# A Rational approach to model-driven development

A. W. Brown
S. Iyengar
S. Johnston

Models, modeling, and model transformation form the basis for a set of software development approaches that are known as model-driven development (MDD). Models are used in reasoning about a problem domain and the corresponding solution domain for some area of interest. In this paper, we explore model-driven approaches to the realization of solutions for today's enterprise systems. In particular, we describe the key elements of MDD as they have emerged from the support of IBM Rational® for model-driven approaches over more than a decade. We discuss the importance of creating model-to-model and model-to-code transformations that guide the development process, and we review the portfolio of IBM Rational tools in the context of their support for modeling and model-driven development.

## INTRODUCTION

The use of models and modeling in the development of software has a long and rich history. From the earliest days of writing programs, the quality and efficiency of the programming task has been closely related to the ability to develop models of the problem domain and to refine them into a solution appropriate to the target environment.[1] Many kinds of models can be created, depending upon their intended use: for establishing a clearer understanding of the problem, for communicating a shared view of the problem and its solution, for formally or informally analyzing key aspects of the solution, for generating lower-level implementation details from a higher-level programming model, and so on. The models are also highly dependent on the characteristics of the problem or solution domain that are of greatest interest, such as static relationships among domain elements, dynamic behavior of key actors in the system, operational aspects of a deployed system, expected performance parameters of a specific logical design, and so on.

Models, modeling, and model transformation form the basis for a set of software development approaches that are known as model-driven development (MDD). Models are used in reasoning about the problem domain and the solution domain for some area of interest. Relationships between these models provide a web of dependencies that record the process by which a solution was created and help to understand the implications of changes at any point in that process.

Although models can be used in many ways in the development process, a particular style of MDD relies on being quite prescriptive in the kinds of models produced, the richness of the semantics they capture, and the relationships among elements in different model types. If precise semantics are carefully adhered to for these models, one may be able to define rules for automating many of the steps needed to convert one model representation to another, tracing between model elements, and analyzing important characteristics of the models.

This prescriptive style of development is called Model Driven Architecture** (MDA**). It involves describing a solution at an abstract level in a model with well-understood semantics and using a series of explicit (often automated) steps to refine that model into more concrete models, eventually producing an executable solution for a specific runtime platform. Standards are emerging to support this architecture. The primary driving force behind MDA approaches based on a standardized set of models, notations, and transformation rules is the Object Management Group, Inc. (OMG**). OMG provides an open, vendor-neutral basis for system interoperability through established modeling standards: the Unified Modeling Language** (UML**), the Meta-Object Facility (MOF**), and the Common Warehouse Metamodel (CWM). Platform-independent descriptions of enterprise solutions can be built using these modeling standards and can be transformed for open or proprietary platforms, including CORBA**, J2EE**, .NET, XMI**/XML, and Web-based platforms.[2]

In the past few years, there have been a number of important advances in how enterprise applications are designed, implemented, deployed, and made to evolve. Hence, it is appropriate to examine the relevance of MDD in the current environment, which is characterized by the ubiquity of distributed systems, extensive use of packaged applications, and the assembly of composite solutions largely from preexisting components. This paper reviews the role of MDD in creating flexible, service-oriented architectures (SOAs) and discusses how model-driven approaches fit into today's development context. It describes the role that MDD plays in creating flexible, on-demand solutions and how the tools and processes provided by IBM Rational* are evolving to make MDD more accessible to a wider audience of practicing software engineers. Although MDD can be applied successfully to a variety of application domains, in this paper we focus on the role of MDD for enterprise solutions in typical business domains such as financial services, retail, and health care.

## CHARACTERIZING TODAY'S ENTERPRISE SOLUTIONS

As in years past, today's business leaders are striving to drive revenue growth, contain costs, and improve the effectiveness of people and processes, at the same time looking for opportunities to innovate and create new business opportunities. This balance has been at the heart of many software engineering initiatives and debates over the past 20 years. On the one hand, organizations are required to focus on containing risk through controlled, repeatable practices with incremental improvements in the context of frameworks and standards such as the Capability Maturity Model (CMM[3]), the ISO 9000 series of standards,[4] Six Sigma,[5] and the IT Infrastructure Library (ITIL**[6]). On the other hand, they must innovate and revolutionize practices to differentiate their offerings with techniques such as agile development,[7] adopting open-source technologies and practices,[8] and dynamically reconfiguring IT (information technology) solutions to optimized business processes.[9]

What sets the current business climate apart from the past is rapid, continuous change. Increasingly, organizations that can respond to change with agility and flexibility are distinguishing themselves from those saddled with complex, brittle processes and systems. Flexible businesses readily adapt to changes that result from mergers and acquisitions, compliance and regulatory mandates, increased competition, evolving technology, and shifting opportunities in outsourcing and insourcing. These businesses often share four guiding principles:

1. *Business-driven development*—The collection of capabilities offered throughout an IT infrastructure is viewed as a set of services that are assembled to meet specific business needs. This provides better insight into how business processes can be realized in IT solutions.
2. *Composite application assembly*—Components are assembled and reassembled into solutions as business and market conditions demand. Solutions typically consist of a heterogeneous mix of components developed by different teams or

acquired from various sources over an extended period of time.

3. *Efficiency through systematic reuse*—Asset management and reuse are exploited to achieve greater business efficiency. The components of a system are managed as key assets of the business and used as a basis for driving consistency and efficiency across a portfolio of solutions.
4. *Explicit support for governance to aid compliance*—The end-to-end life cycle of a system is considered to clarify the development process and to maintain traceability among its artifacts. This supports governance practices that assist in demonstrating conformance to industry standards and best practices for software development by focusing on the artifacts created throughout the development life cycle and the relationships between model elements and between models.

These principles are described in greater detail in the following subsections.

### Business-driven development

In an on demand world, success depends on the ability to rapidly respond to new challenges and opportunities. An on demand business is able to effectively transform business models and processes as needed. Business and technology teams cooperate on a shared view of the organization's needs and priorities for increased flexibility and responsiveness. Organizations that close the gap between the business view of activities and processes on the one hand and the technology that is used to realize these activities on the other can create business models in step with IT solutions. To close this gap and achieve business goals, it is vital that business goals and requirements drive IT development projects. With a flexible technology infrastructure, IT becomes an enabler of responsiveness and adaptability rather than an obstacle to them.

Closing the gap between business and IT depends on improved communication between business analysts and software architects. Solutions are built based on the needs of the business, but are flexible, enabling the business to rapidly understand the impact of changing business needs and to respond to them appropriately. Successfully connecting the business and IT parts of an organization also requires a common conceptual framework for discussing business needs and technology solutions, with a relevance both to business analysts focused on implementing business needs and IT architects focused on implementing a technology solution.

From the initial statistical and analytical models that inform business decisions to system and performance models used in defining IT policies and guiding IT projects, both business analysts and IT architects rely on various kinds of models to understand and communicate their needs. What is often missing, however, is a planned approach to coordinating the integration and use of those models across the entire system life cycle. Business-driven development focuses on this need and places attention on the creation, integration, and use of models across the business and IT domains with the goal of increasing visibility, traceability, and flexibility within the software development process.

One particular approach to software development where business-driven development is particularly relevant is in service-oriented architectures (SOAs). These architectures are gaining much support in the industry today and view enterprise solutions as federations of services connected by well-specified contracts that define their service interfaces. Systems are composed of collections of services invoking operations defined through their service interfaces. Many organizations now express their solutions in terms of services and their interconnections. The ultimate goal of adopting an SOA is to achieve flexibility for the business and IT domains.

Service-oriented development of applications (SODA) incorporates a broad range of capabilities, technologies, tools, and skill sets, including managing the service life cycle (including finding, applying, evolving, and maintaining services), modeling business processes to gain insight into current processes and the potential benefits of process improvements, establishing a platform and programming model (including connecting, deploying, and managing services within a specific runtime platform), and adopting practices and tools that enable teams to effectively create and assemble services in the context of meeting changing business needs. This includes mining existing applications to discover potential services, wrapping existing functionality to make those capabilities accessible as services, creation of new services, and "wiring" services together by connections using their interfaces. Fundamental to these capabilities is the

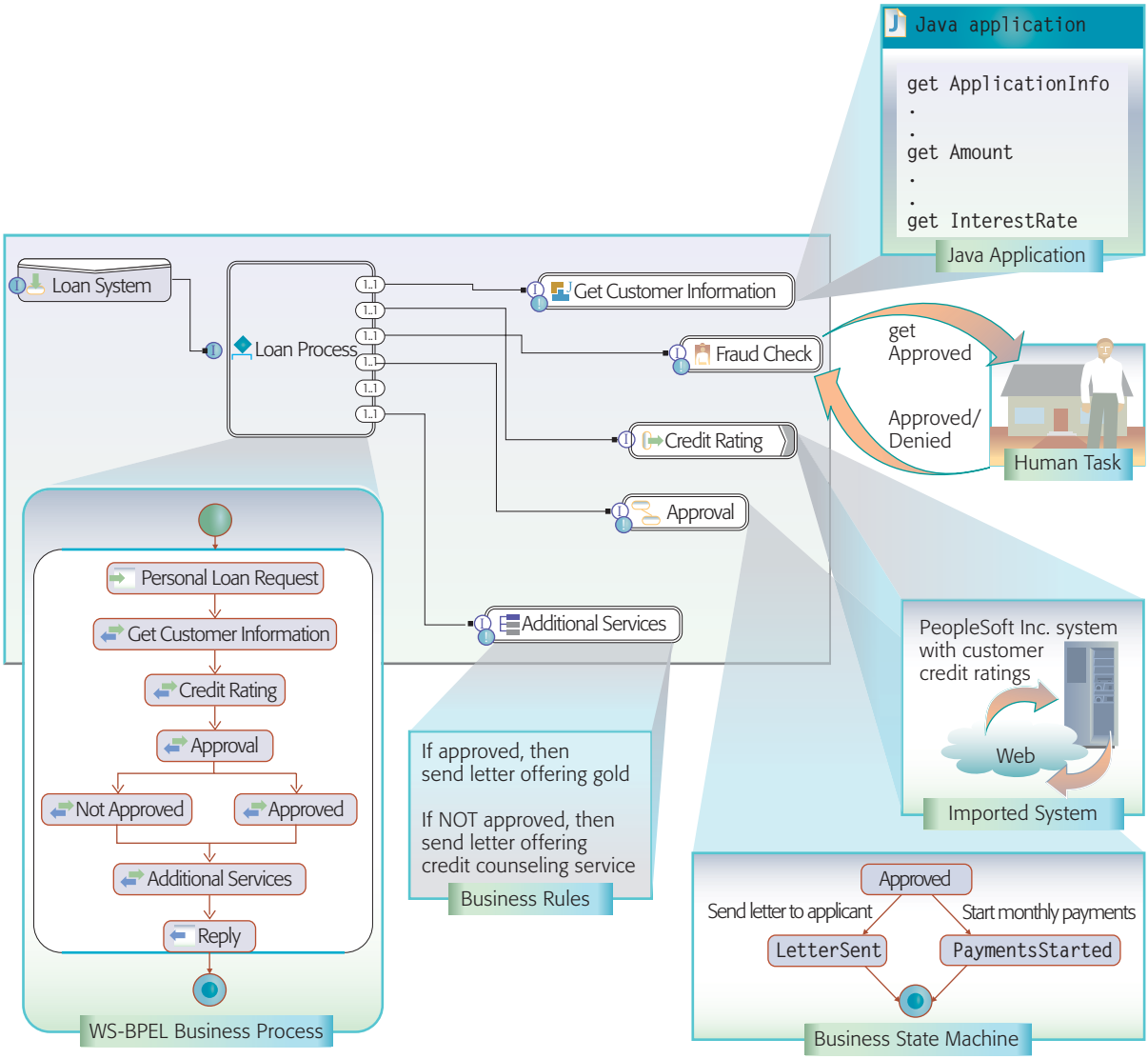


**Figure 1**
Example of a composite application

availability of clear guidance and best practices for designing services-oriented solutions in repeatable, predictable ways.

## Composite application assembly

Traditional enterprise applications were large, monolithic systems targeted at addressing a specific business function. Design techniques, tools, and processes were optimized around the development of these kinds of systems. Today's business systems, in contrast, increasingly involve composite applications. These "*n*-tiered" applications are collections of integrated capabilities and use information and logic from multiple sources. To integrate these capabilities and sources, a composite application invokes multiple transactions and subtransactions across a variety of runtime platforms and systems.

For example, in ***Figure 1*** we illustrate a composite application in which the end user is interacting with a complex system that realizes a complex business process. This process is an assembly of components deployed across Web servers, J2EE (Java** 2 Enterprise Edition) application servers, integration middleware, and legacy systems. The business process, in this case a loan application process, is realized through a combination of automated and manual activities, and delivered through a variety of

technologies. This kind of architecture is typical of enterprise systems today.

In such a complex environment, it becomes increasingly difficult to build, run, and manage applications. There are many challenges, and the current data suggests that software development tools and techniques are not meeting these challenges. For example, according to survey results, only 28 percent of projects were found to have been successful (delivering expected results on time and within budget),[10] and the average J2EE application is unavailable nearly one day per week.[11] Clearly, some new approaches to developing and managing composite applications are required to help organizations create solutions that support these kinds of complex interactions.

## Systematic reuse

When organizations think of services as key assets in the design of systems, the value of reusing these services becomes apparent. As a result, technologies and techniques are developed for the management and governance of assets and for repeatable ways to capture patterns for combining assets. These assets hold critical value for the organization and must be carefully managed. The team infrastructure for managing assets in a consistent way across projects and throughout the enterprise takes on a key role in this approach.

Like any asset, the life cycle of a service includes multiple phases in which the service is identified and discovered, harvested and created, certified and published, reused and measured, and ultimately retired. As illustrated in *Figure 2*, this asset life cycle is described through a set of workflows referred to as asset identification, asset production, asset management, and asset consumption. Services may be consumed or reused at various points in the life cycle, including during development and at runtime.

Many different kinds of artifacts are reused throughout the software development life cycle. Much of the value of any reuse depends on the reused artifact being well-understood by both the producers and consumers of the asset. Detailed knowledge and insight into the artifact becomes essential and enables the reused artifact to play a strategic role in providing consistency and quality in delivered solutions. Hence, models are obviously important in understanding reused artifacts and as

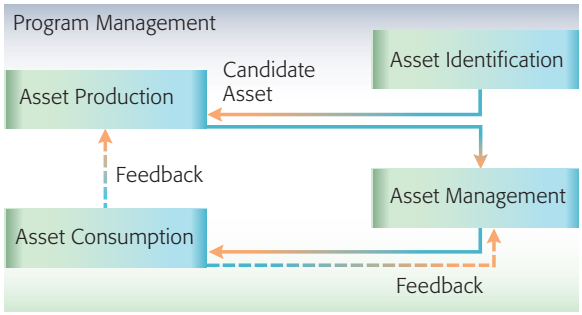


**Figure 2**
A typical reuse life cycle

primary reusable artifacts themselves in any MDD approach.

The key to successful reuse within most large-scale enterprises is the extent to which ad hoc approaches are augmented with supporting technologies and practices to facilitate reuse as part of a systematic approach. Typically, issues that need to be addressed in any successful approach to reuse include providing adequate asset-repository and searching technology, training in architectural approaches that support greater reuse, defining metrics and measures that assess progress in reuse efforts throughout the organization, and introducing reward schemes for creating reusable assets.

## Governance and compliance

As a project is executed, it is essential to ensure that appropriate practices and controls are in place to help it to deliver its promised value. Alignment between business strategy and implementation must be continually monitored, with clear oversight and control during the project's execution. To achieve this, a project must institute effective governance, namely, a clear set of tasks and roles aimed at providing the structures and policies to support management practices and decision making.

The distributed nature of today's enterprise solutions increases the difficulty of managing large projects, projects that typically cross several lines of business (LOBs). The composite nature of such distributed systems requires new techniques and approaches to provide visibility and control for the different organizations both within and outside the enterprise. This leads to much greater focus on how the different parts of a system interact, with each part of the system effectively governed for compli-

ance to requirements dictated by a service-level agreement (SLA) for factors such as security, reliability, performance, and so on. Hence, governance issues have become a very important area for organizations creating enterprise solutions today, requiring a well-defined approach to the entire life cycle of the solutions portfolio of an enterprise.

Adding to this pressure for visibility and accountability is the growing need for compliance to regulatory standards such as HIPAA (Health Insurance Portability and Accountability Act), Sarbanes-Oxley (SOX), and Basel II, fueled by intense public scrutiny of both government organizations and private companies in many industries. For example, the basis of the SOX legislation is to make officers of public companies personally responsible not only for the company's financial statements, but also for ensuring that proper controls are in place to assist in verifying the accuracy of the statements. Such regulations stress the importance of establishing and maintaining corporate accountability and periodically testing its effectiveness through audits.

This focus on accountability has profound impacts at many levels. Most notably, software and software development practices have increasingly become a target of concern because all organizations rely on software as the basis for their operation, and a growing number of them include software as a primary component of their delivered products. IT organizations have an obligation to demonstrate that (1) they are following best industry practices for software development, (2) the software designed is fit for its purpose, (3) the quality of the software being produced has been verified, and (4) effective controls are maintained throughout the software development life cycle to prove that processes are being followed. In all these aspects, models, modeling, and MDD have an important role to play as the basis for the design, construction, and quality of delivered systems. A model-driven approach enhances the visibility of design decisions throughout the project, provides a basis for analysis using different rule bases or bodies of knowledge, and explicitly represents evolutionary changes as refinement dependencies (i.e., relationships that reflect a dependency between elements, where the nature of the dependency is that one or more elements are a refinement of another element) within and across models.

## MODELS, MODELING, AND MDD

Modeling has had a major impact on software engineering and is critical to the success of every enterprise-scale solution. However, there is great variety in what the models represent and how those models are used. MDD refers to a set of approaches in which code is automatically or semiautomatically generated from more abstract models, and which employs standard specification languages for describing those models and the transformations between them. It also supports model-to-model transformations.[12]

Models are the stepping stones on the path between a description of a business need and the deployable runtime components of its solution. As understanding of the business need and the system under development evolve, the models themselves become more complete, accurate, and consistent with each other. The focus of effort also shifts from the models at higher levels of abstraction to those at lower levels. Ultimately, these models are used to directly create the deployable components.

### Models and transformations

The role of models in a model-driven approach is important, but of equal importance are the transformations that relate different models.[13] It is this ability to transform different model representations that differentiates the use of models for "sketching out" a design from a more extensive model-driven software engineering process. For example, in developing service-oriented solutions, high-level models representing business concepts can be transformed into logical models of a service-oriented solution, which in turn are transformed into implementations of services and service assemblies that realize the solution.

The process of MDD can be explored from three perspectives, as described in the following subsections: how models evolve and are related, how model transformations are defined and applied, and how automation of these transformations can increase efficiency in a software project.

#### *How models evolve*

There are two main processes that models undergo: refinement and transformation. *Model refinement* is the gradual change of a model to better match the desired system. A model is refined as more information is collected and the system is better

understood. As a model evolves, models that are dependent on it need to change in response. By the end of each iteration of the development cycle, however, all of the system's models should be consistent with each other.

Models are refined either manually or through some form of automation or assisted automation. Automation can be in the form of execution paths through rules for model refinement. These paths can be expressed as patterns and reused as assets. When a pattern is applied to a model, it modifies or rearranges the model elements to resemble the pattern. The application of a pattern adds new elements or properties to the model. Some user assistance may be involved in applying a pattern—for example, the developer may be prompted to supply an existing model element to bind to a pattern parameter. Other decisions may need to be resolved before the pattern can be executed. In the design of service-oriented solutions, for example, model refinements may include patterns for creating facades and gateways to multiple service providers and refactoring of service interfaces to meet corporate guidelines for quality-of-service criteria, such as security and availability.

*Model transformations* involve two or more models. The most typical example is a high-level abstraction model being transformed into a low-level abstracted and technology-dependent one. The former is referred to as a Platform-Independent Model (PIM), the latter a Platform-Specific Model (PSM). For example, a UML PIM could represent a logical data model and consist of a number of entity classes, each with a number of persistent attributes. This model could be transformed through automation into a UML data model that captures the same underlying entities, but in the form of database tables. The data model could in turn be used to directly generate SQL scripts that define the database, and these scripts could be directly executed on a specific database management system (DBMS). Another example is that of a logical services model describing a business process in terms of the interactions among a set of service providers and consumers. This model can be transformed into a set of instructions in an executable language, such as a Business Process Execution Language for Web Services (BPEL4WS or BPEL for short) workflow, describing that particular choreography of services. A model may also be transformed for purely internal reasons (i.e., refactoring), in which the output model is a refactored version of the input model.

Model transformations are not necessarily unidirectional. An example of a bidirectional transformation is that of a UML PSM of several Enterprise JavaBeans** (EJB**) classes, which is synchronized with the source code implementing these EJBs. New elements (i.e. methods, attributes, associations) defined in the model would generate appropriate elements in the source, and any new elements created in (or removed from) the source would generate appropriate elements in (or be removed from) the model. While UML has been mentioned as a source or destination for many of these transformations, there are many examples where model transformations occur directly without the use of UML as an intermediate representation. Common examples include transformations of relational data to XML data.

### Understanding model transformation

Defining and applying model transformations are critical techniques within any model-driven style of development. The input to the transformation is a model; the outputs can include another model or varying levels of executable code. There are three common model transformations: refactoring transformations, model-to-model transformations, and model-to-code transformations.

*Refactoring transformations* reorganize a model based on some well-defined criteria. In this case, the output is a revision of the original model, called the refactored model. An example is the renaming of all the instances where a UML entity name is used, or replacing a class with a set of classes and relationships both in the metamodel and in all diagrams displaying those model elements.

*Model-to-model transformations* convert information from a model or models to another model or set of models, typically where the flow of information is across abstraction boundaries, or between notations that emphasize different system characteristics (e.g., a process view vs a data view). An example is the conversion of one type of model into another, such as the transformation of a set of entity classes into a matched set of database schema, Plain Old Java Objects (POJOs), and XML-formatted mapping descriptor files.

*Model-to-code transformations* are familiar to anyone who has used the code generation capability of a UML modeling tool. These transformations convert a model element into a code fragment. Examples include generation of code in object-oriented languages such as Java and C++, but these transformations are not limited to output in the form of programming languages. Configuration, deployment, data definitions, message schemas, and other kinds of files can also be generated from models expressed in notations such as UML. Model-to-code transformations can be developed for nearly any form of programming language or declarative specification, such as generating Data Definition Language (DDL) code from a logical data model expressed as a UML class diagram.

### Applying model transformations

There are several ways in which model transformations can be applied. In model-driven approaches, model transformations can be applied manually, with a prepared profile, and by use of patterns.

In manually applying a transformation, the developer examines the input model and manually creates or edits the elements in the transformed model. The developer interprets the information in the model and makes modifications accordingly. The ability to use models manually may facilitate incremental adoption of models and perhaps thereby facilitate the adoption of MDD. A prepared profile is an extension of the UML semantics in which a model type is derived. Applying a profile defines rules by which a model is transformed. A pattern is a particular arrangement of model elements. Patterns can be applied to a model, and this process results in the creation of new model elements in the transformed model.

Apart from raw speed, the significant difference between manual and automated transformations is that automation is ensured to be consistent, while a manual approach is not. Automation ensures consistency when applying transformations using profiles, patterns, or other rules (where profiles and patterns can be considered to embody particular types of rules). Automatic transformations apply a set of changes to one or more models based on predefined transformation rules. These rules may be implicit to the tools being used or may be explicitly defined based on domain-specific knowledge. This type of transformation requires that the input model be sufficiently complete both syntactically and semantically and may require models to be marked with information specific to the transformations being applied.

The use of these transformation approaches usually involves developer input at the time of transformation or requires the input model to be marked. A *marked model* contains extra information not necessarily relevant to the model's viewpoint or level of abstraction. This information is relevant only to the tools or processes that transform the model. For example, a UML analysis model containing entities of the data type "String" may be marked to indicate it is of variable or fixed length, or it may be marked to specify its maximum length. From an analysis viewpoint, just the identification of the String data type is usually sufficient. However, when transforming a String-typed attribute into a database column type, for example, the additional information is required to complete the definition. Hence, although models capture information about the system from specific viewpoints, it is the transformations between models that drive model-to-model and model-to-code traceability across these different viewpoints and between different levels of abstraction.

## Models and visualization

Today, a majority of software developers still take a code-focused approach to development and do not use separately defined abstract models at all. They rely almost entirely on the code they write, and they express their model of the system they are building directly in a third-generation programming language (3GL) such as Java, C++, or C# within an integrated development environment (IDE) such as IBM Rational Application Developer, Eclipse**, or Microsoft Visual Studio**. (For this discussion, we ignore the fact that the code itself is a realization of a programming model that abstracts the developer from the underlying machine code for manipulating individual bits in memory, registers, etc.) The modeling most developers perform is in the form of programming abstractions embedded in the code (e.g., packages, modules, interfaces, etc.), which are managed through mechanisms such as program libraries and object hierarchies. Any separate modeling of architectural designs is informal and intuitive and is sketched out on whiteboards, in Microsoft PowerPoint** slides, or not at all.

Although this may be adequate for individuals and very small teams, in this approach many of the key characteristics of the system are obscured by the details of the business logic implementation. Furthermore, it becomes much more difficult to manage the evolution of these solutions as their scale and complexity increases, as the system evolves over time, or when the original members of the design team are not directly accessible to the team maintaining the system.

Developers can frequently gain additional insights when provided with code visualizations in some appropriate modeling notation. As developers create or analyze an application, they visualize the code through some graphical notation that aids their understanding of the code's structure or behavior. It may also be possible to manipulate the graphical notation as an alternative to editing the text-based code, so that the visual rendering becomes a direct representation of the code. Such rendering is sometimes called a *code model*, or an *implementation model*, although many feel it more appropriate to call these artifact diagrams and reserve the use of the term "model" for higher levels of abstraction. In tools that allow such diagrams (e.g., IBM Rational Application Developer and Borland Together**), the code view and the model view can be displayed simultaneously; as the developer manipulates either view, the other is immediately synchronized with it. In this approach, the diagrams are tightly coupled representations of the code and provide an alternative way to view and possibly edit at the code level.

UML is the most frequently used notation for visualizing static and dynamic aspects of software-intensive systems.[14,15] Originally conceived over a decade ago as an integration of the most successful modeling ideas of the time, UML is widely used by organizations and supported by more than a dozen different product offerings. Its evolution is managed through a standards process governed by the Object Management Group, Inc. The standardization of the UML notation has helped the software industry to communicate understanding of software designs by using a commonly understood visual language.

Many developers use the UML notation informally as a way to describe new and existing systems. In addition, many tools automate the process of visualizing key aspects of a software solution with the UML notation. In the Rational Application Developer product, for example, an existing Java application can be executed and elements of runtime behavior captured and displayed as a UML sequence diagram. This greatly aids understanding and debugging activities.

### Models and domain-specific languages

Models are only useful to the extent that they are clearly mapped to the concepts and characteristics that are essential to the domain of interest and the viewpoint on that domain that is of concern. When someone creates a model, whether using a graphical notation or a textual one, he or she is communicating a set of ideas about a domain of interest. It is much easier to define and share that view of the domain if it is expressed in a language that is customized for that purpose. For example, one would expect that a model that explores the efficiency of human-computer interactions with a Web-based banking system would support descriptions of the flow of interactions in a user session, and a model of a real-time telephone-switching system that is developed to assess timing would support direct representation of various kinds of synchronous and asynchronous events. This leads to a need for domain-specific languages (DSLs) specialized for such situations. To create models that are clearly meaningful requires techniques for creating DSLs.

One of the reasons for the success of UML is its flexibility. It supports the creation of a set of models representing both problem and solution domains. UML can capture and relate multiple perspectives highlighting different viewpoints on these domains, can enable modeling of the system at different levels of abstraction, and can encourage the partitioning of models into manageable pieces as required for shared and iterative development approaches. In addition, the relationships between model elements can be maintained across modeling perspectives and levels of abstraction, and specialized semantics can be created for model elements through built-in UML extension mechanisms (i.e., stereotypes and tagged values bundled into UML profiles). These mechanisms can be seen as a way to create a wide variety of DSLs. Through these mechanisms, UML can support customization for many different user communities.

Using profiles to customize UML is an excellent approach when the domain language can readily be based on existing UML concepts. It allows reuse of

the large base of knowledge, tools, and technologies devoted to supporting modeling in the UML notation. It is also possible to go beyond the bounds of UML when creating DSLs. For example, the IBM Rational Software Architect product (RSA) supports creation of DSLs by profiling UML. It also allows one to visualize and integrate models from different domain formats without the need to transform them into UML. This is possible because RSA maintains all of its internal model representations (i.e., "meta-information") in a consistent way, as Eclipse Modeling Framework (EMF) models. These meta-models (the equivalent of DSLs) leverage EMF and an extension to EMF for domain-specific modeling that provides a visualization service. This provides the ability to create much richer DSLs that extend UML with other concepts. The work at eclipse.org on EMF and the latest revision of OMG MOF 2.0 continue to advance the state of the art in bringing together open-source and open-standards modeling technologies to our industry.

## Models and software evolution

Today's enterprise software systems are rarely (if ever) developed from scratch. Rather, they are created by extending an existing solution framework with domain-specific business logic, by connecting to (and manipulating) information from different sources, and by designing rich user-display and interaction services. Hence, the development approach that is followed is not the classical "waterfall" scenario, where the gathering of requirements is followed by analysis and design, leading to the implementation of the system. Instead, it is one of continual extension and refinement of an existing partial solution toward a desired goal through a set of iterations that add value to the solution.

These partial solutions, forming the heart of a new system, may come from one of several sources:

1. *An existing set of applications*—An existing solution can be extended in many ways, as dictated by the business need.
2. *A proprietary application framework used by the organization*—Having built many kinds of similar solutions in a particular domain, some organizations have extracted core application capabilities as reusable proprietary services to be employed in future solutions.
3. *An acquired application framework*—Recognizing the consistent architectural patterns that are used in designing certain kinds of applications, developers have created a number of technologies to help organizations create solutions conforming to those patterns. The resulting application frameworks are available both commercially and in the open-source community and can be delivered as stand-alone frameworks or bundled with tools that help create, manage, and extend those frameworks.
4. *A set of extensions and customizations of packaged applications*—Many organizations acquire comprehensive solutions for key business processes from packaged-application vendors. These vendors have structured their solutions to support different kinds of extension and customization, offer well-defined APIs (application programming interfaces) to access the internal structure of the packaged application, or augment the packaged applications with detailed design documents, extension examples, and package-specific tools.

The primary task faced by many IT project managers is to create a clear understanding of their domain, to express that understanding in a PIM supporting various kinds of analysis ensuring its correctness and consistency, and to map that domain model to a PSM realized by extending the application framework. Model-to-model transformations help in refining the domain, while model-to-code transformations map the domain model to the implementation.

In model-to-code transformations, the application framework plays a key role, as it constrains and guides the kinds of transformations that are meaningful. For example, if the application framework is based on Struts (an open-source framework for building Web applications based on servlets or JavaServer Pages** [JSPs**] by using the Model-View-Controller design paradigm), the application being created has a well-understood structure, including well-known extension points where business logic can be realized. A set of transformations can be created based on that knowledge. Indeed, "wizard" style tools can be created to automate creation of those transformations for domain models containing appropriate kinds of information. This is the way, for example, in which tools such as the IBM Rational Application Developer use domain-focused visual design tooling to automate code generation for Struts or application frameworks based on JSF (JavaServer** Faces). More generally, by using an application framework as the basis for a

solution, the task of writing model-to-code transformations is significantly facilitated, and greater efficiency, predictability, repeatability, and manageability of the resulting solutions are the consequence.

New software is always built upon existing software and transforming models (to other models or to code) is critical in leveraging existing frameworks. MDA plays a critical role in automating how we extend and customize these frameworks. In light of the constraints inherent in today's components and technologies, some experts view this "custom automation" as the only viable option for creating systems of increasing complexity.

## THE IBM RATIONAL SOFTWARE DEVELOPMENT PLATFORM

The IBM Rational Software Development Platform (RSDP) is a complete, open, modular, and proven solution for developing software.[16] RSDP provides development teams with a powerful foundation for the MDD of enterprise solutions and supports many other styles of development as well. It includes a broad array of best-of-class tools for every phase of the development life cycle and offers deep integration of tools and runtime capabilities across all different aspects of the business in support of a service-oriented view of solutions. Tight integration between tools in RSDP is based, in part, on the Eclipse platform, which also enables integration with third-party ISV (independent software vendor) tools. Eclipse technology enables integration, flexibility, and extensibility in a single, coherent platform for software development and serves as a foundation for the use of RSDP by a broad constituency of users.

RSDP supports key capabilities for building enterprise applications, including requirements analysis, design, construction, software quality, process and portfolio management, and change management and defect tracking. These capabilities are illustrated in *Figure 3*, which shows the RSDP components used by each design team member. RSDP consists of technologies that can be applied to many solution domains for creating a variety of kinds of systems. The remainder of this section will discuss the IBM Rational tools relevant to each stage of the software life cycle, illustrating their use in the context of developing services and SOAs for enterprise IT solutions.

### Business modeling and requirements

Effective business-driven development starts with analyzing business requirements, needs, and existing processes. This analysis is then used to determine the system's requirements and to guide all subsequent stages of development.

IBM WebSphere* Business Modeler enables analysts to model, simulate, and analyze complex business processes quickly and effectively. Analysts use WebSphere Business Modeler to model "as-is" and "to-be" business processes, allocate resources, and perform "what-if" simulations to optimize and estimate business benefits. These models can later be transformed into UML and BPEL to jump-start design and integration activities.

IBM Rational RequisitePro* is used to define business requirements and refine them into software requirements and use cases. Requirements and their management are central to any mature software development process, and business modeling helps ensure that software requirements and use cases reflect real business needs. Well-written use cases provide the foundation for architecture, analysis, user interface (UI) design, and testing.[17] Requirements in Rational RequisitePro can be linked to process models, as well as test cases and test plans, providing full traceability and ensuring that business requirements drive downstream development. The result is a system that meets business goals and requirements and delivers real business value. Within RSDP, requirements are integrated with defect tracking, design, development, and testing tools to jump-start activities, minimize rework, and provide each team member with a direct window into end-user needs.

### Service-oriented solution design

An effective approach to designing a service-oriented application requires the modeling and definition of well-defined, well-documented interfaces for all major service components prior to the construction of the services themselves. RSA is an integrated design and construction tool for creating service-oriented applications. By leveraging MDD with UML and unifying all aspects of software application architecture, RSA enables software architects to design flexible service architectures and automatically apply design patterns for SOA from analysis and design to implementation.

Integration between IBM WebSphere Business Modeler and RSA enables the transformation of

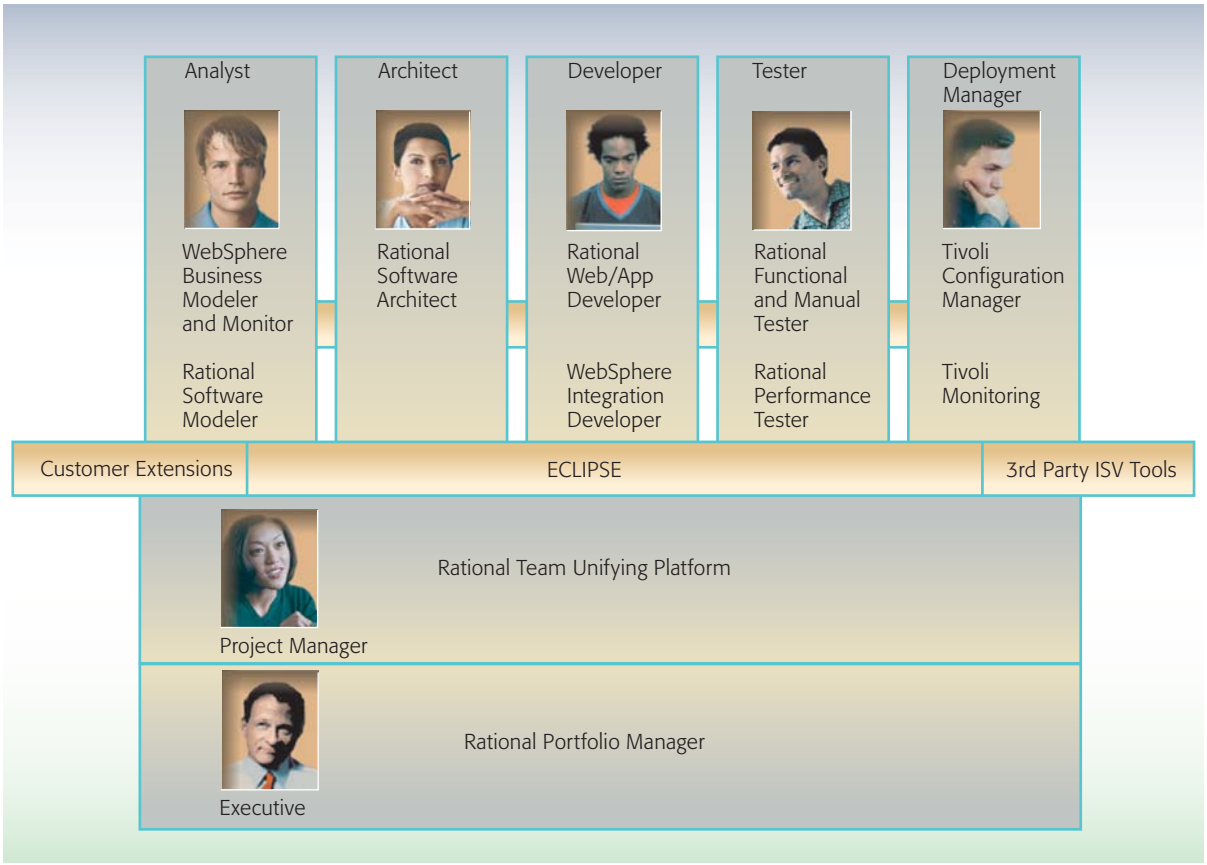


**Figure 3**
The IBM Rational Software Development Platform

business models into software models for automating business activities. The integration automatically maps business notation to software notation (UML), relieving designers from that conversion burden.[18]

To assist with solution design in a service-oriented context, RSA can be specialized with domain-specific notations and DSLs relevant to defining, assembling, and managing services. General mechanisms are available to create customized notations using built-in product extension mechanisms. An example of DSLs created with these mechanisms include the UML Profile for Software Services, which accelerates the transformation of business processes into Web services by providing a common language for describing services that helps software architects to model, map, and partition services.[15] Another example is the UML Profile for Business Modeling, which presents a language for capturing business models in UML and is supported by the Business Modeling Discipline in the Rational Unified Process* (RUP*).[15] (Plug-ins and scripts, including UML profiles, are available at http://www.ibm.com/developerworks/rational/library/1376.html.)

Applying profiles in UML results in a "marked up" model that has specific stereotypes and tagged values applied. Based on these markings, a set of specific transforms can be created that convert those models into some other form. For example, associated with RSA is a transformation that converts models marked with the UML Profile for Software Services into initial service specifications in the Web Services Description Language (WSDL). Those service realizations are then further elaborated and refined within a set of service construction activities.

## Service-oriented solution construction

Java, J2EE, and Web developers are often required to create the core elements supporting an SOA, typically Web services. Manually creating Web services—or even developing systems that use existing Web services—requires a substantial

amount of tedious and error-prone work. Much of this burden can be relieved through appropriate tooling.

IBM Rational Application Developer (RAD) is a comprehensive integrated development environment for service-oriented development that automates many of the tasks commonly performed in the construction and consumption of Web services so that developers can focus on writing the business logic code. RAD can automate everything from WSDL file and code generation to test proxy generation and Web Service Interoperability (WS-I) conformance verification. Developers use RAD to create, build, consume, test, deploy, and publish Web services, either from scratch or by enabling existing applications to achieve WS-I compliance.

RAD greatly simplifies the process of creating a Web service in a top-down manner based on requirements specifications and UML models. Developers save time and reduce errors by using RAD to automatically generate WSDL files that contain an XML schema to describe Web services as well as skeleton JavaBeans** or EJB files. RAD also includes an XML schema definition (XSD) editor for specifying the format of messages.

When working in a bottom-up manner from an existing set of Java classes or EJBs, RAD can also automate much of the process of turning those components into a Web service. After developers use a wizard to specify information about the component, RAD generates the WSDL files describing the Web service, a Simple Object Access Protocol (SOAP) deployment descriptor, and a proxy that a client can use to access the Web service.

The support of RAD's Enterprise Generation Language (EGL) enables business-oriented and procedural developers who may not know Java to use it for developing, testing, and debugging data-driven Web applications, Web services, and business logic by using procedural programming constructs. EGL supports an MDA style of development in which developers create PIMs of aspects of the system to be built and use the predefined transformations to convert those models into a PSM for a choice of different target runtime platforms. EGL's comprehensive support of Web services handles most of the hard work in developing or using Web services. Very little coding is required to use existing Web services or existing assets, leading to a shorter learning curve and high productivity for business-oriented developers.

To enable developers to exercise quality control early in the development cycle, RAD includes automated code review tools to validate coding best practices, component test tools to automate the creation of test stubs, and test harnesses and test input data (based on WSDL file analysis and information from performance profiling tools) to detect and diagnose bottlenecks and deadlock and race conditions.

### Service composition and assembly

Integration developers create and deploy composite applications by refining the business process models created in WebSphere Business Modeler into an executable set of choreographed services. WebSphere Integration Developer is an IDE used by developers to assemble composite applications that deploy to the IBM WebSphere Process Server. Using the visual BPEL editor, integration developers can view the business processes designed by business analysts. They then use WebSphere Integration Developer to choreograph the services, join them into a composite application, optionally test them with a built-in test environment, and deploy them directly to a runtime environment.

At the center of WebSphere Integration Developer is the Service Component Architecture, a programming model that focuses on connecting service components to realize composite applications. WebSphere Integration Developer treats everything as a component. Each component has an interface and can be "wired together" with other components to form a module (with its own interface). This enables changing any part of a composite solution without affecting the other parts. For example, a human task can be replaced with a business rule without the need to update the business process.

As illustrated in ***Figure 4***, there are a number of key elements to WebSphere Integration Developer that support composite application assembly:

- *Business objects*—Business objects can be designed and used in a composite application and are used as input or output parameters for all integrated components in a solution.

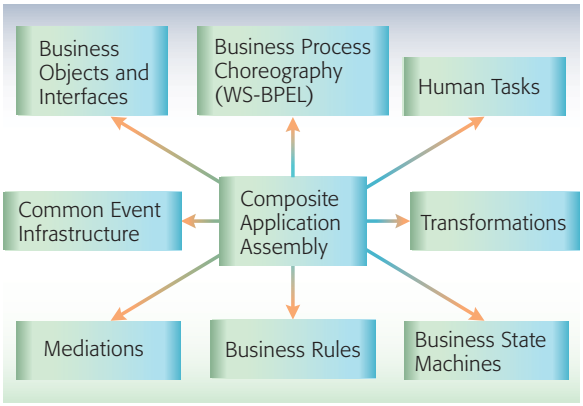


**Figure 4**
Key aspects of WebSphere Integration Developer

- *Business process choreography*—Direct execution of processes expressed in WS-BPEL is possible by use of a specific runtime platform. The platform has built-in capabilities for monitoring and managing the execution of business processes, including support for compensation and rollback.
- *Human tasks*—The process execution engine is augmented with a human task manager that supports modeling and execution of manual activities in a workflow.
- *Mediation*—To assist with wiring services, WebSphere Integration Developer provides common services for data mapping, relationship management, and interface mediation.
- *Adaptive entities*—A state-machine implementation is included for modeling business processes that are more amenable to state machine representation than to a traditional procedural flow representation.
- *Business rules*—Many business decisions can be modeled with decision tables and rule sets supporting simple rule-based logic for process interaction.
- *Common event infrastructure*—Supporting monitoring and management of processes is a foundational set of services for common event reporting and monitoring across a set of components.

WebSphere Integration Developer provides the visual software development tools needed to specify, test, and deploy executable business processes that integrate Web services, enterprise applications, human tasks, and other service components into effective SOA-based business solutions.

## Process and portfolio management

In addition to a comprehensive set of tools for service-oriented development of applications, RSDP also incorporates practical process guidance. In particular, RUP is well-suited to the needs of many styles of development initiative because it is founded upon software-engineering best practices, offers a configurable process framework, and is scalable to support enterprise initiatives. RUP has a long history as the leading commercial process-framework product in the market, providing flexibility through its architecture and tool support for the tailoring, extension, and deployment of processes. Underlying RUP is a sophisticated model of the software development process and its roles and activities. This model is used as the basis for an MDD approach to generate a specific RUP instance, a "RUP configuration," from a collection of core contents and an extensible set of RUP plug-ins specific to particular development approaches and styles. The IBM Rational Method Composer (RMC) is a process management tool platform and conceptual framework. It consists of tools for authoring, configuring, and viewing method content and processes. The kernel RUP framework used by RMC for authoring and extending RUP has been donated to the Eclipse Foundation as part of the Eclipse Process Framework.[19] This framework is influencing a key OMG standard, the Software Process Engineering Metamodel (SPEM).

RUP plug-ins contain a variety of process components customized for specific tools, technologies, or domains. The plug-ins include those tailored to a business-driven reuse-based SOA approach, namely:

- *The RUP plug-in for SOA* integrates support for SOA and service-oriented solutions into the RUP framework, with SOA-specific concepts, guidelines, activities, artifacts, and tool mentors.
- *The RUP plug-in for WebSphere Business Modeler* augments the business modeling capabilities of RUP to leverage WebSphere Business Integration solutions and provide a unified approach for business modeling based on the essential capabilities of RSDP.
- *The RUP plug-in for asset-based development* provides a set of best practices for reuse of software within an organization. It focuses on asset identification, asset production, and asset consumption.

Today's enterprise solutions are increasingly composed of services from many sources. As a result, applications are best viewed as dynamic entities—combinations of services to meet a particular set of business requirements. One effect of taking this view is a need for enterprises to explicitly design, implement, and manage the set of services in the organization as a portfolio of available capabilities. Executives and project managers can use the IBM Rational Portfolio Manager to gain insight into the business benefits, costs, and risks of the portfolio of SOA services. With Rational Portfolio Manager, they can prioritize proposed, existing, and "under-construction" services; track service level financial statistics; manage SOA project-team dependencies; forecast demand for service creation; and better understand the cost of SOA creation, operations, and maintenance.

### Software quality

In addition to tools to assist the developer in testing capabilities in Rational Application Developer and WebSphere Integration Developer, RSDP includes tools to help testers continuously assess the quality of services and composite applications. Test plans and individual test cases can be linked to project requirements in Rational RequisitePro to ensure complete requirements-based test coverage and support traceability across the project life cycle.

With Rational Manual Tester, each sequence of steps defined for a manual task is captured as a test script. These elementary test scripts are then easily combined into longer test scenarios (including entire business processes) without any duplication of content. Test scripts maintain references to other scripts, thus greatly simplifying the maintenance of the test scenarios. The only update required when a manual task changes consists in updating its associated script—all the test scenarios are automatically updated to reflect the change.

Rational Functional Tester strikes a very good balance of UI technology coverage and openness. In particular, it offers scripting using nonproprietary languages (Java or VB.NET), making it flexible enough to accommodate most conditions. In addition, test scripts can be easily recorded on a combination of different user interfaces, potentially built with different technologies. This makes the orchestrated verification of an end-to-end business process virtually seamless. Furthermore, test execution can be triggered automatically from the command line, making it easy to integrate as part of an end-to-end "build-deploy-test" process.

IBM Rational Performance Tester is a test creation, execution, and analysis tool that validates the scalability and reliability of applications under multiple user loads. Performance Tester enables teams to pinpoint system bottlenecks before application deployment. Tests are created by recording a user's activity within a Web browser; teams need little to no programming knowledge to understand and modify these tests. A simple interface initiates a data-pooling capability that ensures unique data for each emulated user. Using an intuitive graphical test scheduler, teams can then organize their tests to accurately simulate the different types of users and user activities the application under test will support.

During test execution, while emulating the desired number of concurrent users, Rational Performance Tester generates reports that clearly highlight poorly performing Web pages, URLs, and transactions. Teams are thus able to discover performance problems in even the most complex systems, increasing the opportunity for problem capture and repair before the system goes live.

## ASSET LIFE-CYCLE MANAGEMENT

With the focus on asset management and reuse, a new emphasis has been placed on creating, harvesting, applying, and managing assets throughout the life cycle. For practical reasons, this is accompanied by strong governance practices for reusable assets, tied to a flexible asset life-cycle management system.

As organizations create solutions that leverage existing assets, the ability to accelerate and manage change becomes more critical. Using IBM Rational ClearCase* and ClearQuest* for change management and defect tracking, teams can automate the software life cycle and establish a consistent process across distributed environments. IBM Rational software configuration management products help teams improve project collaboration and release coordination, increase development responsiveness and agility, and enhance operational efficiency.

In addition, reusable SOA-related development assets can be packaged and used (or "consumed")

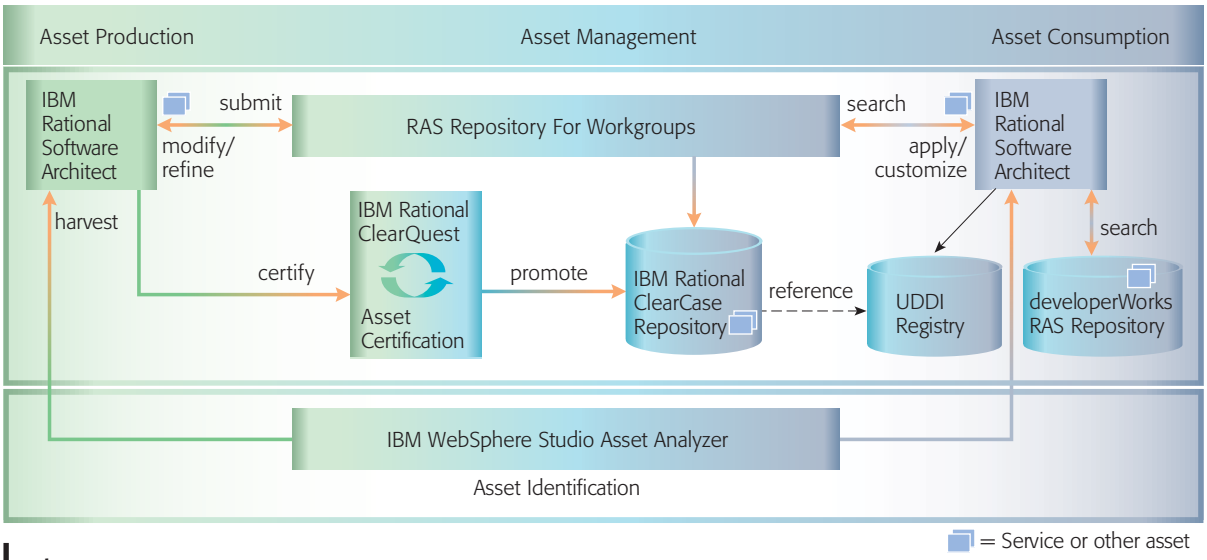


**Figure 5**
An example of the asset management life cycle using IBM Rational tools

within RSDP through direct support of the OMG Reusable Asset Specification (RAS) standard.[20] This is illustrated in ***Figure 5***. As shown in the figure, assets created in the Rational Software Modeler (RSM) are submitted in RAS format to the RAS Repository Service for storage in a ClearCase repository. The governance of the assets, including certification, is managed by customized templates that drive the workflow engine provided by ClearQuest. Other users of RSA/RSM can then search and reuse these RAS-based assets by making queries into the asset repository by using the RAS Repository Service.

RSDP includes several other tools that enable a team to more effectively manage asset workflow. WebSphere Studio Asset Analyzer provides support for the asset identification workflow. RSA provides a RAS client for asset production. This tool also includes a RAS client, called the Asset Explorer, to facilitate asset consumption. Developers and architects use the Asset Explorer to connect to one or more RAS repositories and issue searches, which examine the RAS XML manifest in each of the RAS assets.

The assets and services can be measured in several ways, quantitative and qualitative. Teams can use the RAS repository for work groups to track metrics, such as how many times an asset is searched for, how often its documentation is browsed, and how many times it is imported and reused. Developers can also rate the asset or service and provide textual feedback. These kinds of measurements are critical for the health and functioning of the asset repository.

## SUMMARY

Flexibility is essential to organizations today as they seek to react more quickly to the changing demands of their customers and competitors and the evolving business environment. The role of software in many businesses is now seen as central to their ability to compete effectively and efficiently. A service-based orientation to the systems being developed helps businesses focus on what is essential—the services they offer to customers. It also helps IT professionals to look at the systems that support the business in a different way—as composable solution fragments that must be assembled to meet evolving business needs. This view is an important cornerstone of today's highly reactive business environment.

In this paper, we have focused on the importance of models, modeling, and MDD in creating service-oriented enterprise solutions and on the way in which Rational tools support these concepts. To illustrate these qualities, we reviewed RSDP with particular focus on its support for MDD across the software development life cycle. Further details and discussion of Rational's support for MDD can be found elsewhere.[12,13,21] We have also highlighted

how Rational tools build on a combination of open standards (from OMG, OASIS and W3C**) and open-source technologies (from Eclipse and Apache projects) to create an extensible application life-cycle platform for the industry.

RSDP plays an important role in helping organizations create models of their domain of interest and use those models in a well-defined process that supports traceability and visibility across the life cycle. It combines market-leading products to create a rich, integrated environment for solution development. Furthermore, through support for MDD techniques, RSDP helps to ensure that customers can efficiently deliver service-oriented solutions that meet their business needs.

## ACKNOWLEDGMENTS

This paper draws on the insight and energy of many people at IBM, notably Jim Amsden, Grady Booch, Gary Cernosek, Jim Conallen, Grant Larsen, Sky Matthews, Martin Nally, Jim Palistrant, Jim Rumbaugh, Bran Selic, and Rick Weaver. We also recognize the input and support from all those in the IBM Rational Communities of Practice.

*Trademark, service mark, or registered trademark of International Business Machines Corporation.

**Trademark, service mark, or registered trademark of Object Management Group, Incorporated, Sun Microsystems, Inc., Office of Government Commerce, H. M. Treasury, United Kingdom; Borland Software Corporation, Microsoft Corporation, the Eclipse Foundation, Inc., or Massachusetts Institute of Technology in the United States, other countries, or both.

## CITED REFERENCES

1. M. S. Mahoney, "The Roots of Software Engineering," *CWI Quarterly* **3**, No. 4, 325–334 (1990), http://www.princeton.edu/~mike/articles/sweroots/sweroots.pdf.
2. M. Belaunde, C. Burt, C. Casanave, F. Cummins and D. DSouza, K. Duddy, W. El Kaim, A. Kennedy, W. Frank, D. Frankel, R. Hauch, S. Hendryx, M. Hettinger, R. Hubert, D. Hybertson, S. Iyengar, J. Jourdan, T. Koch, T. Kurokawa, A. Mallia, S. Mellor, J. Miller, J. Mischkinsky, J. Mukerji, C. Mullins, M. Oya, L. Rioux, P. Rivett, E. Seidewitz, B. Selic, J. Siegel, O. Sims, D. Smith, R. Soley, A. Tanaka, S. Tyndale-Biscoe, A. Uhl, A. Watson, D. Weiseand, and B. Wood, *MDA Guide Version 1.0.1*, Object Management Group, Inc. (June 12, 2003), http://www.omg.org/docs/omg/03-06-01.pdf.
3. M. B. Chrissis, M. Conrad, and S. Shrum, *CMMI: Guidelines for Process Integration and Product Improvement*, Addison-Wesley, Reading, MA (2003).
4. J. T. Rabbit and P. A. Bergh, *The ISO 9000 Book: A Global Competitor's Guide to Compliance and Certification*, AMACOM Books, New York (1994).
5. T. Pyzdek, *The Six Sigma Handbook, Revised and Expanded*, McGraw-Hill Professional Books, New York (2003).
6. *An Introduction to ITIL*, Office of Government Commerce (August 2005), http://www.ogc.gov.uk/sdtoolkit/reference/ogc_library/itbusinesschange/ITILIntroduction.pdf.
7. C. Larman, *Agile and Iterative Development: A Manager's Guide*, Pearson Education, Upper Saddle River, NJ (2003).
8. C. DiBona, S. Ockman, and M. Stone, Editors, *Open Sources: Voices from the Open Source Revolution*, O'Reilly Media, Sebastopol, CA (1999), http://www.oreilly.com/catalog/opensources/toc.html.
9. H. Smith and P. Fingar, *Business Process Management: The Third Wave*, Meghan-Kiffer Press, Tampa, FL (2002).
10. *Extreme Chaos*, The Standish Group, Survey report (2001), http://www.standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf.
11. N. Ptak, "The State of J2EE Application Management: Analysis of 2003 Benchmark Survey," Ptak, Noel & Associates (November 2003), http://ptaknoelassociates.com/research.htm.
12. A. W. Brown, J. Conallen, and D. Tropeano, "Models, Modeling, and Model Driven Development," in *Model-Driven Software Development*, S. Beydeda, M. Book, and V. Gruhn, Editors, Springer, Berlin (2005), pp. 1–17.
13. A. W. Brown, J. Conallen, and D. Tropeano, "Practical Insights into MDA: Lessons from the Design and Use of an MDA Toolkit," in *Model-Driven Software Development*, S. Beydeda, M. Book, and V. Gruhn, Editors, Springer, Berlin (2005), pp. 403–432.
14. J. Rumbaugh, G. Booch, and I. Jacobsen, *The UML Reference Manual*, Addison-Wesley, Reading, MA (2004).
15. S. Johnston, "Modeling Service-Oriented Solutions," IBM developerWorks (July 2005), http://www-128.ibm.com/developerworks/rational/library/jul05/johnston/.
16. A. W. Brown, M. Delbaere, P. Eeles, S. Johnston, and R. Weaver, "Realizing Service-Oriented Solutions with the IBM Rational Software Development Platform," *IBM Systems Journal* **44**, No. 4, 727–752 (October 2005).
17. Kurt Bittner and Ian Spence, *Use Case Modeling*, Addison-Wesley, Reading, MA (2002).
18. J. Amsden, "Business Services Modeling," IBM developerWorks (December 2005), http://www.ibm.com/developerworks/rational/library/05/1227_amsden/index.html.
19. Eclipse Process Framework Project (EPF), Eclipse.org, http://www.eclipse.org/epf.
20. *Reusable Asset Specification Version 2.2*, Object Management Group, Inc. (November 2005), http://www.omg.org/cgi-bin/doc?formal/2005-11-02.
21. A. W. Brown, G. Booch, S. Iyengar, J. Rumbaugh, and B. Selic, "An MDA Manifesto," in *Model Driven Architecture Straight from the Masters*, D. Frankel and J. Parodi, Editors, Meghan-Kiffer Press, Tampa, FL (2004).

*Accepted for publication February 22, 2006.*
*Published online July 11, 2006.*

*Alan W. Brown*
*IBM Software Group, Rational, 4205 S. Miami Blvd., Durham, North Carolina 27709 (awbrown@us.ibm.com).* Dr. Brown is an IBM Distinguished Engineer with the IBM Rational software group. He is responsible for aspects of future product strategy in IBM Rational's design and construction products. He defines technical strategy and evangelizes product direction with customers, looking to improve software development efficiency through visual modeling, service-oriented design, generating code from abstract models, and systematic reuse. His current focus is on how service-oriented solutions are created and evolved, with particular interest in software process improvement, Model Driven Architecture, software design and development, and component-based reuse. He received a Ph.D. degree from the University of Newcastle in the United Kingdom.

*Sridhar Iyengar*
*IBM Software Group, Rational, P.O. Box 12195, 3039 Cornwallis Rd., Research Triangle Park, North Carolina 27709 (siyengar@us.ibm.com).* Mr. Iyenghar is an IBM Distinguished Engineer and leads the technical strategy for IBM Rational Software. He works with development teams and architects inside IBM and in the industry to accelerate the use of industry standard models, patterns, and metadata for developing, integrating, and managing applications. His work focuses on the use of models, metadata, and transformation frameworks to create an integrated software platform built using industry standards (J2EE, Web Services, MDA, etc.) and open-source technologies (Eclipse, Apache, etc.). Mr. Iyengar serves on the IBM Software Group Architecture Board Steering Committee and the OMG Architecture Board. He works extensively with customers who are at the forefront of exploiting emerging technologies. He holds several patents, has a Master's degree in computer science, and has been in the IT industry for 24 years working on databases, design and development tools, and integration technology.

*Simon Johnston*
*IBM Software Group, Rational, 4205 S. Miami Blvd., Durham, North Carolina 27709 (skjohn@us.ibm.com).* Mr. Johnston is a member of the IBM Rational strategy team and is responsible for the business-level tooling strategy. He has undertaken a number of standards-related activities for both Rational Software and IBM in the area of XML (W3C Schema working group), Web Services (RosettaNet architecture team), and modeling (OMG UML and OCL teams). He was the author of the UML Profile for Software Services and primary author of the RUP Update for SOA. ■