

UML without Pictures

Conrad Bock, *US National Institute of Standards and Technology*

A lesser-known fact about UML (Unified Modeling Language) is that it defines a repository supporting both graphical and textual notations, as well as compilers for emitting code to multiple platforms. A repository-centered architecture enhances communication between users of different notations for UML concepts by providing a standard, centralized store for these concepts. Because the repository focuses on the meaning behind various notations, it also facilitates the

construction of highly directable model compilers, test-case generators, and consistency checkers.

The UML repository

UML's graphical syntax is the tip of an iceberg. UML also defines a repository for capturing models that is separate from both visual presentation and target implementations, as Figure 1 shows. Users can employ different UML notations, including textual interfaces,¹⁻⁴ and store the common meaning in the repository. Many development environments

are based on text files, and textual presentations for UML are currently used to create production systems. Textual formats coexist with graphical presentations in other languages⁵ and can do so in UML.

Equally important, the UML repository disambiguates meaning for notations that happen to be the same and enables consistency checking between those meanings. For example, arrows have different semantics in the various UML diagrams and even within one diagram. The UML repository stores each of these arrow types distinctly so that the compiler can generate systems according to users' intentions rather than how they express those intentions. Also, consistency checkers can operate on the repository to detect accidental conflicts between those intentions, or a tool might be repository-aware and only allow the construction of consistent models.

Model compilers can use the repository as source "code" that reduces the various nota-

This article reviews the primary concepts of repository-centered development with the Unified Modeling Language, explaining the relation between notation, semantics, and model compilation. It highlights UML's approach to semantics, and flexibility in notation and compilation.

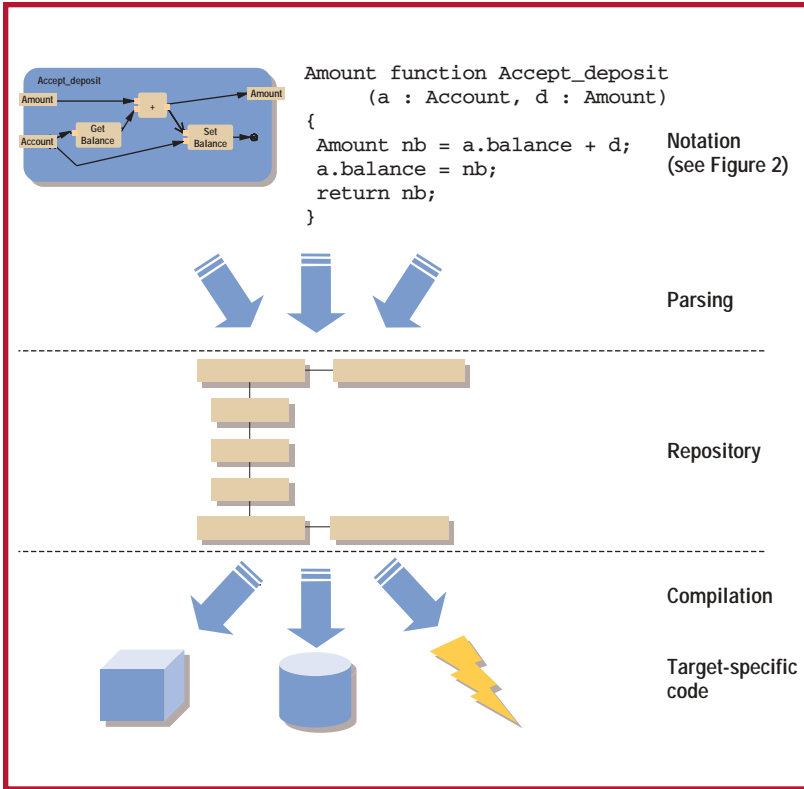


Figure 1. A repository-centered architecture.

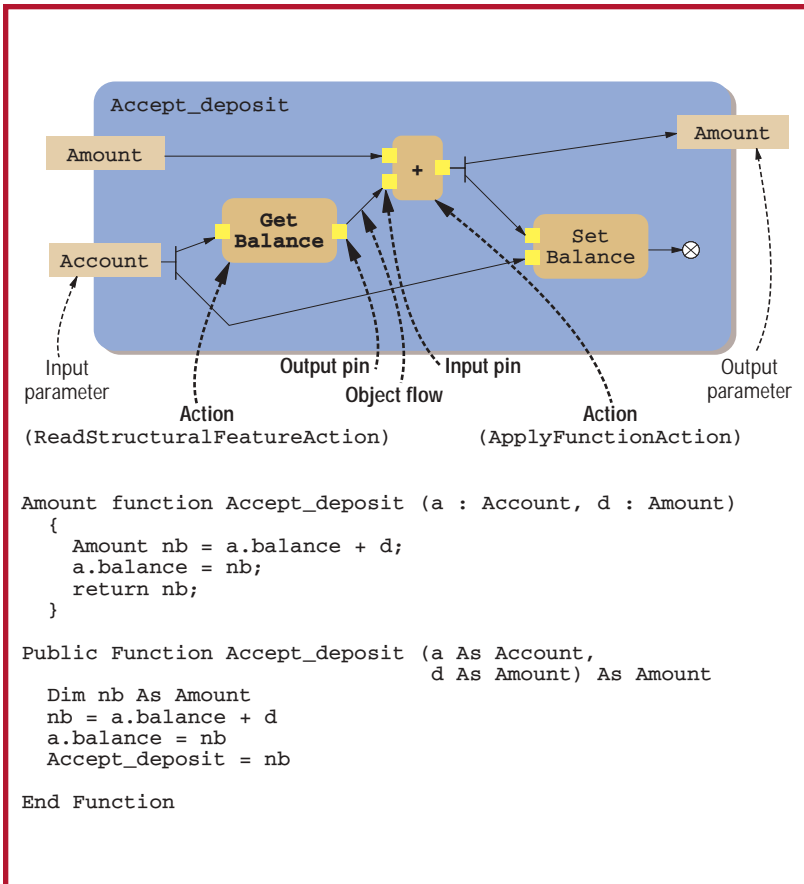


Figure 2. A UML 2.0 activity diagram and equivalent C++ and Visual Basic syntaxes.

tions to their intended effects on the resulting system (see www.omg.org/mda). The separation of repository from notation provides greater flexibility in compiler construction compared to using structures closely related to the ordering of tokens parsed from languages such as XML (Extensible Markup Language) and C++.^{2,3} Repository compilers traverse abstract syntax in a flexible manner rather than being restricted to a tree closely related to the presentation. The architecture provides for sophisticated compiler directives, which makes compilers more configurable.

The UML repository is the basis for generating serialized formats, such as the XML Metadata Interchange, which provides file-based interchange in XML Schema,^{6,7} and application programming interfaces such as the Java Metadata Interface, which provides dynamic access to UML model storage from Java.⁸ The repository enables constraints to be stated and maintained on its contents through languages such as UML's Object Constraint Language, which defines a textual syntax and repository with the same capabilities as UML's.⁹

An example

Figure 2 shows a UML 2.0 activity diagram^{10,11} for a simple procedure that updates a bank account with a deposit. The rectangles on the border are the activity's input and output parameters. The arrows are object flows connecting parameters with input pins and output pins of actions in the activity. Equivalent presentations in C++ and Visual Basic syntaxes appear at the bottom of the figure, assuming that data flow in the activity diagram is presented with variables.

The repository is the same for all the notations in Figure 2, using the integrated activity and action model in UML 2. Figure 3 shows the repository for the part of Figure 2 shown in bold, the flow between `Get Balance` and `+`. Each repository element is an instance of a metaclass defined in the UML specification, which is the name to the right of the colon. The name to the left of the colon is the repository instance itself, blank if it is anonymous. The model shows a `ReadStructuralFeatureAction` called `GetBalance` that gets the value of the `balance` property from an object it receives from the rest of the model (not shown). The result is put on an output pin and

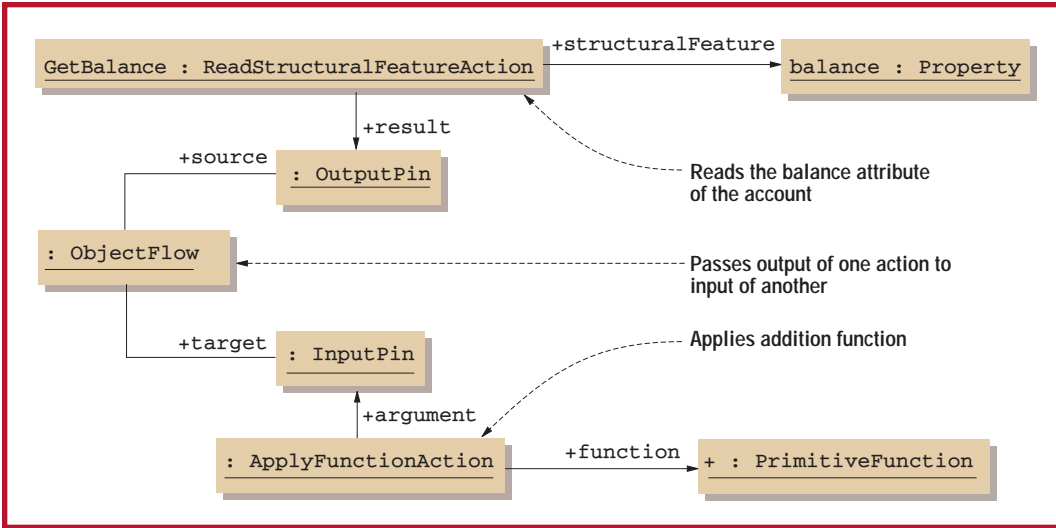



Figure 3. Partial UML 2.0 repository for Figure 2.

passed to an `ApplyFunctionAction` input pin. This applies the primitive function `+` to its inputs (other inputs not shown).

Once the surface notation is translated into the repository, model compilers can generate systems for multiple platforms. For example, the C++ syntax in Figure 2 could be compiled to a platform supporting Visual Basic and vice versa. The compiler might also detect that no object operations are used in Figure 2 and offer compilation to a relational database accessed by Structured Query Language.¹²

Repository-centered architectures extract a system builder's intention expressed in multiple notations for use by highly directable compilers. Users can view notations they feel comfortable with and still interchange models with others who use different presentations for the same concepts. A repository facilitates compiler construction and can be accessed without navigation restrictions implied by the input syntax. A repository focusing on semantic rather than notational distinctions supports a wider set of users and enables more powerful and accurate system generation and consistency checking. 

Acknowledgments

I identify the commercial software languages Java and Visual Basic to adequately describe concepts. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the software languages are necessarily the best available for any purpose.

References

1. *Unified Modeling Language Specification*, Version 1.5, Object Management Group, Mar. 2003; www.omg.org/cgi-bin/doc?formal/03-03-01.
2. *Extensible Markup Language*, W3C, Oct. 2000; www.w3.org/TR/REC-xml.
3. D. Kaley, *ANSI/ISO C++ Professional Programmer's Handbook*, Que, 1999.
4. *Human-Usable Textual Notation*, revised submission, Object Management Group, 1 Apr. 2002; www.omg.org/cgi-bin/doc?ad/2002-03-02.
5. D. Schenck and P. Wilson, *Information Modeling the EXPRESS Way*, Oxford Univ. Press, 1994.
6. *XML Metadata Interchange*, Version 2, Object Management Group, May 2003; www.omg.org/cgi-bin/doc?formal/03-05-02.
7. *UML 2 XML Schema*, Object Management Group, Apr. 2003; www.omg.org/cgi-bin/doc?ad/03-04-02.
8. *The Java Metadata Interface (JMI) Specification (JSR 40)*, Java Community Process, June 2002; www.jcp.org/en/jsr/detail?id=40.
9. *Response to the UML 2.0 OCL RFP*, Object Management Group, Jan. 2003; www.omg.org/cgi-bin/doc?ad/2003-01-07.
10. U2 Partners, *Unified Modeling Language: Superstructure*, Version 2.0, 3rd revised submission to OMG RFP ad/00-09-02, Apr. 2003; www.omg.org/cgi-bin/doc?ad/2003-04-01.
11. C. Bock, "UML 2 Activity and Action Models," *J. Object Technology*, vol. 2, no. 4, July/Aug. 2003, pp. 43-53.
12. H. Darwen, *A Guide to SQL Standard*, Addison-Wesley, 1997.

For more information on this or any other computing topic, please visit our Digital Library at <http://computer.org/publications/dlib>.

About the Author



Conrad Bock is a computer scientist at the US National Institute of Standards and Technology specializing in the Unified Modeling Language and modeling language semantics. He is one of the developers of the UML repository model at the Object Management Group. His research interests include process ontologies and hybrid modeling. He received his MS in computer science from Stanford University.

Contact him at 100 Bureau Dr., Stop 8263, Gaithersburg, MD 20899-8263; conrad.bock@nist.gov.