原文："Turning Clockwise: Using UML in the Real-Time Domain"
Communications of the ACM, Vol.42, No.10, pp.46-54（Oct. 1999）より許可を得て翻訳

リアルタイムドメインへのUML適用

Bran Selic　bran@objectime.com
翻訳：中島　震　nakajima@ccm.cl.nec.co.jp

リアルタイムシステム

時間は僭主だ．他の物理現象と異なり制御不能である．止めることも，伸ばすことも，縮めることもできない．時間は計ることしかできない．このとっつきにくい性質が，リアルタイムソフトウェアの設計を難しくする．その性格上，タイムリーさは最も重要であり，応答の遅れは無応答より悪いことさえある．

しかし，リアルタイムソフトウェアが複雑な理由はタイムリーさだけではない．リアルタイムシステムは物理的な世界とのインタラクションを持つ．この世界は陰謀の塊でイベントは変な時や間違った順序で発生する．したがって，複雑さはこの世界自身に起因し，いかなる技術を用いてもこの根本的な複雑さを除去することはできない．

絶えざる機能向上の要求を満たすためには，この複雑さの問題を何とかしなければならない．より洗練されたシステムを使いたいという自然な思いに加えて，劇化するビジネス競争がますます高度な機能を要求する．たとえば，電話機は約500の異なるサービスである「フィーチャ」が定義されている．転送，音声メール，モバイル電話のローミングサービス等の今では当たり前のものである．フィーチャは独立に定義されていて互いに矛盾することがある．それにもかかわらず，普通は通信システムのソフトウェアがフィーチャをすべて提供し矛盾を解消していると期待する．

多くのリアルタイムシステムで遭遇する複雑さのレベルは我々が認識可能な範囲をはるかに越えている．複雑さに対応するための数少ない道具にモデリングがある．これは，問題の理解ならびに解決を容易にするための抽象化技法である．モデルはシステムの重要な特徴だけを取り込み低レベルの詳細を隠したり無視する．モデルは単純であるが正確な姿を示すので実際のシステムよりもはるかに理解しやすい．

UMLは最近のモデリングツールの1つであり，オブジェクト指向やオブジェクトベースのシステムならびにアプリケーション向けに考案された．1997年，OMGがUMLを採用して以来，UMLへの関心が急激に高まり，ソフトウェア開発の共通語になりつつある．

オブジェクト指向パラダイムとの関連

「リアルタイム」という言葉は非常に多くのソフトウェアで使われている．マイクロ秒単位での応答が必要な組込み制御から秒や分単位の応答時間でよいシステムまである．そのような多様性はあるが，**リアルタイムシステムには「環境と継続的にかつタイムリーにインタラクションを行う」，という共通特性が認められる**．明らかにタイムリーさはリアルタイムシステムの特徴である．では，「継続的なインタラクション」とは何を意味する

のであろうか？　これはリアクティブ性と呼ばれる性質である．リアクティブシステムは，無限のこともあるが，一定の間実行を継続し，入力に応じて応答する．ここで注意すべきは，変わり得る表層の振舞いではなく，変わらないシステムの構造が中心にあることである．これは，設計の中心が，従来の手続き的なアプローチと逆に，アルゴリズムから構造へと移ることを示している．

したがって，オブジェクト指向の考え方は多くのリアルタイムシステムと相性がよい．元来，オブジェクト指向パラダイムは現実世界のシミュレーションのために考えられた．互いにインタラクションを持つ動的なコンポーネントによりソフトウェアを規定する．これはオブジェクト指向技術以前にも多くのリアルタイムシステムで採用された考え方である．

概念レベルでは相性がよいのであるが，リアルタイムシステム開発者はオブジェクト技術がメモリと実行性能にオーバーヘッドを抱えるという不安を持つ．多相性や情報隠蔽といった抽象化機構に起因するオーバーヘッドがあることは事実である．しかし，結論を出す前に，実際のコストを注意深く分析しなくてはならない．初期のオブジェクト技術が持っていた問題点の多くは軽減されたり解消されている．さらに，ハードウェアの性能向上を考えると，リアルタイムシステム向けオブジェクト技術の障壁が低下し続けている．数多くの大規模リアルタイムシステムが全体もしくは一部をすでにオブジェクト指向で開発している．

また，非常に厳しい性能要求が課せられるリアルタイム・アプリケーションであっても，わずかな部分だけが性能を支配することが多く，20％のコードが80％の処理時間を占めるという．これは，オブジェクト指向を適用できるソフトウェアの範囲がますます広がることを意味している．

モデリングへの要求

リアルタイムシステムは他に比べて制約が強く，タイムリーさの要求がその典型である．「ハード」リアルタイムシステムでは，たった1つの時間的な制約違反さえ人命にかかわることがある．システムのモデルを用いて，応答時間，スループット，可用性等の実システムの鍵となる属性を，開発の早い段階で正確に予測することが重要である．多くの属性は定量的であり，計算資源の物理的な性質や量的な制限を反映する．リアルタイムドメインではシステムがすべてのデッドラインを満たすかどうかを決定するＲＭＡ（Rate Monotonic Analysis）[2] 等のスケジュール可能性解析技術，遅れや必要資源を統計的な手法で解析する待ち行列理論等，の技術がすでにある．

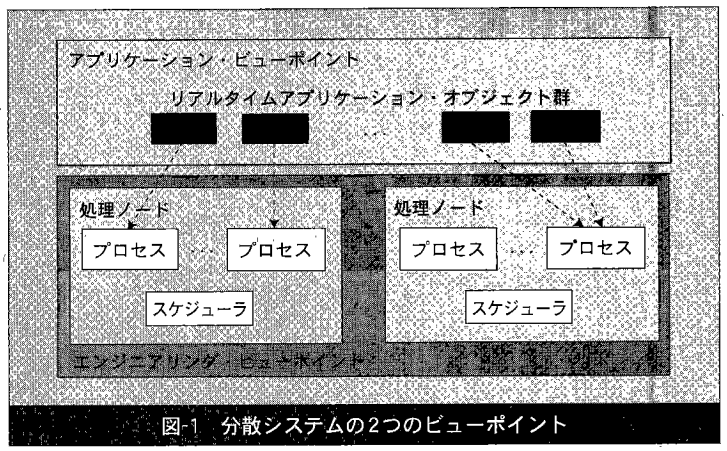

図-1　分散システムの2つのビューポイント

予測可能なモデルを得るためには，対象ソフトウェアの構造ならびに振舞いの側面と共に，前提とする論理的あるいは物理的な資源，すなわち，基盤環境をモデル化する必要がある．この点が，他のシステムとの最も大きな違いであろう．さらに，リアルタイムシステムに固有のモデル化に対する要求もある．

基盤環境のモデリング

基盤環境はプロセッサやネットワークといった物理的な装置と並行プロセス（タスク），ロック，キューのような論理的な「デバイス」からなる（図-1参照）．いずれも特定のタイプに属する資源と考える．一般に，容量や場所といった物理的な属性で資源を特徴付け，属性に値を与えたモデルを用いて実システムの予測を行う．

リアルタイムシステムでは，詳細な資源モデルが必要なことがある．場合によっては，CPUやキャッシュ，メモリ，バスなどのハードウェアを構成する内部アーキテクチャまでモデル化する必要がある．モデリングの詳細度を決める要因は2つある．第1は必要な予測精度であり，第2は資源を操作するシステムに求められる要求である．たとえば，耐故障リアルタイムシステムでは個々のメモリユニットやその他のハードウェア資源を診断するので，基盤環境モデルはソフトウェアモデル全体の一部をなす．また，スケジューリング，タイミング，メモリ管理といったオペレーティングシステムの基本サービスを含むこともある．これらのサービスも応答時間や可用性に関する定量的な属性でモデル化できる．

最後になったが，ソフトウェアと資源の対応関係を規定することを軽視してはならない．例を図-1の破線矢印で示した．ソフトウェアの構成要素を特定資源に対応付けること，その資源に対する要求を明示すること，が必要である．たとえば，性能モデリングでは，プロセッサのメモリ量や処理性能と共に，個々のオブジェクトを

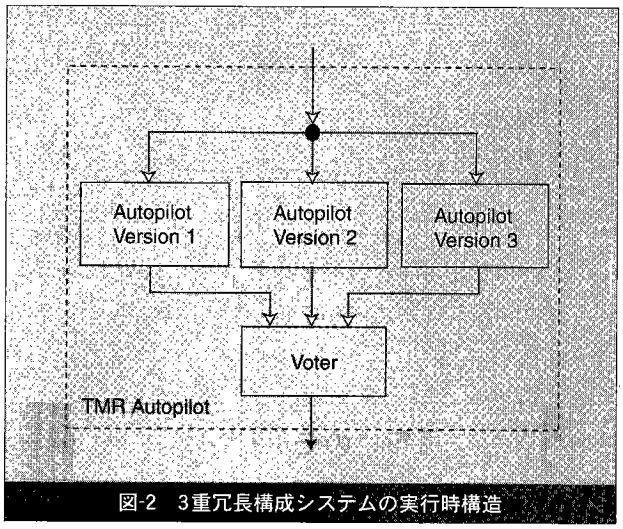

図-2　3重冗長構成システムの実行時構造

各プロセッサへ配置する情報が必要になる．

振舞いモデリング

実世界ではイベントの発生は予測不能であり同時に起こることもあるので，手続き型プログラミングの逐次的なモデルはリアルタイムシステムの振舞い記述に向かない．システムへの入力が持つ性質に応じた2つの方法が考案されている[3]．「イベント駆動」スタイルは並行イベントが非同期的に発生する離散現象をモデル化する場合に用い，イベント到着をトリガーとして離散的なステップで処理を進める．一方，「時間駆動」スタイルは連続的周期的な入力に適する．あるいは，リアルタイム時計に同期して処理が起動され周期的に入力を採取し処理する．また，1つのシステムで異なる周期を持つ複数の並行処理を行うこともある．以上の2つのスタイルを扱う必要がある．

構造モデリング

リアルタイムシステムが実行時に持つ複雑な構造を抽象的なレベルでモデル化しなければならない．耐故障リアルタイムシステムでは，図-2に示すような3重冗長構成を採用することが多い．実行時環境の構造の鍵を握るのは，各オブジェクトインスタンス間の関係のトポロジーである．これは，3つのカテゴリーに分けることができる．

- 対等関係：直接通信するオブジェクト間の関係．図-2のサーバと多数決システムの関係が一例．
- 包含関係：3重化自動操縦システムとその構成要素の関係が一例．構成要素の寿命が包含するコンテナに依存するコンポジションとコンテナが構成要素を単に隠蔽する集約とがある．
- 層関係：クライアントサーバ関係の特別な場合．複数クライアントが1つのサーバを共有する場合に「インプリメンテーション層」あるいは「仮想マシン」を導入して共有する実体を別の階層として抽象化し，隠蔽と同様の効果を得る．

以上の3つの構造の他に，実行時の動的な構造のモデルも必要である．システムの実行負荷に応じたオブジェクトやその間のリンクの生成，消滅である．たとえば，電話交換機では，新たなサービス開始と同時に1つの「呼」オブジェクトを生成し，終了と共に消滅させる．

UMLの利用

UMLは汎用のモデリング言語であり，幅広いアプリケーションドメインに概念的な基礎を与える．リアルタイムシステムへも適用可能なのか，可能な場合，前章の特徴的な要求をどのように取り扱えばよいのだろうか．

UMLを拡張してリアルタイムドメインへ適用するために新たな概念を追加する試みがあったが，我々の経験では拡張は不要である．基本的な概念を適切に特殊化することで，リアルタイム・モデリングが可能である．現行の一般的な概念と矛盾しないで，かつ，補うような意味定義を追加するだけでよい．たとえば，リアルタイム時計は，時刻の設定と読出しの操作を持つUMLクラスの1インスタンスとしてモデル化できる．多くのリアルタイムシステムはこの概念を使うので，特別な「リアルタイム時計」ステレオタイプとして定義し，意味定義の種々の制約をステレオタイプに持たせる．これは，再利用可能なリアルタイムドメイン固有の概念となる．

ステレオタイプを用いると，追加したモデルを知らない技術者であっても，UMLの一般的な知識を持っていれば，モデルの意味を理解できる．もっとも，詳細な意味定義まで理解できるということではない．

基盤環境のモデリング

資源とシステムサービス

多くの資源をUMLの汎用概念であるClassあるいはClassifierの適切なステレオタイプでモデル化できる．プロセッサ等のハードウェア装置は，Classifierの特殊化であるNodeのステレオタイプとしてモデル化する．これによって，複雑な意味定義や内部構造を持つ資源を柔軟にモデル化できる．

各資源は，これを利用する論理的な要素が必要とする特殊な機能を提供する．一般的に個々の資源はインタフェースを通して利用可能なサービス提供サーバと見なせることが多い．インタフェースを通して利用する操作

(フィーチャ) には，アクセスタイムや応答時間といった定量的なサービス品質属性がある．オペレーティングシステムのサービスも同じアプローチで表現する．モデル化の詳細度に応じて，オペレーティングシステム全体を適切なサービス呼出しインタフェースと品質属性を持つ単一オブジェクトで表したり，個別機能を提供するオブジェクトの集まりでモデル化したりする．

1つの資源は1つあるいは複数の論理要素を実装する基盤となる．このことを，資源が論理モデルをサポートするという．逆に，モデルの要素は複数資源のサポートが必要なこともある．論理的な要素と対応する資源間の「サポート」関係を（図-1の破線矢印で示す）ステレオタイプの利用依存関係で表現する．

要素と資源のマッピング

通常，リアルタイムシステムは種々のハードウェア構成で作動するように設計する．たとえば，2台のプロセッサ向けに設計したシステムを，十分な処理パワーがある1台のプロセッサ上でも実行させたい．すなわち，論理モデルと資源の対応付けには種々のマッピングが考えられ，マッピングいくつかをまとめて資源マッピングパッケージ (resource-mapping package) とする．資源マッピングパッケージを用いると柔軟性が増す．特定マシンへの対応を決定しておくことが基本的なシステム仕様の一部のこともあれば，対応付けをロード時あるいは実行時に動的に決定するような指定もできる．

資源と値のマッピング

資源に関して，プロセッサ速度，待ち行列の長さ等の属性値をマシンが持つ資源と対応付けるマッピングが必要である．定量的な解析を行う際には必須である．また，要素と資源のマッピングと同様に，複数の対応関係を可能にするために，資源値パッケージ (resource value package) と呼ぶUMLパッケージにまとめておく．

時間と時間計測

リアルタイムシステムで重要な役割を果たす時刻に対して，上記の資源モデルが適用できる．時刻は種々の意味定義が可能な要素であるが，UMLでは大域的な時刻であるか分散時刻であるかといった仮定をいっさいおかない．Timeと呼ぶデータタイプを標準で定義するが特定の意味付けはない．アプリケーションが自由に時刻のモデルと表現を定義する．

時計，カレンダー，タイミングサービス等の時間に関する機能を資源としてモデル化する．現在値読出しや時間関連サービス要求のサービスインタフェースと，時計の調整やリセットなどの制御インタフェースを持たせる．タイミングサービスは，さらに専用時計を持ち，決められた時間経過や特定の時刻到達でイベントを発生する．これによりイベント駆動の環境でも時間駆動の振舞いをモデル化することができる．

振舞いのモデリング

並列性の基本的な意味はUMLの「アクティブ」オブジェクト概念を使ってモデル化する．標準では並行性から生じる種々の側面のモデル化を意図的に避けていて，プロセス間通信での到着順序や並行プロセスのスケジューリングポリシー等何も規定しない．状況が変われば異なる解決方法が必要となり，すべてのリアルタイムアプリケーションが満足する唯一のアプローチはない．

イベント駆動（リアクティブ）モデリング

イベント駆動の振舞いは，ある種の遷移システムで形式化する．イベント到着に対してある定常状態から別の定常状態に遷移する実体が基本モデルである．全体のアクティビティをイベントと応答の組の列として表せるので，緊急な並行アクティビティがない限り解釈は容易である．しかし，リアルタイムアプリケーションでは，並行性と高い優先度を持つイベントへの即時応答が必須である．

UMLの状態遷移マシンは，David Harelのステートチャートを採用したもので，並行リアルタイムシステムに適した遷移システムである．以下，制御的な振舞いと機能的な振舞いの組合せ方法を例として本手法の特徴を紹介する．ここで，オブジェクトが担うサービス処理を機能的な振舞いと呼ぶ．機能的な振舞いを状態マシンで規定することができ，その例としてビット反転 (alternating-bit) プロトコルハンドラを図-3に示した．

機能的な振舞いは制御的な振舞いと明らかに異なる．制御的な振舞いはシステム全体の中でオブジェクトが制御エンティティとして実行するアクションである．初期化，終了処理，エラー処理と復帰等の制御アクティビティを含む．異なる種類のサービスを提供するオブジェクトに同一の制御的な振舞いを与えると，制御モデルの共通化が可能となり，より単純で制御しやすいシステムとなる．機能から独立した制御オートマトンの例を図-4に示す．

問題なのは，1つのオブジェクトを異なる視点から記述した2つの状態マシンがあり，何らかの方法で組み合わせなければならない，という点である．単純であるが

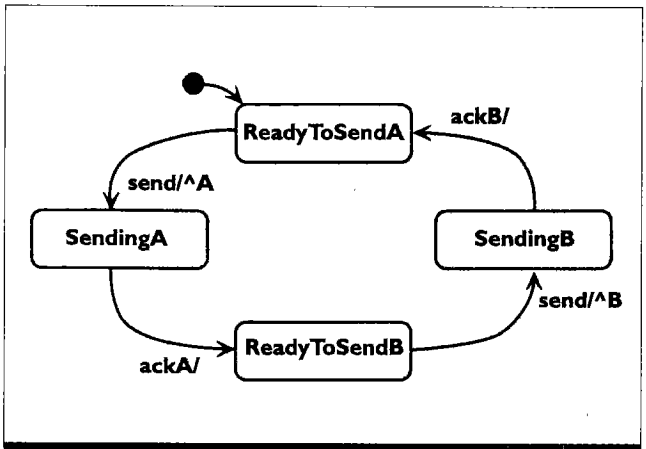

図-3　機能的な振舞い

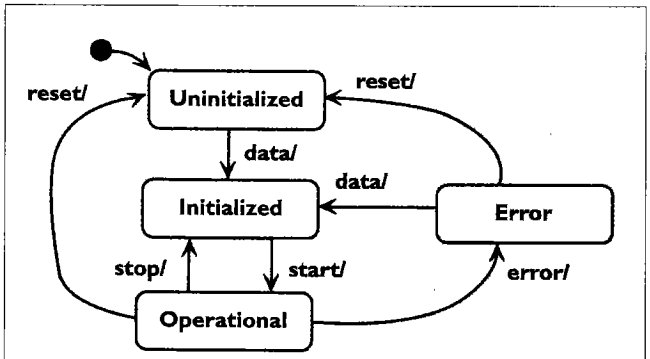

図-4　制御的な振舞い

有効な解決策として，図-5のような階層的状態遷移を用いる方法がある．この方法で2種類の振舞いを明確に分離でき各々を独立に変更することができる．さらに，継承機能を有効に使って制御に関する状態遷移を1つの抽象クラスで定義することもできる．同じ制御的な振舞いを持つすべてのクラスはその抽象クラスのサブクラスとして定義することで，制御的な振舞いを自動的に継承できる．

時間駆動モデリング

時間駆動による振舞いは周期的なアクティビティに基づくので，オブジェクト概念と相性が悪いように思える．先に議論した制御の問題を考慮すると，周期的なアクティビティが開始可能となる前にシステムは操作可能状態に移る必要があることが分かる．同様に，障害が発生した時，システムは障害処理と回復手続きの後，処理を再開する．障害発生，操作対象データの到着等の制御アクティビティは本質的にイベント駆動的なので，時間駆動による振舞いであっても抽象度の高いレベルでは状態遷移マシンを使うことが好ましい．

操作可能状態に到達すると，周期的なタイムアウトイベントが振舞いを起動する．タイムアウトが発生すると，操作可能状態の内部で状態遷移が起こり，その遷移により起動されるアクションは周期的なアクティビティとして実行される．1つのオブジェクトが異なるレートで複数の周期的なアクティビティを行うこともでき，各アクティビティは固有のタイマーが起動する遷移ごとに処理される．オブジェクトの種々のアクティビティを1個所にパッケージ化できるという利点がある．

オブジェクトインタラクション・モデリング

ステートチャート図は個々のオブジェクトのリアクティブな振舞いの記述に有効である．一方，一群のオブジェクトの協調動作を記述することも大切である．UMLはいくつかの道具を提供しており，シーケンス図，インタラクション図，アクティビティ図がある．

リアルタイム開発で最も重要なのはシーケンス図である．リアルタイムという観点から，UMLではシーケンス図にタイミングマークを指定できる点がよい．タイミングマークはあるオブジェクトがメッセージを受けとった時点等の特定の時刻を示す．正確な意味定義や精度はアプリケーション依存であり，メッセージが受け手のキューに入れられた時と処理スケジュールされた時を区別することが重要なこともある．タイミングマークによって時間制約や時間区間の値を図示することができる．

リアルタイムシステムで複数オブジェクトに跨る振舞いの表現にプロトコルを用いる方法がある[4]．プロトコルはUMLのコラボレーションの特殊化であり，2つ以上のアクティブなオブジェクト間で交換される一連の有効なメッセージ群を規定する．プロトコルは特定のオブジェクトやクラスから独立に，プロトコルロールを用いて定義する．UMLのClassifierRoleとしてモデル化されるプロトコルロールは，そのロールに対して有効な入出力メッセージと順序関係を規定する．プロトコルは特定のコンテクストから独立に定義できるので再利用可能である．たとえば，図-2の3重冗長構成システムでは，多数決システム-サーバ間プロトコルを3度使っている．もう1つの利点は，継承機能を用いて特殊化できることである．共通の抽象プロトコルから一連のプロトコルを作成することができる．

構造的な側面のモデリング

設計者は構造的な側面の重要性を見落としがちである．構造的な側面の本質はリアルタイムシステムのアーキテクチャを表現することである．アーキテクチャを注意深く設計してあると，システムを進化させ寿命をのばすことができる．UMLは構造的な側面を厳密にモデリ

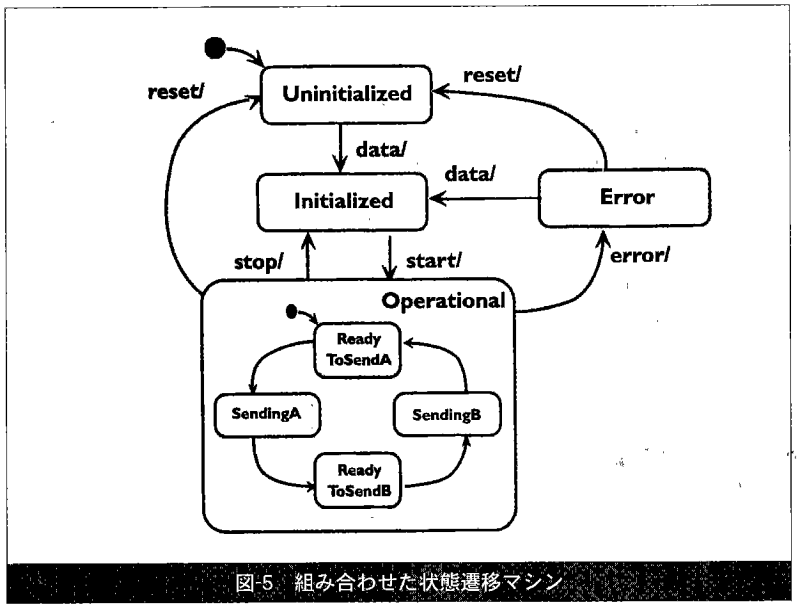

図-5 組み合わせた状態遷移マシン

ングするための基本的な道具として，コラボレーション図を持つ．リアルタイムシステムの基本的な3つの構造パターンのトポロジーを一般的に表現する．特別なクラスを用いる代わりに役割に相当するロールを用いてコラボレーションを定義し，オブジェクトとロールの対応関係を必要に応じて設定する．

アーキテクチャモデリング

最近の話題として，UMLのコラボレーション図を複雑なリアルタイムアーキテクチャに用いる試みがある[5]．重要な点はアーキテクチャ仕様を厳密に規定することにある．第1に，アーキテクチャモデルの一貫性や完全性を機械的に解析することが可能となる．第2に，実行可能なモデルにより異なるアーキテクチャの有効性を早期に厳密に評価できる．最後に，コラボレーション図からのプログラム自動生成技術を用いて，アーキテクチャ仕様からインプリメンテーションを直接得ることが可能となる．インプリメンテーションやその後の進化の過程でアーキテクチャが徐々にゆがむという，従来のアーキテクチャ崩壊を避けることができる．

アーキテクチャ記述の中心はカプセルと呼ぶ実体である．カプセルはアクティブオブジェクトであり，コラボレーション図で内部構造を表現する．コラボレーション図のノードはカプセルであり，それ自身がさらに内部構造を持つ．この繰り返しにより，任意の複雑な構造を表現することができる．仕様をクラスによって与えるので，カプセルを得るためには単にそのクラスのインスタンスを生成するだけでよい．カプセルはポートと呼ぶ特別な属性を1つあるいは複数個持つ．ポートはカプセルの双方向インタフェース機能を担う．カプセルに出入りするすべての通信はポートを通して行われる．その結果，カプセルは外部の実体を直接参照しなくてよい．カプセルは異なるコンテクストで再利用可能なアーキテクチャコンポーネントである．

UMLの重要性

UMLのステレオタイプを用いてリアルタイムに特有な概念を取り込み，従来の有効性が判明しているROOM（Real-time Object-Oriented Modeling）[4]やOCTOPUS[1]といったリアルタイム技法をUMLと融合することができた．これにより，複雑なソフトウェアを効果的に取り扱う最新の設計手法を活用することができる．この手法には，アーキテクチャレベルのモデリング，ステートチャート図による振舞い仕様記述，プログラム自動生成技術等がある．さらに，以上のモデリング技法をOMGで標準化する動きが進行中である．その結果，性能モデリングやスケジュール可能性解析といった従来からの設計技術をUMLモデルに直接適用できるようになる．また，このようなモデリング機能をRational Softwareの製品（Rose RealTime）が提供している．本稿で述べた技法はすでに大規模なプロジェクトで利用されている．

参考文献

1) Awad, M., Kuusela, J. and Ziegler, J.: Object-Oriented Technology for Real-Time Systems, Prentice Hall, NJ (1996).
2) Klein, M. et al.: A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems, Kluwer Academic Publishers (1993).
3) Kopetz, H.: Real-Time Systems: Design Principles for Distributed Embedded Applications, Kluwer Academic Publishers (1997).
4) Selic, B., Gullekson, G. and Ward, P.: Real-Time Object-Oriented Modeling, Wiley, NY (1994).
5) Selic, B. and Rumbaugh, J.: Using UML for Modeling Complex Real-Time Systems, ObjectTime Limited/Rational Software Whitepaper (1998), (http://www.objectime.com/).

(平成12年2月10日受付)