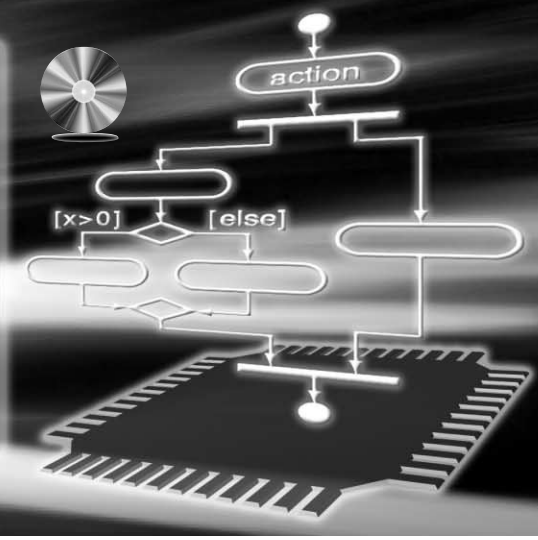
新連載/第1回

組み込みにも対応し始めたUML2.0の行方を探る

―UMLの概要とUML2.0で拡張された部分

坂本 武志



近年の組み込みシステムは，CPUをはじめとする各種ハードウェア性能が飛躍的な向上し，制御対象そのものが複雑で多様になっています．また，システム（製品）に要求される機能が高度になり，ソフトウェアの開発規模が飛躍的に大きくなってきています．さらに，製品の低価格を実現するためにハードウェアの汎用化が進んでおり，制御する側のソフトウェアが複雑になってきています．

その一方で，市場やコンペチタのグローバル化などにより，企業間の開発競争がより激しくなり，新製品の市場へのタイムリな投入，頻繁なモデル・チェンジも要求されています．このため，製品自体の寿命が短くなり，一つのモデルの開発にかけられる期間もどんどん短くなってきています．そして，複雑になってきたソフトウェア開発の遅れが製品リリースのタイミングに致命的な影響を及ぼしてしまう場合もあります．

● ソフトウェア開発における諸々の問題を解決するには

ソフトウェア開発規模の拡大および複雑化と開発期間の短縮，さらには品質の確保…これらの相反する要求に応えるためにはどうすれば良いのでしょうか．

一つの方法として，開発者の人数を増やすことが考えられます．開発チームの規模を拡大し，開発対象のソフトウェア規模が拡大した分をカバーするという考え方です．この方法は，開発対象がある程度の規模までで，一部の開発フェーズに限れば有効な場合もあります．しかし，開発者の数が増えると，開発者間のコミュニケーション・パスが飛躍的に増えてしまうため，実際の開発以外のオーバヘッド（打ち合わせなど）は増加します．その結果，各開発者の開発効率が下がり，場合によっては全体としてマイナスの効果となってしまうこともあります．

もう一つの方法としては，各開発者の開発効率を上げることが考えられます．開発におけるむだをなるべく省き，同じ期間でより多くの成果物を作成できるようにすることで，ソフトウェア規模が拡大した分をカバーするという考え方です．

● 製品開発時のハードウェアとソフトウェアの問題点

一般に製品のバージョンアップを行うときなどは，ゼロからすべてのコードを作り直すのではなく，以前使用したコードをできるだけそのまま流用することが多いと思います．これは，過去の資産をできるだけ再利用することで，新規開発部分をなるべく抑えようとするためです．しかし，ここにも落とし穴があります．

本来の設計者とは違う人が意味もわからずに再利用してしまうと，不具合が発生した場合にそれを解決できない，というパターンです．このパターンに陥ってしまうと，不具合の原因を突き止めるまでに通常よりも時間がかかってしまい，かえって開発効率が落ちてしまうこともあります．

組み込みシステムのソフトウェアは，ハードウェアへ依存する部分が多くあります．そのため，効率的に再利用するためには，ハードウェアへ依存する部分と，そうでない部分の切り分けをきちんと行っておく必要があります．この切り分けが十分でないと，ハードウェア変更時にほとんどすべてのソフトウェアを作り直さなければならない，といった事態にもなってしまいます．

また，ソフトウェア開発では，一般に工程が後になればなるほど，手直しに必要な時間やコストは増大します．そのため，下流工程からの「手戻り」が発生してしまうと，大幅なロスとなってしまいます．したがって，開発の効率を上げるためには，より上流の工程で十分な検討を行い，成果物の精度を上げることで，不具合や手戻りの発生をなるべく抑える必要があると言えます．

この上流工程の精度を上げるためのキーワードの一つが「モデリング」です．

モデリングとは何か

「モデリング」とは「モデル」を作成することです．そして「モデル」とは，実世界の事がらを，関心のある特徴に着目して図や記号，模型など別の表現方法や物体を使って表したものです．組み込みシステムの開発においても，機械的・構造的な特性を検討するための力学モデル，コントローラの設計を行うための制御モデルなど，さまざまなモデルが利用されています．

● 従来は設計書を文章で書いていた

ソフトウェアの開発において，効率を上げるためには設計などの上流工程での検討が重要でした．しかし，従来の手法では文章中心の設計書を用いていることが多いため，

●あいまいさを排除し，設計者の意図を正確に伝達するのが難しい
●文章の読み書きに時間がかかる
●内容をプログラム・コードへ反映することが難しいため，プログラミング時に再度設計が必要になる

といった問題が生じてしまい，なかなか効率を上げることができませんでした．

● ソフトウェアの設計内容を図示化する手法が開発された

そこで，これらの問題を解決し，さらに効率を上げるために，設計内容をソフトウェア・モデルとして図示化する手法がいくつも提案されてきました．

ソフトウェア・モデルは文章と比較して，

●表記法が決まっているので，あいまいさを排除できる
●さまざまな視点からモデルを作成できるので，設計者の意図を正確に伝えることができる
●モデル要素ごとにマッピング・ルールを決めることで，プログラミング・コードへの反映を容易に行える

といった利点があるためです．

しかし，ソフトウェア・モデルを利用する手法が数多く考え出されたため，以下のような新たな問題も発生しています．

●各手法でモデルの表記法が異なる
●利用する手法に応じて，まずは表記法を覚えなければならない

このような状態を解決するために提案されたのが「UML」です．

UMLの歴史と組み込みへの対応

● OMGで策定されたUML

UMLとは，Unified Modeling Language（統一モデリング言語）の略で，オブジェクト指向によって分析・設計されたモデルを表現する言語（表記法＋意味）の一種です．オブジェクト指向技術の標準化団体である，OMG（Object Management Group）で策定されています．

UMLが策定される以前は，多くの有識者によって提唱されたさまざまなシステム開発の方法論ごとに，異なったモデル表記法が定義されていました．このため，同じ内容を表しているにもかかわらず表記がまったく異なっていたり，逆に表記が似通っているのに表している内容が異なったりと，開発者にとってはあまり望ましくない状況でした．このような状態を解決するために，Booch法を提唱していたGrady Booch氏とOMT法を提案していたJames Rumbaugh氏によってモデルの表記法の統一化の動きが始まりました．そして後にOOSE法を提案していたIvar Jacobson氏も加わり，標準的なモデル表記法が検討されました．そして，その成果が「UML1.0」としてリリースされました（**図1**）．

UML1.xシリーズは，リリース後，以下のようなメリットが認められ，広く普及していきました．

▶ コミュニケーションのツールとして効果が高い

●仕様をビジュアル化できるため，内容を把握しやすい

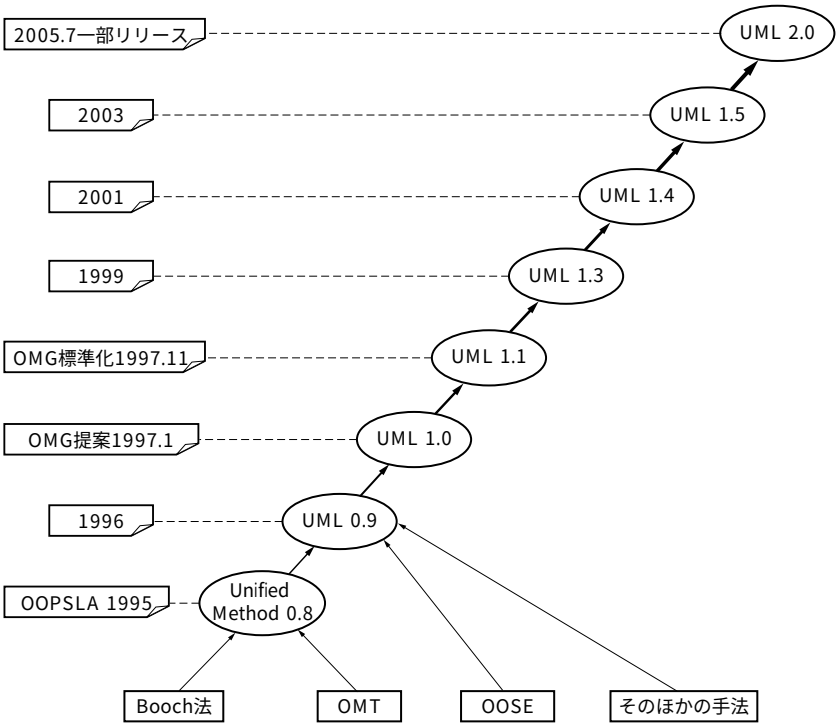


図1
UMLの歴史

●各モデル要素の意味が定義されているため，モデル作成者の意図を明確に表現できる
●設計者や開発者だけでなく，ユーザやテスタなど開発に関係するだれもが使える

▶ 標準化された記法であるため，広く利用することができる

●標準化された記法は，世界中の技術者が理解できる
●分析設計手法が異なっていても，同じダイヤグラムを使って表記できる

▶モデル（ダイヤグラム）間の対応付けが明確になったため，システムのさまざまな側面を表現できる

●分析，設計，実装など各工程間の成果物の対応関係を明確にできる
●全工程で一貫してオブジェクト指向技術が使用できる

▶UML は記法でしかないため，自分たちが選択した分析・設計手法を使うことができる

●各ダイヤグラムをどの工程で，どのように使うかをある程度自由に決めることができる
●必要に応じて使用するダイヤグラムを選択できる

▶ステレオタイプや制約，タグ付き値を利用することでモデル要素を拡張することができる

●対象となるドメイン固有の情報を表現するモデル要素を定義することができる
●開発方法論に応じたモデル要素を定義することができる

組み込み分野においては，モデル・レベルでの検証，実行が可能な ExecutableUML も利用されるようになりました．分析，設計におけるいろいろなテクニックも紹介されています．

● MDA という開発方法論が提案された

さまざまなメリットが認められて普及が進んだ UML ですが，利用が進むにつれて，新たな要求・問題が発生してきました．

そのうちの一つは，モデル化対象範囲の拡大による表現力の不足です．より上流工程のビジネス・モデリングや，組み込み分野における詳細設計など，当初は想定していなかった分野においても利用されるようになったため，UML1.5 では表現力が不足するようになってしまいました．

また，技術の進歩によりモデル化対象であるソフトウェアの大規模化が進んだため，コンポーネント・ベース開発のサポートも望まれるようになりました．

さらには，UML の仕様策定団体である OMG が提唱している MDA（Model Driven Architecture ：モデル駆動型アーキテクチャ）を実現するために，モデル要素の厳密な定義，実装レベルの詳細なモデル表現への対応，モデル・レベルでの検証の実現，メタモデルの整理なども要求されるようになってきました．

MDA とは，大ざっぱに言って，開発の中心をコードからモデルに転換しようという開発方法論です．プラットホームに独立なモデルである PIM（Platform Independent Model）と各種プラットホームに依存したモデルである PSM（Platform Specific Model），およびそれぞれのモデル間と実装言語へのマッピング・ルールを定義し，モデルの変換とソースコードの生成を自動で行うことを目標としています．

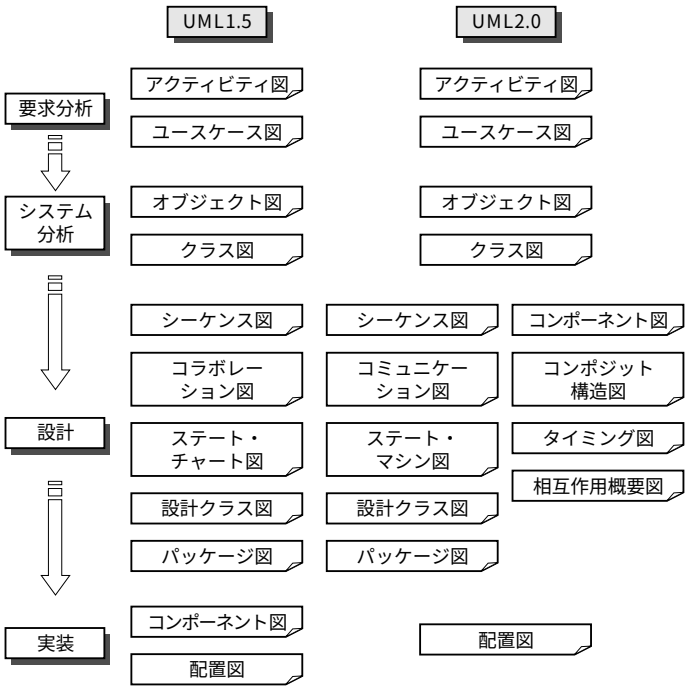


図2　おもな開発工程で使用するダイヤグラム

UML1.5 と UML2.0 の違い

これらの問題を解決するため UML2.0 の仕様が策定されました．

UML2.0 は，UML では初のメジャー・バージョンアップであり，大きく分けて以下の四つの仕様から構成されています．

▶Infrastructure
共通のコア・モデル要素の仕様

▶Superstructure
実際にモデルを作成する時に使用する要素の仕様

▶Diagram Interchange
ツール間などでモデルの交換を行うための仕様

▶OCL（Object Constraint Language）
モデル要素の制約を定義するための仕様

そして，Superstructure の仕様が 2005 年 7 月に正式リリースされました（ほかの三つの仕様は最終調整の段階）．

UML2.0 では，UML1.x の普及により明らかになってきた要求および問題を解決するため，さまざまな改良や拡張が行われています．とくに，

●動的なふるまいの記述能力の強化
●構造とふるまいの連携の強化

が行われています．

なかでも組み込み分野では利用頻度の高いステート・マシン

図やシーケンス図も表現力や再利用性が大幅に向上されており，実装レベルの詳細なふるまいも表現できるようになっています．また，「タイミング図」が追加されたことで，組み込み分野ではとくに重要な「時間」に関する記述がより厳密に行えるようになりました．

図2（p.175）におもな開発工程で使用するUML1.5とUML2.0のダイヤグラムの例を示します．この図からもUML2.0では，実装レベルの詳細なモデル表現の部分が強化されていることがわかります．

次に，UML2.0のそれぞれのダイヤグラムについて，どのような点が変更になったのかについて見ていきます．

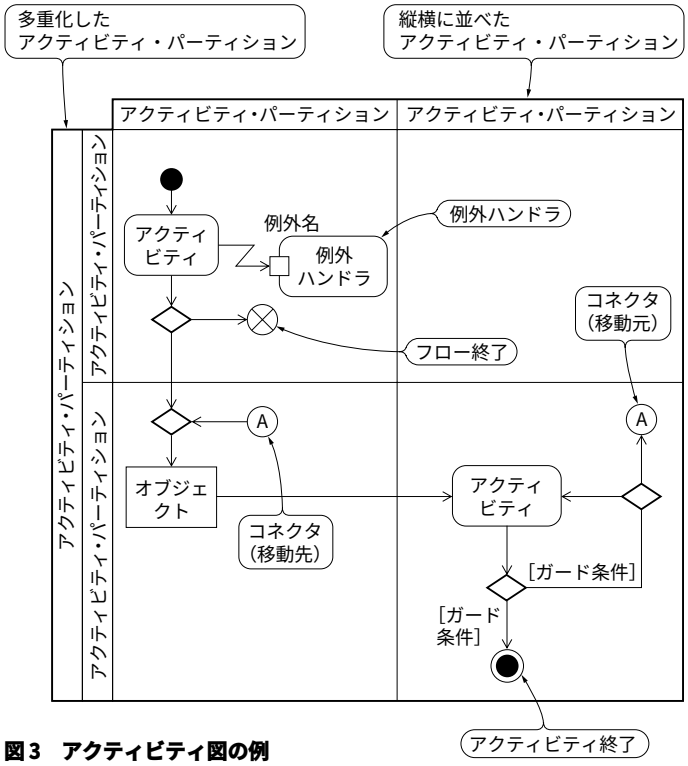


図3　アクティビティ図の例

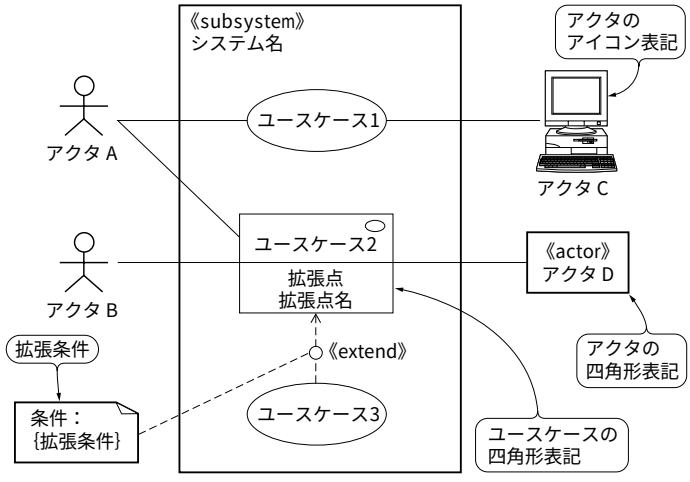


図4　ユースケース図の例

● アクティビティ図（UML1.5 → UML2.0）

アクティビティ図（activity diagram，**図3**）は，処理の実行手順を表すダイヤグラムです．

UML2.0では，各モデル要素の基盤であるメタモデルが大幅に整理され，各要素の意味も明確に再定義されました．さらに，ダイヤグラムの表現力も大幅に強化されています．

具体的には，まず「アクティビティ・パーティション」（UML1.5では「スイムレーン」と呼ばれていた）の多重化や縦横方向の配置が可能となりました．これにより，UML1.5では表現できなかった要素の詳細な分類が表現できるようになりました．

また，動作の終了を表す終了ノードが「アクティビティ終了」と「フロー終了」に分かれました．これにより，動作の正常終了と異常終了を明確に区別して表現できるようになっています．

そして，「例外ハンドラ」という要素が追加になり，アクティビティ内で異常処理が発生した場合の対応処理を表現できるようになりました．UML1.5では分岐条件を使用して表現していた例外処理をより簡潔に表せるようになりました．さらに，「コネクタ」を用いることで，複雑なフローもより見やすく表現することが可能になりました．より実装に近い詳細なふるまいも表現できるようになったと言えます．

● ユースケース図（UML1.5 → UML2.0）

ユースケース図（use case diagram，**図4**）は，ユーザの視点でシステムがもつ機能とシステムの利用者を表すダイヤグラムです．UML2.0では各要素の表記が拡張されています．UML1.5では，ユースケースは楕円のみで表記されていましたが，ステレオタイプ《use case》による表記や楕円のアイコンを用いた表記が可能になりました．またアクターについても，UML1.5ではすべて人型のアイコン（スティックマン）で表現されていましたが，ステレオタイプ《actor》による表記や，コンピュータ型のアイコン表記も行えるようになりました．

組み込みシステムの場合，システムの外部要素は，人（利用者）の場合もあれば，機器の場合もあります．今回の拡張で，より実態に即した形でのモデル表現が可能になりました．

さらに，ユースケースの「拡張」を行うときの条件の表現方法（拡張の途中に丸印を付けて，ノートで記述）が定められました．表記法が統一されたことで，モデルの可読性がより向上したと言えます．

● オブジェクト図（UML1.5 → UML2.0）

オブジェクト図（object diagram，**図5**）は，ある瞬間に着目して，システムの構造を表現するダイヤグラムです．UML2.0では「オブジェクト」の扱い方が内部的に変更になりましたが，表記的には大きな変更はありません．

● クラス図（UML1.5 → UML2.0）

クラス図（class diagram，**図6**）は，システムの静的な構造を表すダイヤグラムです．UML2.0では，メタモデルの構造に変更が加えられているとともに，表記的に細かい部分で改良が加

えられました．UML1.5では，関連に何も印が付いていない場合に，誘導不可能なのか誘導可能性が未定なのかがあいまいでしたが，UML2.0では誘導可能を矢印で，誘導不可能を×印で明確に表現できるようになりました（関連に何も印が付いていない場合，誘導可能性が未定であることを表すようになった）．

また，関連や属性に{ordered}（順序付けあり），{unique}（重複なし）といった制約を記述できるようになりました．実装レベルのプログラム・コードとの対応が強化され，より詳細な表現が可能になったと言えます．

● シーケンス図（UML1.5 → UML2.0）

シーケンス図（sequence diagram，**図7**）は，システムのある機能を実現するためのオブジェクト間の相互作用を，時系列に着目して表現したダイヤグラムです．UML2.0では大幅な変更が加えられ，表現力が格段に向上しました．まず挙げられるのは「複合フラグメント」の導入です．条件分岐を表現するための「alt」，繰り返し処理を表現するための「loop」，並行処理を表すための「par」など全部で12種類が定義されています．UML1.5においても，分岐処理や繰り返し処理はメッセージにガード条件を付記することで表現はできましたが，よりシンプルに記述できるようになり，モデルの可読性が向上しました．

また，UML1.5では複雑なふるまいを表現しようとした場合に，シーケンス図が大きくなりすぎて，モデルの可読性が悪くなってしまったり，作業効率が落ちるという問題点がありました．この問題を解決するために，UML2.0では別のシーケンスを参照するためのしくみとして，「相互作用ユース」が追加されました．相互作用ユースを用いることで，共通のふるまいを抜き出して再利用したり，抽象度に応じて相互作用を階層化することなどが可能となりました．

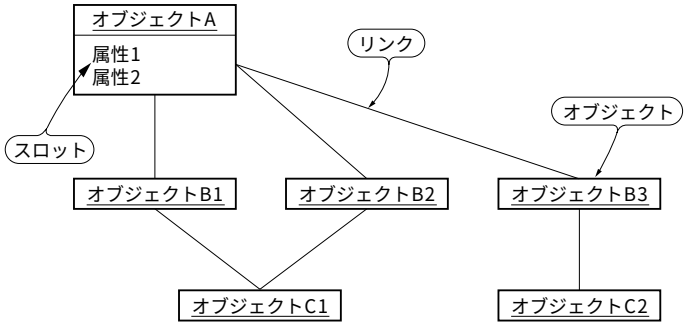


図5　オブジェクト図の例

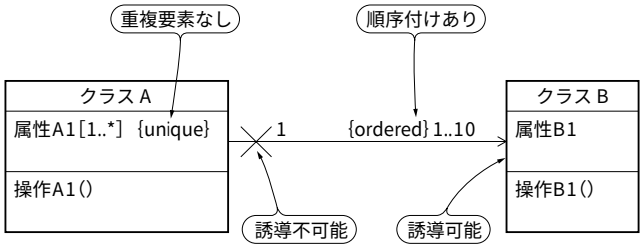


図6　クラス図の例

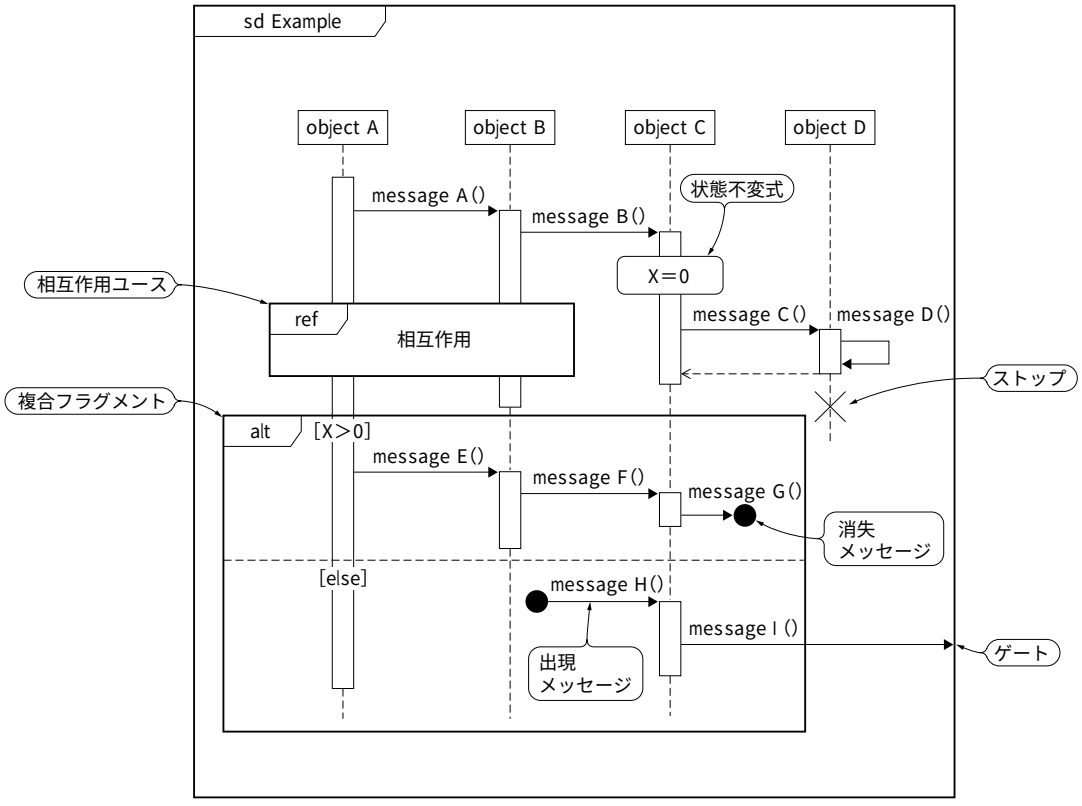


図7
シーケンス図の例

そして，メッセージの送信元や送信先が表現しているシーケンス図の範囲外であることを表す「消失メッセージ」や「出現メッセージ」，メッセージと相互作用の境界点を表現する「ゲート」を用いることで，複数の相互作用をまたぐメッセージも表現できるようになりました．

さらには，相互作用中のある時点での状態を定義するための状態不変式や，複合フラグメントの「assert」が追加されたことで，モデル・レベルでのテストやシミュレーションが可能になりました．MDA の実現に向けて，実装レベルとの対応がより一層強化されたと言えます．

● コミュニケーション図（UML1.5 → UML2.0）

コミュニケーション図（communication diagram，**図 8**）は，システムのある機能を実現するためのオブジェクト間の相互作用を，オブジェクトの関係に着目して表現したダイヤグラムです．UML1.5 では，コラボレーション図と呼ばれていましたが，名称が変更となりました．内容的には UML1.5 から大きな変化はありません．そのため，UML1.5 まではコラボレーション図はシーケンス図と互換性がありましたが，UML2.0 ではシーケンス図の内容が大幅に変更となったため，コミュニケーション図との互換性は保証されなくなってしまいました（入れ子表現などが含まれない単純なシーケンス図とは互換性がある）．

● ステート・マシン図（UML1.5 → UML2.0）

ステート・マシン図（state machine diagram，**図 9**）は，システムを構成するあるオブジェクトについて生存期間中の状態の変化を表現するダイヤグラムです．UML1.5 ではステート・チャート図と呼ばれていましたが，名称が変更となり，表現力が向上しました．UML1.5 では，外部からコンポジット状態内部への遷移はコンポジット状態内の開始状態で，またコンポジット状態から外部への遷移はコンポジット状態内の終了状態でしか表現できませんでした．そのため，履歴状態指示子を使用した場合を除いて，コンポジット状態と外部の遷移は，つねに同じサブ状態からしか行うことができませんでした．UML2.0 では，コンポジット状態と外部の境界として，新たに「入場点」と「退場点」が追加され，任意のサブ状態からの開始・終了が可能になりました．

さらに，履歴状態指示子の種類も追加になり，コンポジット状態が入れ子になっていた場合，どこまでの状態を保持するかを明確に表現できるようになりました．また，UML2.0 ではステート・マシンの継承が可能になりました．クラス間で属性や操作を継承できるように，ステート・マシン間で状態や遷移を継承できるようになりました．

組み込みシステムでは，各オブジェクトの状態管理が非常に重要です．UML2.0 では，より複雑な状態をシンプルに表現できるようになったとともに，状態自体の再利用も可能になり，より使い勝手が向上しました．

● パッケージ図（UML1.5 → UML2.0）

パッケージ図（package diagram，**図 10**）は，モデル要素をグループ化したパッケージ間の関係を表現するダイヤグラムです．UML2.0 ではパッケージ間の関係が強化されました．UML1.5 ではほかのパッケージに含まれる要素を利用する場合，パッケージ間のインポートの関係でしか表現できませんでした．このため，パッケージ内のどの要素を実際に利用しているのかがはっきりしませんでした．UML2.0 では，パッケージ内部の要素を直接指定できるようになったため，どの要素が必要なのかを明確に表現できるようになっています．

さらに，UML2.0 ではパッケージ間の関係としてとしてマージが新しく追加されました．マージの関係を使用した場合，複数のパッケージに存在する同じ名前の要素の特徴を合成することができます（UML2.0 のモデル要素自体もパッケージ・マージを繰り返し利用して定義されている）．パッケージ間の関係が強化されたことで，モデル要素の拡張やパッケージ単位での再利用がより行いやすくなったと言えます．

● コンポーネント図（UML1.5 → UML2.0）

コンポーネント図（component diagram，**図 11**）は，ソフト

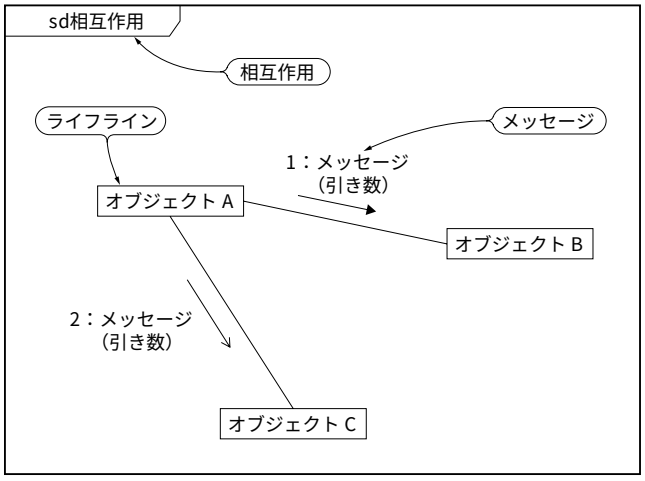


図 8　コミュニケーション図の例

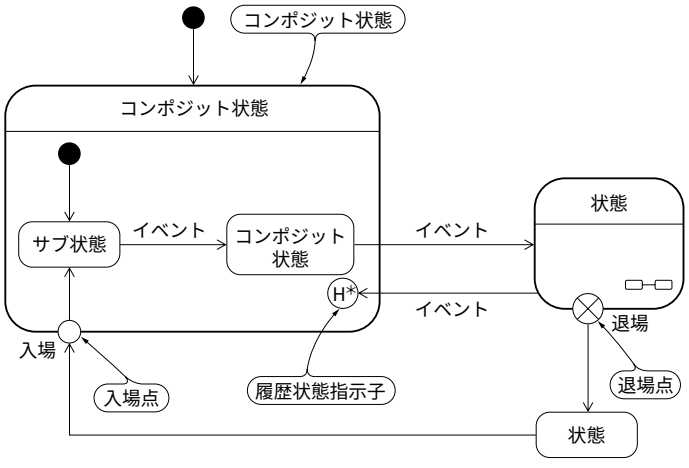


図 9　ステート・マシン図の例

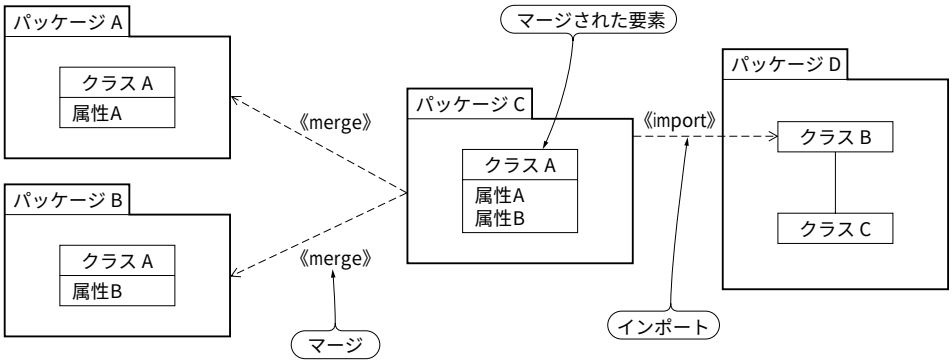


図 10
パッケージ図の例

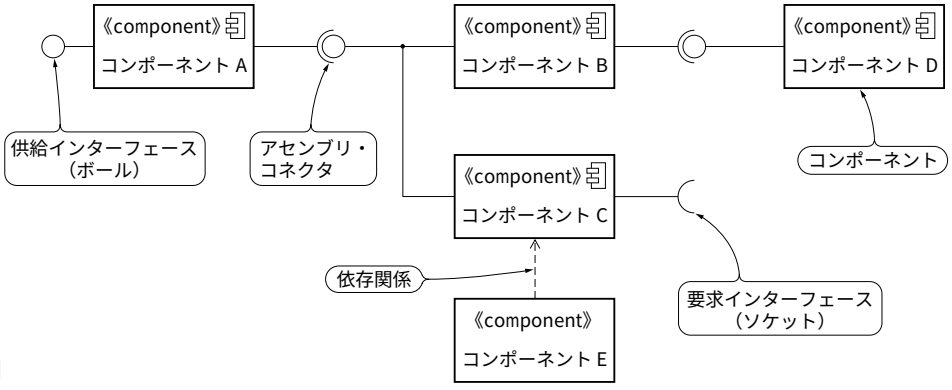


図 11
コンポーネント図の例

ウェア部品の静的な構造を表現するダイヤグラムです．UML1.5にも存在しましたが，UML2.0になって表現する内容がまったく異なるものになりました．UML1.5では，「コンポーネント」はソース・ファイルや実行ファイルなどの物理的な意味でのソフトウェア構成部品として定義されており，コンポーネント図は実装時のソフトウェアの物理的な構造を表現する図という位置付けでした．これに対しUML2.0では，コンポーネントの定義が「インターフェースをもつクラス」と変更になりました．このため，コンポーネント図が表す内容も，分析・設計段階でのソフトウェア部品の論理的な構造に変化しました．

さらに，UML2.0では「提供インターフェース」と「要求インターフェース」という2種類のインターフェースが定義されました．前者は，接続されたコンポーネントが外部に公開する機能を表現し，後者は接続されたコンポーネントが，自分自身の機能を実現するために必要な機能を表します．UML1.5では提供インターフェースに相当する要素しか定義されていませんでした．したがって，UML2.0になってコンポーネントを部品として利用する場合の条件がより明確に表現できるようになったと言えます．

● コンポジット構造図（UML2.0で新規に追加）

コンポジット構造図（composite structure diagram，**図12**）は，UML2.0で新規に追加されたダイヤグラムで，クラスやコンポーネントなどの内部構造や，そこで使用されるコラボレーションを表現します．コンポジット構造図では，内部構造と外

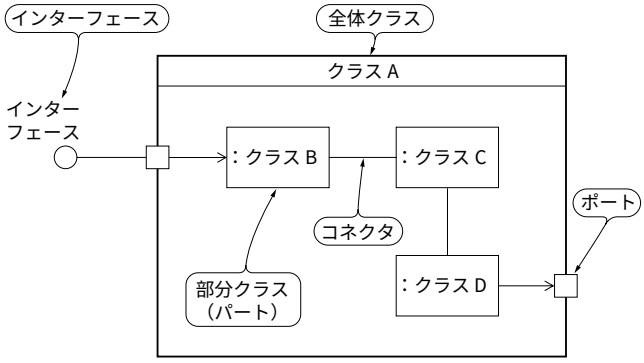


図 12　コンポジット構造図の例

部の境界を「ポート」によって明確に表現することができます．

また，内部構造を「パート」によって，パートとポート間およびパート間の関係を「コネクタ」によって表現します．UML1.5では，クラスの構造を集約やコンポジションといった関連で表現していました．そのため，複雑なシステムでは各要素の境界がわかりにくく，構造の把握が困難でした．コンポジット構造図は，構造化されたクラスや外部と内部の境界をわかりやすく表現できるので，モデルの可読性を上げることができます．

● タイミング図（UML2.0で新規に追加）

タイミング図（timing diagram，**図13**）も，UML2.0で新規に追加となったダイヤグラムです．

オブジェクトの状態変化と，ほかのオブジェクトとのメッ

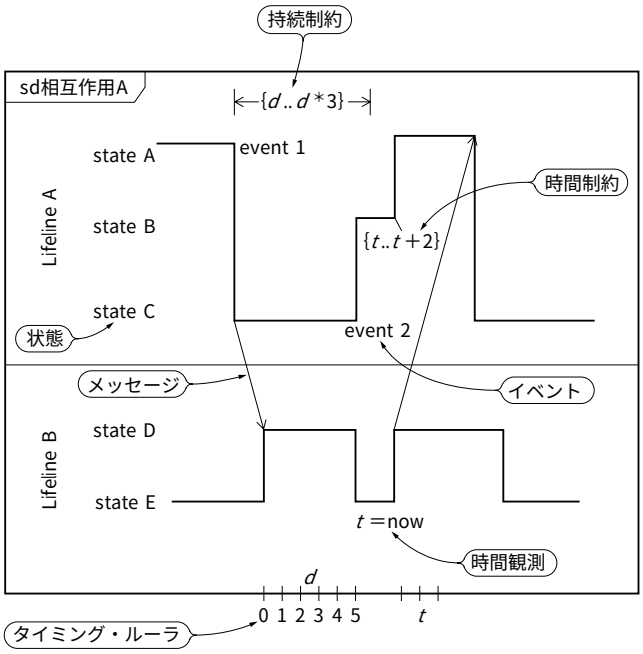


図13　タイミング図の例

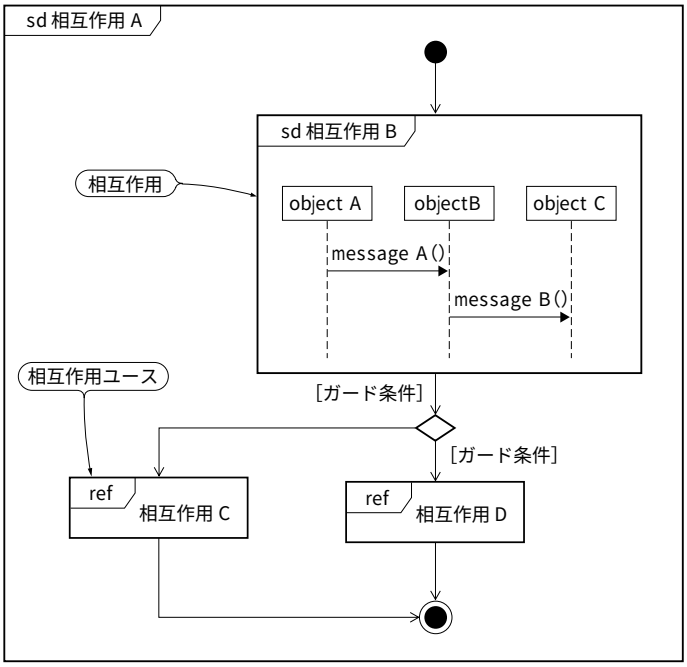


図14　相互作用概要図の例

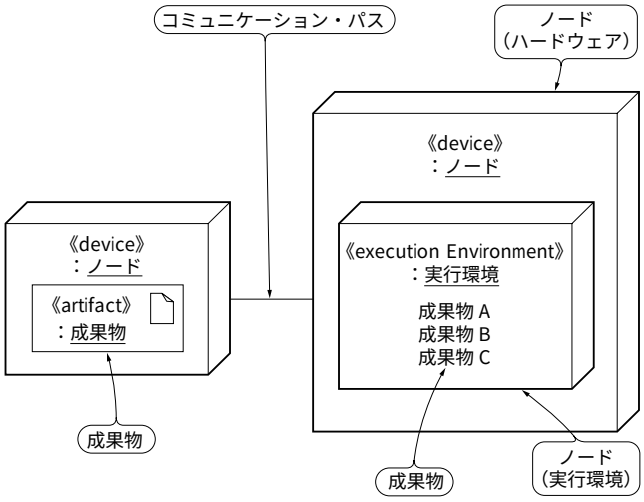


図15　配置図の例

セージのやりとりを，時間に着目して表現することができます．組み込みの分野ではなじみが深く，利用頻度の高いダイヤグラムだと思います．

システムがある機能を実現しようとする場合，構成要素（オブジェクト）間で機能の呼び出し（メッセージのやり取り）を行い，構成要素自身も状態を変化させていきます．しかしUML 1.5では，オブジェクトの状態変化はステート・チャート図，オブジェクト間のメッセージのやり取りはシーケンス図（もしくはコラボレーション図）と，別々のダイヤグラムで表現する必要があり，メッセージとイベント，状態遷移の関係を明確に表現するのが困難でした．タイミング図を利用することで，これらの関係を時間軸に沿った形ですっきりと表現することができます．

また，時間的な制約を表現する「時間制約」と，時間制約の基準となる時間を取得するアクションを表す「時間観測」を用いることで，メッセージの送受信と状態変化の時間的な関係や制約を明確に表現することができます．

● 相互作用概要図（UML2.0で新規に追加）

相互作用概要図（interaction overview diagram，**図14**）は，相互作用の実行手順を表現するダイヤグラムで，UML2.0で新しく追加されました．UML2.0では，何らかの機能を実現するためにオブジェクトやクラス，コンポーネント間で行われるやり取りを「相互作用」として定義しています．

そして，シーケンス図やコミュニケーション図，タイミング図など相互作用を表現するためのダイヤグラムを「相互作用図」と呼んでいます．相互作用概要図は，これらの相互作用間の関係を，アクティビティ図で用いた要素を利用して表現します．

また，「相互作用ユース」を使って，ほかで定義した相互作用を参照することも可能です．相互作用が複雑な場合や，共通化が可能な相互作用を切り出して記述する場合に利用することができます．

● 配置図（UML1.5 → UML2.0）

配置図（deployment diagram，**図15**）は，システムの物理的な構成を表すダイヤグラムです．配置図自体はUML2.0となっても大きな変化はありません．しかし，コンポーネントの定義が大幅に変更されたため，物理的なソフトウェア構成部品を表すものとして「成果物」という要素が追加となっています．また，「ノード」の定義も，配置対象のハードウェアだけでなく，成果物の入れ物を指すものに拡張されています．そのため，プログラムの実行環境などもノードとして表現できるようになっ

ています．

以上，駆け足でしたが，UML2.0の各ダイヤグラムについて，変更・拡張された点を中心に概略を紹介しました．UML2.0では詳細なふるまいや時間に関係する部分の表現力が大幅に強化されたため，組み込み分野でも本格的に使えるようになってきたと言えます．

繰り返しになりますが，モデルはあくまで関係者間のコミュニケーションを円滑にするための道具です．また，UMLはモデルの記述方法と各モデル要素の意味を定義した「言語」であり，開発の方法論までは含まれていません．

また，すべてのダイヤグラムを必ず使用しなければいけないわけではありません．したがって，どの工程でどのダイヤグラムを使用するか，また各ダイヤグラムで何をどこまで表現するかは利用者の選択に任されています．言い換えると，この選択を誤ってしまうと逆にマイナス効果となってしまいます．状況に応じて適用しやすい部分から利用を始め，徐々に利用するダイヤグラムを増やしていくと良いと思います．

おわりに

今回のUML2.0へのバージョンアップでは，コンポーネント・ベース開発への対応が強く意識されており，ソフトウェア部品である「コンポーネント」の表現力が大幅に強化されています．コンポーネントの内部構造や，外部との境界はコンポジット構造図とで明確に表現できるようになり，またコンポーネント間の関係はコンポーネント図で表現できるようになりました．

そして，ふるまいの単位として「相互作用」という考え方が導入され，階層化表現も可能となりました．コンポーネント単位の仕様を明確に表現できるようになったことで，モデル・レベルでコンポーネントの詳細な検討（再利用性やほかのコンポーネントとの互換性など）が行えるようになったと言えます．

組み込み分野では，これまでハードウェアのコンポーネント化が先行して進められてきましたが，UML2.0を使用することでソフトウェアのコンポーネント化，とくに設計レベルでのコンポーネント化を実現することができます．そして，より上流レベルからの再利用を促進することで，開発効率の更なる向上が期待できます．

また，UML2.0では，OMGが提案しているMDAを実現するための改良も数多く行われています．とくに動的なふるまいを表現する部分の強化が行われており，より実装に近いレベルの詳細なふるまいがシンプルに表現できるようになっています．

さらにUML2.0以外にも，OMGではMDAの実現に向けて，MOF（Meta Object Facility：モデリング言語自体を定義しているメタモデルを記述するための仕様．モデル要素のマッピングに利用される）やQVT（Query-View-Transformation：モデルの変換ルールを記述するための仕様）といった仕様の策定も進めています．これらの仕様とUML2.0を組み合わせることで，コミュニケーション・ツールとしてだけではなく，実装手段としてモデルを利用できるようになるかもしれません．

＊　　＊

次回からは，UML2.0で強化された点を具体的な例を使ってさらに詳しく紹介します．

Column　UML2.0対応モデリング・ツール パターンウィーバーについて

パターンウィーバーは，米国Foundatao社によって開発されたUML2.0に対応したUMLモデリング・ツールです．UML2.0で新たに追加されたタイミング図やコンポジット構造図も含めて，13種類すべてのダイヤグラムを記述することができます．

今回，本誌付属のCD-ROM InterGiga No.35にパターンウィーバーの評価版（CE版）が収録されています．UML2.0の各種ダイヤグラムをぜひ実際にお試しください．

なお，パターンウィーバーの詳細情報，最新情報については，以下のURLを参照ください．

http://pw.tech-arts.co.jp/pw/index.html

参考文献

(1) UML2.0 Superstructure Specification, OMG, Document number：ptc/04-10-02.
(2)（株）テクノロジックアート著，長瀬 嘉秀・橋本 大輔　監修；独習UML 第3版，翔泳社.
(3) Stephen J. Mellor, Marc J. Balcer著，Executable UML研究会訳；Executable UML MDAモデル駆動型アーキテクチャの基礎，翔泳社.
(4) Leon Starr著，二上貴夫，長瀬嘉秀訳；Executable UML実践入門-クラス・モデルをいかに作成するか，CQ出版.

さかもと・たけし　（株）テクノロジックアート