Rational Unified Process for Systems Engineering

第 1 部:RUP SE バージョン 2.0 の紹介

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

編集者注記:
IBM Rational Unified Process® (RUP®) は、開発ライフ・サイクルに関連するあらゆるアクティビティーを合理化するフレームワークをソフトウェア開発組織に提供します。RUP は、1996 年に正式に発表されてから、「システム・エンジニアリング」(SE) を含むさまざまな開発要求に対応するように進化してきました。2001 年に、Rational Software の戦略サービス組織によって、システム・エンジニアリングをサポートする RUP Plug-In が初めて提案されました。RUP SE v1 Plug-In は、2002 年に一般に利用可能になり、新たに設置された IBM Rational 製品サービス・チームによってサポートが続けられています。

RUP SE の次世代構想の概要を説明するために、Murray Cantor の記事を今月から 3 回にわたって掲載します。このシリーズは、The Rational Edge の 9 月号と 10 月号に続きます。**これらの記事は現行の RUP SE Plug-In に対応していますが、プロセス・フレームワークへの拡張機能もいくつか紹介しています。現在利用可能な RUP SE Plug-In は RUP SE v1 であることに注意してください。**Rational Developer Network からダウンロード可能です (権限が必要です)。

概要

この記事では、最新の Rational Unified Process for Systems Engineering® (RUP SE®) の進化の概要について説明していますが、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。

RUP SE は、ソフトウェア開発プロセス・フレームワークである Rational Unified Process (RUP) のアプリケーションです。RUP SE は、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から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問題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、それぞれの視点論理的、物理的、情報などをまとめて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、こういったさまざまな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、以下のようなプロジェクトに対応します。

- 並行開発の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。
- アーキテクチャー上に大きな導入の問題があるもの。あるいは
- ビジネス・プロセスの進化に対応するために基盤となる 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 は RUP Plug-In として提供されます。この記事には、執筆時点ではまだ Plug-In に組み込まれていない概念をいくつか記載しています。しかし、システム開発のニーズを満たす方法に関する最新かつ最善の知識を得てもらうために、これらの概念を紹介しています。

システムおよび現在のシステム開発者が直面している課題から話を始めます。この話は RUP SE の設計ポイントの話につながっていきます。つまり、システム開発の課題に対処する方法です。次の 2 つのセクションでは、RUP SE の UML ベース・モデリングおよび要求指定手法、さらに統一モデリング言語 (UML) セマンティクスについて紹介します。1 番目のセクション『**システム仕様**』では、システムのブラック・ボックスについて詳しく説明します。来月掲載する第 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の内部について説明します。10 月の第 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。

システム開発の用語および概念

システム といった場合、エンタープライズがビジネス目的[1] または任務を達成するために利用するサービスを提供するリソース・セットを意味します。一般的にシステム・コンポーネントは、ハードウェア、ソフトウェア、データ、および作業者で構成されます。システムは、提供するサービスと、システムの振る舞い以外の信頼性や所有コストなどの要求によって定義されます。システムの設計は、コンポーネント、属性、および関係の指定で構成されます。

*システム*は、関連語も含めるとさまざまな意味[2] を持つ言葉の 1 つで、技術開発の説明に使用する際に混乱を招くことがあります。一般的には、システムは、まとまって振る舞いを示す要素のセットまたは集合です。システム・エンジニアリングに関する文献にある定義はすべて、この考えに基づいています。前述のシステム定義に加えて、*システム* および*システム・エンジニアリング* の定義をさらにいくつか示します。

INCOSE (International Council on Systems Engineering)[3] から引用します。

> システム・エンジニアリングとは、学際的な方法によって成功裏にシステムを実現することです。以下のようなあらゆる問題を考慮しながら、お客様のニーズおよび必要な機能を開発サイクルの早い段階で定義し、要求を文書化してから、デザイン・シンセシスおよびシステム検証を進めることに重点的に取り組むものです。
>
> - 操作
> - パフォーマンス
> - テスト
> - 製造
> - コストとスケジュール
> - トレーニングとサポート
> - 処分

Mil-STD-499A から引用します。

> エンジニアリング・マネージメント -- 軍の要求事項を運用システムに変換するために必要なエンジニアリングおよび技術活動 (technical effort) のマネージメント。これは次の 4 つを含む。(1) システム性能パラメーター、および要求を満足するに望ましい (preferred) システム形態を決定するために必要なシステム工学。(2) 技術プログラム業務の計画と管理。(3) エンジニアリング・スペシャルティーの総合化。(4) コスト、技術業績 (technical performance)、スケジュールの目標に合わせるための設計工学、スペシャルティー工学、試験工学、後方支援工学 (logistics eng.)、および生産工学の全体にわたる総合活動のマネージメント。

「*The Art of Systems Architecting*」(Maier および Rechtin 著)[4] から引用します。

> ...systems are collections of different things which together produce results unachievable by the elements alone. (～システムとは、複数の要素が全体として機能を発揮し、各要素単独では達成できない結果を生み出す集合体である。)

システムの例として、以下のようなものがあります。

- **IT** システム。主にソフトウェア、コンピューターのハードウェアと周辺機器、および作業者で構成されます。
- **製品**。航空機、衛星、自動車、および電話交換機などがあり、ハードウェアおよびソフトウェアで構成されます。
- **エンタープライズ**。サービスを提供したり、任務を遂行したりします。こういったエンタープライズは、ハードウェアおよびソフトウェアを使用できる作業者によって構成できます。ほとんどの場合、このようなエンタープライズは組織境界を越えることができます。

こういった定義はすべて、システムを次の 2 つの視点から捉えられることを示しています。

- **ブラック・ボックス**の視点: システム全体で提供するサービスおよび満たす要求。
- **ホワイト・ボックス**の視点: システムを構成する要素または部分。

システムの考察、理解、および使用時にこれら 2 つの視点のどちらを使用するかは、関係するニーズに応じて決定されます。システム・エンジニアリングの問題は、以下のようなシステム利害関係者すべてのニーズを満たすシステムを設計して実装することです。

- **ユーザー**。機能およびパフォーマンスに関心があります。
- **所有者**。配置および所有のコストに関心があります。
- **投資家**。市場または業務空間での競争上の優位性に関心があります。

RUP SE は、システム・エンジニアの複数のチームが、すべての利害関係者のニーズを満たすようにシステムのブラック・ボックス視点を決定し、最適なホワイト・ボックス・システム設計を指定する際のメカニズムおよび UML ベース・モデルのフレームワークを提供します。

現在のシステムの課題

システム・エンジニアリング[5] は、20 世紀の中ごろ[6] に 1 つの作業分野として認められました。初期の頃、システム・エンジニアはかつてない機能を提供する複雑で独立したエンティティーを設計して指定することに挑戦していました。システム・アプローチの初期の成功例として、Athena Rocket、NORAD、および Apollo プログラムがあります。現在、システムは、適切な機能セットを提供するだけではなく、新しい課題に対処することも期待されています。

もちろん、初期のシステム・エンジニアリング・プロジェクトにも、多くの課題がありました。例えば、最初の画像衛星の打ち上げでは、大きな技術的成果を上げました。まず、このプロジェクトの開発チームは、ユーザーのニーズを満たすためにこの前例のないシステムに必要な機能は何かを分析する必要がありました。次に、既存の技術を使用して、地上の建造物の画像を描き、その画像を地上局でダウンロードできる十分な長さで保管してから、分析者のワークステーションからアクセスできるようにする方法を考え出なければなりませんでした。当時は、ソリューションに専用リソースが必要であるということはほとんど取り上げられませんでした。機能自体を提供することで利害関係者のニーズを十分に満たしていました。

現在、こういった画像衛星システムの設計では、基本機能の提供への関心は薄れ、より幅広い利害関係者のニーズを満たすためのシステムの最適化への関心が高まっています。例えば、システムによるデータの情報をできるだけ早くしたいだけではなく、分析者はその情報を他のソースからのデータと統合する能力を要求します。システム所有者は、専用ハードウェア (およびソフトウェア) の使用を最小限に抑えたいと考えています。なぜなら、機能ごとに 1 つのシステムを維持することはできず、既存のリソースの再利用を増やしたいためです。また、進行中の業務および保守コストを削減することに非常に大きな関心を寄せています。

さらに、任務およびシステムを実現する技術は、プログラムの存続期間中に確実に何度も変化していきます。また、システムに投資した人は、最低限のコストおよび最短の中断でそういった変化に対応して進化させたいと考えています。さらに、投資が再利用可能な知的財産となることも期待しています。

さまざまな要求タイプ

利害関係者のニーズを満たすために、システム・エンジニアは広範囲にわたる一連の要求を考慮する必要があります。次に、考慮事項の一部のリストを示します。

- **機能:** ビジネス・ニーズを満たすためにユーザーおよび 他のシステムに提供される能力。機能要求には、機能を提供するシステムの振る舞いを示す必要があります。
- **使用可能度:** システム機能へのアクセスのしやすさ。
- **保守容易性:** 障害の発見、切り分け、および除去のしやすさ。
- **拡張性:** 機能の追加のしやすさ。
- **スケーラビリティー:** 時間が経過して要求が増加したときに、ユーザー、データ項目などの数の増加に対応する能力。
- **信頼性:** 安全上の注意などの正しいシステム応答の確率。
- **パフォーマンス:** キャパシティー・ロード中のユースケース内のステップに対して期待されるシステムの応答時間。
- **キャパシティー:** 期待されるユーザーおよびデータ項目の数。
- **サポート可能性:** 許容範囲内のダウン時間などの、現場での保守のしやすさ。
- **製造**

- **配置コスト**
- **運用コスト**

環境に応じて、そのほかにロジスティクス・サポート、セキュリティー、およびリモート・トレーニングのニーズなどのシステム要求がある場合があります。

それらの要求の中には、ソフトウェア開発チームになじみのあるものもあります。中には、ハードウェア、ソフトウェア、および作業者を考慮せずには対応できないものもあり、これら 3 つすべてをシステム設計の作業分野の中に同時に指定する必要があります。

システム開発のもう一つの特徴として、多くのシステム構成を維持しなければならない場合があります。例えば、共通アーキテクチャーを有するが、さまざまなコスト/パフォーマンスの目的を達成するためにさまざまなハードウェアおよびまたはソフトウェアを導入する製品のシステム仕様を維持したい場合などがあります。

急激な変化

最初のシステム・エンジニアリング方法論の大部分が技術革命 (~1955-1980 年) の初期に作成されたときには、今日の技術変化の速度および影響は予想されていませんでした。最初のシステム・エンジニアリング手法は、要求を入念に指定してから、可能な範囲でそれを満たすことに重点を置いていました。

現在、私たちはシステムを 30 年にわたって使用し続ける可能性があることを理解しています。そのため、この 5 年の間にシステムが実行しなければならないことを正確に把握しているエンジニアはおらず、ましてや 30 年になればなおさらで、システムに対してはニーズの進化に適合できることがより求められています。今日のシステム設計者は、システムの運用が進化していく環境またはコンテキストも考慮する必要があります。現在展開されているシステムはいずれも、その存続期間中に予想しなかったシステムとの相互作用の可能性が高くなります。例えば、商用システムの統合がさらに進むとレガシー・システムに新規要求が設定され、新規防御システムがオンラインになると既存のシステムに新規要求が設定されます。

私たちは、システムの実現技術が変化することも分かっています。時間が経過すると、既存のシステムは、競争に勝てなくなるか、維持の費用対効果が低下します。現代のシステム開発アーキテクチャー・フレームワークは、再ホスティングまたは再配置の理由から、技術的に独立した手法を提供する必要があります。

ソリューション空間の拡大

今日の技術によって、設計者はより多くの方法でシステム設計に取り組むことができるため、複雑な課題に対応することができます。例えば、最初の画像衛星システムの設計者の選択肢は限られていて、技術的な制約を賢く乗り越えなければなりませんでしたが、現在のシステム設計者は実際に非常に多くの処理能力があり、システムを最適化するためにサブシステムの責任を振り分け直すことができます。そのため、難しい選択をしなければならない場合があります。この現象は、通信、航空電子工学、IT などの多くの分野で起きています。

論理および物理考慮事項のバランス

システムの概念を説明するには、多くの方法があります。その中には、以下の 2 つのものがあります。

- **物理エンティティーとして**。機械、電気、土木工学などの物理法則および従来の作業分野によって規定されます。
- **大規模な状態マシンとして**。コンピューター・サイエンスおよびソフトウェア・エンジニアリングのインサイトによって規定されます。

ロケット・エンジンのようにシステムによっては、物理エンティティーとして考えることが最も適しているものがあります。ほとんどの IT システムのように、大規模な状態マシンのシステムもあります。実際には、すべてのシステムのいずれかの側面に両方の概念が当てはまります。ロケット・エンジンには、ソフトウェア主導型のコントローラーが内蔵されています。情報システムは、パフォーマンス、待ち時間、および信頼性を制限する物理法則によって規制されます。

これまでプロジェクト・リーダーは一般的に、システムの種類に応じてこういった観点の 1 つを取り入れて、それに応じて管理していました。彼らが主にハードウェアの視点からシステムを物理的なものとして考えたときには、ソフトウェアをハードウェアがジョブを実行できるようにするための「必要悪」と見ていました。システムをソフトウェアの視点から考え、ハードウェアをサービス提供機構と見なした場合、通常はハードウェアを補足部分として扱っていました。

ソフトウェアのサイズおよび複雑さの増加

オブジェクト・テクノロジー、コンポーネント・フレームワーク、およびソフトウェア開発の自動化の出現により、ソフトウェア開発の生産性は 1970 年[7] に比べて倍に向上し、ソフトウェア・アプリケーションのサイズおよび複雑さは増大し続けています。同時に、処理能力、コンピューター・メモリー、使用可能なデータ・ストレージ、およびネットワーク帯域幅も大きく進歩しています。こういった変化は、次々とこれまで以上に高度なオペレーティング環境をもたらしました。ソフトウェアおよびシステム業界は、改良されたあらゆる機能を使用して、ますます大規模で複雑なプログラムを開発しています。実際に、平均サイズのソフトウェア・アプリケーションにおける競争圧力は、ファンクション・ポイント[8] で測定したところ倍にまで増大しています。そして、技術の向上に伴って、この傾向は続いていくと推定されます。

多くの作業分野

現代のシステム開発における一般的な開発チームは、アーキテクト、開発者、設計者、テスターなどの役割を持つ作業者で構成されます。RUP と RUP SE ではいずれも、こういった作業者全員が同時に作業を進め、ライフ・サイクル全体を通じて特定の成果物を進化させていきます。彼ら作業者は、仕事を順番に渡していくことはしません。代わりに、一緒に作業を進めて、詳細さのレベルを発展させ、複数の分野の検討事項に対応していきます。エンジニアリング・プロセスおよびアーキテクチャー・フレームワークの主な目標および課題の 1 つは、さまざまな開発利害関係者がコミュニケーションを取って設計の決定事項を調整する手段を提供することです。現代のシステム設計の複雑さを考えた場合、システム開発チームは、以下のような幅広い検討事項に対処しなければなりません。

- 構築および統合

- ビジネス・モデルの構築
- データのモデル化
- ドメインの課題
- ハードウェア開発
- ヒューマン・ファクター
- IT
- ロジスティクスおよびフィールド・サポート
- ソフトウェア開発
- システム全体の仕様および設計

さまざまな開発チーム・メンバーが、異なる検討事項のセットを重複して担当します。例えば、ソフトウェア・アーキテクチャーが適切に機能要求を満たしていることを確認するだけではなく、*ソフトウェア・アーキテクト* は通常、以下の事項も検討します。

- **使用可能度:** システム機能へのアクセスのしやすさ。
- **保守容易性:** ほかのものを導入せずに障害を切り分けて除去する作業のしやすさ。
- **拡張性:** 既存のソフトウェア製品への新機能の追加のしやすさ。

それに対して、*システム・エンジニア* は通常、以下の課題に対処します。

- **可用性/信頼性:** システムが使用でき、入力に正しく応答できる可能性。
- **パフォーマンス:** ある入力に対するシステムの反応性。
- **キャパシティー:** システムが処理できるユーザーまたはデータ・レコードなどの項目数。
- **スケーラビリティー:** キャパシティーの拡張のしやすさ。
- **サポート可能性:** 現場でのサポートのしやすさ。サポート可能性には、システムのインストールおよびパッチの適用などがあります。

その他の分野固有のシステム・エンジニアリングの検討事項には、セキュリティー、トレーニングのしやすさ、およびロジスティクス・サポートなどがあります。

複雑さの課題の克服

RUP SE は、これまで説明してきた検討事項に対応する成果物と、詳細仕様を進化させるワークフローを提供します。

このシリーズ記事はすべての利害関係者の検討事項すべてを扱っているわけではありま

せんが、モデルのビューポイントを提供して、あらゆるレベルのシステム仕様での検討事項を切り分けることができるアーキテクチャー・フレームワークについては説明しています。これから説明する下位への流れのメカニズムでは、ビューポイント間でモデルの整合性を維持する方法について解説します。このようなメカニズムを使用することで、チームは RUP の事例に従って、開発ライフ・サイクル全体を通じて成果物を継続的に進化させることができます。

RUP SE 設計ポイント

システム開発の問題はソフトウェアのみの開発の問題とは異なり、システム開発では、通常のソフトウェア作業で扱うよりも幅広い要求セットを取り扱います。それでも、ソフトウェア開発作業のほぼすべてにシステム問題の要素の一部が含まれることに注目することが重要です。システムの検討事項を考慮するソフトウェア開発作業の例としては、Web ベース・アプリケーション、ビジネス・アプリケーション、IT 統合、組み込みソフトウェア、また防衛および情報システムなどがあります。

ここで重要なポイントは、前のセクションで取り上げたような今日のシステム・エンジニアが直面する多くの課題をシステム開発プロジェクトで対応しなければならない点です。これらの問題に対処するために、RUP SE は以下の設計ポイントを取り入れています。

- 業界 (事実上の) 標準のシステム定義に従う。
- RUP フレームワークをシステム開発に適用する。
- RUP 4+1 アーキテクチャー・モデルを RUP SE モデル・フレームワークに拡張し、新規ビューを説明する RUP のロール、アクティビティー、成果物、および作業分野を拡張または変更する。
- UML をモデリング言語として使用する。
- ツール資産を提供する。
- すべてのモデル・レベルをプログラム資産として維持する。

以上の設計ポイントのそれぞれについて詳しく見ていきましょう。

業界 (事実上の) 標準のシステム定義に従う

前述の『**用語および概念**』セクションで示したように、システムはブラック・ボックスとホワイト・ボックスの両方の視点から見ることができます。RUP SE はこの原則に従います。ブラック・ボックスの視点については、『**システム仕様**』セクションで解説します (以下を参照してください)。ホワイト・ボックスの視点については、来月号の『**システム・アーキテクチャー**』セクションで解説します。RUP SE で記述される要素には、ハードウェア、ソフトウェア、作業者、およびデータが含まれることに気を付けてください。

RUP フレームワークをシステム開発に適用する

RUP のライフ・サイクルおよび作業分野を図 1 に示します。RUP SE は、次の 3 つの点で RUP に従っています。

- **ライフ・サイクル:** リスクの排除を重視するために、RUP SE は、プロジェクト詳細に対するチームの理解を発展させることで、RUP の 4 つのフェーズに従います。
- **反復:** RUP SE は、リスク識別および軽減に基づいた一連のシステム構築を支援し、通常、反復には少なくとも 1 つのシステム構築が組み込まれます。特に、すべての成果物 (詳細なプロジェクト計画を含む) は反復の間中、発展していきます。RUP SE が RUP から継承する主な特徴は、ウォーターフォール型開発を止めて、反復型開発を使用することです。
- **作業分野:** RUP SE は、フォーカス・エリア、すなわち図 1 に示す「作業分野」に沿って進みます。ここでは、基礎プロセス定義およびチームがシステムを開発する際に実施する作業に多くのビューが提供されます。RUP プロジェクト・チームにはシステム・エンジニアがいますが、システム・エンジニアリング作業分野は分けられていません。正確には、システム・エンジニアは 1 つ以上の RUP ロールを果たし、1 つ以上の RUP 作業分野に参加します。作業分野のワークフローおよびアクティビティーは、より広範囲のシステムの問題に対処するために変更されることに注意してください。こういった変更については、以下のセクションで説明します。

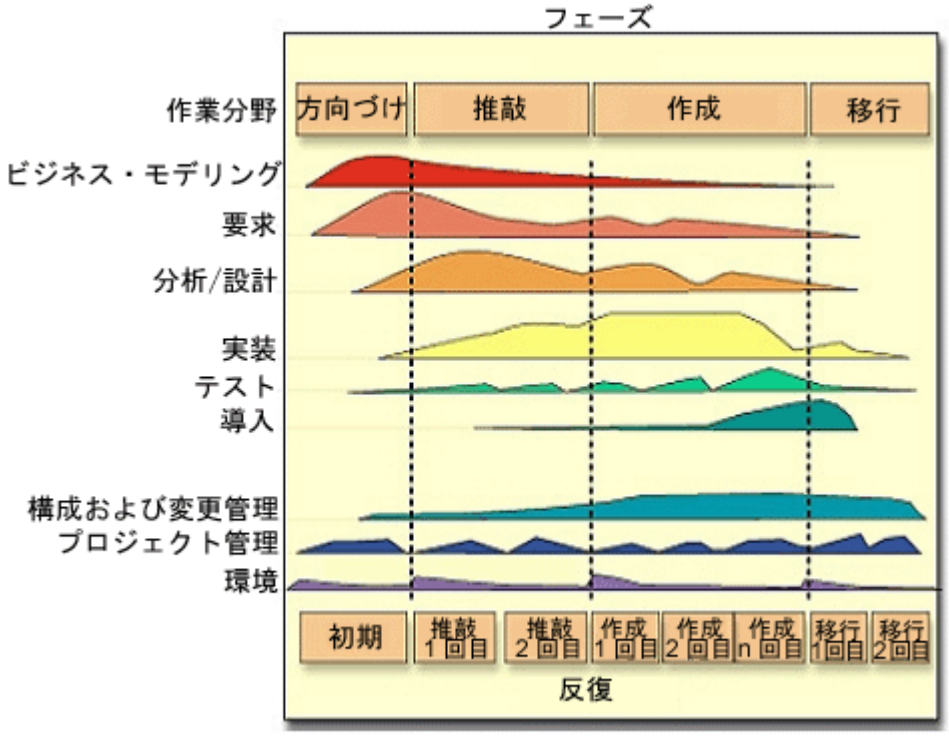

図 1: RUP プロセス・フレームワーク (RUP SE で採用)

以下で説明するように、RUP SE は、追加の成果物と、アクティビティーおよびロールによって RUP を補足し、それらの成果物の作成をサポートします。これらについては、RUP

SE Plug-In で詳しく記述されています。

また、RUP フレームワーク・アプリケーション・プラグインとして、RUP SE では、以下の RUP の基本管理原則をシステム開発に採用することができます。

- 結果ベースの管理
- アーキテクチャー中心の開発

RUP 4+1 アーキテクチャー・モデルを RUP SE モデル・フレームワークに拡張する

開発者は、アーキテクチャー・フレームワークによって、別の仕様および設計の検討事項について推論し、その推論から得た結果を標準的な整合性のある方法で文書化することができます。オリジナルの RUP はソフトウェア開発プロセスであり、その 4 + 1 アーキテクチャー・フレームワークではシステム開発のすべての検討事項を扱うには不十分です。なぜなら、より多くの利害関係者がより多くのビューを必要としているためです。アーキテクチャーに関する新規フレームワークについては、このシリーズの第 2 部で解説します。

UML をモデリング言語して使用する

RUP SE モデル・フレームワークは、UML を使用して、アーキテクチャー・モデル・フレームワークのビューを構成するさまざまな図を表現します。現在のバージョンは、ステレオタイプを含む UML 1.4 セマンティクスを採用しています。次のバージョンでは、UML 2.0 セマンティクスに移行し、採用された場合には、処理中のシステム・エンジニアリング・プロファイルも一緒に移行します。

ツール資産を提供する

RUP SE をサポートするために、IBM Rational Software は RUP Plug-In を用意しています。これには、RUP の拡張が詳細に記述されていて、IBM Rational Rose® および IBM Rational RequisitePro® ツールも追加されています。現在利用可能な Plug-In は 2002 年にリリースされたものです。

すべてのモデル・レベルをプログラム資産として維持する

これまでに説明してきたように、システムの存続期間が初期の要求および実現技術よりも長くなることが多く、それにより、時間が経つにつれて、時代遅れになるか、そうでなければ機能が不十分になるか、所有コストが許容範囲を超えるかのいずれかになります。したがって、効果的なアーキテクチャー・フレームワークでは、指定レベルの上昇に合わせたモデル・ビューを維持する必要があります。トップレベルはコンテキストおよび仕様を設定し、それよりも下のレベルはコンポーネントおよび部品表を設定します。全体を通して追跡可能性を保つ必要があります。このようにレベルを保つことで、変更の影響について推論できる設定値が提供されます。任務の変更は通常、モデル内のトップレベルで変更され、下のレベルに流れていきます。技術の変更によって、別の設計トレードまたは現行設計の別の実現のいずれかが可能になります。RUP SE は、必要なモデル・レベルおよび追跡可能性を提供します。

システム仕様

システム仕様は、システムのブラック・ボックス機能を指定するプロセスです。ここでは、外部から見ることができる機能、システムが提供するサービス、および期待される効率度を指定します。RUP SE では、システム仕様は、コンテキスト中でシステムに期待される実行方法の詳細で構成されます。すなわち、システムを大規模なエンタープライズへの参加者と見なします。システム仕様は、ビジネス目的または任務を達成するためのエンタープライズの分析、および大規模なエンタープライズを実現する際にシステムが果たす役割から得られます。このプロセスについて、以下で説明します。

エンタープライズ

エンタープライズも、システムと同様に、リソースのセット (作業者、ハードウェア、ソフトウェア、およびデータ) を使用してビジネスの目的を達成したり、任務を遂行したりすることに注意してください。システムと同じく、エンタープライズには、エンタープライズで使用またはコラボレーションされるエンティティーである *アクター* があります。実際には、エンタープライズを複数システムからなるシステム[9] と考えることが多くの場面で非常に役に立ちます。

自動車または航空機のような製品も、大規模なエンタープライズの一部です。航空機は、パイロットおよびすべての航空管制システムと対話する必要があります。

コンテキスト分析

開発中のシステムは常に、より大規模なエンタープライズの一部になります。システムを指定するには、大規模なエンタープライズのアクティビティーを理解し、そのエンタープライズをシステムおよびその他のエンティティーに区分けし、エンタープライズにサービスを提供させるためのシステムの参加方法を分析してから、その分析の結果をシステム仕様として獲得する必要があります。このセクションでは、このワークフローについて詳しく見ていきます。

システムに物理と論理の両方のエンティティーの特性があることに注意してください。システムは、サービスを提供し、メッセージを受け渡し、他の論理エンティティーと対話する時には論理エンティティーであり、システムの能力に限界があり、その限界を考慮しなければならない時には物理エンティティーです。そのような限界の例としては、反応性およびキャパシティーがあります。現在のところ、UML にはエンティティーのようなセマンティクスはありません。以下で説明するように、RUP SE では、システムをステレオタイプ化した分類子としてモデリングしていますが、関連する物理的特性をクラス属性として追加しています。

システム仕様は、システムのブラック・ボックスの記述を取得します。ここでは、システムの範囲と境界、提供するサービス、その他の属性、および他のエンティティーとやりとりするものを設定します。この情報は、図 2 に示すシステム・コンテキスト図[10] に取り込まれます。

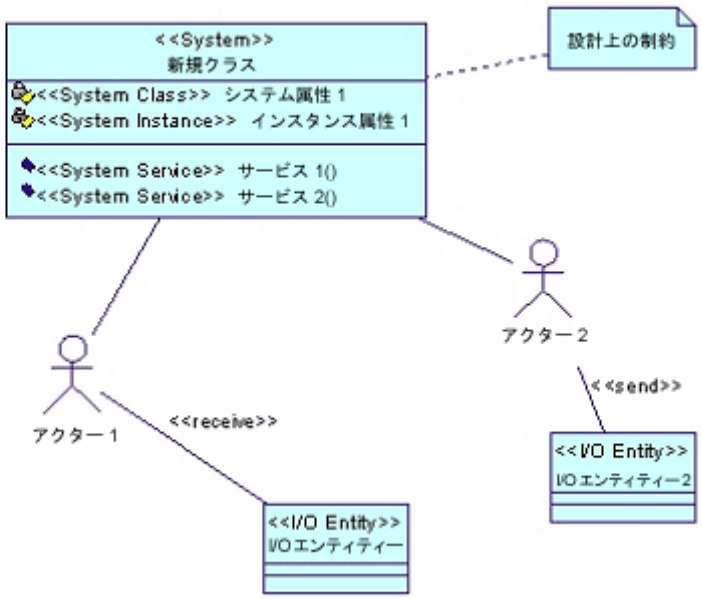

図 2: RUP SE システム・コンテキスト図

図 2 に示すように、関連図は以下のもので構成されています。

- システム分類子
- システム・アクター (外部システムおよびユーザー)
- システム・アクター関係
- 入出力エンティティー
- 入出力エンティティー関係
- 設計上の制約

これらの各要素について見ていきましょう。

システム分類子

システムはエンティティーです。それぞれが同じ仕様に対応する複数のシステムがまとまったものが、システム・クラスのインスタンスになります。例えば、Saab 93 自動車はクラスを形成し、VIN 12345678 の Saab 93 はそのクラス内のインスタンスまたはオブジェクトになります。そのため、システムは UML オブジェクトおよびクラスのセマンティクスで説明される場合があります。IT システムのようにシステムのインスタンスは 1 つのみであることが期待される場合でも、こういったセマンティクスは有効です。

UML セマンティクスに従うと、システム・オブジェクトは、以下の 3 タイプの属性を持つことができます。

- **クラス属性**。各システム・オブジェクトの値は同じです。例えば、Saab

'93 の燃料の総容量または IT システム内の同時ユーザーの総数などがあります。

- **インスタンス属性**。この値は、システム・オブジェクト間で異なる場合があります。例えば、VIN 123456789 の現在の燃料積載量または IT システム内の現在のユーザー数などがあります。
- **効果の基準**。故障までの平均時間または開発の技術的なリスクなどのより一般的な設計目標です。こういった基準は、タグ付き値を通じて獲得できます。

インスタンス属性は、テスト・ケース開発およびシミュレーションにパラメーターを提供することに注意してください。クラス属性はテスト・ケースおよびシミュレーションの範囲を提供しますが、効果の基準は、システム設計が適切かどうかを判断して設計トレードに決定基準を設定する際に使用されます。

システム・クラスの操作は、*システム・サービス* と呼ばれます。UML 用語では、これらは操作になります。ここで、UML では、操作はクラスであり、クラスのインスタンスは、ユースケースの実現の一部であるシーケンス図または相互作用図にあるメッセージであることを思い出してください。システム開発において、これらの操作は、システムがエンタープライズのニーズまたは任務を達成するためにアクターと対話する方法を調査することで見出されます。

さらに、通常の UML セマンティクスに従った場合、サービスを集約してインターフェースにできる場合があります。

システム・アクター

UML では、アクターは、システムと対話するエンティティーで、一般的にはユーザーまたは他のシステムであることを思い出してください。多くのシステム開発者は、アクターの一種として時間または大気条件などの環境要素を考慮することが役に立つことを理解しています。

通常、システム・アクターには、次の 2 種類があります。

- **エンタープライズ・アクター**。システムと直接対話するエンタープライズの外側にあります。
- **内部アクター**。エンタープライズの一部で、システムではありません。このアクターは通常、ビジネス・ワーカーまたはそのほかのエンタープライズ・システムになります。

そして、システム・アクターとシステム・ワーカーを混同しないでください。システムの一部であるワーカーはアクターではありません。以下で説明するように、彼らは一部のシステムのサブシステムへのアクターになることができます。

システム・アクター関係

アクターとシステムのコラボレーションは、UML1.x システムとサブシステムのコラボレーションに似ています。アクター・インスタンスは、システム・サービスを呼び出してコラボレーションでの役割を果たしたり、その逆に呼び出されたりします。したがって、アクターがシステ

ムを使用することを指定するセマンティクスは の依存関係です。依存関係の方向性は、システムがアクター・サービスを呼び出すか、その逆か、あるいは両方かを表します。

アクターとシステムがより密接に結び付いていることがあるため、一方の属性が他方の属性によって影響を受け、アクターとシステムの関係が関連に発展します。例えば、ドライバー (アクター) と自動車 (システム) を考えてみます。ドライバーが自動車を運転しているときには、おそらく駐車しているときよりも迅速に動作できる状態にあります。

システム・サービスを集約してインターフェースを形成できます。インターフェースを使用して、各アクターが使用する操作セットを指定するか、単にサービスのカテゴリーを作成することができます。所定のサービスを複数のインターフェースに組み込むことができることに注意してください。

入出力エンティティー

入出力 (I/O) エンティティーには、システムが生成および/または受け取るあらゆるもの、例えば情報または物理的な項目などが含まれます。小売システムでは、クレジット・カード情報を別のクレジット・カード情報システムに提供することが考えられます。空調システムでは、温風および冷風の空気の入れ替えが考えられます。代表的な入力エンティティーには、データベース照会、ファイル更新、センサー入力、および制御入力があります。代表的な出力エンティティーには、照会結果およびセンサー出力があります。

I/O エンティティーは、属性を持つが、操作がない UML クラスです。

入出力エンティティー関係

図 2 に示すように、I/O エンティティーには、特殊なステレオタイプ化されたアクター関係があることに注意してください。関連は、エンティティーがアクターによって送信されるか、受信されるかを取得します。セマンティクスは、モデル内のアクターとシステムの関係を維持し、コンテキスト図がデータ・フロー図でもあることは示唆しません。

設計上の制約

多くの場合、システム仕様は、厳密にはブラック・ボックスの考慮事項ではありません。仕様には、システム内の制約が含まれます。
そのような設計上の制約の例としては、使用しなければならないコンポーネントおよび準拠しなければならないアルゴリズムなどがあります。設計が進み、ホワイト・ボックス要素が指定されると、次のセクションで説明する RUP SE ビューのいずれかにこのような制約が自然に現れてきます。設計上の制約を表す固有の UML セマンティクスはありません。RUP SE では、次の 2 つのいずれかの方法で取得できます。コンテキスト図の注釈として、または可能であれば、システム・クラスに関連する補足要求文書として取得できます。

システム・ユースケースおよびサービス

RUP SE は、ユースケースとサービス[11] の両方を使用してシステムの振る舞いを取得します。いずれの場合も、以下のように標準セマンティクス[12] が使用されます。

- **ユースケース**は、アクターがシステムを使用する方法を記述します。言い換えれば、アクターとシステムの間で期待されるコラボレーション/シナリオのセットです。ユースケースは、システムを使用して大規模な

エンタープライズ・ビジネス目的または任務を達成する方法を伝達します。RUP SE でのユースケースの記述は、標準 RUP で使用されるものと同じです。特に、システムとそのアクターの間のブラック・ボックスの対話についてのみ記述します。

サービスは、システムがユースケース・シナリオ内で役割を達成できるように提供する (おそらく抽象的な)操作です。サービスのインスタンスは、ユースケース・シナリオを実現するシーケンス図内にあるメッセージです。一般に、ユースケースとサービスの間は nm マッピングされます。

図 3 は、小売システムの一部のコンテキスト図の例です。

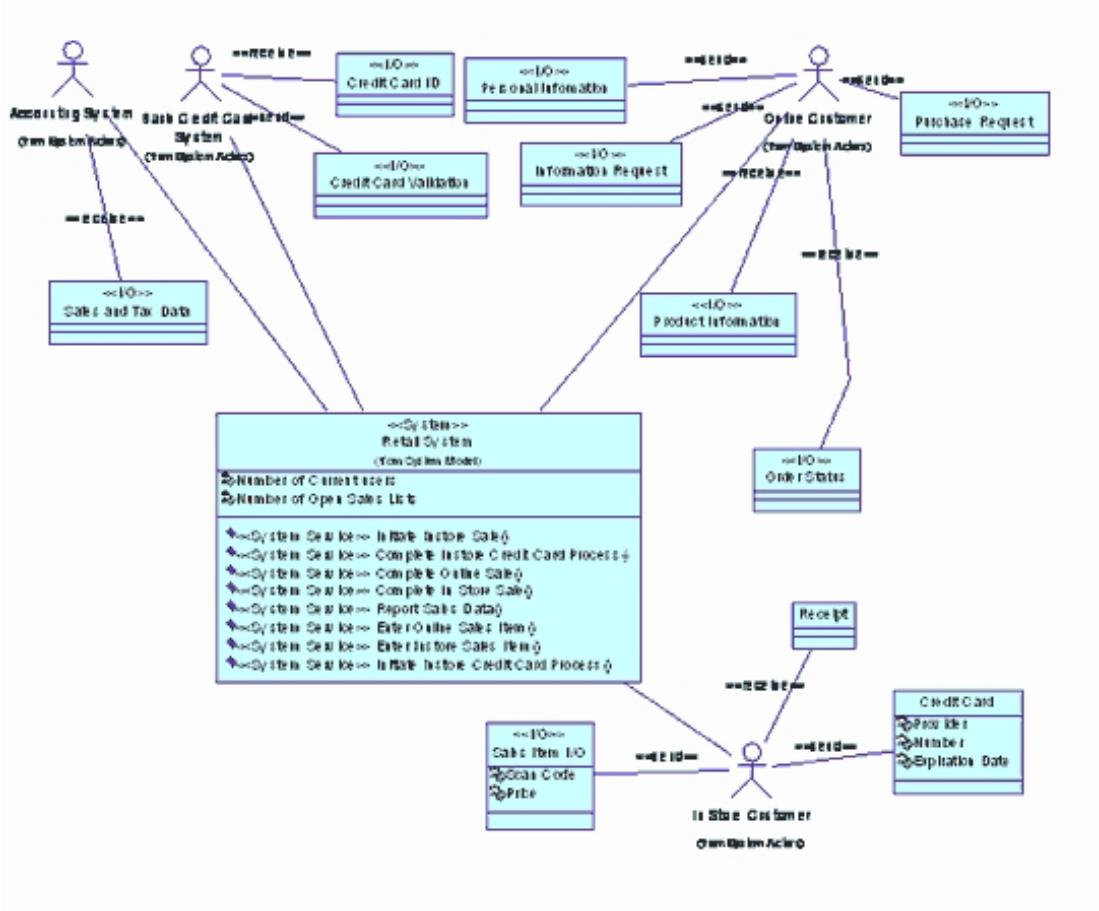

図 3: 小売システム・コンテキスト図
クリックして拡大

来月の予告:

この RUP SE に関するシリーズ記事は、The Rational Edge の 9 月号に続きます。来月は、システム開発に関連するシステム・アーキテクチャーについて解説します。3 回目および最後の回は 10 月版に掲載します。機能および補足要求からアーキテクチャー要素の下位への流れなどの要求分析、プロジェクト組織、反復型開発の問題、および統合について解説します。

注

1 Blanchard、Fabrycky 著「*Systems Engineering and Analysis*」(third edition) Prentice Hall 出版 (1998 年)

2 例えば、*Oxford English Dictionary* には、11 の定義がリストされています。

3 http://www.incose.org/whatis.html

4 Maier W. Maier、Eberhardt Rechtin 著「The Art of Systems Architecting」(second edition) CRC Press 出版 (2000 年) p 8

5 このセクションの大部分は、The Rational Edge に掲載された Murray Cantor 著「Leading Modern System Development」(2003 年 1 月) より引用しています。

http://www.therationaledge.com/content/jan_03/f_modernSystemsDevelopment_mc.jsp

6 Thomas P. Thomas P. Hughes 著「Rescuing Prometheus」Pantheon Books 出版 (1998 年)

7 David Longstreet 著「Software Productivity Since 1970.」(2002 年)
http://www.ifpug.com/Articles/history.htm

8 *Ibid*。

9 Paul Carlock、Robert Fenton 著「System of System (SoS) Enterprise Systems Engineering for Information-Intensive Organizations」Systems Engineering、Vol. 4、No. 4 (2001 年)

10 RUP SE のコンテキスト図は、意味的には、詳細なコンテキスト図 (Sanford Friedenthal Howard Lykins および Abe Meilich が最初に提唱) と同じです。次のアドレスを参照してください。
http://www.omg.org/cgi-bin/doc?syseng/2001-09-05

12 このユースケースとサービスの区別は OOSEM の特徴の1つである *loc cit* です。

13 **UML** におけるサブシステムの分類子セマンティクスの使用については、Fredrick Ferm 著「The Why, What, and How of a Subsystem」(The Rational Edge、2003 年 6 月) を参照してください。
http://www.therationaledge.com/content/jun_03/t_subsystem_ff.jsp

---

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。