White Paper on RFP II: Abstract Syntax Tree Meta-Model

OMG Architecture Driven Modernization Task Force
August 18, 2004

Contributors: Philip Newcomb, The Software Revolution, Inc.
Ed Gentry , Blue Phoenix, Inc.
Carlos Araya, Artinsoft, Inc.
Ivan Sanazria, Artinsoft, Inc.
Charles Dickerson, Relativity, Inc.
William Ulrich, OMG ADM Task Force Co-chair

Introduction

The OMG Architecture Driven Modernization Task Force (OMG ADM TF) will hold discussions pertaining to drafting the ADM Request for Proposal (RFP II), Wednesday, August 23, 2004. A subgroup of the ADM TF was appointed at the June 23rd OMG ADM TF in Orlando to proceed with formulating a draft of RFP II on the subject: "structure artifacts below procedure level." This White Paper is intended to facilitate drafting of RFP II by providing a brief description of the subject area (here in after called the Software Modeling Standard) and summarizes issues related to it to be discussed and considered for standardization. This White Paper is respectfully submitted to the Full ADM TF Task Force to provide a basis for discussions pertaining to preparation of RFP II. The Software Modeling Standard is intended to be compatible with the objectives of the OMG Model Driven Architecture including the OMG Meta Object Framework, and supportive of the broader goals of the ADM Roadmap, the ADM Mission Statement and complementary to ADM RFP I, the KDM standard, and subsequent RFPs to be issued by the OMG ADM TF.

Scope

One of the key goals of standards organizations is to establish rules of the road that facilitate interoperability between the tools and services of the adherents of the standards established by the standards body. Interoperability standards enable integrators and customers to tackle more complex software engineering problems because more comprehensive solutions can be cost-effectively undertaken when tools interoperate effectively because of their adherence to common standards. It is costly, if not impossible, to integrate tools and services that do not have standardized interfaces and well-defined formats for interchanging information. The goal of the Software Modeling Standard is to achieve well-defined interfaces and well-defined formats for interchange of information about software models that are used by the complex and powerful tool sets that are used for software modernization.

The establishment of standards for the interchange of software models is a core requirement for the establishment of the multi-vendor industrial-scale integrated tool frameworks that are needed to support the OMG goal of Architectural Driven Modernization (ADM) and software development and maintenance using Model Driven Architectures.

The purpose of this paper is to commence discussion of what should or should not be defined within the scope of Software Modeling Standards and what should or should not be included in the scope of Software Model Interchange in order to facilitate the OMG ADM TF preparation of a RFP for the Software Modeling Standard.

Software Modeling

High quality software models are the basis of the high quality analysis, transformation and documentation technologies that are the foundation of powerful tool based software modernization solutions. The quality of the software model is the primary limiting factor upon the quality and comprehensiveness of the analysis, transformation, and documentation achieved by software modernization tools. The structure of the software models inherently limits the ability of the software models to be integrated with other models and other tools.

Effort and technology are required to create and exchange software models. The degree of precision of software models and the effort associated with their construction can vary greatly depending upon the approach taken to their construction. The concept of weaker and strong modeling formalisms is concerned with the degree to which models are able to faithfully represent with precision the details of the thing that they seek to represent. A weaker model will gloss over important details, such as scope and type of variables and expressions, while a stronger model will capture these details. Models also differ in their architecture. Weaker models tend to blend modeling layers together without clear boundaries and without expectations for when a particular model property should or should not be present, while stronger models will carefully define properties, define layers between models, and define when model properties should or should not be present.

It is the collective view of the authors that RFP II, the Software Modeling Standard, should provide a descriptive formalism for representing abstract models of the structure of software and the format to be used for the interchange of software models. These software models should provide for the representation of the types, relationships between types, and other properties sufficient to represent the abstract structure of software, and should be adequate for the composition of models of software applications expressed in the language constructs used in software languages. The software modeling standard and the formalism used for its description should be sufficient to permit the representation of generic as well as specific model properties. The standard should permit more specific types to be derived from more generic types, and allow static structural as well as dynamic functional properties of these types to be generalized, specialized and inherited.

Formal software modeling is understood to entail the construction of structured multi-dimensional layered formalisms that describe software and can be manipulated and transmitted by computer programs and transferred between persistent and temporary storage mediums on computers.   It is essential that software models interplay with other modeling layers in such a way that the relationships between software models, and models constituting additional descriptive layers about the software be structurally continuous, so that automated tools can freely use all dimensions and layers of the modeling formalism and easily construct, deconstruct, augment and transform the types that make up software models and the types used to represent models that augment the software models.

Within the context of the ADM other *descriptive modeling layers* are understood to include the KDM as well as other *software analysis models* that supplement the software model with additional semantic properties about software. The boundaries and interface between these model layers should be formally defined and precisely specified sufficiently to allow construction, manipulation and interchange of composite and discrete models, but the specific composition of these modeling layers should not be constrained by the standards body.

Abstract Syntax Trees

Common techniques for software modeling involve the use of parsing technologies to create structured abstract syntax tree representations of software from the concrete (or surface) syntax of the software artifact.

Abstract syntax trees (AST) are models of software that represent the software artifact, either the legacy software or the modern software application, using data structures that represent the types of language constructs, their compositional relationships to other language constructs and a set of direct and derived properties associated with each such language construct. The abstract syntax tree is derived from an analysis process. An abstract syntax tree provides a means for creating a more or less complete representation of the software artifact.

The abstract syntax tree is an extensible and formal representation of the syntactical structure of software that is more amenable to formal analysis techniques than is the concrete or surface syntax of software. An AST may be an invertible representation, that is, it may be possible to traverse the AST and reconstruct the “surface syntax” of the legacy system, or reconstitute it in textual from, from the abstract structures. ASTs may be augmented, that is, the AST may be analyzed and have added to it additional structures that describe additional properties about the software.

Common analyses that augment an AST with additional properties include constraint analysis, data-flow analysis, control-flow analysis, axiomatic and denotational analysis, and so on. The analysis models used to capture such properties can become arbitrarily complex and intricate, and highly varied in form, composition and intended use. ASTs are generated with varying degrees of precision to support differing objectives.

The form and composition of the AST influences the kinds of analyses that can be performed on it. ASTs typically express the structural compositional of the software they model using the syntactical types of the software language in which the software is written. ASTs are traditionally the input to the translation stage of compilers. Machine code is generated by the traversal of the AST structures and application of rewrite rules to translate the AST structures into machine instructions.

ASTs are generally augmented with additional analyses layers, such as type analysis, control-flow analysis or data-flow analysis (to support code optimization) or to support capture of software engineering metrics (control-flow and data-flow complexity) and documentation. The use of AST structures for the abstract representation of the structure of software has become an accepted practice for modeling software; however the definition of the format of AST structures and the mechanisms for representation and interchange of ASTs models has not yet been standardized.

Establishment of standards for software model formats and interchange will facilitate exchanging software models in standard abstract formats between tools. The ability to freely exchange software models between tools will provide industries, corporations and the government the ability to use advanced model-based tools for automated software analysis and transformation, which will bring about a significant advancement in the state of software engineering practice -- perhaps as significant as the introduction of engineering diagrams did for industrial engineering practices. The achievement of deep-level interchange of software models between software tools that have common expectations about the structure of these software models is expected to have a profound impact upon tool providers and tool users in the software industry.

Software Transformation

Software Transformation is generally concerned with the application of rewrite rules to a software artifact in order to change its structure. A general notion of transformation is any mechanized change performed to a software artifact – either directly to the textual form of the software artifact or a change made to an abstraction of the software artifact, such as an AST, which results in a change to the text of the software artifact. There are many variations in approaches used and tools used for software transformation that vary in how they describe software transformation. Some software transformation tools operate on the surface syntax directly. Others operate upon syntax remotely by manipulating abstract syntax.

Among the more commonly used and freely available textual (or surface syntax) software transformation tools today are simple textual substitution commands and macros embedded in editors such emacs and vi and word. Command-line tools for more powerful forms of textual transformation include tools such as sed, awk, and perl. Such tools have in common their manipulation of surface syntax rather than abstract syntax. Tools based upon textual substitution approaches are generally more limited in their range of functionality than transformation tools that operate against abstract software models. Compilers and many software engineering tools use ASTs and AST-like structures produced by lexing and parsing technologies. Some common and freely available tools, such as lex, yacc and bison, are widely used for creating abstract models of software. Such tools have in common their construction of data structures for the representation of software and the use of programs for the manipulation of these data structures. The data structures produced by tools such as lex, yacc and bison are not standardized across industry nor are the techniques for writing the programs that manipulate these data structures.   Among the data structures that can be produced through the use of such tools are abstract syntax trees.

The automated software modernization industry came about in part from the broadened development and utilization of advanced tools for automated software transformation based upon advanced software analysis and transformation tools that operate upon ASTs and AST-like software models. While technologies such as lex, yacc and bison are freely available, the technologies used by vendors for software analysis and transformation of software models are often highly proprietary to the service providers or vendors that originated them. The relatively efficiency and expressiveness of these tools are often the key technical discriminators between the services offered by these vendors.

The software modeling standard should not seek to limit or constrain the variations in composition, mechanisms for definition of the software models, or the techniques and algorithms used for model construction, transformation or interchange. The ADM task force should take care not to define a standard that in any way inhibits the ability of the vendor community to continuously improve the quality of its models and the techniques for creating and transforming software models.   The software modeling standard can establish common formats for model definition and interchange that permit variation in the structure and composition of software models between industry groups.

For example, two vendors may agree or disagree to use a particular abstract syntax tree definition for creating and interchanging software models of software applications written in Ada 83, a software language commonly in use within the DoD and industry. The structure of the Ada 83 ASTs will not be prescribed by the standard, nor will the techniques for model construction. However, the format used for exchanging the AST model between the vendor's tools will be subject to definition by the standard, and will support the definition and interchange of the particular Ada 83 ASTs in a commonly accepted format by the two vendors, using a model interchange format that all industry participants can agree upon.

RFP III, the standard for Software Model Transformation, is expected to support, at a minimum, the transformation between software models of the same language that may exist in different forms. For instance, two vendors working independently are likely to create different AST software models for the representation of the same application in the same software language. It may be the case that the variations in the ASTs produced by different tool vendors are sufficiently divergent that it is impossible to achieve a complete transformational mapping from one to the other.   Such divergences in language model form are expected and acceptable and the software modeling standard will not attempt to constrain them.

Two vendors may choose to base their tools upon the same AST models, and this would obviously facilitate combining the tools to achieve more comprehensive solutions if the tools of the vendors were complementary. The compatibility of models can provide the basis for industry associations and the forging of alliance partnerships between industry participants. To the extent that vendors choose to use models whose AST models are identical or homogenous they will be better able to interchange models defining transformations between their models and are likely to achieve interoperability between their tools more quickly.

Multi-Vendor Tool Suites

A key motivation for the OMG establishment of standards for the interchange of software models within the context of the ADM TF is to facilitate major integrators, tool developers, tool suppliers and customers creating well-integrated solution suites for software modernization.   Such solutions suites will consist of collections of tools and services that facilitate modernization of multiple legacy source languages into one or more target platforms.   Such solution suites will be facilitated by the ability of the vendor community to establish standards for the interchange of software models.

The ultimate OMG goal is to move towards an Architecture Driven Modernization approach to software development and maintenance that facilitates the OMG goal to use Model Driven Architectures for the modernization of legacy systems as well as the development of new systems.   To support this objective, information about the composition of software must be interchangeable through the use of standard interchange formats.

The task of modeling software results in the construction of software models is larger and more complex than in virtually any other engineering discipline. Ideally the interchange formats used for exchanging software models should be consistent with the OMG Meta-Object Facility MOF and XML Model Interchange Formats (XMI). However, while adherence to the MOF is desired, to the extent that additional formalisms for model definition and manipulation are needed that might not exist within MOF, the ADM TF will work with MOF standard committees to reconcile these differences.

Differentiation of Modeling Levels

As mentioned above, software models vary in their levels of precision from superficial to deeply detailed. They may also be augmented with arbitrarily deep levels of auxiliary information (meta-information) that reflect the results of descriptive analysis of the software artifacts. The OMG ADM TF is defining modeling and interchange standards for several of these layers of description. We propose that the software modeling layer be restricted to the collection of AST-like data structures that represent the abstract structure of software. This layer does not consist of the analysis layers that can be superimposed upon ASTs to augment the AST with potentially unlimited additional layers of semantic description, which should more properly be called the software analysis layer.

By bounding the software modeling layer to the more concrete collection of structures that provide the abstract representation of software constructs for multiple languages we restrict the complexity of the format used for the interchange of software information. This restriction can expedite the process of standards adoption by limiting the breadth of the area subjected to standards adoption. This restriction can thus achieve benefit more quickly by facilitating a shorter standards adoption cycle, and hence more rapid adoption of the standard by leading participants in the industry.

By taking this more limited approach to standardization, the structures used for control-flow analysis, data-flow analysis, and the structures used for representation of scope, type, and those used for denotational semantics or axiomatic semantics are not considered part of the software model layer. We would characterize the modeling layers used for these kinds of semantic augmentations to the software modeling layer as the Software Analysis Layer and would propose that a separate RFP be issued to address the Software Analysis Standard.

Furthermore, we propose that the standards for representation of software modeling information not attempt to constrain the set of language constructs to a common set of constructs universal to all languages. Even though there is commonality between many software languages, software languages vary greatly in their syntactical and semantic forms.

In order to faithfully capture the abstract structures associated with these software languages the standard for software interchange must allow for all possible variations in language constructs. Thus, the standard for software model interchange should not propose a universal set of language constructs. It must instead focus upon the mechanisms and structures that facilitate representing AST-like structures and the formats for transferring such models between tools.

Finally, the software model layer must be augmentable, that is, it should be possible to form composite models that contain both the software models for a particular language as well as the software analysis models that pertain to particular software analysis formalism.

Surface Syntax and Abstract Syntax

The mechanisms for the creation of abstract syntax from surface syntax are often achieved through the use of proprietary technologies. There are widely used tools that provide a one-way translation from surface syntax (textual form of software) into the Abstract Syntax Tree Form. The degree of precision of Abstract Syntax Tree Models varies as well, with some modeling techniques being more complete than others. There are also tools that support bi-directional mapping between surface-syntax and abstract syntax.

In principal the approach to this mapping between concrete and abstract form should not be the focus of the software modeling standards. The mapping from syntactical form to abstract form is more of a mapping issue than a modeling issue, in any case. There are many ways to approach this problem, and the establishment of standards for interchange of software models does not require addressing the mechanisms that are used for the creation of the software models, nor does it need to address the mechanism by which the software artifact takes on abstract or concrete forms.

Variation in industry approach is expected in the technologies used for abstracting software model from their concrete forms and generating software from abstract forms.

Conclusion

This paper has attempted to delimit the scope of the ADM TF RFP for software model interchange. Its perspective is that there are many areas ripe for standardization that properly lies outside of the purview of the software modeling RFP.

The RFP must anticipate the need for the software modeling standards to be augmented with auxiliary models, but it should not establish standards in areas that are extraneous to modeling the structure of software, nor should it establish a standard that inhibits contiguity between modeling layers or inhibit the interchange of composite models.

In particular, the principal concern of the ADM TF RFP for software modeling is to provide formats and mechanisms that facilitate the interchange of abstract software models for formal software languages. Such formal languages are to include all software languages.

Glossary

Abstract Syntax Tree	A data structure consisting of types that represent language constructs connected by sequence and unit valued relationships to other types. Additional properties associated with each type represent names of identifiers, numbers and other literal values associated with the language construct. An abstract syntax tree is an acyclic graph with a single root node, connecting nodes and leaf nodes.
Concrete Syntax or (Surface Syntax)	The textual form of a software artifact expressed in the software language in which the software artifact was written.
Software Artifact	The model or textual form used for writing or expressing software or information about software.
Software Analysis Layer	The modeling layer used for capturing and expressing

	information about semantic models of software artifacts.
Software Analysis Model	The model formalism used for capturing and expressing information about semantic models of software artifacts.
Software Analysis Interchange Format	The format used for transmission of software analysis models.
Software Model	The model formalism used for capturing and expressing information about syntax models of software artifacts.
Software Modeling Layer	The modeling layer used for capturing and expressing information about syntax models of software artifacts.
Software Model Interchange Format	The format used for transmission of software models.
Software Transformation	The application of a program that takes as input a software artifact and produces as output a modified form of the software artifact, or the application of a program that takes as input a model of a software artifact and produces as output a modified form of the model of the software artifact.