THE OMG'S UNIFIED MODELING LANGUAGE™ (UML™) ELEMENTS

Cezar Botezatu, Ph.D., Cornelia Botezatu, Ph.D.
Romanian-American University, Bucharest, Romania

In the beginning of the 90s two approaches came up, which should unify the communication and the integration of software components. Microsoft® introduced the Component Object Model (COM) and the Object Management Group initiated with the Common Object Request Broker Architecture (CORBA®) a quite similar model.

CORBA, or Common Object Request Broker Architecture, is a standard architecture for distributed object systems. It allows a distributed, heterogeneous collection of objects to interoperate.

The Object Management Group (OMG) is responsible for defining CORBA. The OMG comprises over 700 companies and organizations, including almost all the major vendors and developers of distributed object technology, including platform, database, and application vendors as well as software tool and corporate developers.

CORBA products provide a framework for the development and execution of *distributed applications*. Distribution introduces a whole new set of difficult issues. However, sometimes there is no choice; some applications by their very nature are distributed across multiple computers because of one or more of the following reasons:

- The *data* used by the application are distributed
- The *computation* is distributed
- The *users* of the application are distributed

CORBA defines an architecture for distributed objects. The basic CORBA paradigm is that of a request for services of a distributed object. Everything else defined by the OMG is in terms of this basic paradigm.

The services that an object provides are given by its *interface*. Interfaces are defined in OMG's Interface Definition Language (IDL). Distributed objects are identified by object references, which are typed by IDL interfaces.

The ORB (Object Request Broker)

The ORB is the distributed service that implements the request to the remote object. It locates the remote object on the network, communicates the request to the object, waits for the results and when available communicates those results back to the client.

The ORB implements location transparency. Exactly the same request mechanism is used by the client and the CORBA object regardless of where the object is located. It might be in the same process with the client, down the hall or across the planet. The client cannot tell the difference.

The ORB implements programming language independence for the request. The client issuing the request can be written in a different programming language from the implementation of the CORBA object. The ORB does the necessary translation between programming languages. Language bindings are defined for all popular programming languages.

Information about OMG's specifications:

1) Model Driven Arhitecture ™;
2) Unified Modeling Language ™;
3) Common Warehouse Metamodel ™;
4) XML ™ Metadata Interchange (XMI ™);
5) Common Object Request Broker Architecture (CORBA®)

Unified Modeling Language is not only a de facto modeling language standard. It is fast becoming a de jure standard. Nearly two years ago the Object Management Group (OMG) adopted UML as its standard modeling language. As an approved Publicly Available Specification (PAS) submitter to the International Organization for Standardization (ISO), the OMG is proposing the UML specification for international standardization. The major benefits of international standardization for a specification include wide recognition and acceptance, which typically enlarge the market for products based on it. However, these benefits often demand a high price. Standardization processes are typically formal and protracted, seeking to accommodate a diverse range of technical and business requirements. From a business perspective, the timescales of standards usually conflict with the competitive need to use the latest technology as early as possible. From a technical perspective, the need to achieve consensus encourages "design by committee" processes. In this sort of environment, sound technical tradeoffs are often overridden by inferior political compromises. Too frequently the resulting specifications become bloated with patches in a manner similar to the way laws become fattened with riders in "pork belly" legislation.

It assumes the reader is generally familiar with the use of UML, and instead focuses on the language's recent and future evolution. The processes and architectures for UML change management are examined.

Although the infrastructure and most of the superstructure of the language were sound, several significant problems were known at the time of the final submission:

- *Incomplete semantics and notation for activity graphs.* Activity graph semantics, which were added relatively later in the process, were not fully integrated with the state machine semantics on which they depended. In addition, some notation conveniences required by business modelers were missing.

- *Standard elements bloat.* The language specification included many standard elements (stereotypes, tagged values, and constraints) that were hastily added to address the requirements of various competing methods groups. Many of these standard elements had sparse semantics and were inconsistently named and organized.
- *Architectural misalignment.* The submitters fell short of their goal of implementing a 4-layer metamodel architecture using a strict metamodeling approach.

Instead they settled for the pragmatic, but less rigorous, loose (non-strict) metamodeling approach. This adversely affected the integration of UML with other OMG modeling standards, such as the Meta Object Facility (MOF) .

Rather than delay the standardization of UML, the submitters resolved to address some of these problems in the next revision of the language.

***The OMG's Unified Modeling Language*™ (UML™)** helps you specify, visualize, and document models of software systems, including their structure and design, in a way that meets all of these requirements. (You can use UML for business modeling and modeling of other non-software systems too.) Using any one of the large number of UML-based tools on the market, you can analyze your future application's requirements and design a solution that meets them, representing the results using UML's twelve standard diagram types.

You can model just about any type of application, running on any type and combination of hardware, operating system, programming language, and network, in UML. Its flexibility lets you model distributed applications that use just about any middleware on the market. Built upon the MOF™ metamodel which defines *class* and *operation* as fundamental concepts, it's a natural fit for object-oriented languages and environments such as C++, Java or C++, but you can use it to model non-OO applications as well in, for example, Fortran, VB, or COBOL. *UML Profiles* (that is, subsets of UML tailored for specific purposes) help you model Transactional, Real-time, and Fault-Tolerant systems in a natural way.

You can do other useful things with UML too: For example, some tools analyze existing source code (or, some claim, object code!) and reverse-engineer it into a set of UML diagrams. Another example: In spite of UML's focus on design rather than execution, some tools on the market *execute* UML models, typically in one of two ways: Some tools execute your model interpretively in a way that lets you confirm that it really does what you want, but without the scalability and speed that you'll need in your deployed application. Other tools (typically designed to work only within a restricted application domain such as telecommunications or finance) generate program language code from UML, producing most of a bug-free, deployable application that runs quickly if the code generator incorporates best-practice scalable patterns for, e.g., transactional database operations or other common program tasks. Our final entry in this category: A number of tools on the market generate Test and Verification Suites from UML models.

UML and OMG's Model Driven Architecture™ (MDA™): A few years ago, the biggest problem a developer faced when starting a distributed programming project was finding a middleware with the functionality that he needed, that ran on the hardware and operating systems running in his shop. Today, faced with an embarrassingly rich array of middleware platforms, the developer has three different middleware problems: First, selecting one; second, getting it to work with the other platforms already deployed not only in his own shop, but also those of his customers and suppliers; and third, interfacing to (or, worse yet, migrating to) a new "Next Best Thing" when a new platform comes along and catches the fancy of the analysts and, necessarily, CIOs everywhere.

Models vs. Methodologies: The process of gathering and analyzing an application's requirements, and incorporating them into a program design, is a complex one and the industry currently supports many *methodologies* that define formal procedures specifying how to go about it. One characteristic of UML - in fact, the one that enables the widespread industry support that the language enjoys - is that it is *methodology-independent*. Regardless of the methodology that you use to perform your analysis and design, you can use UML to express the results. And, using XMI (XML Metadata Interchange, another OMG standard), you can transfer your UML model from one tool into a repository, or into another tool for refinement or the next step in your chosen development process. These are the benefits of standardization!

UML defines twelve types of diagrams, divided into three categories: Four diagram types represent static application structure; five represent different aspects of dynamic behavior; and three represent ways you can organize and manage your application modules.

Structural Diagrams include the Class Diagram, Object Diagram, Component Diagram, and Deployment Diagram.

Behavior Diagrams include the Use Case Diagram (used by some methodologies during requirements gathering); Sequence Diagram, Activity Diagram, Collaboration Diagram, and Statechart Diagram.

Model Management Diagrams include Packages, Subsystems, and Models.

I'm About to Start my First UML-based Development Project. What do I Need to do? Three things, probably (but not necessarily) in this order:

1) Select a methodology: A methodology formally defines the process that you use to gather requirements, analyze them, and design an application that meets them in every way. There are many methodologies, each differing in some way or ways from the others. There are many reasons why one methodology may be better than another for your particular project: For example, some are better suited for large enterprise applications while others are built to design small embedded or safety-critical systems. On another axis, some methods better support large numbers of architects and designers working on the same project, while others work better when used by one person or a small group.

2) Select a UML Development Tool: Because most (although not all) UML-based tools implement a particular methodology, in some cases it might not be practical to pick a tool and then try to use it with a methodology that it wasn't built for. (For other tool/methodology combinations, this might not be an issue, or might be easy to work around.) But, some methodologies have been implemented on multiple tools so this is not strictly a one-choice environment.

You may find a tool so well-suited to your application or organization that you're willing to switch methodologies in order to use it. If that's the case, go ahead - our advice to pick a methodology first is general, and may not apply to a specific project. Another possibility: You may find a methodology that you like, which isn't implemented in a tool that fits your project size, or your budget, so you have to switch. If either of these cases happens to you, try to pick an alternative methodology that doesn't differ too much from the one you preferred originally.

3) Get Training: You and your staff (unless you're lucky enough to hire UML-experienced architects) will need training in UML. It's best to get training that teaches how to use your chosen tool with your chosen methodology, typically provided by either the tool supplier or methologist.

Two features add to the expressiveness of UML. Object Constraint Language (OCL) has been part of UML since its beginning, while the Action Semantics extension is a recent addition:

Object Constraint Language lets you express conditions on an invocation in a formally defined way. You can specify invariants, preconditions, postconditions, whether an object reference is allowed to be null, and some other restrictions using OCL. As you might expect, the MDA relies on OCL to add a necessary level of detail to PIMs and PSMs.

Action Semantics UML Extensions let you express actions as UML objects. An Action object may take a set of inputs and transform it into a set of outputs (although one or both sets may be empty), or may change the state of the system, or both. Actions may be chained, with one Action's outputs being another Action's inputs. Actions are assumed to occur independently - that is, there is infinite concurrency in the system, unless you chain them or specify this in another way. This concurrency model is a natural fit to the distributed execution environment of modern enterprise and Internet applications.

UML 2.0 - A Major Upgrade: In mid-2001, OMG members started work on a major upgrade to UML 2.0.

March, 2003: over the last four years, development tool suppliers have devoted a great amount of attention to two key design standards: UML (Unified Modeling Language) and MDA (Model-Driven Architecture). These standards are the spearheads of a supplier effort to increase the importance of design tools and design in users' development processes. Two key barriers to the success of this effort are the enormous gap between high-level design standards and low-level, real-world Java coding, and strong developer resistance to the "formalism," or over-tight control, that design tools tend to impose on their users. Nevertheless, Aberdeen believes that the new design-driven development can bring major benefits in programmer productivity, software quality, and flexibility, but only if implemented carefully - and not imposed in inappropriate situations.

Finally, UML 2.0 SEEMS Final

March 15, 2003 — **After three years of work, including gathering requirements and building consensus, the Analysis and Design Platform Task Force of Object Management Group Inc. is set to finalize version 2.0 of the Unified Modeling Language specification that is expected to lead to the creation of a new generation of tools that bring together modeling and requirements management within a component-based development environment.**

The task force is scheduled to vote at an OMG technical meeting March 24 in Orlando, Fla., to recommend the submission by a group called U2 Partners, which has the broadest support behind it. While other submissions remain active, task force co-chairman Cris Kobryn of Telelogic Inc. said there is little activity going on with the other submissions.

“We’re moving toward the endgame,” Kobryn said. “It’s taken longer than any of us would like, but we have broad-based support” within OMG’s membership. Kobryn explained that much of the delay has involved getting UML in line with other OMG specifications.

Jon Siegel, OMG’s vice president of technology transfer, said, “The upcoming vote on UML 2.0 demonstrates that the specification has taken its final form, and this adoption effort has reached a successful conclusion. UML 2.0 will soon be implemented in tools from not only the 35 companies participating in the adoption effort, but also from any other company—OMG member or not—that downloads the final specification from the OMG Web site once the adoption process completes.”

UML 2.0 furthers the cause of component-based development by adding support for business processes and workflows, and is more tightly joined to OMG architectural specifications, Kobryn said. A key new element of the language is the ability to decompose structures at a hierarchical level to an arbitrary level of complexity, making it easier for developers to manage the complexity of an application, he explained.

“In UML 1, with simple classes and components, there was no way to drill down into Sequence diagrams,” Kobryn said. “Now, with the integration between structure and behavior, you could right-click on a Use Case diagram to get a Sequence diagram, which provides the behavioral detail for the Structure diagrams. You can decompose complex behaviors and flexibly integrate them” into new applications.

Bringing new tools into the hands of developers, where there has been some resistance to modeling, should help UML tool vendors gain market share and mind share, according to Tim Sloane of Aberdeen Group Inc., a research and analysis firm.

"UML 2 is a huge step forward," Sloane said. "It's moving modeling forward by enabling it to be expanded to a broader audience."

Sloane, however, said it is "harder to discern" if UML narrows the business-IT gap he believes is critical for modeling to become pervasive throughout corporate development shops and smaller organizations. "That's where the character of OMG being architects and software development suppliers comes to the fore. There are not many people in OMG who are business modeling people, who have thought this through to represent business models."

Looking ahead, Telelogic's Kobryn can envision vendors bringing out tools to move requirements from their basis in text into a visual environment. "The language is timely," he said. "We're catching up with component-based development, J2EE, .NET and the convergence of the IDE and modeling tools."

INFRASTRUCTURE SPECS

In addition, OMG task forces are expected to recommend as standards a mapping from Web Services Description Language (WSDL) and SOAP to CORBA, and a way to represent enterprise collaboration as a Web service, according to OMG. The group already has agreed upon the reverse CORBA to SOAP and WSDL mapping.

Other standards expected to be acted on include a connection between CORBA Notification and Java Message Service, a way to deploy and distribute CORBA components, and data distribution in real-time systems.

The Business Enterprise Integration task force will review proposals to standardize interfaces to business processes such as Web services, workflow and e-commerce within OMG's Model Driven Architecture, OMG announced.

The Model for UML 2.0

- Alignment of the language metamodel with the MetaObject Facility (MOF) metamodel will simplify model interchange via XML Metadata Interchange (XMI) and cross-tool interoperability.
- An extension mechanism will allow modelers to add their own metaclasses, making it easier to define new UML Profiles and to extend modeling to new application areas.
- Built-in support for component-based development will ease modeling of applications realized in Enterprise JavaBeans, CORBA components or COM+.
- Support for runtime architectures will allow modeling of object and data flow among different parts of a system. Support for executable models will be improved in general.
- More accurate and precise representation of relationships will improve modeling of inheritance, composition and aggregation, and state machines.
- Better behavioral modeling will improve support for encapsulation and scalability, remove restrictions on mapping of activity graphs to state machines, and improve Sequence diagram structure.
- Overall improvements to the language will simplify syntax and semantics, and better organize its overall structure.

SOURCE:

Object Management Group Inc.

1. Finally, UML 2.0 Seems Final March 15, 2003, SD Times by David Rubinstein
2. Principles of SOA March 2003, Application Development Trends, Jason Bloomberg
   Rational's Devlin Eyes Tight IBM Ties March 3, 2003, eWeek, Darryl K. Taft
3. Sun, Microsoft Officials Face off on Java vs. .Net Debate January 30, 2003, Infoworld, Paul Krill
4. Enterprise Application Integration: The Problem that Won't Go Away January 29, 2003, DevX, Glen Kunene
5. MDA Driving App Designs January 20, 2003, eWeek, Darryl K. Taft
6. Driving Toward Better Software January 1, 2003, SD Times, David Rubinstein

Надійшла до редакції 23 березня 2004р.