Rational Unified Process for Systems Engineering

第 3 部: 要求分析/設計

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

The Rational Edge の 8 月版から、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。第 2 部では、システム・アーキテクチャーに注目し、システムの内部を複数のビューポイントから記述する RUP SE アーキテクチャー・フレームワークについて紹介しました。そして、第 3 部では、要求分析および下位への流れ、さらに RUP SE フレームワークの要素の仕様について取り上げます。この中では、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導き出す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。また、RUP SE を使用したシステム開発についても簡単に解説します。

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 (http://www.rational.net) からダウンロード可能です (権限が必要です)。

RUP SE および要求

システムの慣例に従って、RUP SE では以下の 2 種類の要求に対応します。

1. **振る舞い要求** -- システムがエンタープライズ内での役割を果たすために行うこと。RUP SE では、システムの振る舞いは、ユースケースおよびその分析によってサービスに取得されます。ユースケースおよびサービスには、性能要求が関連付けられている場合があります。
2. **補足要求** -- 設計目標 (例えば、信頼性や所有コストなど) およびシステム属性 (例えば、データ容量や総重量) などの機能外要求。

RUP SE には、アーキテクチャー要素への要求を派生する以下のようなプロセス・パターンもあります。

1. 所定のモデル要素のブラック・ボックス要求または仕様を決定します。
2. そのモデルをホワイト・ボックス要素に分解し、それらの要素に役割および責任を割り当てます。
3. 各要素がブラック・ボックス要求を満たすために共同でコラボレーションする方法を詳しく調べます。通常、これには、何らかの形式のコラボレーション図が含まれます。
4. コラボレーションの分析を統合して、要素のブラック・ボックス要求を決定します。

このプロセス・パターンはよく知られているもので[1]、Friedenthal などが OOSEM (Object Oriented System Engineering Method)[2] に採用していることは特に興味深い点です。

したがって、システム仕様は、ユースケース、システム・サービス、および補足要求を定義することを意味し、それらに対処することで、システムはビジネス目的またはミッションを達成できます。RUP SE では、要求の*割り振り* と*派生* を区別しています。ブラック・ボックス要求がホワイト・ボックス要素に割り当てられている場合、要求は*割り振られています*。ホワイト・ボックス要素がブラック・ボックス要求に対応するために他の要素とコラボレーションする方法を調査することで決定される場合は、ホワイト・ボックス要求は*派生されています*。振る舞い要求も補足要求も、派生させることができます。派生した要求はシステム設計におけるアーキテクチャー要素の役割を考慮することに注意してください。

次の例を考えてみます。ほとんどの自動車には差動装置があり、その装置によって駆動軸を車軸に接続して、自動車がカーブを曲がるときに動輪の速度を変えて走行できるようにします。この機能は、内側の車輪の回転を外側の車輪よりも遅くして両方の車輪のトラクションを維持しなければならないことから不可欠です。

いずれかの車輪がトラクションを失うと、車輪が空転して横滑りするおそれがあり、ひっくり返ることもあるかもしれません。それでも、差動装置には自動車の*システム* 要求はありません。自動車のシステム要求は、自動車がカーブを走行するときにトラクションを維持するという単純なものです。これは、差動装置を伴わないさまざまな方法で実現することができます。例

えば、次のいずれかの代替手段が可能です。

- 単動輪 (SF 映画に出てくることがある三輪自動車など)。
- 動輪ごとに 1 個ずつで 2 個のモーター。何らかの電子制御方式による運転ソリューションを使用します。

この差動装置の規則は、技術的な観点から (すなわち、トラクション・コントロールだけでなく、全体的な安定性の維持、車内容積の最適化、資材メンテナンス・コストの管理などのさまざまな要求に対応する際により良い働きをする)、またはエンジニアがそれまでの優れた技術を活用しなければならないという設計上の制約から優勢であるため、主流になっています。

現在では、差動装置は自動車の必須要素ではありませんが、差動装置にシステム要求を割り当てるメカニズムはありません。むしろ、差動装置は、自動車の他の要素 (ステアリング、ブレーキなど) とのコラボレーションにおいて、連携して必要な動作を実現し、自動車が安全にカーブを走行できるようにするという役割を果たします。「車輪速度の調節」などの差動装置の動作は、システム要求および差動装置が担う役割から派生します。この動作は、派生であり、割り振りではありません。

さらに、差動装置は、システムの定義済み補足要求をサポートするために、派生した補足要求に対応する必要があります。例えば、差動装置には、重量および容積の予算、さらに信頼性の測定も含まれます。

ユースケースを実行するためにコラボレーションするサブシステムに派生した要求を使用することを*論理分解* と呼びます。同様に、割り振りによりサブシステム要求を決定することを*機能分解* と呼びます。一般的に、論理分解は品質システムに不可欠です。[3]

したがって、システム要求は、エンタープライズ・サービスおよびエンタープライズ内でシステムが果たす役割を理解することから派生します。分析モデルでは、システム・アーキテクチャー要素は、このシリーズの第 2 部の『**システム・アーキテクチャー**』セクションで説明したように、サブシステム、局所、およびプロセスになります。これは、各種類のアーキテクチャー要素の要求を決定する要求分析作業分野に含まれます。例えば、ビジネス・モデルを適切に使用することで、RUP SE では、エンタープライズをシステムとそのアクターにパーティショニングし、システム要求を派生することを提案します。そして、システム要求を決定するために、分析者はビジネス要求に対応するようにシステムとそのアクターをコラボレーションする方法を調査することができます。

以降のセクションでは、分析モデルのシステムおよび要素への機能要求を派生させる RUP SE のアプローチについて説明します。

ユースケースの下位への流れからの機能要求の派生

ユースケースの下位への流れは、システムおよびその要素への機能要求を派生するアクティビティーです。下位への流れは、あるモデル・レベル内をより詳細にするか、下位モデル・レベルで要素を指定する場合に適用できます。例えば、下位への流れを使用して、コンテキスト・レベルでシステム・サービスを決定できます。同様に、分析レベルで使用した場合は、サブシステム・サービスを識別して、サブシステムをさらにサブシステムに分けることができます。

ここで、*下位への流れは再帰的に適用できる* ということに注意することが重要です。言い換えれば、ホワイト・ボックス要素が次の適用ではブラック・ボックスになるということです。これにより、チームは、適切な特性レベルで大規模なシステムを推論できます。下位への流れのアクティビティーを繰り返し適用することで、チームは、抽象化レベルの整合性を保ちながら詳細さを増すことができます。また、ここでは並行設計が可能です。つまり、別々のチームがさらに設計を進めるために各ホワイト・ボックス・エンティティーをブラック・ボックス・エンティティーとして扱えるように十分に指定できます。このアプローチは、分析モデルの要素への要求を派生させるだけではなく、わずかな変更でビジネス要求からシステム要求を決定する際にも使用できます。

上記で説明した階層方式で下位への流れを実施すると、サービスとユースケースの間に、*ブラック・ボックス・サービスがホワイト・ボックス・ユースケースになる* という興味深い関係ができます。ユースケースには、ある目的を達成するためにコンテキスト内のエンティティーおよび要素がコラボレーションする方法を記述します。ここで、ユースケースの下位への流れの目的は、システム・サービスの配布をサポートすることです。サービスの実現はユースケース・シナリオで構成されます。各 UML サブシステムには、サブシステムがコラボレーションするシステムのアクターと、依存関係を共有する対等なサブシステムを示すコンテキスト図を作成できます (これらは、第 1 部の『**システム仕様**』で紹介したエンタープライズと内部アクターと同様です)。サブシステムの観点から見ると、サービスの実現というのは、厳密にはアクターとコラボレーションして役割を果たす方法です。これがまさにユースケース・シナリオです。下位への流れは、ユースケース分析で共通の価値観を発見することで変わることに注意してください。下位への流れの中でユースケースは、ブラック・ボックス・エンティティーに価値を持たせますが、参加しているアクターには持たせない場合があります。

シンプルな実現

ユースケースの下位への流れは、オブジェクト分析の基本的な慣例である*ユースケースの実現* の拡張機能です。ユースケースの実現は、ユースケース・シナリオの実施に参加するクラスを検出し、さまざまなクラスのオブジェクトのコラボレーション方法を見出すことから成り立ちます。実現では、コラボレーション中にやりとりされ、シーケンス図またはコラボレーション図に取り込まれるオブジェクトのメッセージの順序を指定する必要があります。実際には、シーケンス図を作成することで、多くの場合、オブジェクトをユースケースの実現に参加させるためにクラス操作で提供すべきメッセージを見つけられます。

RUP SE では、この実現の概念がいくつかの方法で拡大されます。まず、実現は設計よりも上のモデル・レベルに適用されます。例えば、エンタープライズとシステムの間に適用された下位への流れの結果によって、システム・サービスを識別できます。システムとそのモデル要素の間に適用された場合、下位への流れにより、以下の結果が得られます。

- サブシステムのユースケース調査。
- サブシステム・サービスおよびインターフェースの識別。
- 局所に対してホスティングされるサブシステム・サービスおよび/または サポートされるインターフェースの調査。

UML サブシステムに対するユースケースの実現を拡張するという考えは目新しいものではありません。例えば、UML サブシステムの実現は、多くの場合、*アーキテクチャー相互作用図* と呼ばれます。

以下に、システム・コンテキスト図を作成して、システム・サービスを識別するための下位への流れのステップを示します。

1. エンタープライズ・ホワイト・ボックスをコラボレーションするシステムのセットとしてモデリングします。
2. エンタープライズ・サービス、ミッションなどを実現するためのシステムのコラボレーション方法をモデリングします。
3. システムのコンテキスト図を作成します。
4. アクター (すなわち、システムとコラボレーションするエンティティー) を決定します。
5. I/O エンティティーを識別します。
6. システムとそのアクターの間の類似のコラボレーションをユースケースに集約します。
7. パフォーマンス、事前および事後条件などのユースケースの詳細を追加します。
8. システム・サービス、すなわち、ユースケースをサポートするためにシステムが行うことを識別し、類似のホワイト・ボックス・ステップを集約します。
9. エンタープライズ・ニーズの分析からシステム属性を追加します。

実現がクラスまたは UML サブシステムのような 1 種類のホワイト・ボックス要素で構成されるときには、これを*シンプルな実現* と呼びます。例としては、以下で表すエンタープライズからシステムへの下位への流れがあります。

手順 1: 結合実現

将来のバージョンの RUP SE では、上記のシンプルな実現は次のように*結合実現* に拡張され、サービスを実施する際に複数のビューポイントの要素をコラボレーションする方法を分析します。例えば、結合実現では、下位への流れは、論理、物理、および情報の要素のコラボレーションを同時に決定することで構成できます。

結合実現は、以下の手順から成り立ちます。

1. 参加ビューポイントを選択します。論理ビューポイントは必須です。
2. ブラック・ボックス・サービスを実現するホワイト・ボックス・ステップごとに、以下のことを行う必要があります。
    - 実行する論理要素を指定します。
    - 追加ビューポイントの参加方法をモデリングします。例えば、以下のようなものを組み込みます。
        - *物理ビューポイント* では、ホスティングする局所を指定します。局所が 2 つある場合には、2 ステップに分解します。
        - *プロセス・ビューポイント* では、実行するプロセスを指定します。プロセスが 2 つある場合には、2 ステップに分解し

ます。-*情報ビューポイント* では、使用する情報の処理をサポートするデータ・スキーマ要素を指定します。

このプロセス全体をとおして、以下の*結合実現規則* を適用してください。所定の論理要素のホワイト・ボックス・ステップで他のビューポイントに複数の要素が必要であれば、各ステップでそれぞれのビューポイントから必要なホワイト・ボックス要素が 1 つのみになるように、そのステップをさらに複数のステップに分割します。

3. 各ビューポイントに以下の相互作用図を作成します。
   - アーキテクチャー相互作用図
   - 局所相互作用図
   - プロセス相互作用図

4. 各ステップに対するパフォーマンス、正確性などの補足要求の予算を立て、相互作用図で評価/確認します。

手順 2: 結合実現でのリソースの指定

結合実現には、さまざまなアプリケーションが用意されています。例えば、システムから論理および作業者ビューへの下位への流れに使用して、自動化の決定について推論することができます。あるいは、システムから論理、物理、およびプロセス要素への下位への流れに使用できます (このアプリケーションについては、以下で詳しく説明します)。システムの物理リソースの仕様を明確にするには、以下のことを行う必要があります。

1. 初期分析モデル・レベル・ビュー (システム・ホワイト・ボックス) を作成します。これを行うには、以下のようにします。
   - 論理ビューにオブジェクト指向分析手法を使用します。
   - 局所ビューに物理的な考慮事項を適用します。

2. 結合実現を使用して、以下のような (アーキテクチャー上重要な) 各システム・サービス仕様をモデリングします。
   - UML サブシステムのコラボレーション・ステップ。
   - ホスティングする局所。
   - 実行するプロセス。

3. ホワイト・ボックス性能要求を取り込みます。つまり、ホワイト・ボックス・ステップへのブラック・ボックス性能要求の 予算を立てます。これを行うには、以下のようにします。
   - UML サブシステム・ユースケースを識別します。つまり、サブシステムごとに、そのサブシステムに伴うシステム・サービスを識別します。
   - サブシステムごとに、コラボレーション時のメッセージ

に集約手法を適用することでサービスを識別します。

- 局所ごとに、ホスティングされるサブシステム・サービスを調査します。
- プロセスごとに、実行したサブシステム・サービスを調査します。

4. システムとサブシステムのユースケース、および/またはシステムとサブシステムのサービスの間の追跡可能性を文書化します。

手順 3: コンテキストから分析モデル・レベルへの下位への流れ

サブシステム、局所、およびプロセスへのホワイト・ボックス・ステップの割り当てには、設計の決定事項のセットが必要です。各決定事項は、それぞれの分析要素がシステム全体の設計で果たす役割に詳細を追加します。割り当てプロセスの中で、チームが設計をリファクタリングして、ある要素から別の要素に所定のビュー内の責任を切り替えることを決定する場合があります。また、下位への流れによって、適切な詳細レベルが追加され、サブシステム、局所、およびプロセスの役割と責任をリファクタリングすることができます。

表 1 に、「クレジット・カード利用での閉店セール」システム・サービスのホワイト・ボックスの下位への流れの例を示します。ここでは、クリック・アンド・モルタル小売システムにサブシステム (図 6) および局所モデル 1 (図 8) を使用しています。

表 1: 結合実現表

システム・アクターのアクション	ブラック・ボックス予算編成済み要求	作業者のアクション	サブシステム・アクション	ホワイト・ボックス予算編成済み要求	局所	プロセス
お客様はクレジット・カードを提供します。	30 秒	店員はクレジット・カードを機械に通します。	POS 端末はクレジット・カード・サービスが検証を可能にするように要求します。	.5	POS 端末	端末
			クレジット・カード・サービスは銀行のクレジット・カード・システムを照会します。	28 秒	店舗サーバー	オーダー処理
			有効であれば、POS は領収書を印刷します。	.5 秒	POS 端末	端末
		店員は領収書に署名するようにお客様に要求します。				

次のステップは、UML サブシステムのユースケースおよびコンテキストを決定することです。UML サブシステム・コンテキスト・ビューは、システム・コンテキストと同様に、サブシステム、

そのアクター、および関連する I/O エンティティーで構成されます。サブシステムの場合、そのアクターは対等なサブシステムと、場合によっては、システム・アクターで構成されます。図 1 は、サブシステム・コンテキスト図の例を示しています。

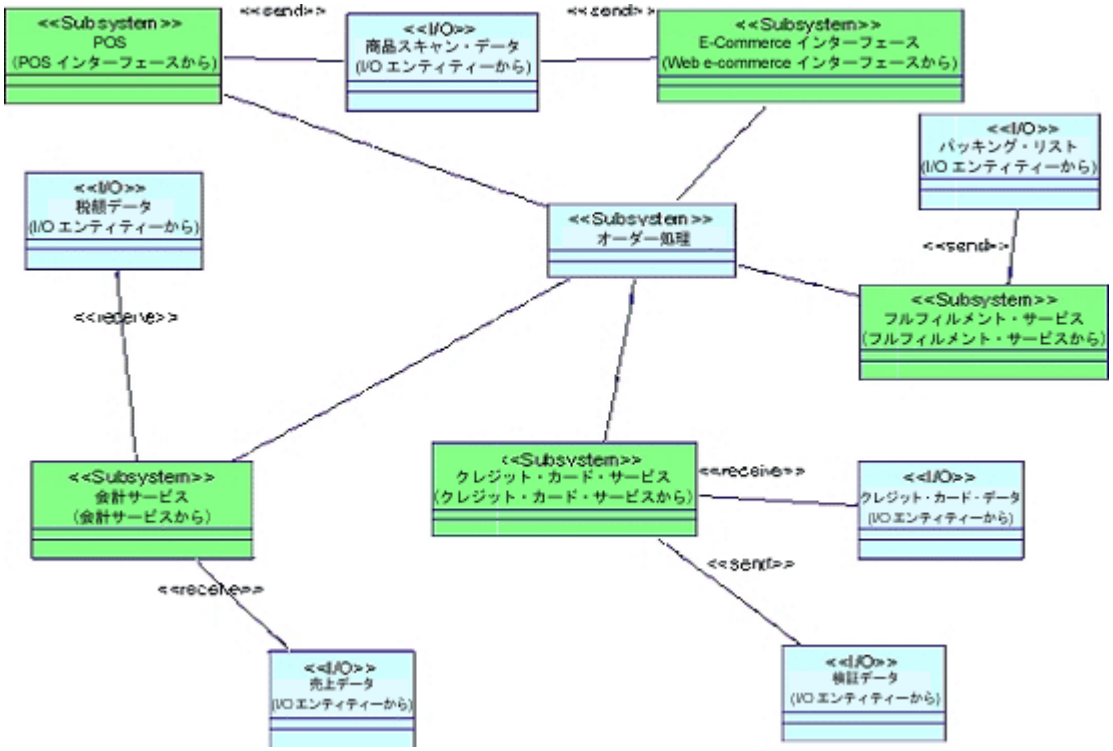

図 1: サブシステム・コンテキスト図

ユースケースには、価値サービスを提供するためにシステムとそのアクターがコラボレーションする方法を記述することを思い出してください。サブシステムの場合、システム・サービス自体が価値サービスになります。したがって、サブシステムごとに、そのユースケースが、サブシステムとコラボレーションするシステム・サービスになります。開発作業をサブシステム境界に沿って、またはテスト・ケース作成の基礎として分割する場合、サブシステムのユースケース調査を続けることが役に立ちます。

サブシステム・サービスは、サブシステムによってサービスのホワイト・ボックス・ステップを分類することで検出できます。サブシステムごとに、表 2 に示すようにホワイト・ボックス・ステップを分類し、類似のステップを集約します。その結果、それぞれのサブシステムによってサービスの仕様が提供されます。

表 2: 局所でホスティングされるサービスの調査の例

局所名:サービス 店舗処理			
局所責任: この局所は、中央店舗の販売取引および会計をホスティングします。中央局およびクレジット・カード処理へのインターフェースを提供します。			
サブシステム・サービス	**サブシステム**	**システム・サービス**	**サービス・ホワイト・ボックス・テキスト**
クレジット・カード・セールの開始	オーダー処理	売り上げを入力	オーダー処理によりセールス・リスト作成開始
商品データを追加	オーダー処理	売り上げを入力	スキャナー・データがオーダー処理に送信され、オーダー処理では在庫から、品名、価格、および課税状況を取得し、リストを更新
合計を計算	オーダー処理	売り上げを入力	オーダー処理で金額を合計し、税額を計算

サブシステム・サービスを決定したら、局所またはプロセスによってサブシステム・サービス・セットを分類できます。局所ごとにホスティングされるサービスの調査は、局所で実行される計算と、関連する性能要求を表します。この情報によって、局所に配置される物理コンポーネントの仕様への入力データが提供されます。同様に、各プロセスで実行されたサービスの調査は、ソフトウェア・コンポーネントの仕様への入力データになります。これらのアクティビティーにより、結合実現プロセスに以下のステップが追加されます。

- 局所ごとに、ホスティングされるサービスの調査 (表 2 に示すようなもの) を行います。
- プロセスごとに、実行したサービスの調査を行います。

コンポーネント仕様について以下でさらに詳しく説明します。

局所に対してサブシステム・サービスを関連付けるもう一つのアプローチは、局所でホスティングされる複数のサービスを含むサブシステム・インターフェースを定義してから、そのインターフェースを局所に関連付ける方法です。このアプローチには、サービスから局所への関連が UML モデル内に完全に包含されるという利点があります。

イベントのホワイト・ボックス・フローの中のテキスト記述は、シーケンスまたはコラボレーション図のセットとして表すこともできます。これらの図は分析要素間のトラフィックを伝達し、それぞれの図はオブジェクトが分析要素のプロキシー・クラスであるシーケンス図です。
各メッセージによってサブシステム・サービスが呼び出されます。図 2 および 3 に、表 2 に記述したイベントのフローに対するサブシステムおよび局所の相互作用図を示します。

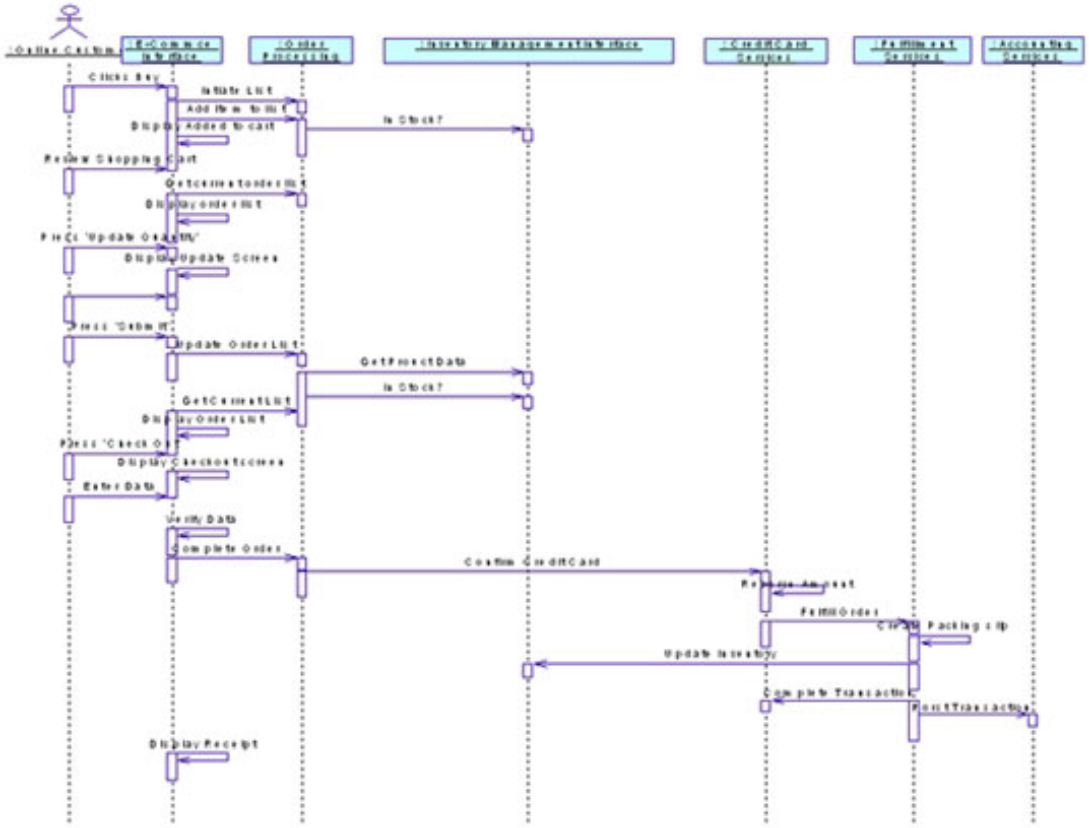

図 2: サブシステム相互作用図の例
クリックして拡大

図 2 は、サブシステムの結合および結束へのインサイトを提供しています。このインサイトは、サブシステム設計のリファクタリングに使用できます。例えば、ペアのサブシステム間のトラフィックが多い場合に、それらを結合するとその効果が分かります。図 3 に、局所相互作用図の例を示します。

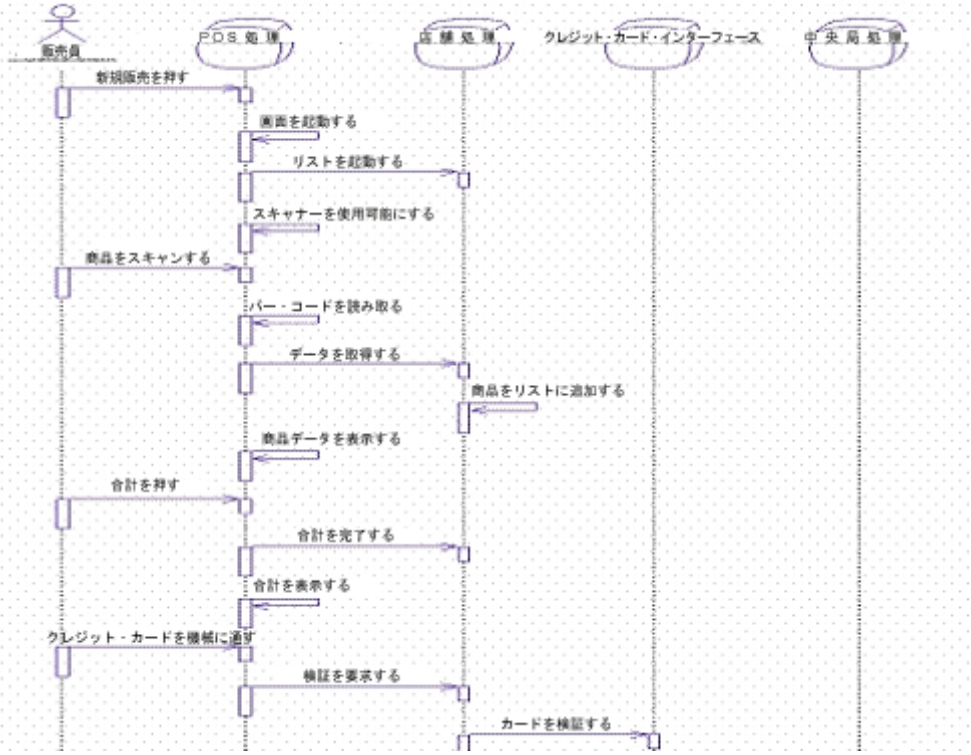

図 3: 局所相互作用図の例

図 3 のトラフィックは、局所間を流れる必要があるデータを示しています。
この情報を使用して、局所間の関連を指定します。

補足要求の下位への流れ

補足要求は、元々はシステム・クラス属性またはタグ付き値として取り込まれます。分析プロセスの一部として、システム設計者は初期局所図を作成します。局所図は、機能外の考慮事項を統合したもので、信頼性およびキャパシティーなどの機能外要求に対応する方法を扱うコンテキストを提供します。

標準的な技法では、キャパシティー、許容される障害率などの予算管理が可能です。これにより、派生された補足要求セットが局所要素ごとに出来上がります。局所特性は、それらの要求から決定されます。

コンポーネント仕様

分析からアーキテクチャーの設計レベルへの移行には、ハードウェアおよびソフトウェアのコンポーネント設計の決定が伴います。この設計レベルの仕様には、配置すべきハードウェア、ソフトウェア、および作業者の各コンポーネントで構成されます。

ハードウェア・コンポーネントは、局所の分析と、そこから派生した特性、およびホスティングされるサブシステム・サービスによって決定されます。この情報を使用して、局所の記述子レベルでの実現が可能です。記述子ノード図は、コンポーネント、サーバー、ワークステーション、作業者などを、特定の技術的な選択項目を示さずに指定します。図 4 は、図 8 に示す局所図を実現する記述子ノード図の例を示しています。フルフィルメント局所は、ウェアハ

ウス・ゲートウェイ、メール/郵便システム、および 2 人の作業者の 4 つのコンポーネントとして実現されます。

記述子ノードは、局所から割り振りまたは予算管理プロセスを通じて特性を継承します。

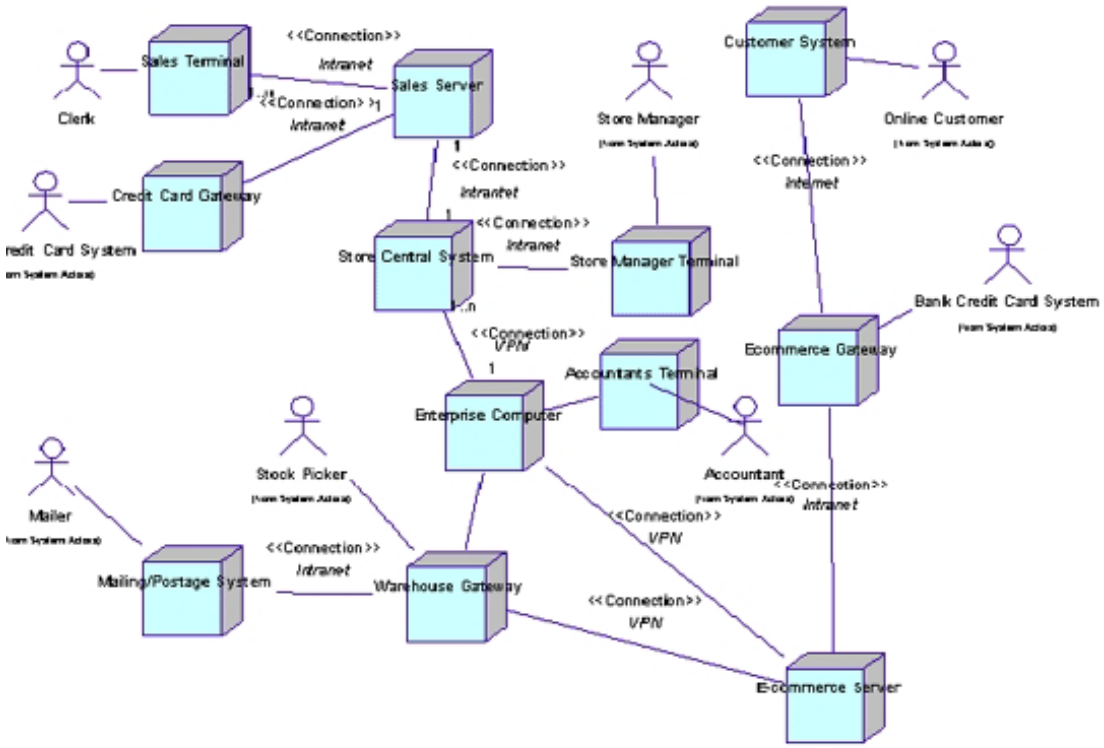

図 4: 記述子ノード図の例

実装ハードウェア・コンポーネント、つまり、実際に配置されるハードウェア・セットは、記述子ビューからコスト/パフォーマンス/キャパシティーのトレードを行うことで決定されます。実際には、システムは、さまざまな価格/パフォーマンスの目的を達成するために、複数のハードウェア構成を持つ場合があります。

コンポーネントは、オブジェクト・クラスのセットを指定してから、それらのクラスに関連するコードをコンパイルして実行可能ファイルにアセンブルすることで決定されます。十分に検討されたソフトウェア・コンポーネント設計には、以下のさまざまな検討事項が反映されていなければなりません。

- **局所** -- コンポーネントを実行する必要がある場所。
- **ホスティング** -- 実行コードのプロセッサー命令セットおよびメモリー 制限。
- **並行性** -- 信頼性および関連する検討事項に対応するための、別々のホストまたはメモリー・スペースへの処理の分割。

したがって、コンポーネントを指定するために必要な情報には、局所およびそこで実現されるハードウェア・コンポーネントでホスティングされるサブシステム・サービスの調査、プロセスで実行されるサービスの調査、およびサブシステム・サービスの VOPC (参加クラスのビュー) が含まれます。

RUP SE 手法では、それぞれのハードウェア構成に、各ノードでホスティングされるすべてのサブシステム・サービスに参加するクラスからコンポーネントを作成する必要があります。そういったサービスを複数のプロセスで実行しなければならない場合、それぞれのプロセスで実行されるサブシステム・サービスの参加クラスを割り当てることによって、コンポーネントをさらに分割します。サブシステム・サービスによっては複数のプロセスで実行されるものがあるため、そのクラスが複数のコンポーネントに含まれることがあることに注意してください。メモリーの制約事項 (.exe および .dll のトレードオフなど)、出荷メディアの制限などに対処するようにコンポーネントをさらに分割することで、プロセスが完成します。

これらのアクティビティーにより、システムを構成するハードウェアおよびソフトウェア・コンポーネント・セットが特定されます。

システム開発

RUP SE プロジェクトは、RUP プロジェクトとほぼ同じように管理されます。しかし、ほとんどのシステム・エンジニアリング作業の規模およびそこで必要な追加アクティビティーの数によって、このセクションで簡単に説明するようないくつかの相違点があります。

プロジェクト編成

従来の直列プロセス (「ウォーターフォール型」プロセス) から反復型プロセスに移行することは、プロジェクトの編成方法に対して深い意味を持ちます。直列プロセスでは、多くの場合、スタッフ・メンバーは成果物が完成するまでプロジェクトに割り当てられます。例えば、技術スタッフは、仕様を完成させて開発スタッフに渡し、次のプロジェクトに進みます。RUP ベースのプロジェクトでは、そういった引き渡しは行われません。むしろ、成果物は、開発プロセスの間中、反復して進化していますこのためには、要求データベースおよび UML アーキテクチャーなどのプロジェクト成果物を担当するスタッフ・メンバーを、そのライフ・サイクル中は開発プロジェクトに割り当てたままにする必要があります。

図 5 に、代表的な RUP SE プロジェクトの編成を示します。プロジェクトは、それぞれにプロジェクト管理者および技術リーダーが配置された複数の開発チームの集合体で構成されます。システム全体のアーキテクチャーおよびプロジェクト管理を扱うチームもあります。

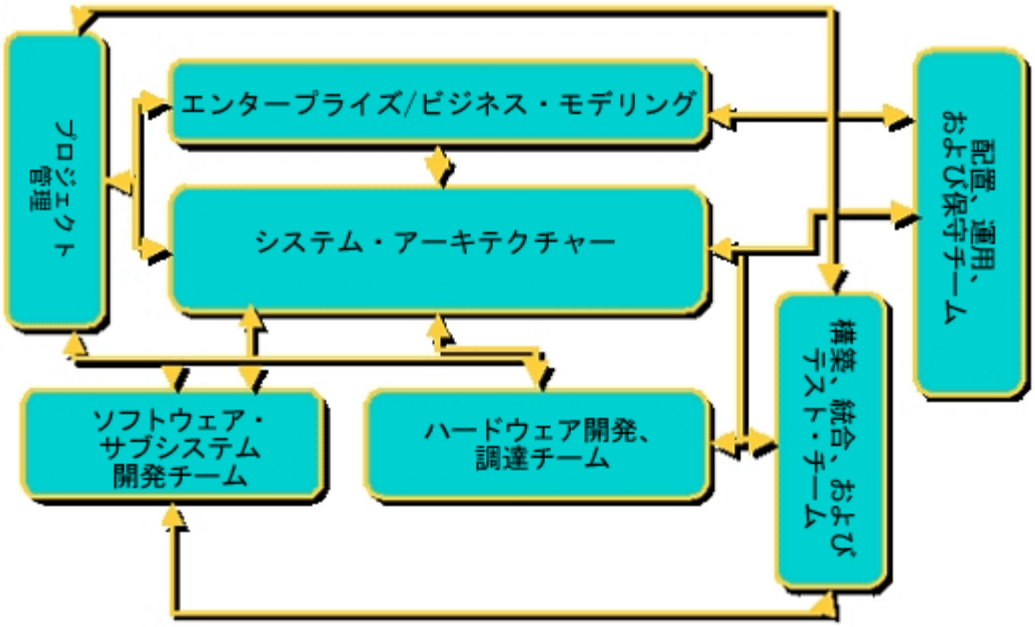

図 5: RUP SE **組織図**

この図のチームには、以下の機能があります。

- **エンタープライズ・モデリング・チーム**は、ビジネス・ニーズを分析し、ビジネス・モデルおよび/または操作の概念に関する文書などの関連する成果物を生成します。

- **システム・アーキテクチャー・チーム**は、エンタープライズ・モデリング・チームと協力して、システム・コンテキストを作成し、システム要求を派生させます。このチームは、サブシステムおよび局所ビューを作成し、さらに要求を派生させます。開発プロセス全体をとおして、システム・アーキテクチャー・チームは技術的な発展の基点としての機能を果たし、アーキテクチャーおよびエンジニアリングの問題を解決します。このチームは、開発チームと協力して、ソフトウェア・コンポーネント・アーキテクチャーを特定します。チーム・メンバーの中には、開発チームの技術リーダーが入っています。

- **プロジェクト管理チーム**は、プロジェクトのレビュー、リソース計画、予算の追跡、出来高と差額、および調整された反復計画などの標準的なプロジェクトの問題を管理します。

- それぞれの反復ごとに、**統合およびテスト・チーム**は、開発チームからコードおよびハードウェア・コンポーネントを受け取り、ソフトウェア・コンポーネントを構築し、ハードウェアおよびソフトウェアのコンポーネントを実験用の設定でインストールします。また、このチームは、反復ごとにシステム・テストを計画して実行し、レポートを作成します。

- **サブシステム開発チーム**は、1 つ以上のサブシステムのソフトウェアを実現するための設計および実装を担当します。このチームは、下位への流れのアクティビティーの中で得られた派生ユースケースに基づいて作業します。システムのサイズおよび複雑さに応じて、サブシステムのユースケースがクラス設計および関連するコード・モジュールとし

て実現されるか、サブシステムがさらにサブシステムに分解される可能性があります。

後のケースの場合、1 つのサブシステム・チームがさらに複数のサブシステム・チームに分割され、サブシステム・アーキテクチャー・チームが 1 つ作成される場合があります。このプロセスにより、RUP SE アプローチのスケーラビリティーが得られます。

- **ハードウェア開発および調達チーム**は、ハードウェア・コンポーネントの設計、仕様、および配布を担当します。
- **配置運用および保守チーム**は、運用上の問題に対処し、ユーザーとの窓口としての機能を果たします。このチームは、現場でシステムのインストールおよび保守を行う場合があります。別のケースでは、このチームは、ユーザーの障害レポートを処理し、現場にパッチを提供することもあります。

並行設計および実装

RUP SE 編成アプローチの魅力的な特徴は、非常に大規模なプログラムに拡張できることです。派生された要求を使用してシステムをサブシステムおよび局所に分解すると、それぞれの分析モデル要素は並行設計および開発に適したものになります。これまでに述べたように、UML サブシステムを別々の開発チームに割り当て、局所をハードウェア開発または調達チームに割り当てることができます。各チームは、ホスティングされるサービスまたは割り当てられたインターフェースから派生した調査から作業を進めて、設計モデルおよび実装モデルの一部を作成します。これは、設計要素の設計および実装を並行して進められることを意味します。

非常に大規模なシステムでは、複数システムからなるシステム・アプローチを採用できます。その場合、各 UML サブシステムには独自の局所モードが用意されるため、対応すべきことは論理的な検討事項のみになります。これにより、図 5 に示す組織構造をサブシステム・レベルで適用でき、さらにスケーラビリティーが増すことになります。

反復型開発、統合、およびテスト

RUP アプローチの主要点の 1 つは、一連の反復の中でシステムを開発し、それぞれで新機能を徐々に増やしながら作業プロトタイプを作成することです。システムは各反復で統合およびテストされます。反復テストはシステム・テストのサブセットです。その結果、最後の反復によって、運用時の設定に移行できる全テスト済みシステムの準備が整います。

反復のタイミングおよび内容は、プロジェクトの早い段階で反復計画に取り込まれます。しかし、RUP 成果物と同様に、反復計画は、システムがまとまってきたときの新たな認識を反映するために継続的に更新されます。

システム反復計画 に取り込まれる反復の内容は、反復で作成されるコンポーネントによって実現されるユースケースおよび補足要求によって指定されます。各反復は、適切なシステム・テスト・ケースのサブセットによってテストされます。

サブシステムでは、派生されたサービスがシステム・サービスから追跡されることを思い出してください。この追跡により、サブシステムおよび局所に対して、派生した反復計画の基礎が提供されます。つまり、反復をサポートするためにサブシステムおよび局所が提供しなければならない機能への各システム反復の内容を追跡することができます。実際には、開発チームは開発の実用性を反映させるように反復内容を協議します。例えば、初期のシステム反復では、サブシステムの全機能は必要ありません。妥協が必要になります。

優れたシステム反復計画では、ウォーターフォール・ベースの統合およびテスト・フェーズの後期で起こる典型的なパニック状態とは対照的に、システムの技術的リスクを早い段階で識別して解決することができます。技術的なリスクは、機能要求と機能外要求の両方に影響を及ぼす可能性があります。例えば、初期の統合では、設計およびインターフェース仕様を詳細にするだけでは完全に把握できないシステムの立ち上げおよびフェイルオーバーの問題を振り落とすことができます。実際には、初期の反復は、そういった機能外要求にアーキテクチャーが十分に対応しているかどうかを確認する必要があります。

反復システム開発は、テスト回数が増え、初期の反復をサポートするためにさらにハードウェア環境を整えたり、シミュレーションする必要があるため、費用が高くなると思われるかもしれません。開発チーム間で各反復の内容を調整することによって、さらにプロジェクト管理作業も必要になります。しかし、これらの目に見えるコストは、システム・アーキテクチャーに関連するリスクを早期に識別して軽減することで実現されるコスト削減によって相殺されます。アーキテクチャーの問題を初期の段階で取り除くよりも、プロジェクトの後期で取り除く方がかなりコストが高くつくことは、標準的な工学原則です。経費が増えるだけでなく、プロセスの後期に問題を取り除くことによる不確実性と、プロジェクト後期のスケジュールおよび予算のリスクも増します。

反復プロジェクト内のテスト組織の役割は、直列のウォーターフォール・ベース・プロジェクト内のテスト役割とは異なります。システム全体の開発後の統合に非常に多くの時間を割り振る代わりに、反復ベースのテスト組織では、プロジェクトのライフ・サイクル全体をとおして統合、テスト、および問題のレポート作成に時間を費やします。

まとめ

RUP SE は Rational Unified Process® (RUP) Plug-In として提供される RUP フレームワークのアプリケーションで、ソフトウェア、ハードウェア、作業者、および情報コンポーネントで構成される大規模システムの開発をサポートします。RUP SE には、さまざまな開発利害関係者の検討事項に対処するソリューションを提供するためのアーキテクチャー・モデル・フレームワークが組み込まれ、それぞれの形式の視点 (論理的、物理的、情報など) を検討できるようになっています。
RUP SE の際立った特徴は、システム全体の要求をより明確に特定することで、システム・コンポーネントの要求が同時に得られるという点です。

RUP SE は、概念上では以下のようなプロジェクトに適しています。

- 並行開発を行うために複数のチームを必要とする大規模なもの。
- ハードウェアとソフトウェアを並行開発するもの。
- 構造的に重大な配置の問題があるもの。
- ビジネス・プロセスの進化に対応するために基盤となる IT

インフラストラクチャーの再設計を必要とするもの。

RUP SE では、システム開発チームにシステム全体の問題に対処するフレームワークを提供しながら、RUP のベスト・プラクティスを活用できます。RUP SE の利点としては、以下のようなものがあります。

- **システム・チーム・サポート** -- ビジネス分析者、アーキテクト、システム・エンジニア、ソフトウェア開発者、ハードウェア開発者、およびテスター間の継続的なコラボレーションを可能にします。
- **システム品質** – チームがアーキテクチャー主導プロセスのシステム品質の問題に対処するためのビューを提供します。
- **システム・ビジュアル・モデリング** -- システム・アーキテクチャーで UML をサポートします。
- **スケーラビリティー** -- 非常に大規模なシステムに拡張できます。
- **コンポーネント開発** -- ハードウェアおよびソフトウェア・コンポーネントを決定するワークフローを提供します。
- **システムの反復型設計および開発** -- 並行設計、ハードウェアおよびソフトウェア・コンポーネントの反復型開発をサポートします。

注

1 Maria Ericsson 著、IBM Rational ホワイト・ペーパー「Developing Large Scale Systems Using the Rational Unified Process」(http://www.rational.com/products/whitepapers/sis.jsp) を参照してください。

2 Sanford Friedenthal などの「Adapting UML for an Object-Oriented Systems Engineering Method」。2000 年 INCOSE シンポジウム会報。

3 Murray Cantor 著「Thoughts on Functional Decomposition」The Rational Edge (2003 年 4 月)。

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。