Rational Software Architect: A tool for domain-specific modeling

D. Leroux
M. Nally
K. Hussey

Rational Software Architect (RSA), the latest generation Rational® modeling tool, is based on Eclipse™ Modeling Framework (EMF) technology. RSA offers all the important features of the previous generation of Rational modeling tools, while supporting a much wider range of model formats. RSA diagrams can be used in editing and displaying models derived from any EMF-based metamodel. The combination of RSA and EMF provides a powerful capability for integrating domain-specific languages (DSLs) with UML® in a single toolset. This paper describes how RSA and EMF provide these capabilities and explores some of the ways that IBM is currently exploiting them.

INTRODUCTION

The development of the Unified Modeling Language**[1] (UML**), standardized by the Object Management Group, Inc. (OMG**) in 1997, was an important step in focusing efforts to create a single object-oriented modeling language. An industry of services, consultants, and tools has sprung up around UML, and tools from IBM Rational* are among the market leaders. In 2005, a major revision of UML, UML Version 2.0[2] (also called UML 2), was created, expanding the scope of concepts described in the UML standard.

One of the reasons for UML's success is that it contains abstractions for many standard object-oriented modeling concepts, such as class diagrams, state-machine diagrams, use-case diagrams, and sequence diagrams, with which users can describe the architecture, design, and even implementation of software systems. Despite this richness, users sometimes want to capture information in models not provided for in UML. For this purpose, UML provides the concept of a *UML profile*. UML profiles allow the definition of *stereotypes*, which are designed extensions of UML elements that allow users of UML to annotate UML elements with extra information. Stereotypes provide a simple but powerful mechanism for extending and adapting UML. If users want to model something that is not exactly the same as a UML concept, they can often find a UML concept that is close to what they want and customize it with a stereotype. Because UML tools support the concept of a stereotype, users can

create their own UML modeling extensions and still exploit UML tools to display and edit their models.

This use of stereotypes to extend UML and UML tools is a good example of the value of reuse; defining a whole new modeling language and toolset would usually be prohibitively expensive. Still, the definition of stereotypes is not adequate if one needs to define a concept for which there is no similar UML concept. There are also cases where users need more control over the behavior of the modeling tool than can be achieved by customizing a UML modeling tool with stereotypes. Models that define concepts different from the standard concepts in UML are sometimes called domain-specific languages (DSLs) or domain-specific modeling (DSM) languages. Some DSLs describe concepts that are specific to a particular technical domain but outside the scope of UML. Other DSLs describe concepts unique to a particular application or solution domain. Many articles and books have described the value of DSM languages[3] or "software factories"[4] for particular domains.

Although UML is a powerful force for unifying modeling concepts within the object-oriented domain, there are many modeling languages from other domains, some of which predate UML. The entity-relationship (E-R) model,[5] first proposed in 1976, inspired some of the ideas that found their way into UML, but E-R modeling itself has also continued to be successful, especially in the relational database domain. There are enough differences between the E-R model and the concepts it inspired in UML, such as support for explicit keys, that E-R modeling has never been subsumed by UML. More recently, the Extensible Markup Language[6] (XML) and Internet standards have created modeling languages such as XML Schema[7] (XSD) and Web Services Description Language[8] (WSDL). XML Schema is a language for modeling XML data, and WSDL models the interfaces to Web services.

Recognizing the need for supporting the development of DSLs, and the continuing and emerging importance of non-UML standard modeling languages, the designers of Rational's next-generation modeling tools set out to make sure the tools could support a broad range of modeling languages equally. RSA supports the same UML extension capabilities as the previous Rational modeling tools, Rational Rose* and Rational XDE* (Extended Development Environment). These extensions enable users to quickly create UML profiles to address domain-specific concerns, but unlike the previous tools, RSA offers other means for integrating non-UML standard modeling languages and DSLs.

Rational Rose and XDE relied on reverse engineering and round-trip engineering to create UML from domain models such as Enterprise JavaBeans** (EJBs**), Java**, and C++. (*Reverse engineering* in this context is the process of converting source code files into UML model elements. *Round-trip engineering*, found in XDE and Rose, allows the user to synchronize the contents of a UML model with a set of source code files.) A domain-specific profile was used in order to visualize, model, and reference existing domain models. This led to significant redundancy between the UML models and the domain models, and many issues related to synchronization of these artifacts. Users had to create large UML "library" models for referencing existing domain-specific libraries; for example, with Rational Rose, using a type from the standard Java library requires importing the Java library package into a model.

Unlike these previous tools, the modeling capabilities of RSA allow users to visualize and integrate models and model elements from different domain formats without having to create, store, and synchronize reverse-engineered versions of these models transformed into UML models. Because RSA's internal model representations are based on EMF metamodels (a *metamodel* is a model that defines another model), this task is made easier. RSA includes EMF representations for UML2, EJBs and C++, among others. (For an example of an EMF-based model, see Reference 9, which provides an overview of the open implementation of the UML2 metamodel underlying RSA.) RSA integrates these metamodels, allowing them to reference one another by leveraging EMF and the RSA-specific extensions to EMF for DSM. These extensions are called the visualization and metamodel integration services. These services have many capabilities, including allowing the user to leverage existing artifacts when designing new systems with UML.

A quick overview of the metamodel integration capabilities of EMF and RSA follows. We then describe two different ways that RSA provides visualization and integration of a domain-specific

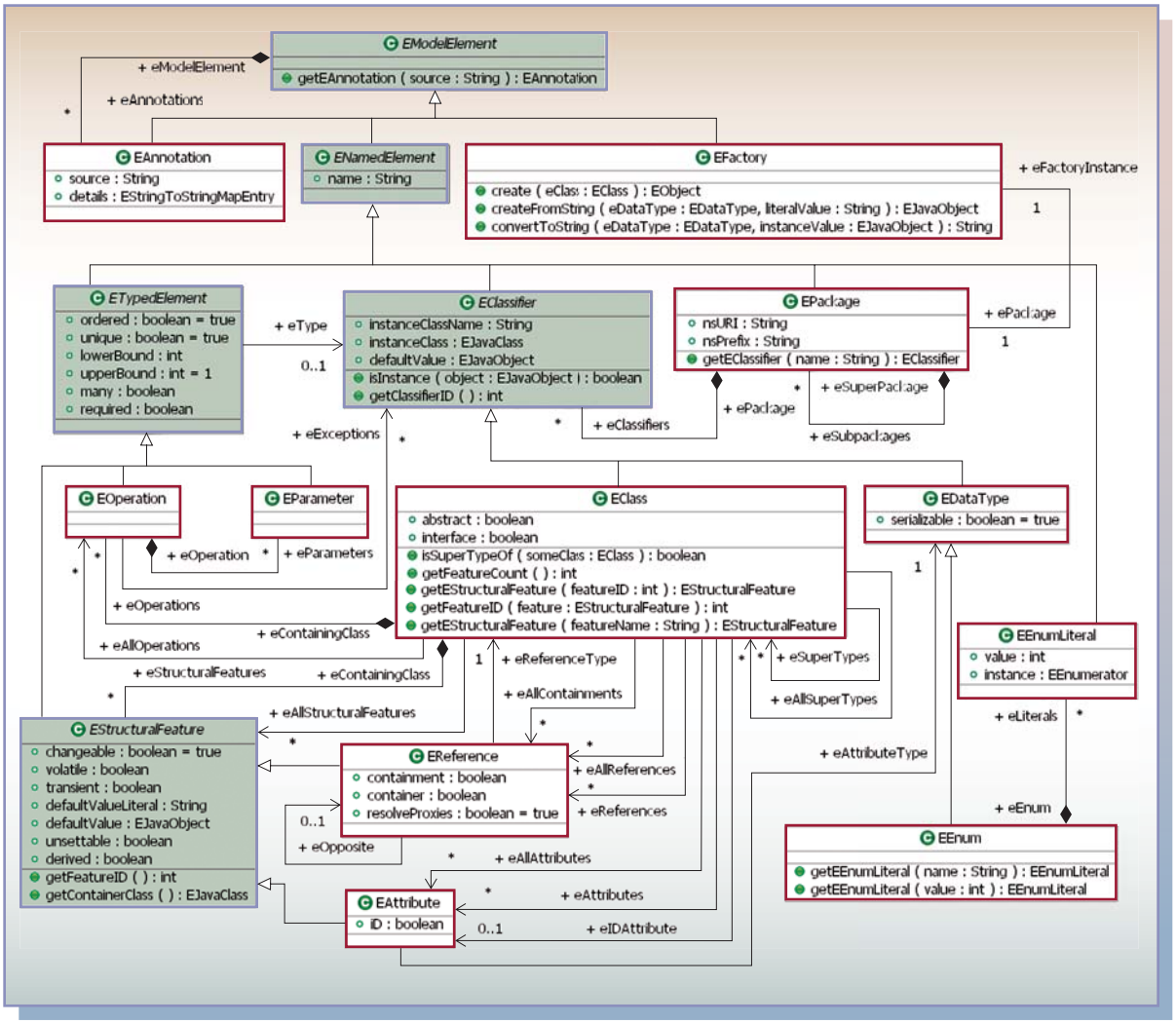

Figure 1
The Ecore metamodel, which is the basis for EMF

model. The first is through the use of the visualization technology; the second is through the use of a custom EMF resource implementation. Like all new technologies, there are areas of these solutions that still need to be improved. Adding modeling support for a new DSL in RSA 6.0 is very code-intensive and requires the use of many proprietary extension points. We also discuss new technologies that will be incorporated into future versions of RSA to further improve this integration.

OVERVIEW OF EMF

EMF, the Eclipse Modeling Framework,[10] is a key infrastructural element of many IBM products. It offers a simple, pragmatic approach to modeling and metamodeling. Based on proven technology (available since 2002), it can be used to generate code that is typically written repeatedly, allowing developers to focus on more complex aspects of the systems they are developing.

EMF consists of an underlying metamodel (called Ecore, for EMF's "core" model, as shown in ***Figure 1***) and tools for importing and generating code from source models in various formats. It includes an efficient runtime model, persistence and validation frameworks, utilities for modeling and recording changes, user-interface-independent support for viewing and editing data, and much more.

The source model for an EMF project contains a description of an application's data, including

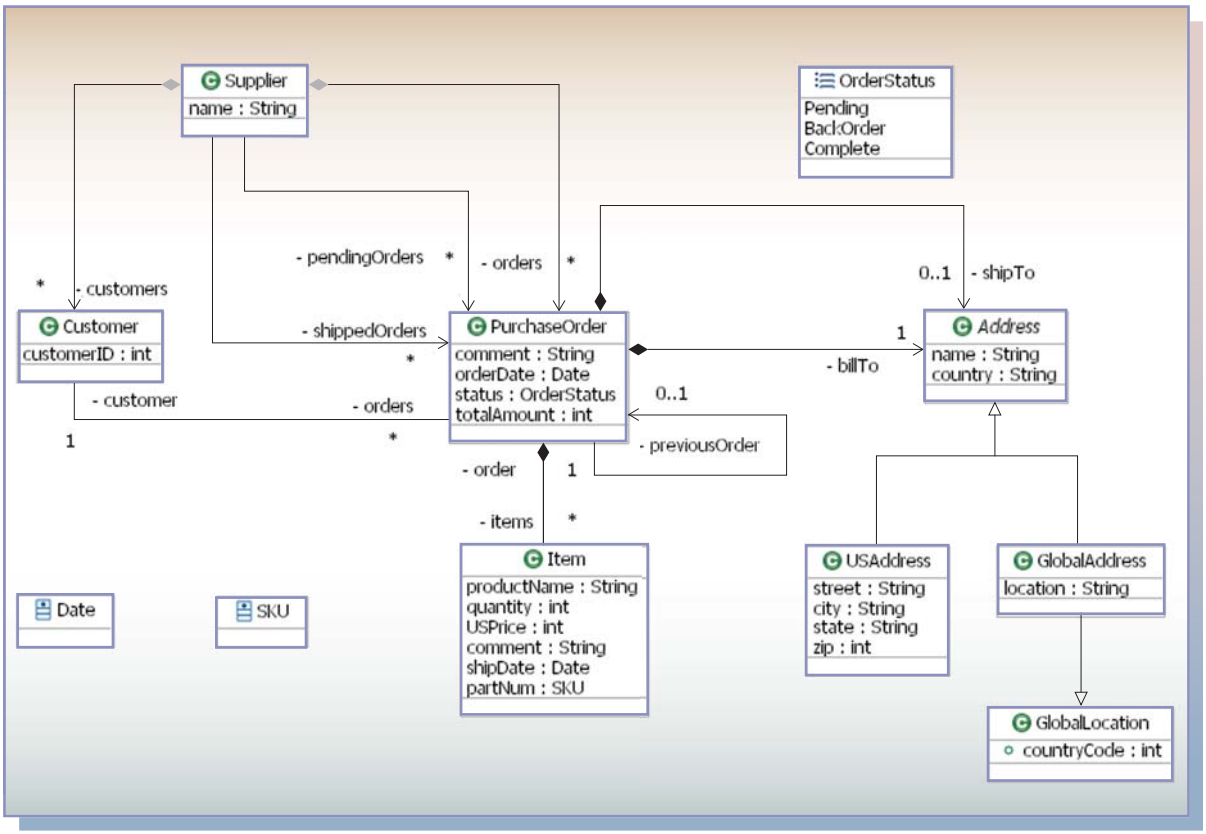

Figure 2
Sample EMF model for PurchaseOrder

objects and their attributes, relationships, operations, and constraints—essentially the basic concepts from the Meta Object Facility[11] (MOF**). It can be in any of several formats (effectively different views of the same information), including annotated Java interfaces, UML, XML Schema, or Ecore itself. Given a source model, EMF can generate Java implementation code (including user interface code), XML schemas, and Eclipse plug-in artifacts. The generated code includes efficient support for change notification, bidirectional handshaking, type-safe enumerations (i.e., those that do not permit assigning to an object a value of the wrong type), and reflection, based on the `EObject` interface. Reflection is the ability to query the metamodel for its classes and their structural features and thus navigate objects in a generic fashion.

Every EMF-generated application model is an instance of the EMF metamodel, Ecore. For example, an application model of a simple purchase order is shown in ***Figure 2***. Here, `PurchaseOrder` and `Item` would be instances of `EClass`; `shipTo`, `billTo`, and `productName` would be instances of `EAttribute`; and `items` would be an instance of `EReference`.

"Persisted data" (i.e., data made persistent) in EMF is referred to as a *resource*. EMF provides a generic, customizable XML or XML Metadata Interchange[12] (XMI**) resource implementation, but other resource implementations (e.g., those backed by a database) are possible. Data can be spread over a number of resources, each of which is identified by a Uniform Resource Identifier (URI). A collection of related resources is known as a *resource set*; EMF uses proxies to represent references between objects in different resources within a resource set. EMF uses these proxies to support on-demand loading of referenced resources.

As an example, two purchase orders, `po1` and `po2` (based on the application model above) may be saved in resources `po1.epo2` and `po2.epo2`. Purchase order `po2` references `po1` as its previous order,

so a cross-resource reference is serialized in `po2.epo2` (using URI `po1.epo2#/`). When the `po2.epo2` resource is loaded, its contents are parsed and instantiated in memory. However, the first time `po2`'s previous order is accessed, the proxy for `po1` is resolved (based on the URI `po1.epo2#/`), the resource containing it is loaded into the resource set (i.e., the contents of `po1.epo2` are parsed and instantiated in memory), and the reference between `po2` and `po1` is established, as shown in the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<epo2:PurchaseOrder xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance"
xmlns:epo2="http:///epo2.ecore"
  comment="po1">
<shipTo xsi:type="epo2:USAddress" name="John
  Doe"/>
</epo2:PurchaseOrder>

<?xml version="1.0" encoding="UTF-8"?>
<epo2:PurchaseOrder xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance"
xmlns:epo2="http:///epo2.ecore"
  comment="po2">
<shipTo xsi:type="epo2:USAddress" name="Jane
  Doe"/>
<previousOrder href="po1.epo2#/"/>
</epo2:PurchaseOrder>
```

An Ecore model of application data can also be defined at runtime (without requiring code generation) by using the Ecore API (application programming interface) or by loading it from a persistent form. A dynamic implementation of the EObject reflective API provides the same runtime behavior as that for a generated implementation. EMF treats all model instances the same, regardless of whether they are defined statically (i.e. generated) or dynamically.

OVERVIEW OF RSA METAMODEL INTEGRATION

As mentioned previously, RSA includes a set of EMF metamodels (UML2, EJBs, data, etc.). EMF provides a basic language (Ecore) and a set of services for working with instances of these models.

The visualization and metamodel-integration services component of the RSA architecture allows different domain-specific models to be visualized and integrated. The primary purpose of this component is to allow for the integration and dynamic mapping of elements from a source metamodel to a different target metamodel. This set of services allows any EMF metamodel to be mapped to, and therefore be integrated with, another EMF metamodel.

This component is required because metamodels from standard specifications such as UML are usually closed systems. UML elements can only have relationships to other UML elements; therefore, to reference non-UML elements from a UML model, one needs some way of mapping the element to a concept that UML understands. In RSA 6.0, the services provided by this component allow EJB, Java, and C++ elements to be visualized and integrated into UML models.

As an example, in ***Figure 3*** an EJB called 'Customer' can be dragged onto a UML diagram and visualized by using UML component notation. The same EJB can reference a data table called 'Customer'. This reference crosses a metamodel boundary. The data table is itself referenced from a UML2 use case called 'UpdateCustomer'. This reference also crosses a metamodel boundary. The CustomerLocal Java interface that is exposed by the EJB and a legacy C++ class called 'LegacyCPPSystem' that is also traceable to the use case are also shown in the figure. These relationships also cross metamodel boundaries.

The key concept underlying this ability is called a *visualization reference*, that is, a URI that uses the `vizref` schema and can represent a domain element or an association between elements. These references support the ability to specify a mapping and extra properties that should be applied to the target when the reference is resolved.

If a domain element is visualized on a diagram or referenced from an element from a different metamodel, all that is persisted (i.e., made persistent) is a reference to the domain element, which consists of a URI with a `vizref` schema that includes the following fields: `VizRefHandler` (its creator), target language kind (what it maps to), supporting `vizref`s, and other properties. The data stored in a

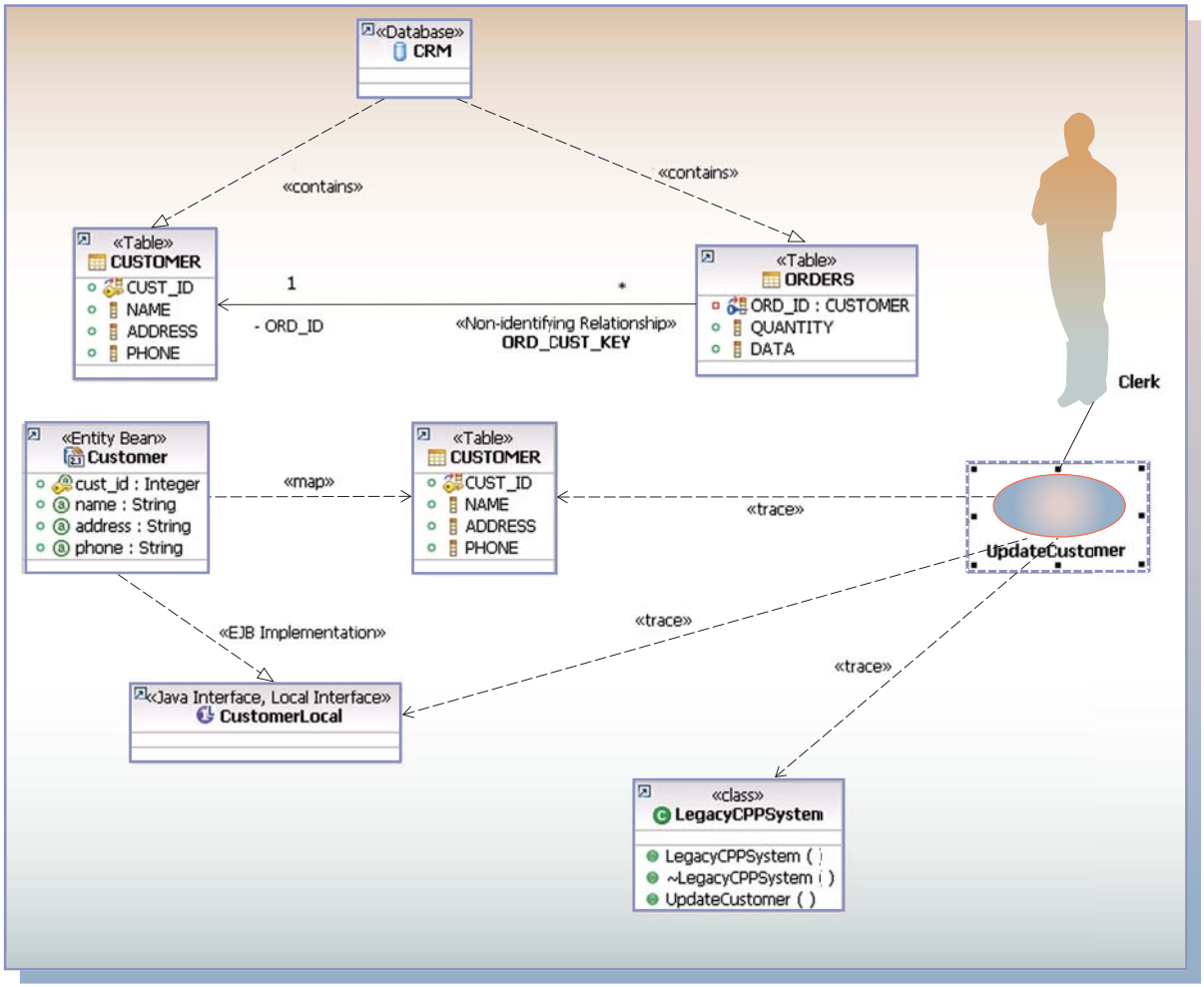

Figure 3
RSA diagram showing elements from various domain models

`vizref` is completely defined by the person who implements the domain mapping.

This sounds fairly complicated, but it actually is not. The following examples of visualization references illustrate this. In order to duplicate these examples, one simply drags various Java elements onto an RSA diagram and then opens up the `.dnx` or `.emx` file with a text editor.

In the first example, a Java class named `com.ibm.demo.JavaSubclass` in a project named `TraceDemoJava` is visualized as a UML class. We therefore have a `vizref`, with a type of `uml:Class`, containing the class name, package name, and Java file name properties. There is also a supporting `vizref` (composite) to the Eclipse project in which the Java file resides. A Java project is mapped to `uml2.Model`.

```
<element xmi:type="uml:Class"
href="vizref:///#jsrctype^vcore.target
=uml2.Class^name
  =JavaSubclass[jcu^name
    =JavaSubclass.java[jpack^name
      =com.ibm.demo[jsrcroot^srcfolder
        =[project^vcore.target
          =uml2.Model^id
            =TraceDemoJava]]]]"/>
```

In the second example, the relationship between Java classes named `JavaSubClass` and `JavaSuperclass` (an "extends" relationship) is represented as a composite `vizref` of type `uml:Generalization` containing two supporting

vizrefs that represent the Java class (JavaSubclass) and its superclass (JavaSuperclass).

```
<element xmi:type="uml:Generalization"
href="vizref:///#jgen^vcore.target
=uml2.Generalization[jsrctype^name
  =JavaSubclass[jcu^name
    =JavaSubclass.java[jpack^name
      =com.ibm.demo[jsrcroot^srcfolder
        =[project^vcore.target
          =uml2.Model^id
            =TraceDemoJava]]]]]
[jsrctype^name
  =JavaSuperclass[jcu^name
    =JavaSuperclass.java[jpack^name
      =com.ibm.demo[jsrcroot^srcfolder
      =[project^vcore.target
      =uml2.Model^id
         =TraceDemoJava]]]]]"/>
```

We use composite visualization references in order to improve performance. These references allow us to reuse reference objects in memory and thus greatly improve the performance of refactoring operations.

In light of the previous discussion, in the following section we examine how RSA leverages this technology to provide a new, better integrated workflow when dealing with a mix of general UML2 models for design- and domain-specific models for construction or implementation. The easiest way to do this is to compare the workflows of past Rational tools with those of RSA.

COMPARISON OF RATIONAL ROSE AND XDE WITH RSA

Modeling tools such as Rose and XDE are used for many different purposes. For the purpose of this paper we examine three categories of use: (1) visualizing, understanding, and documenting existing code; (2) creating new designs and converting them into implementations; and (3) forward engineering (i.e., the process of transforming UML model elements into source-code files). The first category of use is a very common use of modeling tools and is often the one first used when a developer is introduced to modeling. The second category of use occurs when the tools are used to create new designs, possibly reusing elements from existing implementations. These new designs are then converted into initial implementations. In some cases, some users may iteratively add to the model and regenerate the implementation. The third category of use occurs when the tool is used in a true forward-engineering model-driven development fashion. In this case, the model becomes the "code" for the user, and the implementation that is derived from the model becomes a derived artifact.

In the following, we review each of these three common usage categories and compare how they are performed in Rose and XDE as compared with RSA.

Visualization

One of the main reasons people want to create diagrams of an existing implementation is to either help them understand it or to help them communicate it to others. UML is useful for this. For the code segments from the AutoWorld sample project shown in ***Figure 4***, one may want to create a diagram, also shown in Figure 4, showing the inheritance relationships between the various entity beans that are found in this project. (The AutoWorld sample is available in XDE and RSA.) In order to create this diagram in XDE, it is necessary to first "reverse engineer" the code into a model by right-clicking on the AutoWorld Project in the Navigator and selecting the Reverse Engineer option. This operation creates a model called "Java 1.3 Code Model" with the content shown in the model explorer in ***Figure 5***.

As the figure shows, XDE reverse-engineers a great deal of detailed information into the UML model, and therefore, navigating through the content found in the model explorer is not as easy as navigating in the project explorer shown in Figure 4. Once the reverse-engineered version of the four desired EJBs in the XDE model has been located, a new diagram can be created, and the EJBs can be dragged onto it to create the visualization shown in Figure 5.

The XDE diagram in Figure 5 also is not quite as precise as the one in Figure 4. It lacks the key indication on the 'id' field of the Vehicle EJB, and because XDE is not aware of the domain-specific WebSphere* Application Server 5.x extension that allows EJB inheritance, it was necessary to draw the diagram by using the bean implementation classes instead of the logical EJBs.

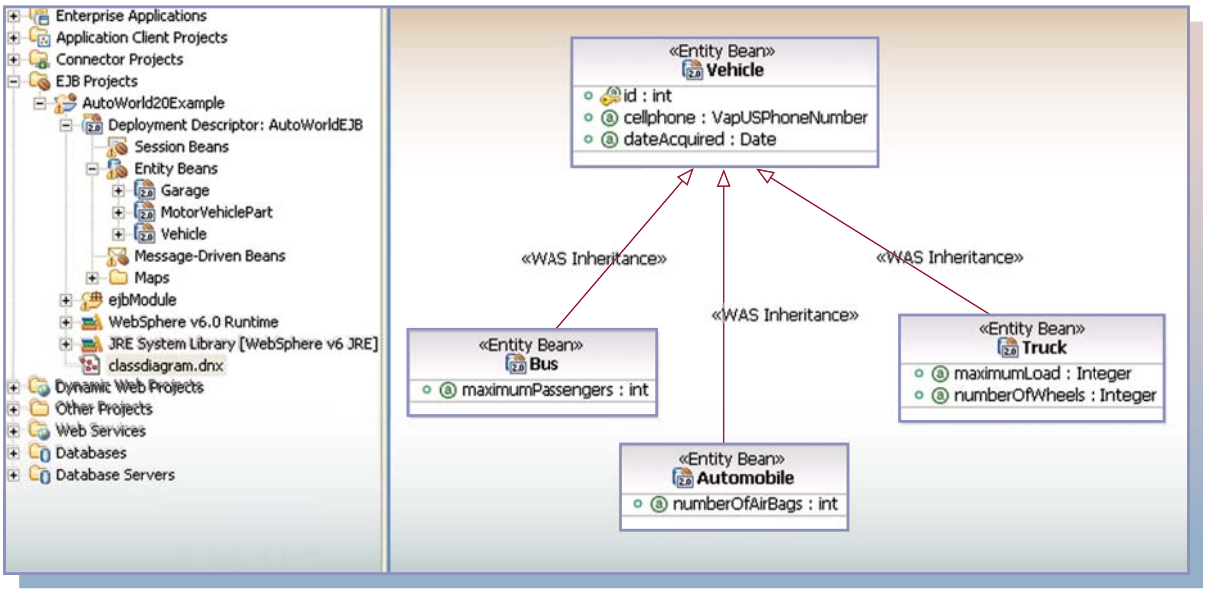

Figure 4
AutoWorld EJB project and corresponding EJB inheritance diagram

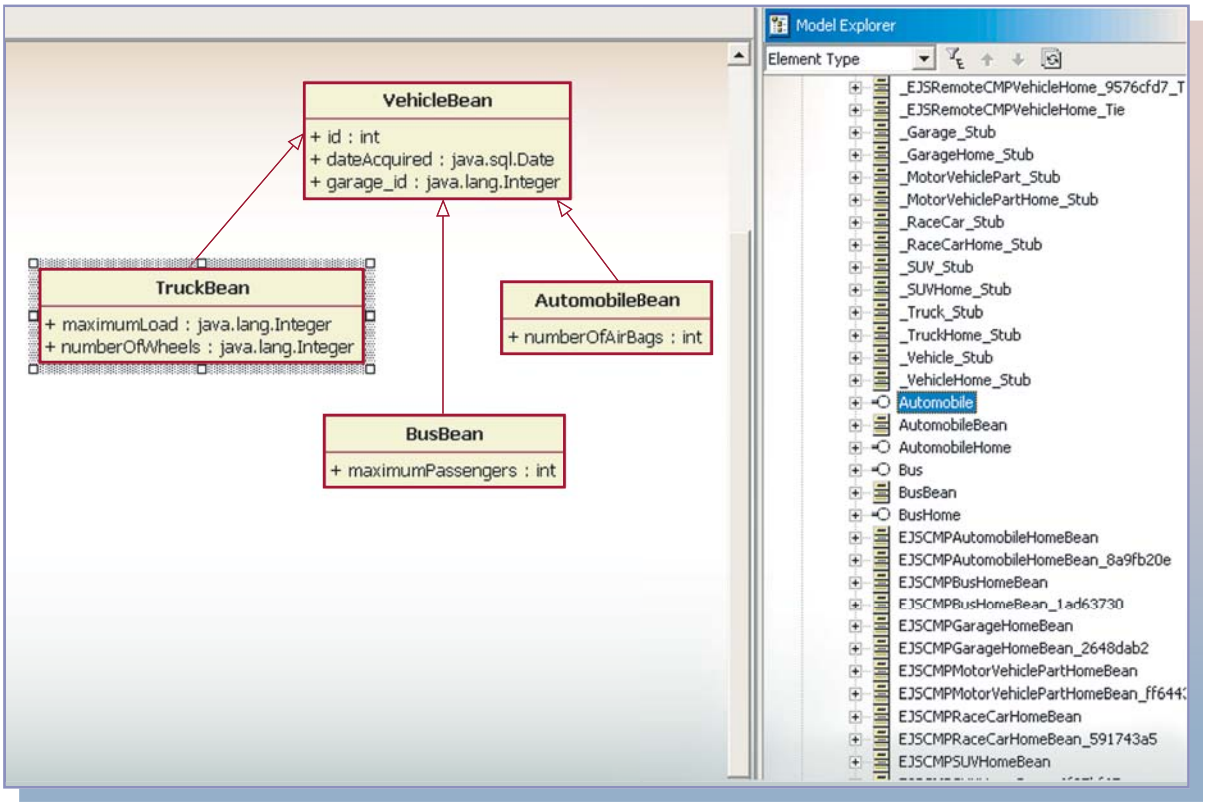

Figure 5
AutoWorld EJB code model, Model Explorer, and diagram from XDE

In comparison, to create this diagram in RSA, one simply selects the desired EJBs in the project explorer and then selects "Visualize" and "Add to New Diagram" from the context menu. The diagram in Figure 4 was captured from RSA. In RSA, it is also possible to simply use the "Browse Diagram" or "Topic Diagram" support and completely skip the diagram-creation step by selecting the EJBs in the project explorer and selecting "Visualize" and "Browse In Diagram." This functionality allows users to visualize elements in diagrams and navigate through their domain-specific and UML models without persisting the diagram itself, similar to the way one uses a Web browser.

Thus far, the differences could be attributed to "UI trickery." Examining the files that are created (the `.mdx/.mdl` file of XDE/Rose versus the `.dnx/.emx` file of RSA), however, reveals a very significant difference. In Rose and XDE, the UML objects are persisted; therefore, a redundant copy of the information is stored in the Rose and XDE model. In RSA, the UML objects are not persisted, and the views simply use the "visualization reference" mechanism that we described previously to reference the EJB or Java elements directly. As a result, the XDE file we obtained was 2.5 MB, whereas the RSA file was 25 KB.

To simulate what happens in a real team environment, we can update the source code or deployment descriptor. For example, we add a container-managed persistence (CMP) field (`fuelConsumption:int`) to `Vehicle` by using the deployment descriptor editor. In XDE, we need to select the model in the model explorer and select "Synchronize" (or "Reverse Engineer") from the context menu. The code model is synchronized, and then the diagram is updated. In RSA, the diagram is updated automatically as soon as we persist our changes into the Java source files or deployment descriptor.

In Rose and XDE, this scenario becomes much more complicated in a real team environment. If two different users are updating the source or the UML models, then, because of the duplication of information between models and source, during a merge session it is necessary to reconcile the model changes and code changes separately.

The preceding example, though simple, showed that referencing non-UML instead of reverse-engineering into UML allows RSA to integrate its notation and presentation model (UML) with underlying domain-specific technologies without duplicating information. In this particular example, RSA leveraged the existing EJB EMF models that are available in RSA to provide first-class modeling of EJB concepts without requiring the use of intermediate UML code models and profiles.

Creating new designs and reusing existing elements

Users often want to model a new system without having to fill in all the implementation details. It is also helpful to be able to use elements from an existing implementation. In the following example, we use an existing Java interface called `ExistingJavaInterface` and an existing Java class called `ExistingJavaClass` to represent elements from an existing implementation. The `NewClass` UML class represents our new UML design. The desired outcome is illustrated in the diagram shown in ***Figure 6***.

In order to use the existing Java elements in Rose and XDE, one must first reverse engineer the existing Java code into a reference UML code model. This model is shown in ***Figure 7*** as "Java 1.3 Code Model." It is also required to import a model of the Java software development kit (SDK) (`jdk_min`) for any Java types we wish to use, such as `java.lang.String`. A new UML class, "Class1", is created in a new diagram and model, the existing elements are dragged into it, and relationships are created that reference the UML version of the interface. The result is illustrated in the diagram portion of Figure 7.

In order to draw the diagram in Figure 6 with RSA, one simply drags the existing Java interface and class into a diagram from the explorer and then draws the UML implementation and aggregation relationships directly from the UML class to the Java elements. It is not necessary to reverse engineer the existing elements into a UML model.

The ability to create a UML implementation relationship that references a Java interface is again accomplished by using a `vizref`, as shown in:

```
<implementation xmi:id="_le2AbZW50w"
    client="_hVc0d8AbZW50w">
  <supplier xmi:type="uml:Interface"
```

Figure 6
RSA diagram mixing UML and Java elements

```
  href="vizref:///#jsrctype^vcore.target
    =uml2.Interface^name
     =ExistingJavaInterface[jcu^name
       =ExistingJavaInterface.java[jpack^name
         =com.ibm.existingsystem
             [jsrcroot^srcfolder
           =[project^vcore.target
           =uml2.Model^id
     =ArticleExamples]]]]"/>
  <mapping xmi:id="_le2iOS64EdqhOd8AbZW50w"/>
</implementation>
```

If the new design in XDE or Rose is satisfactory, the new class can be forward engineered into a Java implementation. If one wishes to retain the UML model, it is necessary to keep this model in sync with the code from this point on, as we described in the previous section.

In RSA, the UML-to-Java transformation can be used to generate an implementation for the new class. (In RSA and other new Rational products, such as Rational Software Modeler and Rational Systems Developer, the term "transformation" is used for a

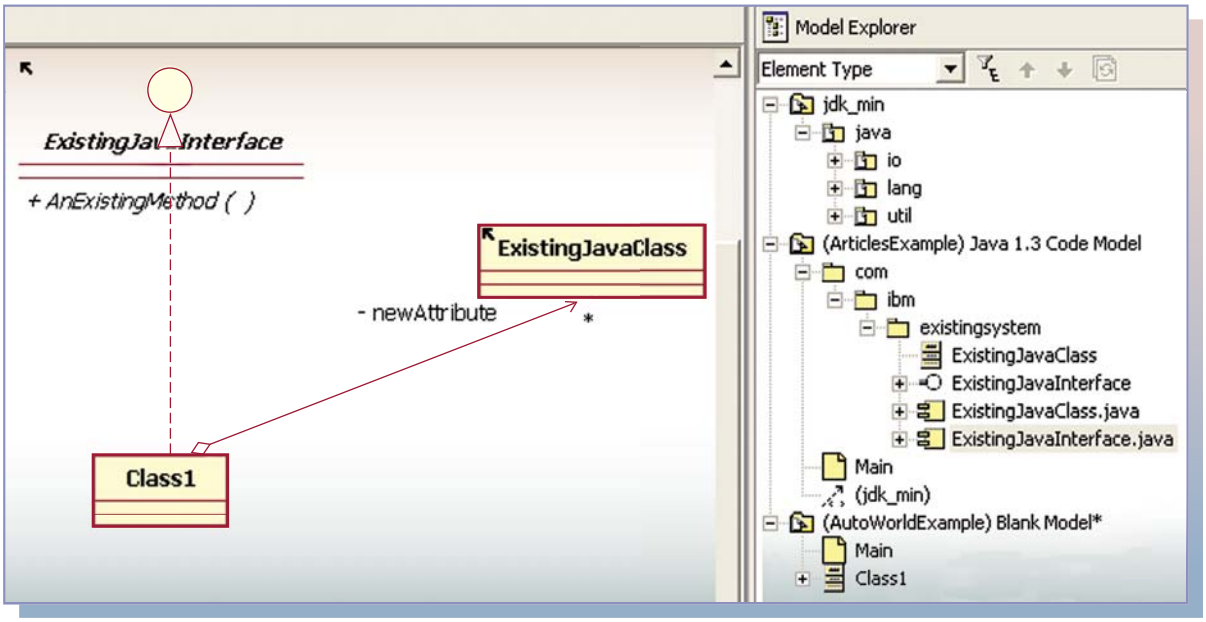

Figure 7
XDE Diagram and Model Explorer showing links between the reverse-engineered existing code and the new model

process that transforms elements of one model into elements of another model.) If the "Replace UML Elements" option is selected as part of the transformation configuration, the reference on the diagram for "NewClass" will be updated to reference the newly created Java class. In this way, the diagrams always stay in sync with the code. As part of the transformation, the diagram is updated as shown in *Figure 8* to include a direct reference to the new Java class.

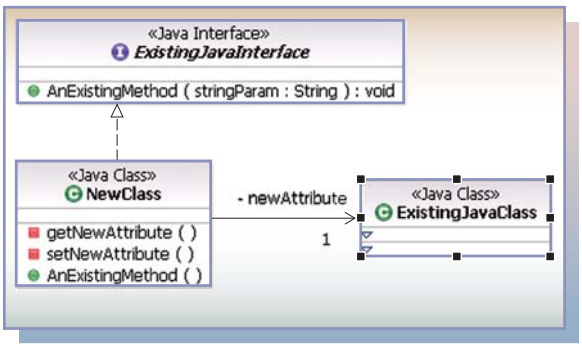

Figure 8
RSA post-transformation model (using replacement elements)

Forward engineering using the UML model

A user with a forward-engineering model-driven development approach treats the UML model as the only persistent artifact in the process. All other artifacts are considered derived from the UML model. The model is transformed to the target domain and executed.

Although RSA introduces improved support for UML patterns and transformations to help users who take this approach, there is no significant difference between the capabilities and usage of RSA versus the capabilities and usage of Rose and XDE. One still models in UML and then does one-way transformations into the target domain.

In some of these cases, users are creating large UML profiles and storing a great deal of extra information in models in order to avoid having to reconcile changes between external artifacts and the UML models. In the section "Future of RSA," we discuss some new options for creating a custom domain-specific modeler as an option for replacing large profiles on UML models.

WebSphere Business Modeler integration example

The second main example that we examine is the Websphere Business Modeler (WBM)-to-RSA integration that is now available. WBM models are stored in a format called Business Object Model (BOM). BOM is similar to parts of UML2 but has significant differences and enhancements to address the needs of business process modelers.

In the past, integration between IBM business modeling tools and IBM software modeling tools was one way in nature. The business modeler exported the model to UML, and then, the UML modeler simply imported the model into the UML tool. No traceability was maintained, and keeping the models in sync was almost impossible. For example, if one wanted to use a WBM 5.x model and XDE together, it was necessary to export to UML from WBM and then import into XDE.

In RSA, the WBM models can simply be opened. Once the model is opened as a read-only UML model, links can be created to elements in this model from other UML models, enabling the business model to be used as a "contract model" for developing a new system or a set of services. If the WBM model is updated, changes are immediately reflected in the UML version, thereby removing the duplication and traceability issues that existed in previous integrations. This functionality allows the user to create models that have elements from both the BOM metamodel and the UML metamodel, providing the same expressiveness as previous tools without their issues.

How does this work? RSA registers a custom EMF resource implementation that loads a WBM resource and dynamically maps its contents to UML elements. Proxies from UML elements (in other models) to WBM elements, when resolved, cause the correct WBM resources to be loaded. For example, the XMI segment in *Figure 9* shows a portion of a persisted UML model that contains a class named "Class1" which has an implementation dependency on an interface from another (WBM) model. The reference is persisted as a URI comprised of the resource name (`resources.XMI`) and the unique identifier of the element in that resource (`BLM-df04d4826b07b524c236754b558965ac`).

The XMI segment in *Figure 10* shows a portion of the WBM model that contains the element refer-

```
<ownedMember xmi:type="uml:Class" xmi:id="_bV9UMDBNEdq_sa2-mTt1NA" name="Class1"
                                    clientDependency="_dd3b4DBNEdq_sa2-mTt1NA">

 <implementation xmi:id="_dd3b4DBNEdq_sa2-mTt1NA"
                                                client="_bV9UMDBNEdq_sa2-mTt1NA">

   <supplier xmi:type="uml:Interface"
                   href="resources.XMI#BLM-df04d4826b07b524c236754b558965ac"/>

   <mapping xmi:id="_dd9igDBNEdq_sa2-mTt1NA"/>

   <realizingClassifier xmi:type="uml:Interface"
                   href="resources.XMI#BLM-df04d4826b07b524c236754b558965ac"/>

   <contract href="resources.XMI#BLM-df04d4826b07b524c236754b558965ac"/>

 </implementation>

 <ownedOperation xmi:id="_jl6YMDBNEdq_sa2-mTt1NA" name="Task"/>

</ownedMember>
```

Figure 9
XMI segment illustrating a UML model with a dependency on a WBM model element

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:com.ibm.btools.bom.model.resources="http:///com/ibm/btools/
bom/model/resources.ecore"
xmlns:com.ibm.btools.model.resourcemanager.versioncontrol="http:/
//com/ibm/btools/model/resourcemanager/versioncontrol.ecore">
  <com.ibm.btools.model.resourcemanager.versioncontrol:Version
versionID="5.1.1.0"/>
  <com.ibm.btools.bom.model.resources:Role xmi:id="BLM-
df04d4826b07b524c236754b558965ac" uid="BLM-
df04d4826b07b524c236754b558965ac" name="Role1">
    <ownedComment xmi:id="BLM-4159976408b85db39a867b8fb99c3eca"
uid="BLM-4159976408b85db39a867b8fb99c3eca" body=""/>
    <ownedComment xmi:id="BLM-5a9576deb9cfa051bcc093f4730589c7"
uid="BLM-5a9576deb9cfa051bcc093f4730589c7" body=""/>
    <owningPackage href="RID-
00000000000000000000000000000011.xmi#FID-
00000000000000000000000000000011"/>
  </com.ibm.btools.bom.model.resources:Role>
</xmi:XMI>
```

Figure 10
XMI segment containing a referenced element in a WBM model

enced by the class in the UML model—a role named "Role1" with an identifier of `BLM-df04d4826b07b524c236754b558965ac`.

When the UML resource is loaded, its contents are created in memory, including Class1, and a proxy is created for the reference to the role in the WBM model. The first time an attempt is made to access the implemented interface, the proxy is resolved: the resource containing the referenced element (`resources.XMI`) is added to the resource set, the custom resource implementation is used to convert its contents to UML elements (the WBM role is converted to a UML interface), and

the proxy is replaced with a reference to the element with the referenced identifier (i.e., the interface).

FUTURE WORK ON RSA

IBM Rational is continuing to develop and enhance the capabilities described in this paper for future versions of RSA. One of the key technologies that will make it easier to add new DSL support to RSA is currently being developed in open source on eclipse.org under the Eclipse Graphical Modeling Framework (GMF) project.

Eclipse GMF project

The Eclipse GMF project, as described in the GMF tutorial,[13] has the goal of providing an open domain-specific graphical modeling toolkit. Tool developers will be able to use this toolkit to design and generate a custom graphical modeling tool from an EMF model of a domain. The toolkit will also be usable to provide extended or custom notation support for existing metamodels such as UML.

The GMF project consists of two major components. The first is a runtime component that helps provide a common platform for developing graphical DSM tools that are extensible and integrated with one another. The second is a set of tools that will make it easy to create a set of graphical figures and map them to a domain model expressed in EMF.

The IBM RSA development team is a key contributor to the runtime components section of the GMF project. Many of the components underlying RSA 6.0 were donated to the project in order to seed its capabilities and to help ensure that any modeling editor generated with the GMF tools or built directly on the GMF runtime will be consistent with RSA's existing graphical editing capabilities and can be easily integrated with other GMF-based modeling tools.

RSA enhancements for lightweight metamodel integration

Although the GMF project will go a long way toward simplifying the process of building domain-specific editors by providing the ability to define visualizations without requiring the entire domain to be mapped to UML, there are still many circumstances when the ability to integrate various metamodels is desired. For example, in the Java modeling example earlier, one might have built a custom DSM editor for Java instead of using the visualization and metamodel integration services from RSA. Although using GMF would allow Java interfaces to be drawn or visualized on diagrams, it would not have enabled support of the use case where a new UML class extends an existing Java interface. In other words, relationships or references across elements from different metamodels could not be created.

In order to address this, the RSA team is working on a more lightweight version of visualization and metamodel integration services that will allow models to be integrated without having to share common visualizations.

CONCLUSION

As the examples in this paper have demonstrated, the RSA 6.0 product shows significant progress by IBM Rational in supporting the ability to model in different DSLs. This allows users familiar with a domain to quickly obtain value from visualizations based on these models.

In the section "Future work on RSA," we briefly discussed some new technologies that will make it easier for tool developers, partners, and advanced customers to develop their own integrated DSM capabilities.

Several examples, such as the one displayed in Figure 3, illustrate another key focus in RSA, namely, the integration of DSM capabilities with general UML modeling and transformations. Because of this integration, RSA allows the user to enhance domain-specific diagrams by adding links to use cases, interactions, and collaborations. This capability increases and supplements the expressive capabilities of the tool and leverages UML and RSA's rich capabilities, without having to reproduce them in each DSL.

CITED REFERENCES AND NOTE

1. *Unified Modeling Language (UML), Version 1.4.2, ISO/IEC 19501*, International Organization for Standardization, (April 13, 2005), http://www.iso.org/iso/en/

CatalogueDetailPage.CatalogueDetail?CSNUMBER=32620.

2. *Unified Modeling Language (UML): Superstructure Specification, Version 2.0*, Object Management Group, Inc. (May 7, 2004), http://www.omg.org/cgi-bin/doc?formal/05-07-04.
3. For a list of articles and publications on domain-specific modeling, see http://www.dsmforum.org/publications.html.
4. J. Greenfield, K. Short, S. Cook, S. Kent, and J. Crupie, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, Wiley Publishing, Inc., Hoboken, NJ (2004).
5. P. P. Chen, "The Entity-Relationship Model—Toward a Unified View of Data," *ACM Transactions on Database Systems (TODS)* **1**, No. 1, 9–36 (1976), http://bit.csc.lsu.edu/~chen/pdf/erd.pdf.
6. *Extensible Markup Language (XML) 1.0 (Third Edition)*, World Wide Web Consortium (W3C) Recommendation (February 4, 2004), http://www.w3.org/TR/REC-xml/.
7. *XML Schema Part 0: Primer Second Edition*, World Wide Web Consortium (W3C) Recommendation (October 28, 2004), http://www.w3.org/TR/xmlschema-0/.
8. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, *Web Services Description Language (WSDL) 1.1*, World Wide Web Consortium (W3C) Note (March 15, 2001), http://www.w3.org/TR/wsdl.
9. K. Hussey, *Getting Started with UML2*, eclipse.org (August 4, 2005), http://www.eclipse.org/uml2.
10. F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. J. Grose, *Eclipse Modeling Framework*, Addison-Wesley, Reading, MA (2004).
11. *Meta-Object Facility (MOF) 2.0 Core Specification*, Object Management Group, Inc. (March 10, 2004), http://www.omg.org/cgi-bin/doc?ptc/03-10-04.
12. *XML Metadata Interchange Specification, Version 2.0.1*, Object Management Group, Inc. (May 5, 2006), http://www.omg.org/docs/formal/05-05-06.pdf.
13. R. C. Gronback, *GMF Tutorial*, eclipse.org (January 22, 2006), http://wiki.eclipse.org/index.php/GMF_Tutorial.

Accepted for publication February 16, 2006.
Published online July 11, 2006.

Daniel Leroux
IBM Software Group, Rational, 770 Palladium Drive, Ottawa, Ontario, Canada, K2V 1C8 (dleroux@ca.ibm.com). Mr. Leroux is a Senior Technical Staff Member and senior development manager with IBM Rational Software. He has been with IBM Rational Software for eight years and has held various management and development roles for the Rational modeling family of products. Over the last four years, he has led the architecture and development of the Rational Software Architect/Modeler product line.

Martin Nally
IBM Software Group, Rational, 1090 Katella St, Laguna Beach, CA 92651 (nally@us.ibm.com). Mr. Nally is an IBM Distinguished Engineer who joined IBM in 1990 with 10 years' prior industry experience. He was the lead architect and developer for IBM VisualAge/Smalltalk and lead architect and overall development manager for IBM WebSphere Studio. His current title is Chief Technical Officer, IBM Rational Software.

Kenneth Hussey
IBM Software Group, Rational, 770 Palladium Drive, Ottawa, Ontario, Canada, K2V 1C8 (khussey@ca.ibm.com). Mr. Hussey is a senior software developer for IBM Rational Software. He is a committer (i.e., a developer with write access to the source code repository) on the EMF project and lead of the UML2 project, both of which are open-source tool subprojects at Eclipse. ■