Rational Unified Process for Systems Engineering

第 2 部: システム・アーキテクチャー

執筆者: **Murray Cantor**
IBM Software Group
Rational 製品サービス
主席エンジニア

先月から、最新の Rational Unified Process for Systems Engineering® (RUP SE®) の進化の概要について解説する 3 部シリーズを開始しました。RUP SE は、Rational Unified Process® (RUP®) ソフトウェア・エンジニアリング・プロセス・フレームワークのアプリケーションです。RUP を使用する方は、現在利用可能な RUP Plug-In for SE が 2002 年に使用可能になった RUP SE v1 Plug-In であることにご注意ください。

第 1 部では、システムの考察、現代のシステム開発者が直面する課題と RUP SE がそれらを処理する方法、RUP SE 統一モデリング言語 (UML) ベース・モデリングと要求指定手法、および UML セマンティクスの使用について解説しました。今月の第 2 部では、システム・アーキテクチャーに注目し、RUP SE アーキテクチャー・フレームワークを紹介します。ここでは、複数のビューポイントからシステム内部について説明します。10 月に掲載される第 3 部は、要求分析および下位への流れについて説明し、RUP SE フレームワークの要素に対する要求および仕様を派生する方法を紹介します。この中で、複数のビューポイント全体からアーキテクチャー要素の仕様を一緒に派生するという今までにない手法の結合実現メソッドについて説明しています。第 3 部では、RUP SE のプログラム論についても説明しています。

編集者注記: RUP SE v1 Plug-In は 2002 年に一般に利用可能になり、このプラグインの v2 は 2003 年の 6 月に利用可能になりました。このシリーズの情報は v2 に対応していますが、この記事の中ではプロセス・フレームワークで可能な拡張機能についてもいくつか紹介しています。RUP SE Plug-In (v1 および v2) は、IBM Rational Developer Network からダウンロード可能です (権限が必要です)。

定義

RUP SE を明確に理解するには、いくつかの用語および概念に関する基礎知識が不可欠です。[ほかの標準では、これらの用語に別の定義をしている場合があります。ここで私たちが目指していることは内部的な整合性です。]

システムの分解: システム・エンジニアリングの成功は、多くの物事を一度に推論できる能力によって決まります。システム・レベルの分解は、この能力を実現するための強力な手法の 1 つです。

システムは、次の 2 つの方法で分解できます。

- 論理分解を使用してさらにシステムに分解します。これは、「システムのシステム」分解と呼ばれます。
- デリバリー・システムを構成するシステム・コンポーネントに分解します。

システム・モデル・ディメンション: RUP SE システム・モデルには 2 つのディメンションがあり、システムの設計および構成に携わる複数のチームで検討事項を分けることができます。

- **ビューポイント・ディメンション:** 限定された 1 組の品質の検討事項を扱うコンテキスト。
- **モデル・レベル・ディメンション:** 特定の設計レベルの詳細を取得する UML 図。

モデル: 全分野の検討事項、特性レベル、およびモデル・エンティティー関係を取得するビューを含む、システムの表現。

モデル・レベル: 各モデルを作成できる抽象化のレベルです。詳細を隠したり、カプセル化した概略的なものから、詳細および明示的な設計の決定事項を明らかにした具体的なものまでがあります。

ビューポイント: 名前が示すように、ビューポイントは、システムに関するいくつかの局面または検討事項を確認できる概念的な「位置」で、概念上のフィルターを形成する概念および規則セットの適用を意味します。システムを理解するには、通常は実際のシステム自体を検討するだけでは不十分です。そのため、関連するさまざまなビューポイントを表すようにモデルを作成します。

ビュー: 特定のビューポイントから関係のあるエンティティーを示すモデル・レベルの射影です。この射影は、一般的にいずれかの図によって示されます。ビューポイントと (抽象化の) モデル・レベルが交差するところには、その抽象化レベルでそのビューポイント (検討事項) に関係のあるモデルのビューが含まれます (あるいは少なくとも識別されます)。

ビューポイント

RUP SE フレームワークは、表 1 に示すビューポイント・セットを提供します。

表 1: システム・モデルのビューポイント

ビューポイントが表すもの		検討事項
作業者	システム・ワーカーのロールおよび責任	• 作業者のアクティビティー • 自動決定 • 人とシステムの対話 • ヒューマン・パフォーマンス仕様
論理	希望する振る舞いのためにコラボレーションする UML サブシステムの結合セットとしてのシステムの論理分解	• ユースケースの実現に適したシステム機能 • 拡張性および保守容易性 • 内部での再利用 • 良好な結束性および接続性
物理	物理コンポーネントのシステムおよび仕様の物理分解	• 機能を提供し、補足要求を満たすために適切な物理的特性
情報	システムによって保管および処理される情報	• データを保管できる十分なキャパシティー • データにタイムリーにアクセスできる十分なスループット
プロセス	制御のスレッド。計算要素を実行します。	• 並行性および信頼性のニーズに対応する処理の十分なパーティショニング

表 1 のビューポイントは、ソフトウェア集約的なシステムにおける最も一般的なものの一部です。多くのシステム・アーキテクチャーでは、ドメイン固有のビューポイントがさらに必要です。例えば、安全性、セキュリティー、および機械的なビューポイントなどです。

ビューポイントは、システム・アーキテクチャーおよび設計で取り上げなければならないさまざまな分野の検討事項を表します。システムの利害関係者または専門家の検討事項がアーキテクチャー全体にとって重要である場合、おそらく設計の決定を把握するビュー・セットが必要になります。

さまざまなビューポイントを管理できるスキルがあるスタッフ・メンバーによってシステム・アーキテクチャー・チームを構成することが重要です。このチームは、作業者ビューポイントに対する責務を負うビジネス分析者とユーザー、論理ビューポイントに注目するソフトウェア・アーキテクト、物理ビューポイントに注意を払うエンジニア、およびドメイン固有のビューポイントに関する専門家などで構成されます。

モデル・レベル

ビューポイントに加えて、システム・アーキテクチャーを構築するには、仕様のレベルも必要です。アーキテクチャーを作成すると、概略的で抽象的な仕様からより具体的で詳細な仕様に発展していきます。RUP SE には、RUP のガイドラインに合わせて、表 2 に示す 4 つのアーキテクチャー・モデル・レベルがあります。

表 2: RUP SE モデル・レベル

モデル・レベルが表すもの	
コンテキスト	システムおよびそのアクター
分析	概念的なアプローチを確立するビューポイントごとの初期システム・パーティショニング
設計	ハードウェア、ソフトウェア、および作業者への分析レベルの実現
実装	固有の構成への設計モデルの実現

これらのレベルを通じて、抽象的な設計から物理的な設計に進んで行きます。コンテキスト・モデル・レベルは、システムと対話するすべての外部エンティティー (アクター) を取得します。これらのアクターは、システムを配置するエンタープライズの外部または内部のいずれにすることも可能です。いずれの場合も、アクターは人または他のシステムにできます。分析レベルでは、ハードウェア、ソフトウェア、および作業者の選択項目はパーティションに反映されません。代わりに、システムが実行する必要があるものとその作業を提供する方法を分類する設計アプローチが反映されます。設計レベルでは、必要なハードウェアとソフトウェアのコンポーネントおよび作業者の役割などに関する決定を下します。実装レベルでは、ハードウェアおよびソフトウェア技術に固有の選択を行い、設計を実装します。例えば、設計レベルでは、データ・サーバーを指定します。実装レベルでは、ある特定のデータベース・アプリケーションを実行する特定のプラットフォームを使用することを決定します。

以上のレベルで追跡可能性を維持することが重要です。エンタープライズまたは任務が変更されたら、コンテキスト・レベル・ビューと、それよりも低いレベルの影響を受けるビューを一緒に修正する必要があります。基盤となる技術が変更されると、実装レベルが影響を受け、設計レベルも影響を受ける場合があります。まとめると、エンタープライズの変更の影響が下位に流れるのに対して、技術の変更の影響は上位に流れます。

システム・アーキテクチャー・ビュー

次のステップは、さまざまなビューポイントおよびモデル・レベルから、アーキテクチャーを表すビュー・セットにシステム・アーキテクチャーを取得することです。表 3 の各セルに、システムのビューを示します。実装レベルでは、システム構成ごとのハードウェアおよびソフトウェアのコンポーネントの実現を単一の図に取得することに注意してください。

表 3: RUP SE モデル・フレームワーク

モデル・レベル	**モデル・ビューポイント**				
	作業者	***論理***	***情報***	***物理***	***プロセス***
コンテキスト	UML 組織ビュー	システム・コンテキスト図	エンタープライズ・データ・ビュー	エンタープライズ局所 (エンタープライズ・リソースの分配)	ビジネス・プロセス
分析	汎化システム・ワーカー・ビュー	サブシステム・ビュー	システム・データ・ビュー	システム局所ビュー	システム・プロセス・ビュー
設計	システム・ワーカー・ビュー	サブシステム・クラス・ビュー ソフトウェア・コンポーネント・ビュー	システム・データ・スキーマ	記述子ノード・ビュー	詳細プロセス・ビュー
実装	*作業者の役割仕様および指示*	構成: 配置図 (ソフトウェアおよびハードウェア・システム・コンポーネント使用)			

モデル・レベル、ビューポイント、およびビューの関係は次の図 1 のようになります。

モデル・レベル	モデル・ビュー・ポイント				
	作業者	論理	物理	情報	プロセス
コンテキスト	UML 組織ビュー	・システム・コンテキスト図 ・システム・ユースケース図	エンタープライズ局所（エンタープライズ・リソースの分配）	エンタープライズ・データ・ビュー	ビジネス・プロセス
分析	汎化システム・ワーカー・ビュー	・サブシステム・ビュー ・サブシステム・コンテキスト図 ・サブシステム・ユースケース図	システム局所ビュー	システム・データ・ビュー	システム・プロセス・ビュー
設計	システム・ワーカー・ビュー	・サブシステム・クラス・ビュー ・ソフトウェア・コンポーネント・ビュー	記述ノード・ビュー	システム・データ・スキーマ	詳細プロセス・ビュー
実装	作業者の役割仕様および指示	構成：配置図（ソフトウェアおよびハードウェア・システム・コンポーネント使用）			

図 1: モデル・レベル、ビューポイント、およびビュー

ドメイン固有のビューポイントでも、1 つ以上のレベルについて適切な成果物が必要です。このフレームワーク内のプロジェクト成果物セットは、プロジェクト開発ケースの一部になっていなければなりません。それぞれのビューポイントについて簡単に見ていきましょう。

注: この文書の執筆時点では、UML 2.0 は導入準備中で、OMG は UML のシステム・プロファイルへの提案依頼書をリリースしています。UML 2.0 およびシステム・プロファイルが導入されると、RUP SE ビューのセマンティクスが更新され、これらの標準をフル活用できるようになります。

作業者ビューポイント

作業者は独自にビューポイントを持つことが保証されています。作業者は論理エンティティーでもあり、物理エンティティーでもあります。サービスを提供して、他の論理エンティティーとコラボレーションするように指示された場合は、論理エンティティーになります。パフォーマンス、反応性、および能力に関して制限されるという点では物理エンティティーです。もちろん、作業者は、モデル内でさらにサブディビジョンに分割されないブラック・ボックス・エンティティーです。

作業者とシステムの自動化された部分との対話方法を推論することは、システム・エンジニアリングの得意とすることです。作業者ビューポイントは、この推論を可能にする設定値を提供します。

また、*システム・ワーカー*と*ビジネス・ワーカー*は同じではないことに注意してください。システム・ワーカーは、システムの一部である人間です。彼らはシステム・アクターでは*ない* ため、システム・サービスの提供の一部を担当します。RUP SE フレームワークでは、システム・ワーカーはステレオタイプ化されたクラスとして表されます。作業者は、お互いに依存関係があるか、他の関係に依存している場合には関連付けることができます。汎化システム・ビューでは、一般のシステム・ワーカーはあまり詳細に表されません。

アプリケーションによっては、システムの自動化された部分に抽象化を導入すると便利です。マシン は、実現に作業者が含まれないという点で汎用システムとは異なります。汎化された作業者およびマシンを下位へ流れるワークフローで使用して、作業者の仕様を決定し、自動化の決定について推論することができます。図 2 に、作業者図の例を示します。

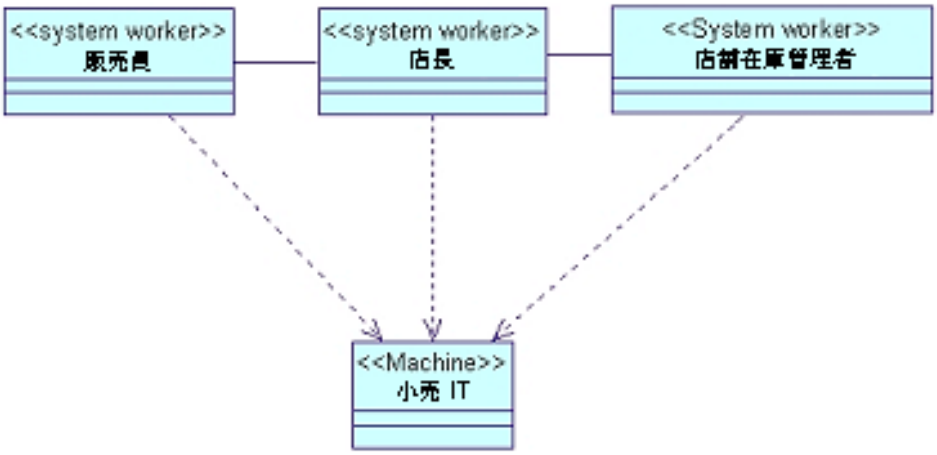

図 2: RUP SE 作業者図

例えば、船のシステムをモデリングした場合、分析モデル・レベルでは船員を一般のシステム・ワーカーとして表すことができます。しかし、設計レベルでは、多くの固有の船員役割を定義できます。

注: 作業者図にステレオタイプ化された分類子の *マシン*をさらに組み込み、自動化の決定をサポートすることもできます。以下で説明する結合実現メソッドでは、マシンは自動化を通じてサポートされるホワイト・ボックスの論理ステップを実行します。

論理ビューポイント

論理ビューポイントは、オブジェクト分析者の間で最もよく知られているものです。ここには、さまざまな抽象化レベルで、システムを実現するオブジェクトの種類を記述します。論理ビューポイント内のビューの要素は、クラスおよび サブシステムです。UML 1.4 では、システムとサブシステムは分類子とパッケージから継承されます。サブシステムの分類子とパッケージの両方の性質を取得する UML 構文はありません。通常、UML では、サブシステムは依存関係を持つパッケージとして表されます。RUP および RUP SE では、プロキシー・クラスを使用して分類子セマンティクスを表します。RUP SE では、必要に応じてプロキシーおよびパッケージを *システム* または *サブシステム* としてステレオタイプ化し、必要に応じて前述のシステムのセマンティクスをサブシステムに追加します。図 3 に、共通表記を使用したクリック・アンド・モルタル小売システムの UML サブシステム・ビューを示します。この図のパッケージの箇所に、サブシステム分類子を使用することも選択できます。

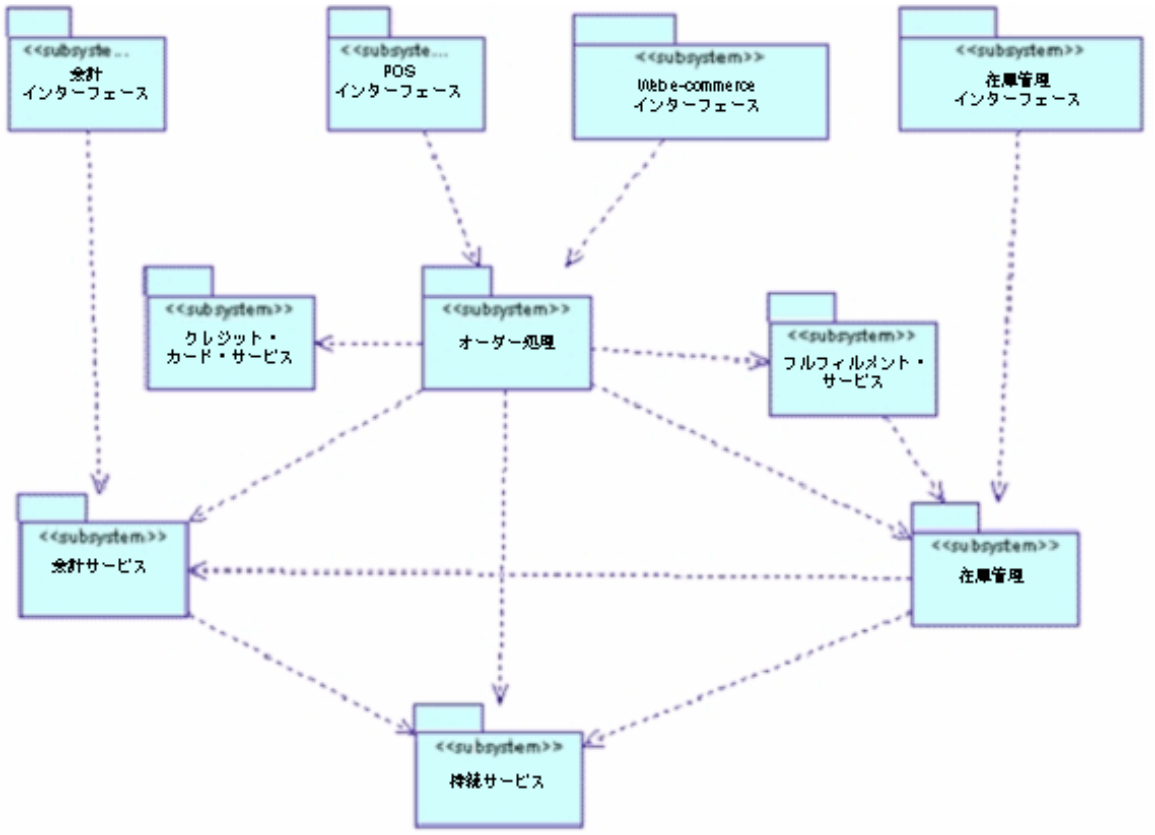

図 3: クリック・アンド・モルタルのサブシステム図
クリックして拡大

物理ビューポイント

システム・エンジニアリングでは、物理リソースはシステムの一部分または局面です。したがって、システムの物理的実現の要素のプロパティーについて推論できるセマンティクスを示す必要があります。つまり、システム・エンジニアリング環境には結果として、ハードウェアの詳細仕様が作成または獲得されます。システム・エンジニアリングにはハードウェア・エンジニアリングの作業分野 (機械、電気) は含まれませんが、ハードウェア設計チームへの入力データとして使用できる仕様は含まれます。

表 3 に示すように、RUP SE は分析レベルの *システム局所ビュー* という物理ビューポイント図を使用します。物理ビューポイントでは、システムは論理サブシステム・サービスを提供する要素に分解されます。局所図はこの分解の最も抽象化度が高い表現です。この図は、特定の地理的位置に限定せずに処理を行うか、特定のハードウェアに対する処理機能の実現もせずに処理を行うことを表します。局所は、リソースが隣接しているかどうかを参照し、設計モデルに取得されるロケーションを参照するとは限りません。例えば、局所ビューは、システムが宇宙衛星と地上局での処理が可能であることを示すことができます。各局所でホスティングされる処理は、設計時の重要な考慮事項です。

局所図は、初期パーティショニング、システムの物理要素の配布方法、およびそれらの要素の接続方法を示します。処理の局所性のほとんどが、主に機能外要求を考慮するときに発生する問題であることから *局所* という用語が使用されます。

図 4 に示すように、局所図は次の 2 つの要素で構成されます。

- **局所:** システムの概念上の物理パーティショニングを可能にするための物理リソースのグループ化。このアイコンは丸みを帯びた立方体です。
- **接続:** データ、サービス要求、または I/O エンティティーの受け渡しに使用するための局所間のリンク。UML では、接続はステレオタイプ化された関連として表されます。

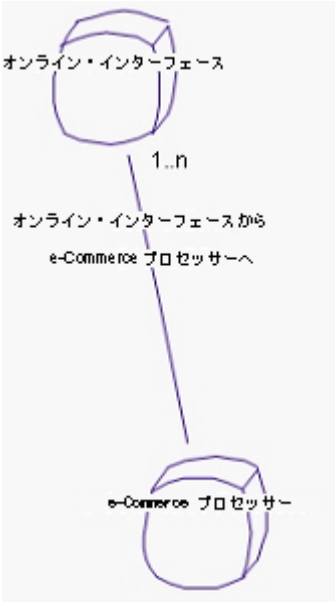

図 4: 局所図の要素

局所セマンティクス

局所は、システム・クラスの物理的特性を実現する際に使用され、そのセマンティクスはシステムの物理的性質に関連付けられているものから得られます。特に、局所にはクラスおよびインスタンス属性があり、効率度はタグ付き値で獲得されます。局所には、次の 2 つのタグのデフォルト・セットがあります。

- **品質:** 信頼性、可用性、パフォーマンス、キャパシティーなど
- **保守容易性:** コストおよび技術的なリスク

これらの局所特性は名目上のセットを形成します。各開発チームは、プロジェクトに最適な特性セットを決定する必要があります。この決定が、開発ケース仕様アクティビティーになる場合があります。

局所特性は、派生された要求を満たすように設定されます。特性と要求には微妙な違いがあります。例えば、正当なエンジニアリング上の理由から、要求よりも上の局所を指定する場合があります。

以下の『**局所、サービス、およびインターフェース**』のセクションで、局所によるサブシステム・サービスのホスティングを示します。

接続セマンティクス

局所は、接続によって結合され、局所間の物理リンクを表します。接続は、タグ付き値を持つステレオタイプ化された関連で、ここでも特性を獲得します。名目上の接続タグには、以下のものがあります。

- **スループット:** 転送速度、サポートされるプロトコル
- **保守容易性:** コスト、技術的なリスク

局所がサービスをホスティングするため、接続はサービス呼び出しを渡さなければなりません。実際に、システム内で考慮すべきフローには少なくとも以下の 3 種類があります。

- 制御フロー
- データ・フロー
- マテリアル・フロー

例えば、自動車のスロットルについて考えてみます。スロットル・リンクは、サービス要求 (開閉) をスロットルに送信する*コントロール* 接続です。ガス・ラインもスロットルへの接続です。ガソリン自体はサービス要求ではなく、むしろスロットルがサービスの実行に使用する*原材料* です。最後に、スロットルへの応答の調整に使用される環境および自動車の状況データ・ストリームが進んで行くスロットルへのネットワーク・*データ* 接続があるかもしれません。

局所およびノード

UML ノードが処理能力とメモリーを持つ分類子であることを思い出してください。[1]
配置図で使用した場合、UML ノードのセマンティクスでは、ソフトウェア・コンポーネントのプロセッサーのホスティングについての推論が可能です。暗黙の前提条件は、物理リソースが考慮対象のソフトウェアの外にあることです。例えば、ソフトウェア・エンジニアリングにおいて、ほとんどの場合、ハードウェアはオペレーティング・システムよりも下の層を使用可能にするものと見なされます。

UML は、配置図に以下の設計および実装レベルの成果物を提供します。

- **記述子図** (設計レベルに対して)
- **インスタンス図** (実装レベルに対して)

特に、インスタンス配置図は、ハードウェアおよびソフトウェアの構成および実際の選択を把握し、物理ビューポイント内で実装レベルとして機能するシステムの分析および設計の基本を提供するものです。「UML Reference Manual」では、配置図のインスタンス・バージョンを「ランタイム処理ノードと、そこに存在するコンポーネント・インスタンスおよびオブジェクトの構成を示す図」と説明しています。[1]

RUP SE でも、この意図が保たれています。ここでのノードは、設計および実装モデル・レベルでソフトウェアを実行する物理リソースを指定する際に使用される特殊な局所です。しかし、局所の一種として、RUP SE ノードをステレオタイプ化して、すべての局所セマンティクスを含めるこ

とができます。これらのセマンティクスと UML の標準ノードは異なることに注意してください。局所がステレオタイプ化されたノードというよりは、むしろノードがステレオタイプ化された局所です。UML 2.0 では、より良い方法で物理パーティショニングを処理できます。

局所、サービス、およびインターフェース

局所は、論理サービスを提供する物理リソースを指定します。実際には、各局所は、1 つ以上の論理サブシステムのサービスのサブセットを提供します。それらのサービスは、下記の結合実現ワークフローの結果によって決定されます。

指定された局所でホスティングされるサブシステム・サービスのセットは、次の 2 つの方法で取得できます。

- ホスティングされるサブシステム・サービス

  文書の調査

- 関連するサブシステム・インターフェース

最初の方法は、よりシンプルで、要求文書を局所に関連付けます。2 番目の方法では、より高度な UML の使用を求められます。サブシステムは分類子で、そのサービスは分類子操作です。それに加えて、UML では、サブシステム・サービスをインターフェースに編成する操作が可能です。すなわち、インターフェースはサブシステム・サービスのサブセットです。この 2 番目のアプローチでは、それぞれのサブシステムに必要なインターフェースを定義してから、適切な局所に割り当てます。一般的に、局所には複数のインターフェースが関連付けられます。

設計トレード

「設計トレード」は、共通システム・エンジニアリング手法の名前です。代替設計アプローチ・セットの作成、その代替策のコスト、品質、および実現可能性の分析、および最適なソリューションの選択を行います。局所ビューは、複数の局所図を組み込むことで設計トレードをサポートします。それぞれの図は、システムの物理分解について別個の概念的なアプローチを提示します。

図 5 および 6 は、多くの小売店、中央倉庫、および Web が存在するクリック・アンド・モルタル・エンタープライズへの異なるエンジニアリング・アプローチを文書化した局所図です。最初のソリューション (図 5) は店舗の処理能力を表現しています。2 番目のソリューション (図 6) は、中央局のプロセッサーに直接接続されているすべての端末を示しています。いずれの場合も、設計を実現するために必要な局所の特性を設定できます。現在、多くの方は図 5 の方が良い設計を示していると考えるでしょう。しかし、数年後には図 6 のソリューションの方が優れていると見なされるかもしれません。

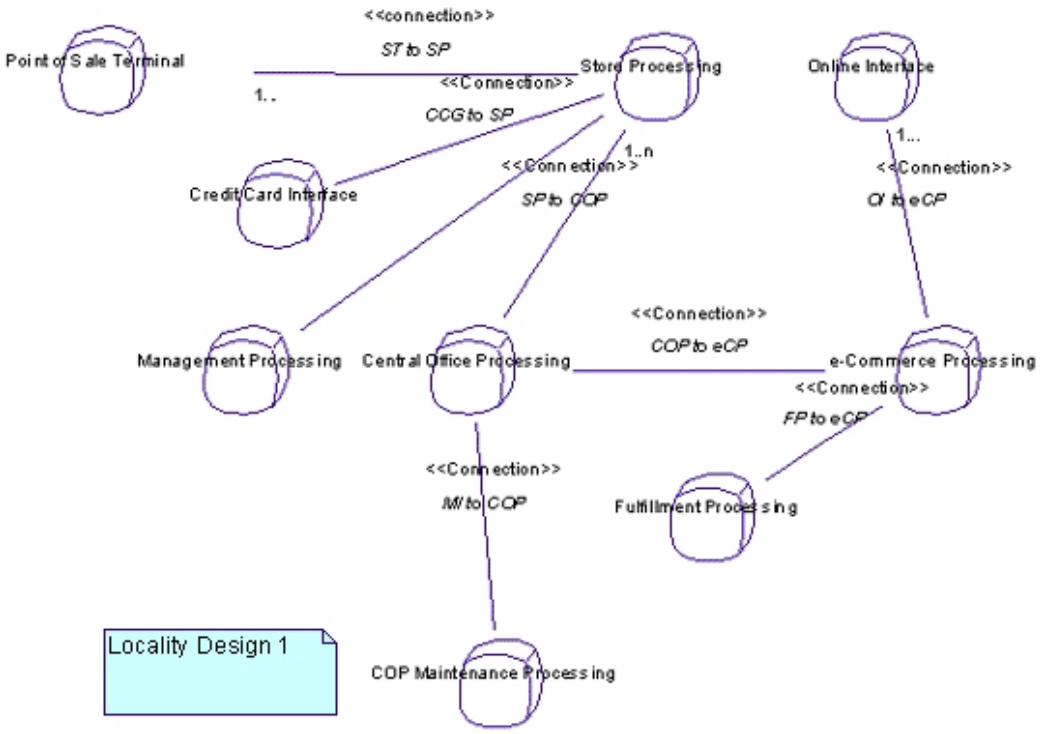

図 5: システム局所ビュー -- 例 1

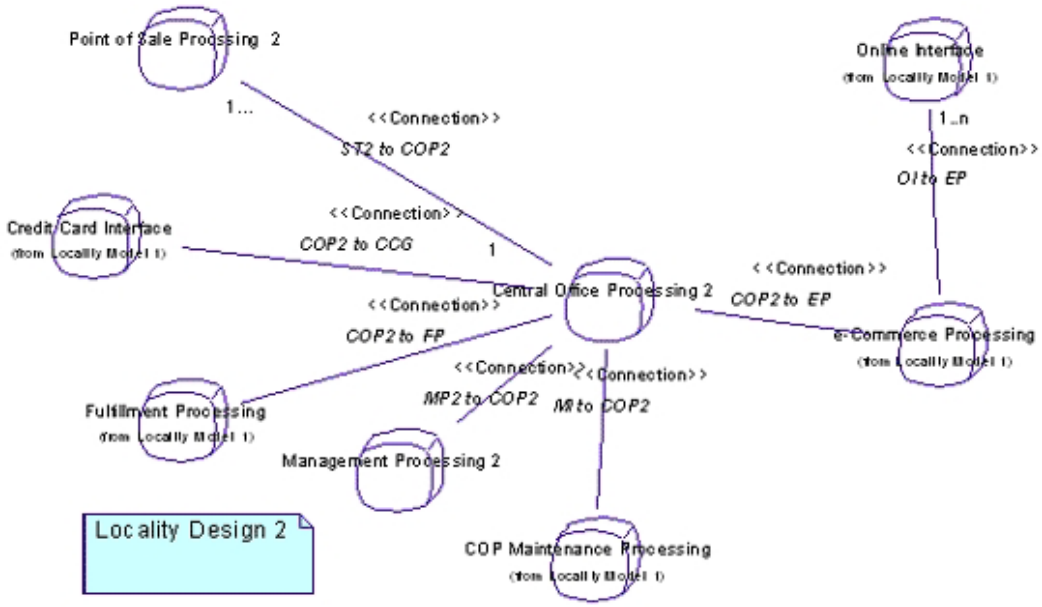

図 6: システム局所ビュー -- 例 2

局所の分解および実現

サブシステムと同様、局所は、階層によってさらに詳細な局所に分解できます。集約を使用して、局所とサブ局所を関連付けようとします。しかし、物理分解での全体と部分の関係と、通常はクラス集約で表される関係の間には重要な違いがあります。一般的な使用法では、クラス・オブジェクト (全体) が他のクラス・オブジェクト (部分) から集約されると、全体の属性には各部分の属性が含まれます。システム局所内の全体の属性は、部分の属性の*関数* です。単純な例として、全体の重みは各部分の重みの合計になります。ほとんどの場合、全体の属性と部分の属性の関係はより複雑で、現在 UML には属性間の機能的な関係を表すセマンティクスはまだありません。機能を実現する専用属性および操作を使用して、関係を取得するワークアラウンドをモデルに挿入することができます。

局所を物理コンポーネントとして実現するときには、階層による実現をお勧めします。すなわち、各コンポーネントは、ただ 1 つの局所の実現の一部になります。そうしなければ、局所とコンポーネントの間に派生された機能外要求の追跡可能性を保つことは困難です。しかし、局所間でコンポーネントを再利用可能にしたい場合は、この提案に従う必要はありません。その場合は、もちろん、コンポーネントは、局所間のシステム要求の下位への流れの中で見出された最も厳しい要求を満たさなければなりません。

それだけでなく、同じ局所に複数の実現を作成して、複数システムの実装への下位の流れを作ることに意味がある場合があります。例えば、さまざまな実装により、幅広い価格/パフォーマンスのポイントをサポートする製品ラインがある場合などが考えられます。

情報ビューポイント

オブジェクトとリレーショナル・データベースのいずれのモデリングにも UML を使用するというよく練られたプラクティスにより、RUP SE では情報ビューポイントを利用することができます。システム・モデル内でデータベース・モデリングを維持するということは、データ・クラスと機能クラスの間の関連をサポートし、かつデータベース・コンポーネントを局所に割り当てることで、システム全体の整合性が保たれることに注意してください。

プロセス・ビューポイント

プロセス・ビューポイントも、標準 UML を使用して表現されます。[3] 図 7 に、システム・プロセス・ビューの例を示します。

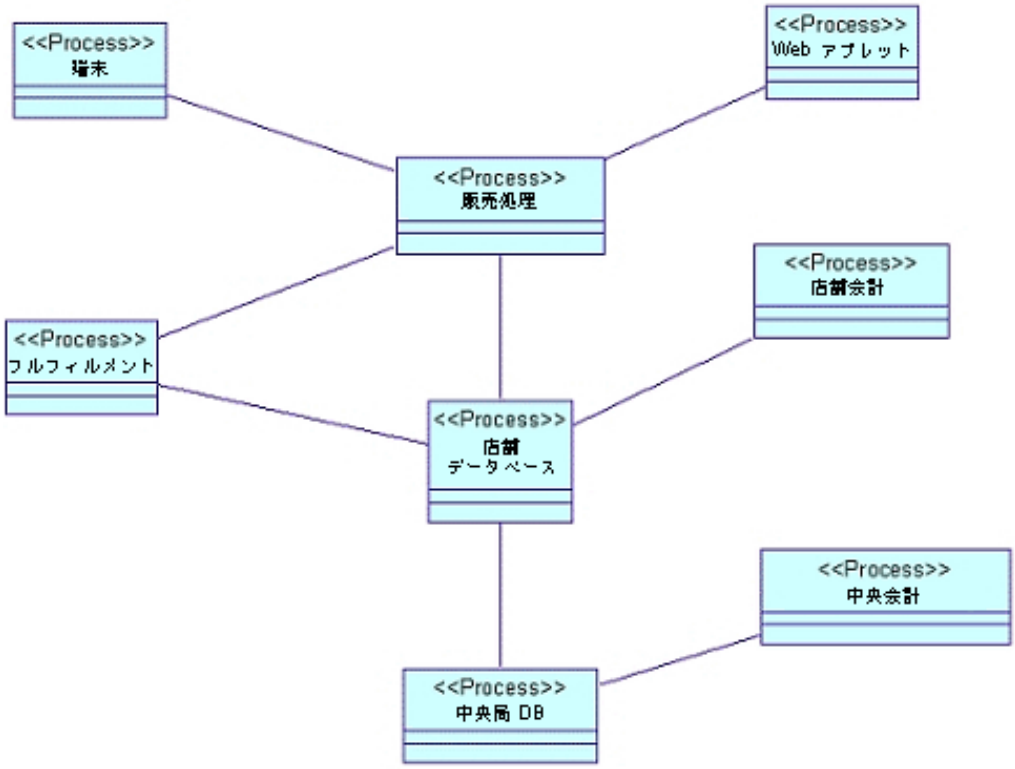

図 7: システム・プロセス・ビューの例

モデル・レベル間の移動

モデル・レベルを下に移動すると、モデルは正確さではなく、明確さが増します。各レベルでモデル要素を指定する際にはできるだけ正確にする必要があります。なぜなら、各レベルでの正確さにより、プロセスのシステムおよび作業分野の理解が増すためです。レベルを下げると、各ビューがより明確に決定され、実装レベルでの構成項目が得られます。重要なことは、*あるレベルのモデル要素が次のレベルの要求を設定する* ということです。あるいは、図 8 に示すように、各モデル・レベルが、それよりも上のレベルで見出された要求を*実現* すると言えます。例えば、以下のようなものがあります。

- **分析モデル・レベル**は、コンテキスト・モデル・レベルに指定された要求に対応する方法を示します。
- **設計モデル・レベル**は、システム分析モデル・レベルから得られた要求に対応する方法を示します。
- **実装モデル・レベル**は、設計仕様に対応します。

図 8: 下位モデル・レベルが上位モデル・レベルで設定された要求を実現する
クリックして拡大

図 9 は、設計レベルで物理ビューポイントに記述子ノード図を組み込む方法の例を示しています。ここでは、各局所を実現する物理設計を示します。

各モデル・レベルはその次に低いレベルで実現される要求または仕様を設定することから、設計要素がどの程度仕様を満たすかを把握することで、レベル間の追跡可能性を保つことができます。

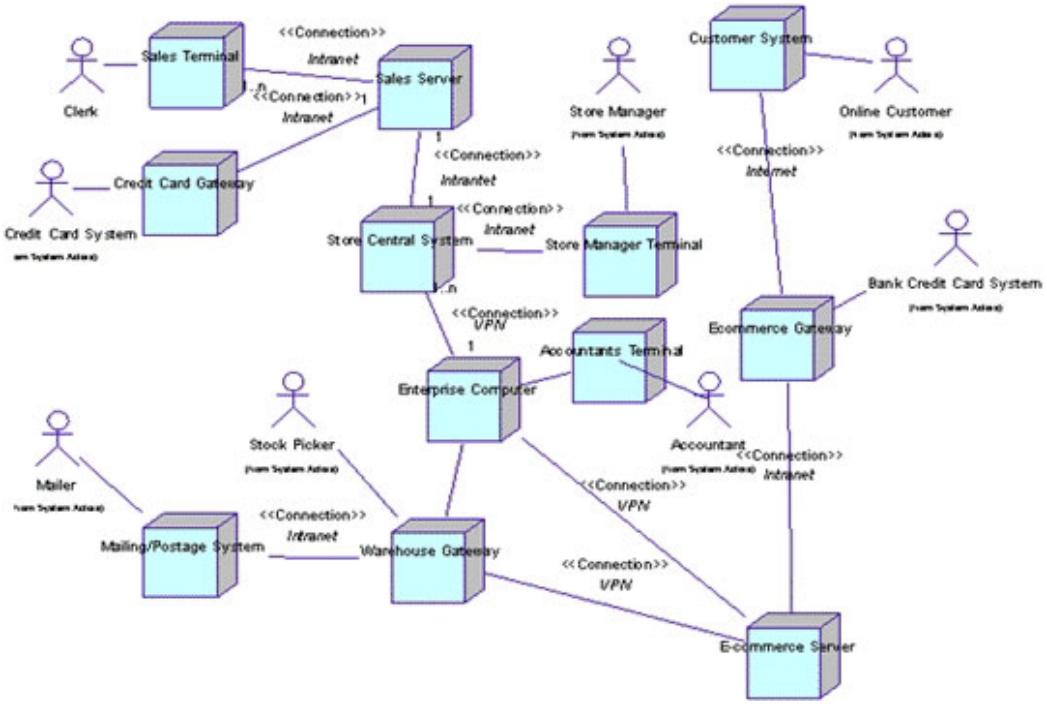

図 9: クリック・アンド・モルタル局所ビューの実現
クリックして拡大

実際には、チームが 1 つのモデル・レベルを作成すると、例えば、そこに指定された 1 つ以上の要素を実現できないなどのために、上位レベルを改訂しなければならないことが分かる場合があります。したがって、開発が進んでいくときに、実際に「凍結」されるレベルはありません。各レベルは開発の間中、保守されます。しかし、開発が進むと、通常、作業の焦点は 1 レベルずつ下がっていきます。

注

[1] Grady Booch、James Rumbaugh、James Jacobson 著「The Unified Modeling Language User Guide」Addison Wesley 発行 (1999 年)、358 ページより。

[2] Ibid、p.252ff。

[3] Ibid、p.455。

この記事で取り上げている製品またはサービスについて詳しくはここをクリックして、表示される手順に従ってください。ありがとうございました。