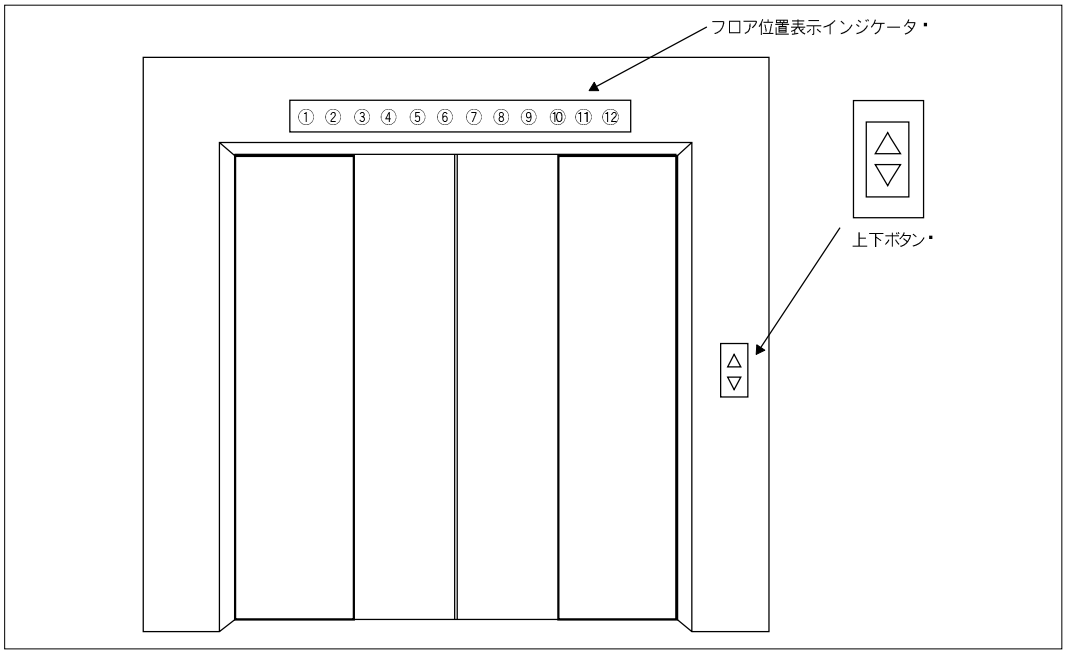
●エレベータ制御システムの概要

①ビルに地下・屋上はない．1Fから12Fまでのビルの，3基のエレベータ制御システムのソフトウェアである．エレベータは3基存在するが，それぞれ高層階，中層階，低層階に分けて運用する（図1）．

②高層階エレベータは1Fから9Fまでは直通，9Fから12Fまでは各階止まり．

③中層階エレベータは1Fから5Fまでは直通，5Fから

▌図2▐フロアのデバイスイメージ



9Fまでは各階止まり．

④低層階エレベータは1Fから5Fまで各階止まり．

●フロアに関する仕様

①各フロアには上／下（△または▽／）ボタンが3基のエレベータに対応して3セット存在する．ただし，1Fは上ボタン，12Fは下ボタンのみが存在する（**図2**）．

②3セットのうちどれか1セットの「△」または「▽」ボタンを押すと，押されたボタンが点灯し，同時に他の2セットのボタンも押されたボタンと同じボタンが点灯する．ただし，運用ルールに従うため，フロアに止まらないエレベータボタンは点灯しない．

③エレベータフロアに到着した場合は，対応した「△」または「▽」ボタンの点灯を消灯させる．

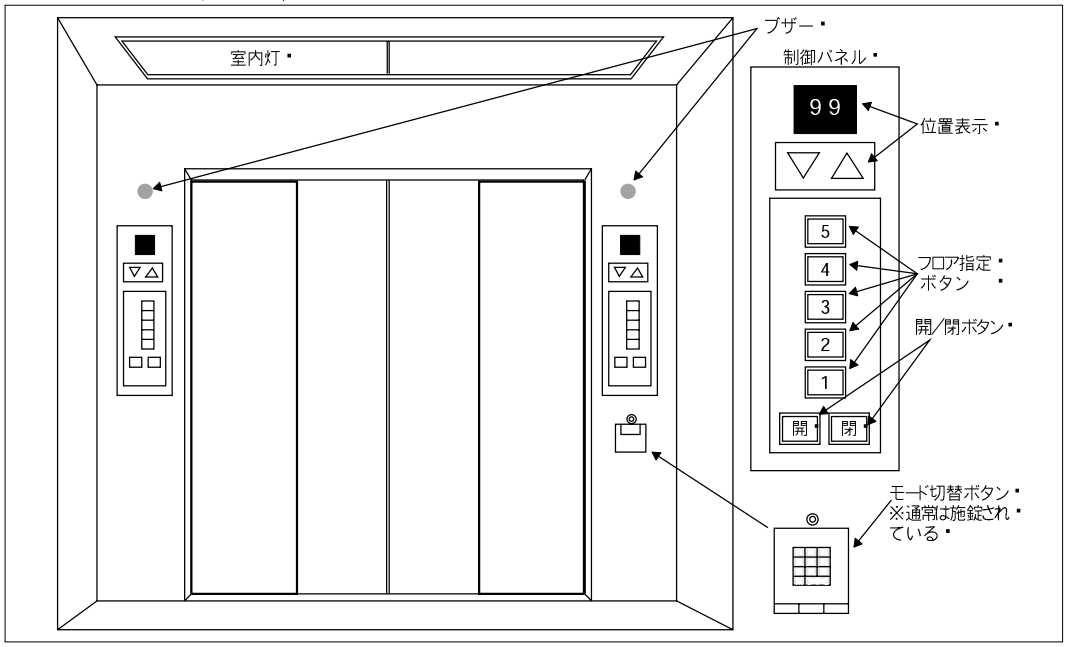
④3セットのうちどれか1セットの「△」または「▽」ボタンが押されると，エレベータ制御システムにエレベータをリクエストする．ただし，運用ルールに従わないリクエストは無視する．

⑤3基の各エレベータに対応したフロア位置表示インジケータが存在し，現在位置もしくは一番近いフロアの階の番号を点灯させる．

⑥3基のエレベータそれぞれ運用モードとして「点検モード」，「通常モード」，「非常モード」が存在する．各モードに関する仕様は次のとおり．

- 「点検モード」：エレベータ定期点検用．各フロアからの要求は受け付けない
- 「通常モード」から「点検モード」へのモード移行は「手動」のみ
- 「点検モード」から「通常モード」へのモード移行は「手動」のみ
- 「点検モード」から「非常モード」へのモード移行は「手動」および「自動
- 「通常モード」：通常のエレベータ運用モード
- 「通常モード」から「点検モード」へのモード移行は「手動」のみ
- 「点検モード」から「通常モード」へのモード移行は「手動」のみ
- 「通常モード」から「非常モード」へのモード移行は「手動」および「自動」
- 「非常モード」：エレベータが何らかの問題や故障で利用できないときのモード．地震や火災時もこのモードになる．「非常モード」時は，最寄りのフロアに停止し，ドアを開く．各フロアおよびエレベータ部（キャビン）からのフロア要求，ドアの「閉じ

■図3■エレベータ内部（キャビン）のデバイスイメージ



る」要求は受け付けない

- 「非常モード」から「通常モード」および「点検モード」へのモード移行は「手動」のみ
- エレベータは地震や火災を感知可能であり，感知した場合は「非常モード」へと遷移する

⑦火災を感知する火災センサがついており，温度が50度以上となると火災と判断する．

⑧地震を感知する地震センサがついており，震度3以上の揺れを感知した場合は，地震が発生と判断する．

●エレベータ内部（キャビン）に関する仕様

①各エレベータ内部（**図3**）には，行き先（フロア）を指定するボタンがある．押されたフロアの階のボタンは点灯させ，エレベータ制御システムに通知する．ただし，運用ルールによって止まらないフロアの要求のボタンが押されても無視する．

②目的のフロアにエレベータが到着した場合は，点灯している行き先（フロア）を指定するボタンを消灯させる．

③各エレベータは，フロア位置表示インジケータが存在し，現在位置もしくは一番近いフロアの階の番号を点灯させる．

④各エレベータにはドアの「開」，「閉」ボタンがある．押されたときは対応してドアを開く，または閉じる．同時に押された場合は，ドアを開く処理を優先する．

⑤エレベータフロアに停止中以外には，ドアは開かない．「開」ボタンが押されても無視する．

⑥ドアにはセンサがついており，物に触れるとドアが開く．

⑦タイマにより，ドアが開いている状態で，10秒で経過すると自動的にドアが閉まる．

⑧エレベータ部の天井には蛍光灯が付いており，エレベータ稼働中は点灯している．ただし，エレベータフロアのいずれかで停止中かつドアが閉まっている状態で30秒経つと蛍光灯は消灯する．再び蛍光灯が点灯するのは，エレベータ制御装置から要求があり，目的フロアに移動するときと，停止中のフロアの「△」または「▽」ボタンが押されてドアを開くときである．

⑨各エレベータは，規定の重量制限が設けられており，500kgまでである．そこで各エレベータは過重センサが取り付けられており，500kgを超える乗員または荷物がある場合には，ブザーを鳴らし，ドア

も閉まらないようにする．ブザーは500kg未満になるまで継続して鳴り続ける．

●その他の要件事項

①ビルが変わり，エレベータ数やフロア階数が変わっても，再利用できるエレベータ制御システムであること．これは，エレベータ制御システムに新たな機能が追加されても，複雑な改修作業なく拡張できるソフトウェアアーキテクチャを持つエレベータ制御システムであることを要求する．この要求には，利用するオペレーティングシステムや各種ハードウェアの対応も含まれる．

②エレベータ制御アルゴリズム（エレベータの割り当て含む）は，ビルのフロアの階数の数や運用パターンに応じた対応ができること．

③3基のエレベータはそれぞれエレベータを移動させるモータがビル最上部に設置されている．

④3基のエレベータはそれぞれブレーキが用意されており，各エレベータが減速，停止する場合にはブレーキ装置を利用する．ブレーキ装置はモータと同様にビル最上部に設置されている．

⑤各エレベータは過重センサがエレベータ部（キャビン）の床の設置されている．

⑥各エレベータドアには，センサが設置されており，人・物に触れると感知する．

⑦エレベータワイヤをモータで制御して移動させる．

⑧エレベータの目的のフロアまでの距離や移動速度の制御は，すべてエレベータ制御パラメータに入力された情報を利用して行う．

⑨エレベータおよびビルのフロアは地震や火災を感知可能なセンサが取り付けられている．

●エレベータの駆動系デバイス

ここでは，トラクション式の機械室ありタイプを想定しています．このタイプは，「かご（キャビン）」と「つり合いおもり」の重量をバランスさせ，上部に取り付けた巻上機（モータ，ブレーキ）で効率よく駆動する，エレベータでもっとも基本的なタイプです（図4）．

●エレベータ間のシステム関連

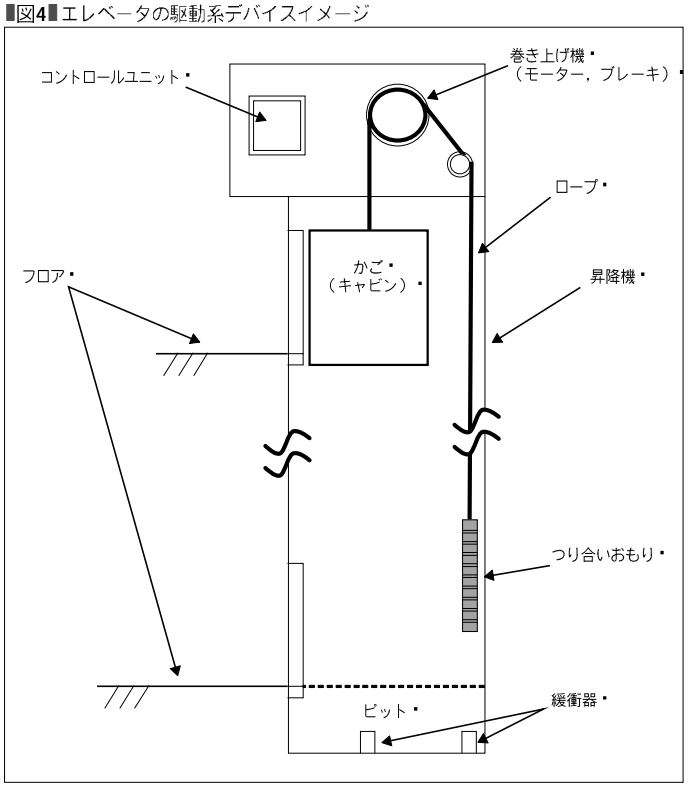
各エレベータにはコントロールユニットが存在し，ネットワークで接続され，お互いの位置や運行状態などの情報をやり取りします（図5）．ただしスレイブコントロールユニット同士が直接データのやり取りを行うことはなく，すべての情報は，マスタコントロールユニットが管理します．

●エレベータの運行ルール&スケジューリング

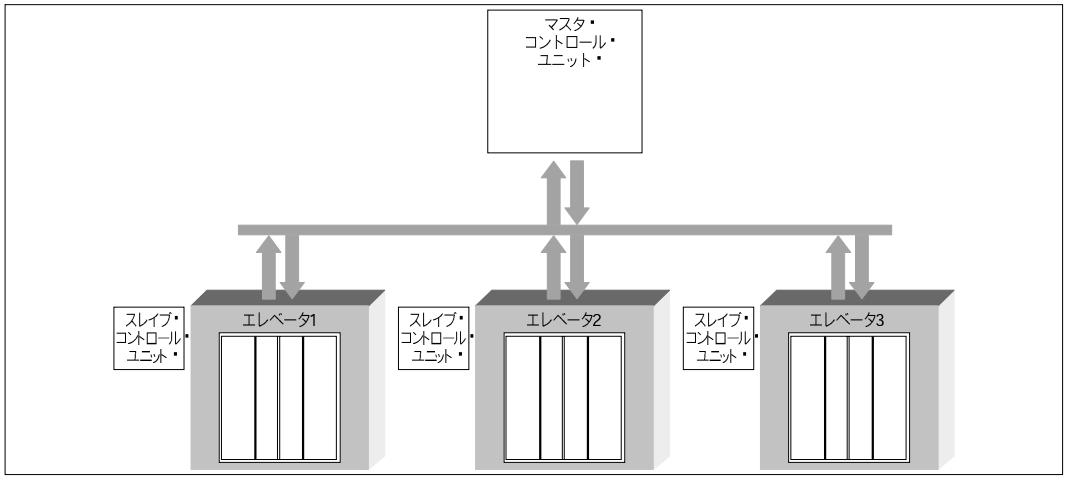
運行スケジュールを管理するテーブルは1次テーブル，2次テーブルの2つ用意します．

①フロアボタンが押された，または室内のフロア指定ボタンが押された場合は，運行スケジュールテーブルに停止予定階として登録する．ただし，エレベータの状態により登録されるテーブルが異なる．

■図4■エレベータの駆動系デバイスイメージ



■図5■ エレベータ間のシステム関連図



●エレベータ休止状態

リクエストのあった階を1次運行スケジュールテーブルに登録する．

●エレベータ停止状態

リクエストのあった階を1次運行スケジュールテーブルに登録する．

●エレベータ運行準備状態

リクエストのあった階がエレベータの進行予定方向と同じ場合は，1次運行スケジュールテーブルに登録する．進行予定方法と逆なら2次運行スケジュールテーブルに登録する．

●エレベータ運行状態

エレベータ進行方向にある階からリクエストがあった場合，その時点でエレベータ位置と4階以上離れていたら1次運行スケジュールテーブルに登録する．その他の場合は，2次運行スケジュールテーブルに登録する．

②エレベータ運行スケジュールテーブルに登録されている階に止まったら，停止予定スケジュールはテーブルから削除される．

③1次運行スケジュールテーブルに登録されていたすべての階に止まり，テーブルがすべて削除された場合，2次運行スケジュールテーブルに停止予定階が登録されていたら，2次運行スケジュールテーブルの内容を1次運行スケジュールテーブルにコピーし，2次運行スケジュールテーブルをクリアする．

●エレベータの進行方向

エレベータが休止・停止状態のときに，最初に運行スケジュールテーブルに登録された階のある方向に進行方向を設定する．

●利用環境

●リアルタイムOS

リアルタイムOSについては特定のOSを指定せず，一般的なリアルタイムOSが持つ機能を備えたリアルタイムOSを利用していると仮定します．

①リアルタイムOSを利用する．今回利用するリアルタイムOSは，タスクの優先度に基づくスケジューリング方式のプリエンプション（優先）機能を持つ．

②タスク間通信機，排他制御などのメカニズムは典型的リアルタイムOSが持つ機能を持つ．

③タスク優先度は十分な128の優先度が付与し，異なるタスクに同じ優先度を付与できる．同じ優先度のタスクのスケジューリングはラウンドロビンである．

●開発言語

開発言語は，C言語およびC++言語およびアセンブリ言語とします．注

MDD

Chapter 4

要件定義

ブロック図の作成とユースケースによる定義

エレベータ制御システムの要件定義

本章では要件定義について解説します．要件定義には多くの作業が含まれますが，ここでは，ユースケースによる要件定義の記述を中心に紹介したいと思います．

ユースケースについては，UMLの普及に伴い認知度がかなり高くなりました．ユースケースを取り扱った書籍も多岐に渡ってきていますので，「ユースケースとは何か」の具体的な紹介は割愛し，ユースケースによる要件定義についてのポイントを，例を用いて紹介していくことにします．

さて，要件定義では，組込みシステムにおいても開発対象のスコープ定義が重要です．組込みシステムの特徴として，ハードウェアへの考慮が必要ですので，「ブロック図」を用いることが効果的になります．そこでまずは，ブロック図についてみていきます．ブロック図がどのようなもので，どのように役立てられるのかを理解しておいてください．ブロック図による作業の次は，ユースケースによる要件定義を紹介します．ブロック図の作成と分析作業からユースケースを作成する入力情報が得られますが，どのように利用されるのかを見ていただきたいと思います．

ブロック図の作成

ブロック図を作成する目的

ブロック図にはいくつかの種類があります．よく作成されるブロック図として，「機能ブロック図」，「ハードウェアブロック図」などがあります．ブロック図の種類や開発チーム，開発製品により作成理由や目的は変わりますが，典型的なところではブロック図は，要件仕様定義作業と並行して，以下の目的のために作成します．

●ブロック図の作成目的とメリット

①システムの詳細にとらわれず，システムの全体像を掴む
②システムの要点を把握する
③システム（ハードウェア）構成を掴む
④システムに要求されている機能の概要を掴む
⑤外部およびシステム内部の処理の流れを掴む
⑥外部およびシステム内部のデータの流れを掴む

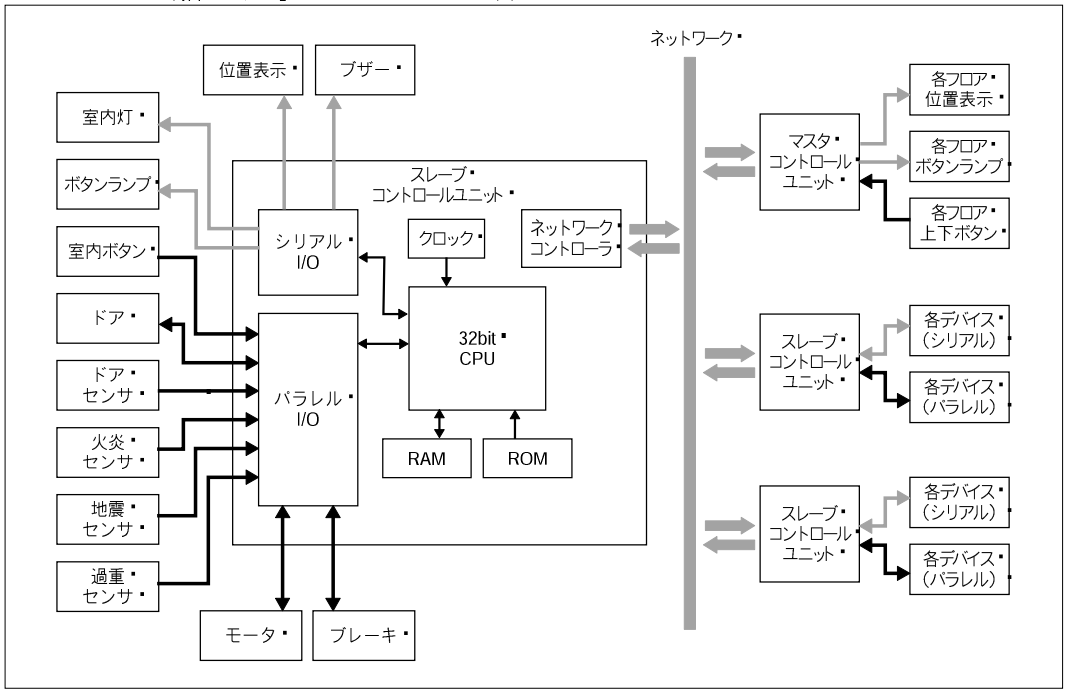
これらのことがわかりやすいように，目的にあったブロック図を何枚か作成します．

ブロック図の利用目的

ブロック図の種類に応じて作成理由や得られる情報は異なるものの，開発を進める上で大きく2つの情報に利用されていきます．

第1に従来の組込みシステム開発で利用されてきたように，開発対象となるシステムの機能の概要や処理・データの流れからタスク構成やモジュール分割を行うために利用します．システムに接続されているハードウェアデバイス類を明確にし，外部のデバイスとのI/Oデータ，イベントの分析を実施します．この作業

■図1■「エレベータ制御システム」のハードウェアブロック図



から得られた情報を元に，システムの大まかなタスク構成やモジュール分割を行ったら，それに基づいてタスク，モジュール間のデータの流れを整理します．

第2に，詳細な要件定義の入力情報になります．近年はユースケースによる要件定義が主流になりつつあります．このことは組込みシステム開発にも当てはまりますが，ビジネス系のシステムと異なり，ユースケースによる要件定義を実施する場合は，システムの持つハードウェア構成の理解と接続されている外部デバイス類の分析は不可欠になります．本稿では，どのようにブロック図の情報がユースケースによる要件定義の入力情報となっているかを解説していきます．

今回作成したブロック図について

最初は要件記述などを参照しながら，まずはデバイス（入出力）を抜き出します（表1）．

次に，抜き出した各デバイスが，システムとどのように接続されているか，入出力を考慮してブロック図にします．具体的に「エレベータ制御システム」のハードウェアブロック図を見ていきましょう（図1）．

「エレベータ制御システム」のハードウェアブロック図

●マスタコントロールユニット

各フロアの上下ボタン情報の取得やフロア位置表示などを管理します．各エレベータ運行スケジュール[注1]を管理

■表1■「エレベータ制御システム」のデバイス

●フロアにあるデバイス ・上下ボタン（入力） ・ボタンランプ（出力） ・エレベータ数表示パネル（出力） ●エレベータにあるデバイス ・フロア指定ボタン（入力） ・開／閉ボタン（入力） ・モード変更ボタン（入力） ・ボタンランプ（出力）	・エレベータ位置表示パネル（出力） ・ドアセンサ（入力） ・過重センサ（入力） ・地震センサ（入力） ・火災センサ（入力） ・室内灯（出力） ・ブザー（出力） ・ドア（入出力） ・モータ（入出力） ・ブレーキ（入出力）

注1●エレベータ運行ルール（189ページ）参照．

するのもマスタコントロールユニットです．各フロアからの要求と各エレベータからの要求を管理し，エレベータ運行スケジュールを決定し，エレベータの配車をします．

●スレーブコントロールユニット

エレベータボタン情報，各センサからの情報を取得し，マスタコントロールユニットに知らせます．マスタコントロールユニットが決定した運行スケジュールに従い，モータ・ブレーキを制御し，エレベータを運行します．

●パラレルI/O

各センサやモータ，ブレーキ，ドア，室内ボタンが接続されます．火災センサ，地震センサ，過重センサからはON/OFF信号と割り込み信号を検知します．割込みがあったセンサに対しては，500ミリ秒ごとにポーリングして情報を取得します．ドアセンサからは，割り込み信号を検知します．モータ，ブレーキは，制御パラメーターをI/Oポートに出力することにより制御します．モータ，ブレーキの動作状態は10ミリ秒ごとに監視します．ドアは，開閉信号をI/Oポートに出力することにより制御します．ドアの動作状態は，100ミリ秒ごとに監視します．

●シリアルI/O

ボタンランプや室内灯，ブザー，位置表示が接続されます．

ユースケースによる要件定義の記述

要件定義とは，開発するソフトウェアの機能要件や制約事項などを洗い出し，定義する作業を指します．要件定義の成果物が「要件仕様書」とか「要件定義書」と呼ばれるものであり，これらは開発するソフトウェアの機能要件や制約事項などを記述したドキュメントです．

近年，ソフトウェアが複雑になるにつれて，要件定義作業が非常に困難になってきました．複雑かつ大規模化するソフトウェアの機能や制約事項を，矛盾や過不足なく正確に記述することは大変な作業だからです．そこで，現在ではUMLにも定義されているユースケースを用いた要件定義が一般化しつつあります．ユースケースによる要件定義は，多くの優れた点を備えているからです．

機能の定義〜ユースケースによる定義

ユースケースは，正確にはユースケース図とユースケース記述（ユースケーススクリプトとも呼ぶ）の2つを指しています．「ユースケース図」は，UMLに定義されているアイコンを用いて作成するアクタと楕円で表現するユースケースの図です．一方，「ユースケース記述」は文章によるユースケースの詳細なシナリオの記述となります．

●ユースケース図の利用

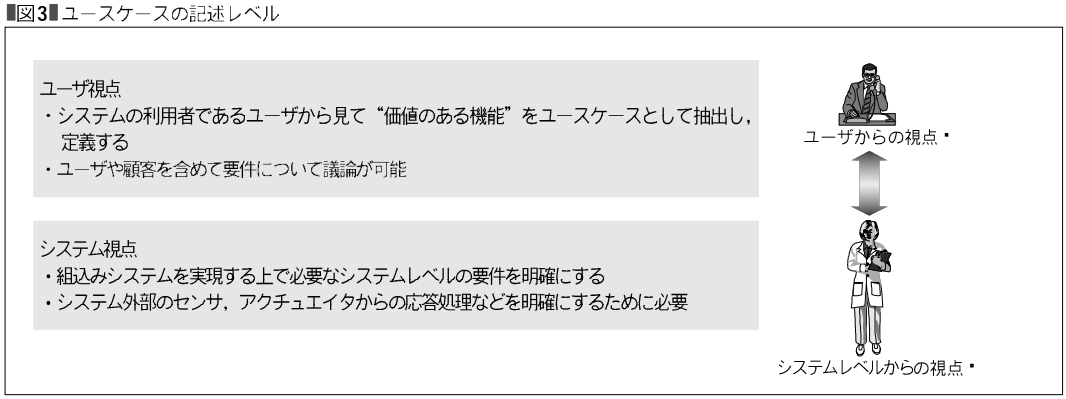
ユースケース図の狙いは，最初から詳細なユースケース記述を書く前に，システム全体を俯瞰しながら，どのようなアクタ（利用者，システムにアクセスする人・物）があるかを明確にし，システムに求められている要件があるかを洗い出していくことにあります．

ユースケースのメリットとしては，以下のような点が挙げられます．

①顧客およびユーザ視点による要件定義が行える．利用者，顧客がシステムに何を期待し，どのように使用するかの視点を見失わない
②顧客およびユーザ視点による要件記述ができるため，顧客やユーザを要件定義の作業に巻き込める
③ユースケース間の依存関係，共通の機能が明確にできる
④反復型開発を駆動する
⑤要件定義の機能とオブジェクト指向によるアーキテクチャを結びつける

ユースケースを開発基本単位として進めていく開発を「ユースケース駆動開発」と呼びます．反復型開発を用い，ユースケース駆動で開発を実施する場合は，明確にしたユースケースにプライオリティをつけ，反復型開発のときにリスクの高いところから作業を行うことが一般的です．反復型開発と言ってもユースケースが主流になる以前は，どのように反復開発を進めれ

■図3■ユースケースの記述レベル



ば良いのかの判断が不十分な状態でした．

さて，ユースケースを作成する際には基本となるステップがありますので，ご参考に紹介しましょう．

①システムを使用するユーザであるアクタを探す
②アクタがシステムとどのような関わり合いを持つか考える（システムに働きかける，システムから情報を受け取る）
③ユースケースがユーザ（アクタ）の視点で見たときに，ユーザ（アクタ）が期待する処理を表現させる．また，ユースケース1つで処理が完結する処理を表すようにする
④ユースケースが大きい場合は複数のユースケースに分けられないか考えてみる

●ユースケース記述の利用

ユースケース記述は，ユースケース図によって洗い出されたユースケースを，シナリオという形で詳細に記述していきますが，特徴はシステムの利用を常に意識しながら記述することです．

さて，組込みシステムではもう少し複雑です．人工衛星に搭載されているシステム，家電製品，AV情報家電などには，人の直接的な操作がなく動作するシステムがあります．これは，見方を変えると，システムに働きかけるアクタが人ではないということです．

ユーザ視点のユースケースを作成後，組み込みシステム特有の視点で，ユースケースを詳細化してみることが必要になります．このときのユースケースは，アクタは人ではなく，センサやアクチュエイタなどになることが多くなります（図3）．このことから，ユーザの視点ということだけでは，組み込みシステムはユースケースとして不十分になることが多くなります．ユーザの視点のユースケース図およびユースケース記述と，システム分析の視点のユースケース図およびユースケース記述を用意することになります．

さらに組込みシステムに特徴的なのは，正常系の処理よりも異常系の処理が非常に多いということです．正常系・異常系の処理をユースケースという単位で明確に記述し，定義するという点で，ユースケースは，従来の方法以上に要件定義と分析に優れています．各ユースケースにおけるこの正常系・異常系の処理を「シナリオ」と呼び，システムの処理の振る舞いを定義していきます．

非機能的要件の定義

ビジネス系のような非組込みシステムの開発手法や開発方法論を見ると，非機能的要件に対する記述が少ないことに気づきます．ビジネス系ではシビアなリアルタイム性やROM/RAMなどのリソース制約，フォールトトレランス性，ハードウェアの制約がないか，ほとんど開発に影響を与えないと考えて良いほどであるからです．そのためでしょう，多くの開発手法や開発方法論では，非機能的要件について考慮することがあれば，それを検討するのはかなり設計作業が進んでから扱うように書かれています．つまり，ビジネス系では非機能的要件は副次的な要件なのです．

しかし，組込みシステムでは違います．組込みシステムは性能重視の開発です．開発初期にこの非機能的要件に対する分析を確実に実施することが非常に重要になります．組込みシステムは，開発開始の段階で利用できる（しなければならない）ROM/RAMなどのリソース制約やリアルタイムOSの種類，開発言語，厳

しい処理応答時間であるリアルタイム性が課せられているからです（表2）．

そのため，この後の開発作業や特にアーキテクチャ設計，詳細設計で重要になりますので，この要件定義で非機能的要件を明確にしておくことになります．

非機能要件はユースケースでは書きづらい場合があります．多くは，特定のユースケースでなくシステム全体に対しての制約などの場合です．この場合は別途「非機能的要件」を整理した文書を作成することになります．ただし，非機能要件は特定のユースケース，つまり特定の機能に対しての制約もあります．その場合は該当するユースケース記述に記述することになります．

ユースケースで用いるテクニック

ユースケース図とユースケース記述にはいくつかのテクニックを用います．ここでは簡単に，エレベータ制御システムのユースケース定義に登場するものの中から「ユースケースの構造化」に関係するものを解説しておきます．ユースケースではユースケースの構造化を行うことがよくあります．このとき「包含」，「拡張」，「汎化」関係を利用するのですが，この関係間の違いと利用するときのポイントを紹介します．

さらにユースケースが大きくなるときに，ユースケースを効果的に分割したり，階層化したりしたいことがあります．そのとき用いるものに「ユースケースパッケージ」があります．ユースケースパッケージについても最後に解説をします．

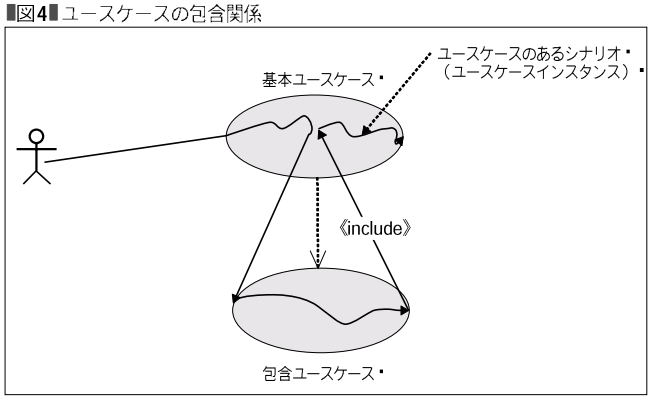
●包含関係（《include》）

包含関係は，基本ユースケースから包含されるユースケースとの関係です（図4）．包含関係によって，基本ユースケースが包含ユースケースに矢印で接続され，《include》で表現されます．矢印の向きが，この後紹介する拡張関係《extend》と逆になるので注意してください．

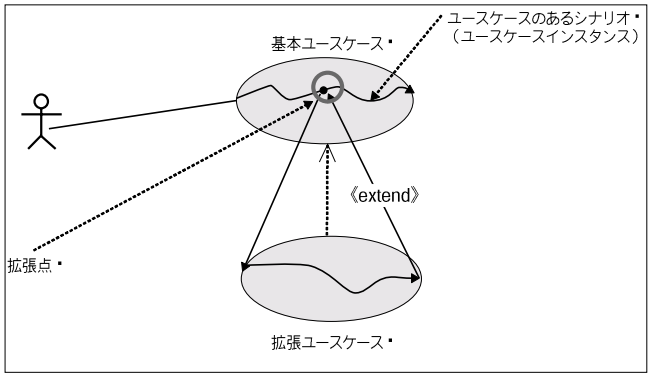
基本ユースケースには複数の包含が存在する可能性があります．また，1つの包含ユースケースは，複数の基本ユースケースに含まれる場合があります．このことは，基本ユースケース間の関係には関係ありません．同一の包含ユースケースと基本ユースケースの間で複数の包含関係が存在する場合もあり，包含が基本ユースケースの異なる場所に挿入されていることもあります．包含関係により，場所が定義されます．追加されるものはすべてネストされ，1つの包含ユースケースが別の包含ユースケースの基本ユースケースとなることもあります．

包含ユースケースは，共通の振る舞いを切り出したユースケースであるため，アクタが関連付けられている必要は必ずしもありません．包含ユースケースだけで，あるアクタにとって目的を達成する機能を提供する場合は，アクタへのコミュニケーションが発生し，関連が必要となります．

■図4■ユースケースの包含関係



■図5■ユースケースの拡張関係



■表2■組込みシステムで求められる主な非機能的要件

①プログラム言語
②処理応答時間などのシステムの処理に対するリアルタイム制約
③利用するリアルタイムOS
④ROM・RAMメモリ制約からのコードサイズの制約
⑤保守性や拡張性への要求

包含関係はもともと1つのユースケースの振る舞いを切り出したものですから，基本ユースケースと包含されるユースケースの2つを合わせることで意味のある振る舞いとなります．そのため，基本ユースケースだけでは意味を持ちません．

ただし，ユースケースは通常基本フロー，代替フローなどいくつものシナリオを持ちます．シナリオによっては，包含されるユースケースの振る舞いを実行しない場合もありえます．

●拡張ユースケース（《extend》）

拡張ユースケースから基本ユースケースへの関係で，基本ユースケースに定義した振る舞いに，ある条件のときのみ実施したい機能を追加するしくみを提供します（図5）．拡張関係によって，拡張ユースケースを基本ユースケースに「拡張点」を用いて《extend》関係で接続します．基本ユースケースにある拡張点を定義することで，拡張を挿入する位置を定義できます．拡張ユースケースを用いるのは拡張点を定義し，条件が必要となります．

基本ユースケースは，包含関係とは異なり，基本ユースケース自体で完結している必要がある点に注意してください．これは拡張関係にあるユースケースが，ある条件のときにのみ実施されるオプションだからです．また，基本ユースケースには1つだけではなく複数の拡張関係を持たすことが可能です．拡張関係にあるユースケースは，アクタから基本ユースケースが実行されて，拡張点が定義された位置に達すると条件をチェックします．条件が成立すれば拡張関係にあるユースケースの振る舞いが実行されます．拡張関係の条件が偽のとき，拡張ユースケースの振る舞いは実行されません．

●ユースケースの汎化

ユースケースの汎化では，子ユースケースから親ユースケースへの関係で，親ユースケースに対する振る舞いや特徴が子ユースケースによって具体化（特化）されることを定義します．親も子も必ずしも抽象ユースケースではないですが，汎化関係にある場合に親ユースケースは抽象ユースケースとして定義されることも多くあります．子は親の構造，振る舞い，関係をすべて継承します．この点に注意して，汎化関係を用いるか他の関係を用いるかを判断します．

同じ親を持つ子ユースケースは，どれも親ユースケースを具体化（特化）したものです．似たユースケースがいくつも登場したときは，汎化関係を用いる検討をする価値があります．ユースケースの汎化は，子となるユースケース群の共通部分を抽出します．そのため，クラスの汎化と同様に，しばしば抽象的なユースケースとして定義されることがあります．

子ユースケースでは親ユースケースの定義に依存しつつ，親ユースケースの振る舞いに固有の機能（振る舞い）を追加できます．

親ユースケースが抽象的なものである場合，その振る舞いは抽象的な振る舞いが記述されているために，そのままでは不完全であることに注意しましょう．特化される子ユースケースは，アクタにとって価値のある意味のある振る舞いを定義します．

なお，親ユースケースが抽象的なユースケースである場合，アクタと関係を持つ必要は必ずしもありません．

●ユースケースパッケージ

ユースケースパッケージは，ユースケース，アクタ，関係，図，そしてその他のパッケージを分割し，構造化するときに利用します．よほど小さなシステムでない限り，組込みシステムのユースケースは，かなりの数のユースケースが登場することが多くなります．ユースケースの粒度は，開発対象のシステムやユースケースの定義者の意向も影響しますが，それでも通常はかなりの数が登場します．

ユースケースパッケージは，要件定義の機能別やユースケースの定義の抽象度別にユースケースを管理したいときに便利です．外部に委託するときや，過去のプロジェクトから一部のユースケースを再利用するなどのときも便利です．

「エレベータ制御システム」のユースケース図の作成

ユーザ目的のユースケース

さて，ここからはケーススタディ「エレベータ制御シ

ステム」のユースケースを見ていくことにしましょう．

最初はユーザ目的（視点）のユースケース図から作成していきます．

エレベータ制御システムは，エレベータを利用するユーザに対して，目的階に運ぶというサービスを提供します．また，管理者によって3種類の運用モードに切り替えられるようになっており，定期的に点検を実施するなど，安全に運行できるように設計されます．

前者は，エレベータの乗客というアクタに対して，“エレベータ目的階に行く”というユースケースとなり，後者は，管理者というアクタに対して“運用モードを変更する”というユースケースとなります．

ユーザは，エレベータを利用して目的の階に行くだけでなく，他の人の乗り降りが済むまでドアを開いておき，乗り降りが済んだ時点で，閉ボタンでドアを閉める，などを操作することもあります．これらは，“ドアを開く”と“ドアを閉じる”というユースケースで表現します．ユーザレベルのユースケース図は，**図6**のようになります．

ユースケース図の作成で最初に登場するのはユーザ目的（ユーザ視点）のユースケース図です．このときは，「エレベータ制御システム」にどのような機能を利用者が求めているかという「視点」からユースケース図を描きます．アクタは利用者や管理者が登場しています．アクタである利用者や管理者がどのような機能をエレベータ制御システム」に機能として要求するか？を検討すると，図6のようなユースケース図ができ上がります．

次にユーザ目的（ユーザ視点）のユースケース図に対するユースケース記述（スクリプト）も作成しますが，今回は誌面の都合上，割愛します．

●システム分析レベルのユースケース

ユーザ目的のユースケースを作成した次は，システムレベルのユースケースを作成していきます．システムレベルのユースケースは，ユーザ目的のユースケースをサブ機能レベルに分割する際に，どのようにそのユースケースの目的が達成されるのかを考えます．

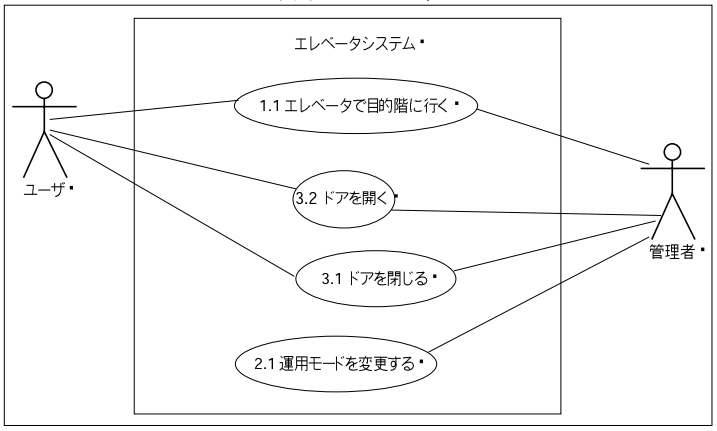
具体的には，「ハードウェアブロック図」などから，システムは実際に外部からどのような刺激（イベント，割り込みなど）を受けて，機能を実施するかを考えます．

組込みシステムは，飛行機や劇場のチケット予約システムなどと異なり，GUIを通じて処理が開始されるのではなく，ハードウェアデバイスから処理のトリガを受けることになります．

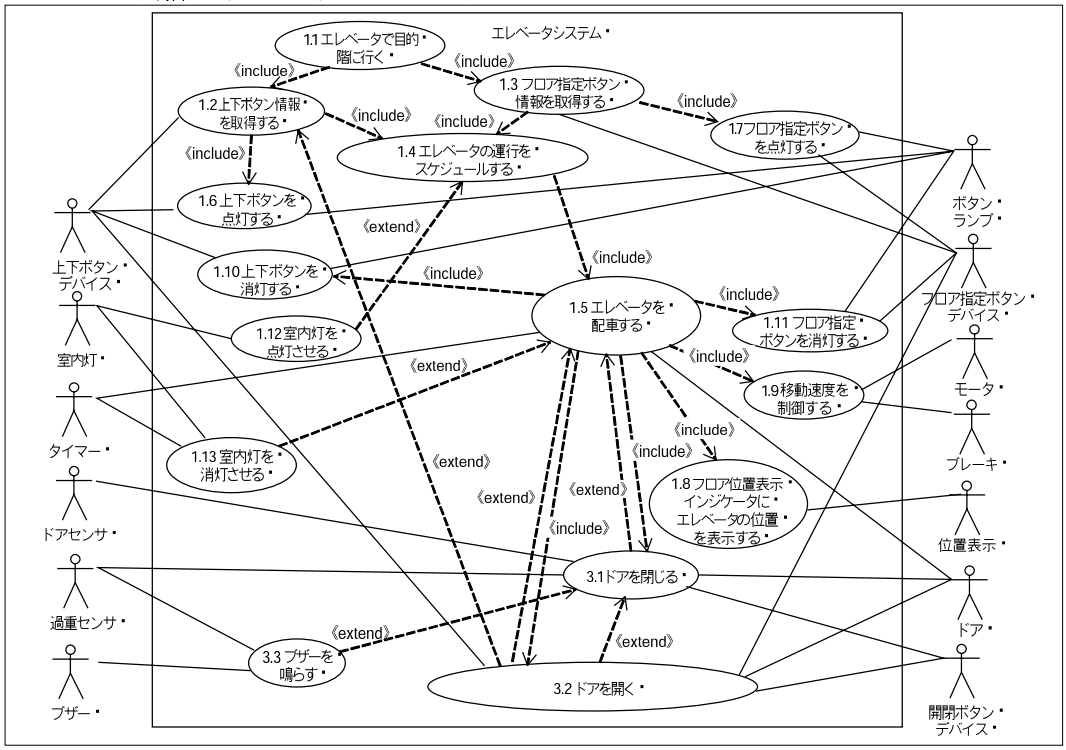
ユーザ目的のユースケースで登場するアクタは，実はハードウェアブロック図を見てわかるとおり，ボタンデバイスやセンサからの情報がシステムとやり取りすることがわかります．通常，組込みシステムは，ユーザ目的のユースケースで記述されているユースケースを実現するには，ユーザの視点からは見えない，システムレベルの機能が複数処理されて実現されることが多くなりますから，システムレベルの機能を的確に洗い出すことが重要です．システムレベルのユースケースは技術者の視点で，システムに必要な処理はどのようなものかを検討して作成します．「ハードウェアブロック図」を参考にシステムに接続されているデバイスをアクタとして検討してみるところから始めるのがヒントになります．

また，ユースケースのメインフローや代替フローの中で，ユースケースとして切り出せる処理があるかどうかもヒントになります．ユーザ目的レベルのユースケースがいくつかのサブ機能のユースケースとして切り出されます．本稿末の参考文献❶で紹介している「どのように／なぜ」の関係を利用するのも1つの効果

▮図6▮エレベータのユースケース図（ユーザレベル）



■図7■エレベータ制御システムのシステムレベルユースケース1



的な方法です．ここでは，“エレベータで目的階に行く”というユーザレベルのユースケースをどのように分割したかを紹介します．

“エレベータで目的階に行く”のユースケース記述では，アクタであるユーザが実行する4つのステップが記述されています．

①昇り降りを指定する
②配車されたエレベータに乗車する
③目的階を指定する
④目的階で降車する

このうち，ステップ1と2は，それぞれの次のステップまでの間にシステムレベルで処理されるシナリオのトリガとなります．ステップ1では，昇り降りを指定するために，ユーザはフロアに設置されている上下ボタンを押します．この処理は，システムレベルである“上下ボタン情報を取得する”というユースケースを呼び出し，上下ボタンの点灯やエレベータ配車などが実行されます．同様に，ステップ3に関係する処理は，“フロア指定ボタン情報を取得する”というユースケースで表現され，これらの関係は，図7で表現されています．この2つのシステムレベルのユースケースは，ユーザレベルのユースケース“エレベータで目的階に行く”にincludeされるサブユースケースとなります（“ユースケース記述：エレベータ目的階に行く”参照）．

ユースケースの分割は，ユースケース記述をまとめながら，アクタとユースケース間の関係や，ユースケース間の関係に矛盾がないかを確認しながら進めます．図7では，<<include>>と<<extend>>という2種類のユースケース間の関係が用いられています．<<extend>>関係は，あるユースケースの処理の中で，オプションとして実行されるユースケースがある場合に使用します．たとえば，“エレベータ運行をスケジュールする”というユースケースでは，エレベータリクエストされた階に配車する際に室内灯が消灯している場合には，点灯させなければなりません．このとき，“エレベータ運行をスケジュールする”と“室内灯を点灯する”という2つのユースケースの間には，<<extend>>関係が成り立ちます．

ユースケース間の<<include>>関係や<<extend>>関

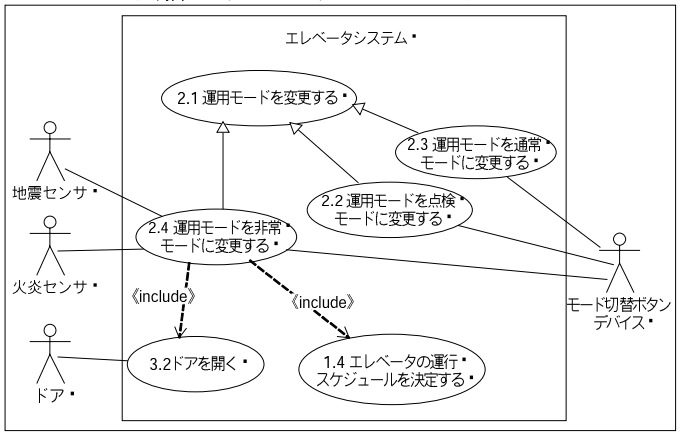
係をユースケース記述に記載しておくと，そのユースケースがどのような関係を持っているかがひと目で確認できるようになるため，非常に便利です．

図8は，“運行モードを変更する”というユーザレベルのユースケースを分割したシステムレベルユースケース図です．図7，図8のようなシステムレベルでのユースケース図では，アクタは対象システムのデバイスになります．

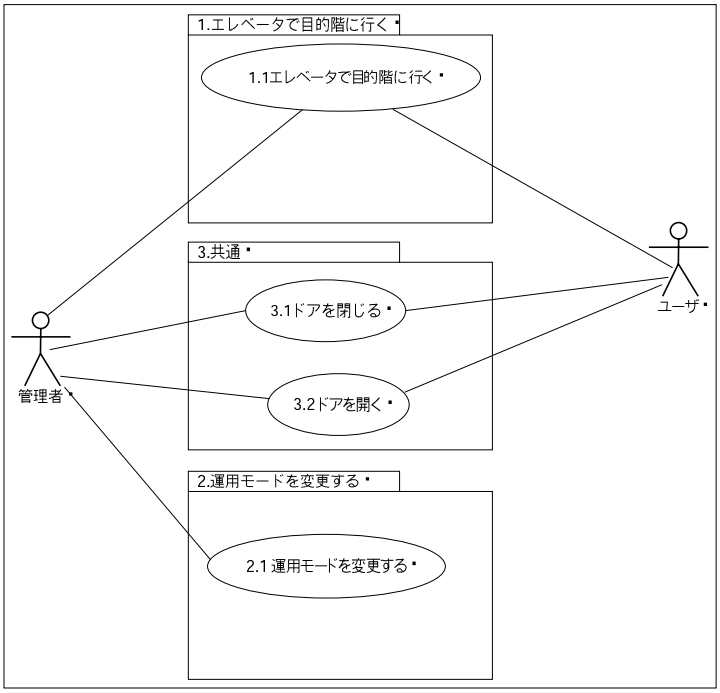
これまでに紹介した3つのユースケース図は，図7は図6のユースケース“エレベータ目的階に行く”に，図8はユースケース“運用モードを変更する”に注目して，その詳細な処理の内容を表現しています．つまり，この3つのダイアグラムは，特定のユースケースを分割するという点において，深く関係しています．このような場合には，各ダイアグラムに対して（あるいは，パッケージやユースケースに対して），関係するダイアグラムやモデル要素などをリンクさせ，モデル要素間で情報を追跡できるようにすることで，UMLモデルをよりわかりやすく表現することができます[注2]．

図6で表現されている4つのユーザ目的レベルのユースケースは，図7，図8のように，目的レベルがさらに詳細化され，サブ機能レベルのユースケースに分けられました．ここで，分割されたサブ機能レベルのユースケースをまとめるためのパッケージを考えます．このようなパッケージを用意することで，サブ機能レベルのユースケース群がどのユーザ目的レベルのユースケースの処理を表現したものかを，明確に表すことができます．また，このように問題範囲ごとにまとめることで，この後の設計プロセスにおいて，開発作業を分担しやすくなるというメリットがあります．この例では，“エレベータで目的階に行く”と，“運用モードを変更する”という2つのパッケージと，さらにこの2つのユースケースに共有されるユースケース“ドアを閉じる”と“ドアを開く”を，共通パッケージにまとめます（図9）．

■図8■エレベータ制御システムのシステムレベルユースケース2



■図9■ユースケースパッケージ図



エレベータ制御システムのユースケース記述の作成

実際にエレベータ制御システムのユースケース記述を作成していきましょう．次のようになります．

注2●Rhapsodyでは，モデル要素を関連する情報にリンクさせるためのハイパーリンク機能が用意されています．この機能では，Rhapsodyのモデル要素を，モデル内の情報だけでなく外部のドキュメントなどにもリンクさせることができます．

●ユースケース記述：エレベータ目的階に行く

ユースケース番号：
1.1
ユースケース名：エレベータで目的階に行く
目的：
エレベータで目的階に行く
アクタ：
ユーザ，管理者
説明：
目的階に行く
事前条件：
エレベータ制御システムが，通常モードで動作していること
メインフロー：
1.昇り降りを指定する．<<include>>上下ボタン情報を取得する
2.配車されたエレベータに乗車する
3.目的階を指定する．<<include>>フロア指定ボタン情報を取得する
4．目的階で降車する
<<include>>ユースケースリスト：
1.2上下ボタン情報を取得する
1.3フロア指定ボタン情報を取得する
制約：
止まらない階がある

●ユースケース記述：上下ボタン情報を取得する

ユースケース番号：
1.2
ユースケース名：
上下ボタン情報を取得する
アクタ：
上下ボタンデバイス
目的：
ユーザが押した上下ボタン情報を取得する
説明：
ユーザが押した上下ボタン情報を取得し，そのボタンの押されたフロアにエレベータリクエストする
事前条件：
エレベータ制御システムが動作している，かつ，通常モードである
メインフロー：
1.乗客が押した上下ボタン情報を取得する　（A1）
2.上下ボタンを点灯する <<include>>上下ボタンを点灯する
3.上下ボタンが押された階にエレベータ配車するようにスケジュールする<<include>>エレベータ運行をスケジュールする
代替フロー：
A1.上下ボタンが押された階にエレベータ停車中の場合：<<extend>>ドアを開いて処理を終了する
事後条件：
上下ボタンを押した乗客がいるフロアにエレベータを配車する
<<include>>ユースケースリスト：
1.6上下ボタンを点灯する
1.4エレベータ運行をスケジュールする
<<extend>>ユースケースリスト：
3.2ドアを開く

●ユースケース記述：エレベータ運行をスケジュールする

ユースケース番号：
1.4
ユースケース名：
エレベータ運行をスケジュールする
アクタ：
上下ボタンデバイス，フロア指定ボタンデバイス
目的：
乗客からの要求に応じて，エレベータ配車を計算する
説明：
ユーザの上下ボタン，フロア指定ボタンの操作から，3基のうちから最適なエレベータ，指示のあったフロアに配車するようにスケジュールする
事前条件：
配車がリクエストされたこと
メインフロー：

1.各フロアからの要求と，各エレベータらの要求を取得して，各エレベータ停車フロアを計算する（A1．A2）
2.各エレベータを配車する <<include>> エレベータを配車する
代替フロー：
A1.非常モードの場合：最寄りのフロアに配車する
　点検モードの場合：無視する．
A2.室内灯が消灯している場合には，<<extend>> 室内灯を点灯させる
事後条件：
リクエストされた階に1基のエレベータ配車する
<<include>> ユースケースリスト：
1.5エレベータを配車する

<<extend>> ユースケースリスト：
1.12室内灯を点灯させる
制約：
止まらない階がある
エレベータフロアに停止中以外には，ドアは開かない（制約1）
運行ルールを参照すること．

●ユースケース記述：エレベータを配車する

ユースケース番号：
1.5
ユースケース名：
エレベータを配車する

アクタ：
タイマー，ドア

目的：
エレベータを配車する

説明：
エレベータを，指示のあったフロアに配車する

事前条件：
配車がリクエストされたこと

メインフロー：
1.ドアが開いていたら<<extend>> ドアを閉じる
2.各エレベータの<<include>>移動速度を制御し，それぞれ目的の階に配車する
3.<<include>> フロア位置インジケーターに各エレベータの位置を表示する
4.目的のフロアに到着したことを確認する
5.対応した<<include>>行き先ボタンを消灯する
6.対応した<<include>>上下ボタンを消灯する
7.<<include>> ドアを開く
8.ドアが完全に開いた状態になったことを確認し，Doorタイマーをセットする（A1）
9.Doorタイマーで10秒経過していることを確認し，<<include>> ドアを閉じる
10.Idleタイマーをセットし，再度リクエストが来るまでカウントする（A2）

代替フロー：
A1.開ボタンが押される：<<extend>> ドアを開く
　閉ボタンが押される：<<extend>> ドアを閉じる
A2.<<extend>> 室内灯を消灯させる

事後条件：
リクエストされた階に1基のエレベータを配車する

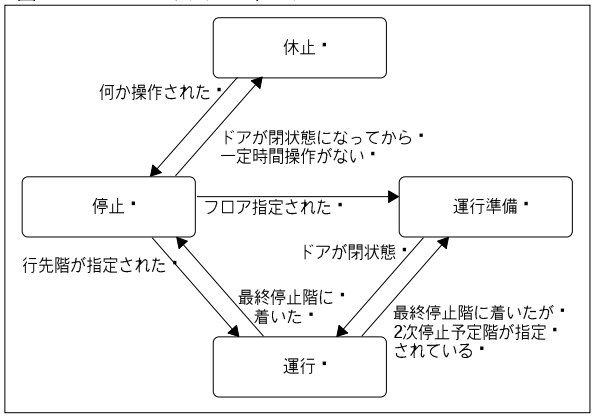
<<Include>> ユースケースリスト：
1.9移動速度を制御する
1.8フロア位置表示インジケータにエレベータの位置を表示する
1.11フロア指定ボタンを消灯する
1.10上下ボタンを消灯する
3.2ドアを開く
3.1ドアを閉じる

<<extend>> ユースケースリスト：
3.2ドアを開く
3.1ドアを閉じる
1.12室内灯を点灯させる

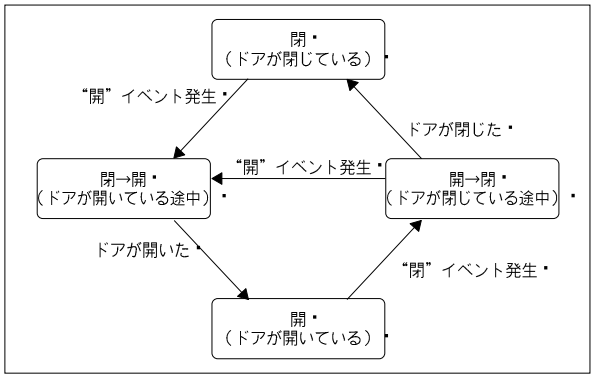
1.13室内灯を消灯させる

制約：
止まらない階がある
エレベータがフロアに停止中以外には，ドアは開かない（制約1）

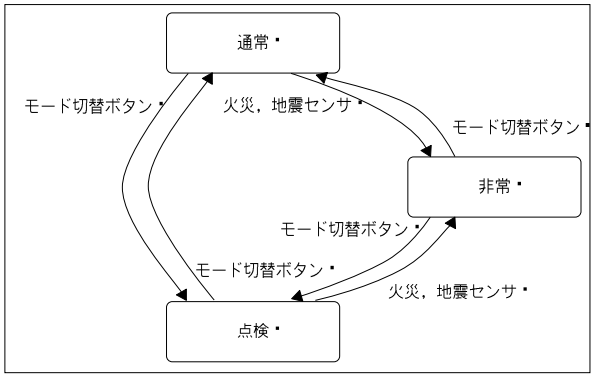
▌図10▐エレベータ（キャビン）の状態



▌図11▐ドアの状態



▌図12▐エレベータ運用モード



イベント分析

要件定義の作業には「ハードウェアブロック図」から「イベント分析」を行うことも有効です．イベント分析を行う際に，エレベータ制御システムの条件として必要になるエレベータの各状態を明確にして，状態図として表現・定義します．

「ハードウェアブロック図」のところで解説しましたが，ハードウェアブロック図およびイベント分析は，従来の組込みシステム開発で利用されてきたように，開発対象のシステムの機能の概要や処理・データの流れからタスク構成やモジュール分割を行うために利用します．システムに接続されているハードウェアデバイス類を明確にし，外部のデバイスとのI/Oデータ，外部とのイベントのやり取りから，システムの大まかなタスク構成やモジュール分割を実施する上で必要となるからです．設計ではそれに基づいてタスク，モジュール間のデータの流れを整理し，具体的なタスク設計，タスク間通信の詳細設計と実装を行っていきます．

エレベータ（キャビン）の状態

エレベータ（キャビン）の状態は図10～12のようになります．

イベント分析一覧表

イベント分析の一覧表は，表3のとおりです．

次回は

第1回目はケーススタディの紹介と要件定義のポイントを紹介しました．要件定義作業にはまだ数多くの作業がありますが，本稿ではブロック図，ユースケースおよびイベント分析の作業をピックアップして解説しています．なお，ここではケーススタディであるエレベータ制御システムのユースケース記述を作成しましたが，誌面の都合ですべてのユースケースについて掲載できませんでした．今後，読者の皆さんには何らかの形でユースケース記述を含む，モデル図などを提供することを検討しています．

さて，次回は分析作業に焦点を当てて解説を展開していきます．第2回および第3回は，MDDによる開発作業で最も中心的な作業となる内容を紹介する予定です．組

参考文献

❶『ユースケース実践ガイド―効果的なユースケースの書き方』／Alistair Corkburn著／ウルシステムズ㈱，山崎耕二，矢崎博英，水谷雅宏，篠原明子訳／ウルシステムズ㈱監修／翔泳社／ISBN4-7981-0127-3

❷『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』／Bruce Douglass著／㈱オージス総研 訳／渡辺博之 監訳／翔泳社／ISBN4-8813-5979-7

▌表3▐イベント分析一覧表

デバイス	イベント	エレベータ運行モード	エレベータ状態	ドア状態	条件	動作
フロアボタン	上下ボタン	通常	休止	―	エレベータが同じ階にある	・室内灯を点ける ・エレベータを“停止”状態に移行する ・ドアを開く
			停止	―		・ドアを開く
		通常	休止	―	エレベータが別の階にある	・ボタンの押された階を運行スケジュールに登録 ・室内灯を点ける ・エレベータを“運行”状態に移行する ・フロア指定ボタン点灯 ・エレベータ位置表示 ・運行スケジュールに従いエレベータを運行する
			停止	開		・ボタンの押された階を運行スケジュールに登録 ・エレベータを“運行準備”状態に移行する
				閉		・ボタンの押された階を運行スケジュールに登録 ・エレベータを“運行”状態に移行する ・運行スケジュールに従いエレベータを運行する
			運行準備	―		・ボタンの押された階を運行スケジュールに登録
			運行	―		・ボタンの押された階を運行スケジュールに登録
室内ボタン	フロア指定ボタン	通常	休止	―	―	・ボタンの押された階を運行スケジュールに登録 ・室内灯を点ける ・エレベータを“運行”状態に移行する ・フロア指定ボタン点灯 ・エレベータ位置表示
			停止	開		・ボタンの押された階を運行スケジュールに登録 ・エレベータを“運行準備”状態に移行する
				閉		・ボタンの押された階を運行スケジュールに登録 ・エレベータを“運行”状態に移行する
			運行準備	―		・ボタンの押された階を運行スケジュールに登録
			運行	―		・ボタンの押された階を運行スケジュールに登録
	“閉”ボタン	通常	休止	―	―	・室内灯を点ける ・エレベータを停止状態に移行する
			停止	開		・ドアを閉じる
			運行準備	開		・ドアを閉じる
	“開”ボタン	通常	休止	―	―	・室内灯を点ける ・エレベータを“停止”状態に移行する ・ドアを開く
			停止	閉		・ドアを開く
			運行準備	開→閉		・ドアを閉じる
	点検モード	通常	―	―	―	・点検モードにする
	通常モード	点検			―	・通常モードにする
		非常			モード変更許可状態	

（次ページに続く）

■表3■イベント分析一覧表（続き）

デバイス	イベント	エレベータ運行モード	エレベータ状態	ドア状態	条件	動作
過重センサ	ON	通常	停止 運行準備	閉	―	・ブザーON ・閉るボタンを無効にする
			停止 運行準備	開→閉		・ブザーON ・ドアを開く ・閉るボタンを無効にする
	OFF	通常	停止 運行準備	開→閉	―	・ブザーOFF ・閉じるボタンを有効にする
火災センサ	ON	通常 点検	運行	―	―	・進行方向の最寄階を運行スケジュールに登録 ・非常モードにする
			運行以外	閉		・非常モードにする
				―		・ドアを開く ・非常モードにする ・モード変更を“禁止”状態にする
	OFF	非常	―	―	―	・モード変更を許可状態にする
地震センサ	ON	通常点検	運行	―	―	・進行方向の最寄階を運行スケジュールに登録 ・非常モードにする
			運行以外	閉		・非常モードにする
				―		・ドアを開く ・非常モードにする ・モード変更を“禁止”状態にする
	OFF	非常	―	―	―	・モード変更を“許可”状態にする
ドアセンサ	ON	非常以外	運行以外	開→閉	―	・ドアを開く
ブザー	ON	―	―	―	―	・ブザーを鳴らす
	OFF					・ブザーを止める
室内灯	ON	―	―	―	―	・室内灯を点ける
	OFF					・室内灯を消す
位置表示	ON	―	―	―	―	・現在エレベータがある階を表示する
	OFF					・表示を消す
ボタンランプ	ON	―	―	―	―	・ボタンのランプを点ける
	OFF					・ボタンのランプを消す
ドア	開く	―	運行以外	開く以外	―	・ドアを開く ・ドアの状態を“閉→開”にする
	閉じる	―	―	開		・ドアを閉じる ・ドアの状態を“開→閉”にする
	開いた	通常	―	閉→開		・タイマイベント（ドア閉）セット ・ドアの状態を“開”にする
	閉じた		停止	―		・タイマイベント（休止移行）セット ・ドアの状態を“閉”にする
			運行準備	―		・エレベータを“運行”に移行する ・ドアの状態を“閉”にする ・運行スケジュールに従いエレベータを運行する
タイマ	Door	通常	停止	開	イベントがセットされてから10秒経過	・ドアを閉じる ・タイマイベントセット
	Idle			閉	イベントがセットされてから30秒経過	・室内灯を消す ・エレベータの状態を“休止”へ移行する
モータ	作動	通常	運行	―	モータが停止中	・モータを作動させる
	停止				モータが作動中	・モータを停止させる
	ステータス確認				―	・モータのステータスを確認する
ブレーキ	作動	通常	運行	―	―	・ブレーキを作動させる
	解除					・ブレーキを解除する
	ステータス確認					・ブレーキのステータスを確認する