ソフトウェア開発は、より抽象化とアーキテクチャー主導の形式へ

～ キーとなる技術はモデル駆動、ドメイン特化、そしてアスペクト指向 ～

IBMコーポレーション
ソフトウェア・グループ
Rationalソフトウェア
ディスティングイッシュト・エンジニア
ジェームズ・ランボー

James Rumbaugh
Destinguished Engineer
Software Group - Rational Software
IBM Corporation

オブジェクト指向方法論OMT（Object Modeling Technique）やUML®（Unified Modeling Language：統一モデリング言語）、RUP（Rational Unified Process®：ラショナル統一プロセス）をつくり上げた功労者の一人であるジェームズ・ランボー博士は、現在IBMにおいて、MDA（Model Driven Architecture：モデル駆動型アーキテクチャー）のカギを握るとされているモデル変換（transformation）技術の研究と、OMG（Object Management Group）におけるQVT（Query-Views-Transformations：MDAの中核を支える上位メタモデル・レベルでの変換ルール記述言語の標準化コンセプト）のコア・レビューにも携わっています。
来日したランボー博士が、MDAを中心とした開発手法の最新動向に関して意見を述べます。聞き手は日本アイ・ビー・エム エグゼクティブITアーキテクト 榊原 彰です。

Management Forefront—1

SPECIAL ISSUE: **Software Engineering**

Software development: Shifting Toward More Abstract and Architecture-led Formats

-- Key technologies are: Model Driven, Domain Specific and Aspect-Oriented --

Dr. James Rumbaugh, one of the key architects of the OMT (Object Modeling Technique), UML®(United Modeling Language) and RUP (Rational Unified Process®), is now working on the transformations, a key technology for Model-Driven Architecture or MDA. He has also been working as one of the main reviewer of OMG's QVT (Query-Views-Transformations) – the concept to standardize the language to write the transformations rules at higher meta model level that is essential technology of MDA.
During his recent visit to Japan, we had a chance to listen to Dr. Rumbaugh's views on MDA and other latest trends in the software development. Interviewer is Akira Sakakibara, Executive IT Architect of IBM Japan, Ltd.

モデル駆動でオンデマンドの実現へ

MDA（Model Driven Architecture：モデル駆動型アーキテクチャー）はその名が示す通り、モデル駆動によって開発を進めていく技術です。モデル駆動とは抽象度の異なるモデルを駆使して、システムの実装を導き出すことを意味します。基本的にはPIM（Platform Independent Model：プラットフォーム独立モデル）からPSM（Platform Specific Model：プラットフォーム特化モデル）へと変換し、さらにPSMからプログラム・コードを生成する技術です。この技術が完全に実現されれば、J2EE（Java™ 2 Platform Enterprise Edition）やMicrosoft® .NETなど企業内に実行環境が複数あっても抽象度が高く実装環境に依存しないPIMを保守しておくことで、ビジネスの変化に合わせて最適な実装環境に向けたモデル（PSM）を導出し、プログラムを作り出すことができます。これはIBMが掲げる「オンデマンド」を実現する一つの形として、実に有益な技術であるといえます。

モデル駆動を以前から推進してきたRational® Softwareがこの流れを加速させることができたのは、やはりIBMとの統合によるところが大きいのです。分析・設計から開発・テストさらには運用時まで、統一された思想の下に製品が提供されることは、「オンデマンド」の推進に欠かすことのできない条件です。IBMのもともとのお客様がRationalに対して抱いてくださる興味は非常に心強いものがあります。IBMが本腰を入れてトータルな開発支援ソフトウェアに取り組み始めたことで、Rational製品はさらに発展することでしょう。これからはIBMが得意とする金融や企業マネジメントの分野でもRationalのノウハウを展開したいと思います。

さて、その際に最も重要となることは、「お客様向けのメッセージの簡素化」だと考えています。「オンデマンド」の標榜（ぼう）は実に明確なメッセージですが、製品ラインにはまだ分かりにくいところがあるように思います。ソフトウェア開発チームは、まず製品ラインの簡素化を図り、お客様向けのメッセージを簡素化することで、お客様がご自分のニーズに合った製品を簡単に見つけやすくしなければなりません。現在、MDA研究に関しては、その理解度を深めるのと同時に、既存の製品とどう組み合わせ、どの方向に製品展開を図るかについての戦略が練られています。戦略は理想を追いつつも現実的なものにしなければなりません。CASE（Computer Aided Software Engineering）ムーブメントの二の舞を演じることのないようにしなければならないのです。

「MDAなんてCASEと何も違わないじゃないか」と言い放つ人が少なからずいるようです。確かに「自動化」というキーワードで一見すると、MDAは1980年代にブームを起こしつつ、期待された効果を上げられなかったCASEと同じように見えるかもしれません。しかしながらCASEが目指したものとMDAとでは根本的に異なる部分があります。CASEは個々のソフトウェアが大きく重いものでした。設計プロセスに役立つ、あるいはコードを生成する、というプログラマーが必要としていた機能を部分的にしか備えられなかったことも衰退した原因の一つです。端的に言えば、CASEツールの多くは米国国防総省仕様をベースに開発された「文書作成」ソフトウェアであって、多くのプログラマーが切望していた「ソフトウェア開発の簡素化を助ける」ソフトウェアではなかったのです。CASEに限らずコンピューター・ソフトウェアは、目的に合致し、使いやすく、ユーザーに自発的に「これを使いたい」と思わせるものでなければなりません。

一方、MDAの目的は「プログラミング作業の簡素化」にあります。プログラミングにおける、繰り返しのコードを取り除き、プログラム自体にコードを作り出す機能を持たせることが、MDAの目標です。多くのシステムでコードの繰り返しが問題になっていますが、ある「機能」をそのデザインに付け加えることで、一つの作業のみで多くのコードを自動的に作り出すことが可能になるのです。この機能の開発こそ、プログラミング作業の簡素化であり、同時にコードの質的向上にもつながります。

MDAを支えるポイント ～ 変換と翻訳

MDAのポイントは「変換：transformations」と「翻訳：translation」にあります。モデルを使ってコードを作成するには、翻訳ソフトウェアの作成が必要になります。難しい技術ですが、過去40年にわたって培われてきたコンパイラー技術を使えば、翻訳ソフトウェアの制作は可能です。基本的には、コンパイラーは翻訳ソフトウェアの一種といえます。ただしオブジェクト指向のコミュニティーにおいては、従来のコンパイラー研究が持つ重要性と有益性がまだ十分に認知されていません。今後は、オブジェクト指向に慣れた若いプログラマーや、オブジェクト指向分野の専門家に、コンパイラー研究の歩みを理解してもらい、それを現代的にアレンジし直すことが必要になると思います。

では変換技術とは何でしょうか。それは、一つのモデルから抽象度の異なるほかのモデルを導出する際の技術です。すべてのプログラミング言語は、それぞれ歴史的におのおのの「特殊な要素」を幾つか含んでいます。それに比べるとUML®（Unified Modeling Language：統一モデリング言語）は抽象的で、より数学的であり、各分野に特有の「特殊な要素」を含まないという利便性があります。しかしUMLそのものには、モデルを簡素化する機能はありません。より高い言語レベルで事象をとらえることは可能になりますが、UMLそのものが、作業を簡素化することはありません。

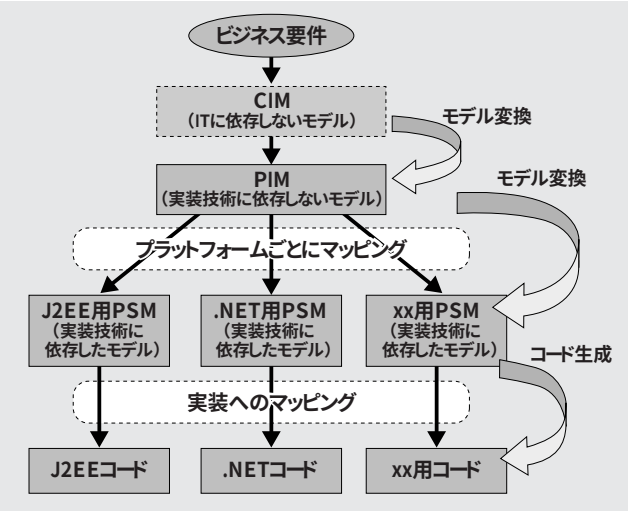

図1. MDA概念図

作業の簡素化につながる言語が、ドメイン特化言語です。数百ラインのコードを意味するドメイン特化言語を使うことで、プログラミング作業は簡素化するのです。メタモデルとしてのUMLは抽象化に優れた特質を持ちますが、ドメイン定義にはドメインに特化した言語の文法が必要になることもあります。前述したPIMや、IT（Information Technology：情報技術）を前提としないCIM（Computing Independent Model）には、ドメイン特化言語を作成して使用することが必要になります。

ドメイン特化言語を作成する際には、まずその分野の専門家が参加するモデル作りが必要になります。当然ながら、金融について何も知らないプログラマーのみで、金融特化型ドメイン・モデルは作れません。ドメイン特化言語の構築は、そのドメインでの情報のUMLモデル、つまりドメインの概念および概念間の関係を処理できるようにする「分類子」の構築から始まります。つまり、各分野に専門用語があるのは、理由があるということです。ですから各専門分野の専門知識と専門用語を網羅することが、ドメイン特化言語作成の第一歩となります。もちろんこの時点では優秀なUMLモデラーが必要になります。モデルを見ただけで、専門家に何を聞いて確かめるべきかを理解できなければならないからです。言語の作成は、正確さが肝要です。そして言葉の持つ真の意味と役割を追求するには、まずその言語の持つパターンを探し出すことが必要です。ドメイン特化言語の「ステートメント」にはある一定のパターンがあり、MDAの「ステートメント」も例外ではありません。つまりある言語の「ステートメント」を見ると、その言語に共通のパターンが見えてきます。ドメイン特化言語を作成する際には、その分野における言語の持つ「パターン」を見つけ出すことが重要になります。パターンが見つかれば、それに基づいてUMLモデルを使うこともできますし、またドメイン特化言語を作ることもできます。現在、ドメイン特化言語は主にITのベーシックな技術領域で発展してきました。例えばユーザー・インターフェース構築に用いるGUI（Graphical User Interface）ビルダーや、

音楽データをデジタル・インターフェース化するためのMIDI（Musical Instrument Digital Interface）などはその例です。ただし現時点では、ビジネス・エリアでのドメイン特化言語は極めて少ないのが実情です。ITU（International Telecommunication Union：国際電気通信連合）標準に準拠したSDL（Specification Description Language）を使用しているテレコミュニケーション業界などは、ビジネス・エリアにドメイン特化言語を持ち込んでいるまれな例といえます。モデル駆動のパラダイムに確実に移行する必要がある今日、こうしたビジネス・エリアに向けたドメイン特化言語を早急に作り上げる必要があるでしょう。OMG（Object Management Group）ではドメイン・タスクフォースを設けてこうした取り組みを推進している最中です。

アーキテクチャー主導、そしてアスペクト指向へ

このように、MDA時代に向けて、言語の持つパターンをとらえることの重要性を教えることが、より優れたモデルを作るためにもまず必要です。それよりも、もっと大切なことは「優れたアーキテクチャー」を教えることです。MDAによる構築は、基礎となる優れたアーキテクチャーがあって初めて可能となります。「MDAによる構築」とは、アーキテクチャー構築の一環としてコードを作り出すことです。従って優れたアーキテクチャーがなければ、そもそもMDAを適用する意味がありません。Rational Softwareではアーキテクチャーの重要性をずっと説いてきたので、優れたアーキテクチャーを持つ意義は教えることはできます。しかし、完璧にMDAをサポートする製品は残念ながらまだ存在していません。これが現在、IBMをはじめ各MDAソリューション・プロバイダーが直面しているジレンマです。ちょうど、まだ完成していない車を売るようなものですから、現時点で何もかもが自動的に開発されるというような夢物語や多大な期待を抱くことは避けなければなりません。もちろん、早い時期での製品化が必要であり、そのために私たちも日々努力を続けています。理論は、製品化により初めて生きてきます。もちろん業界トップで製品化を実現させたいとは考えていますが、それには幾つものハードルがあります。ドメイン特化言語や、たくさんのコードを生み出すプログラムの開発には時間がかかります。一朝一夕で完成できる技術ではないからです。段階を踏んだ開発が必要になりますが、有効な機能を備えたツールを早急に開発するとともに、お客様にはそれぞれの段階ごとに最適なソリューションをお届けしていきたいと思います。

アーキテクチャー構築という観点から、もう一つ難しい点として非機能要件の実現が挙げられます。ご存じのようにアーキテクチャーには機能的な側面と非機能的な側面があります。伝統的なUMLモデルの場合、非機能的な側面のモデル化は非常に困難です。例えばパフォーマンスや信頼性を語る場合、電話回線システムの信頼性が話題になっているのなら「ダウン時間は1年間で2分未満に抑えたい」というように簡単に言葉で表現できます。しかしUMLモデルにおいては、その表現が大変難しくなります。つまり非機能的側面の要求は、現時点では、言葉による表現の方がずっと簡単だということになってしまいます。

そのような意味でアスペクト指向に期待する点や、アスペクト指向発展への動機付けとなる点はたくさんあります。アスペクトとは、異なった事柄をふるいにかけ、選別することであり、前出の非機能的な要求、つまり「信頼性」も一つのアスペクトです。異なったアスペクトで物事を考えることにより、例えばビルを建築するときに、電気の配線関係、配管工事、エレベーター工事、エアコンの工事などを、それぞれ専門のチームに依頼するのと同じメリットがあります。つまり、それぞれの専門職がお互いに独立して、同時に作業を進行していくことで、結果として一つの大きなプロジェクトを完成させることができるからです。この考え方は、ソフトウェア開発にも十分に応用できます。しかしながらMDAソフトウェアと同様に、アスペクト指向においても、現時点ではまだ高度なアスペクト言語が完成していないという問題があります。現在のアスペクト指向言語はまだまだ低レベルで、それをうまく使いこな

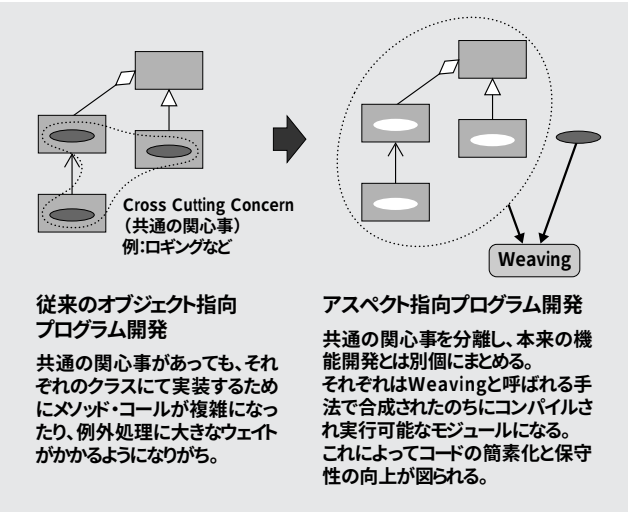

図2. アスペクト指向の概念図

すにはユーザー自身がたくさんの情報をインプットし、実行したい命令を明確にしなければなりません。「煩雑な作業工程」が伴うのです。つまり現在のアスペクト指向言語は、時間や労力のカットにはあまりつながってはいないということです。

しかし、アスペクト指向の考え方は、前述したドメイン特化言語とも共通点が多いのも事実です。ドメインの基本も、異なったアスペクトを選別し、ふるい落としていくことにあるからです。しかしそのためにも、より高度な言語、つまりユーザーを煩雑な作業から解放してくれるような優れた自動「取捨選択」機能を持つ翻訳ソフトウェアが必要となります。この機能が実現し、アスペクト指向テクノロジーが完成したあかつきには、より人間の思考回路に近い「取捨選択」機能を持つ、まったく新しい考え方が登場することでしょう。

UML 2.0

UML 2.0は、UML 1.xが持つ大きな欠陥を修正した改良版です。UML 1.xの大きな欠陥の一つが、内部構造モデル構築機能がなかったことです。UML 2.0には、この機能が備わりました。実はこれは、前述したSDLとRationalの組み込みオブジェクト指向方法論であったROOM（Real-time Object-Oriented Modeling）の両方のアイデアから生まれたもので、Rational Softwareのリアルタイム・ソリューションの対象になったものです。また、組み込み関連以外にも幾つかのメリットがあります。例えば、アクティビティー図は、ビジネスの詳細なプロセスをより的確に表現できるようになりました。UMLは研究段階を終了し、実践的に使いながらさらなる改良を加えるという新たな段階に入ったのです。UMLには既に十分な機能が備わっていますから、今後はMDAを使用した構築、特にMDAのポイントとされる変換技術の開発にもっと時間を使うべきです。実際私たちは、この作業を進めている最中ですが、まだまだ難問が山積みの状態です。

例えばアスペクト指向を考えてみましょう。残念ながら、UML 2.0はアスペクト指向をサポートできていないので、アスペクトをモデルとして表すことは困難であり（工夫次第ではそれなりに見せ掛けることはできるでしょうが）、アスペクト言語への翻訳は期待できません。実際、アスペクト指向言語が書けるプログラム言語は、まだほとんどないのが現実です。現在の実用レベルでは問題になることは少ないでしょうが、アスペクト指向ではないことが、UML 2.0の最大の欠陥だといえるかもしれません。UMLが抱える真の問題は、「異なった雑多な考え方が集まってできた言語である」ということかもしれません。UML 2.0を作るに当たって、本来ならばUML 1.0の考え方から脱却し、アスペクト言語が書けるように改良をすべきだったと思います。作業過程で意見がまとまらなかったことが、今になっても悔やまれます。UML 2.0を改良することは私のこれからの最大の課題になると思います。

しかしこれはUMLに限ったことではありません。コンピューター・サイエンスの世界においては、「どうやったら、多様な考え方をプログラムに上手に組み入れられるか？」が今後の課題になります。多様な考え方は、多様なアスペクトとも言い換えられます。多様なアスペクトが、それぞれ完全に独立しているのであれば、何の問題も生じませんが、多くの考え方がさまざまな面で少しずつ重複していることが問題なのです。この点においてUML 2.0はUML 1.0とまったく変わりません。改良が加えられなかった理由は、「や

り方が分からなかった」からです。まだまだ研究中で、必要なものを製品に組み込むことができなかったというのが正直なところです。

最新技術の習得においてもトレーニングの基本は同じ

さて、こうして最新動向をお話しすると、最新技術をどのように身に付けるのかということを、よく質問されます。開発技術がどのように変遷していく中にあっても、開発者のトレーニングの基本は変わりません。私はコンピューターを工学の一種としてとらえているので、エンジニアの訓練法もほかの工学分野に近づくべきだと考えています。まずは大学で基礎となるさまざまな科学や工学理論を学んでほしいのです。同時に多くのパターンを学んでほしいと思います。パターンを覚えることで、TPOに合わせた判断ができるようになるからです。土木工学では、地形や立地条件や目的などを考慮して、例えば「吊り橋」を架けるか、「鉄橋」を架けるかを判断することが必要なため、学生時代にさまざまなパターンを叩き込まれます。理論とパターンを学んだ後で、さまざまなプロジェクトに参加し、訓練を通じて経験を積んでいくのです。

一流の建築家を目指す学生は、単純な同じプロジェクトを何度も繰り返し、その都度、前よりもいいものを作ろうと努力します。モデルを作り、それを作り替え、担当教授からの批判を受けることで、学生たちは理論とパターンを基に建造物を造り出す訓練を積み、「仕事のコツ」や「経験則」を学んで行きます。とにかく「練習」を積まないことには、技術の上達はないのです。

しかしコンピューターのプログラマーには、コンピューター・サイエンスの授業を受けずに、プログラミングだけを学んだ者も少なくありません。原理を学ぶ必要がありますが、その目的のためにCやJavaなどのプログラミング言語を習得することを第一義とするのは不適切です。優れた芸術家を目指すなら、一流の美術館に足しげく通い、一流の芸術家の作品に触れることが必要なように、一流のプログラマーを目指す者は、一流のプログラムに触れる必要があるのです。それができないことが、優れたプログラマーの育成を妨げています。プログラマーを目指す者には、ケーススタディーとして、実際に使われているいろいろなプログラムを検証し、そのメリット／デメリットを徹底的に分析する機会が与えられるべきです。そのためにも、公開プログラムの数を大幅に増やす必要があります。これはプログラマーのみならずすべての開発者にいえることです。

そして、さらに大学を卒業後しばらくはベテランのプログラマーの下で、経験を積み、同時に先達の技や知識を見て覚える期間も必要です。またプログラマー同士の交流を深め、自分の作品に対するフィードバックを求めることも重要です。IBMでは、若いプログラマーは必ずベテランと組んで実地訓練を行い、常にベテランのプログラマーからフィードバックをもらえる環境をつくっています。エンジニアリングとは知識習得と体験を、繰り返し繰り返し行うことなのです。

新しい技術への挑戦

日本のお客様は、素晴らしい品質のシステムを確実につくり上げる能力をお持ちです。多数のお客様や学会関係者とお会いして、大変感服いたしましたし、実に多くのことを学びました。しかしながらあえて一点だけアドバイスをさせていただくことにします。それは結果を重視するあまり、過度に用心深くなりすぎていると感じる場合も少なくなかったということです。用心深いのは結構ですが、せっかくのチャンスを逃されないように。確信が持てるまで待っていると、誰かに先を越されてチャンスを失う場合もあります。リスクを承知で、新しい技術に挑戦することも時には必要であることを、再度ご確認ください。私たちもぜひそのお手伝いをさせていただきます。