Towards A Standard For Model Specification And Storage

D. Deva, J. Sprinkle, G. Nordstrom, and M. Maroti
Vanderbilt University
Nashville, TN, 37203

Abstract

Software production has become an industrial task usually involving teams of programmers working on complex problems to produce large, even huge software systems. Globally distributed teams are doing a growing share of all software development work. The management of software engineering teamwork, especially of a temporally and/or spatially distributed team, presents an enormous organizational challenge as well as an intricate technical problem, as such distributed teamwork requires tool support for coordination of cooperative activities, maintenance of project control, and sharing of information. Domain-specific Model Integrated Program Synthesis environments are created according to a modeling paradigm – a description of the class of models that can be created using the system. Just as model integrated computing applications are executable instances of domain models, domain models can be viewed as instances of metamodels. The representation of these models and the modeling paradigm is unique to the specific modeling environment. This poses a major problem for portability of models from one modeling environment to another. The purpose of this paper is to explore the possibility of a common standard for the storage of models, in what framework the standard should exist, and who should define the standard.

1 Introduction

Modeling is the process of creating an abstract representation of an engineering system. Models serve several useful purposes, such as testing a system before building it, communicating with customers, visualization, and reduction in complexity etc. [4]. Domain-specific modeling environments (DSME's) differ in their modeling languages and their model representation. There cannot exist a single standard for all DSME's within the entire computer systems design community. The reason for this is that DSME's are designed to solve problems within a certain domain and some modeling concepts are very important for some applications but trivial or redundant in other DSME's. Large companies with physically dispersed divisions create distributed teams to work together on software projects. Cross-organizational projects also occur with greater frequency, such as a subcontractor working closely with a primary systems-integration contractor on a large project. In such cases models developed by one team using a DSME, should be compatible with the DSME used by the other. Here, there arises a need for DSME's to exchange models with one another.

Although there exist several standard implementations for specifying models (e.g. UML [11]) and at least one that specifies interchange formats (CDIF [1]), none of these implementations support data conformance or validation. It is for this reason that we will explore the possibilities of a different medium for specification and storage.

2 Background

Large Computer Based Systems (CBS's) are among the most significant technological developments of the past 20 years [12]. Within the domain of CBS's the technology with which to develop them has evolved. In the beginning of CBS's, the structure of the system and its components was certainly not computerized, because of the enormous expense of visualization, but was instead hand-drawn [4]. Today, with the development of inexpensive computer hardware, large CBS's are being developed using the concept of Model Integrated Computing (MIC) [5][6][8].

2.1 Why Models Are Used To Solve Problems

Computers and computing systems are ubiquitous today. Therefore, not only does it make sense to construct software that reflects the tangible world that it describes, but also it is necessary to construct that software in a way that allows the software to evolve as the world evolves. MIC uses models as the basis for system development. These models reflect the components of the system, and the interactions between them, but not the particular implementation in which those components operate. MIC is used to create a solution for the *class* of problems [13] (e.g. production flow management), and that solution is then employed to describe the particular problem (e.g. production flow management for the Saturn plant [8]).

The MIC approach describes the environment in terms of models. The direct benefit of a model based solution is found in the notion of Model-Integrated Program Synthesis (MIPS) [9][13]. MIPS allows the generation of programs from the model-integrated solution. This means that to update the software solution, one merely updates the model of the environment, and recompiles the software. This means fewer errors in software updates, and faster updates of the software solutions.

2.2 GME Modeling Concepts

DSME'S are made up of basic concepts. In the Graphical Modeling Environment (GME), developed at Vanderbilt University, the following modeling concepts apply, on a separate level from the final modeling solution. For a more detailed description of the modeling concepts, please see [7].

- Paradigm: a description of what modeling concepts are allowed in a particular solution. More information on the role of the paradigm is described in section 2.3.
- Category: a container for models. It is an organizational tool (similar to a file system folder).
- Atom: the smallest entity of a solution. It cannot contain any parts, but can connect to other atoms and have attributes.
- Model: can contain atoms and other models. In addition it can also contain connections, references, attributes, and conditionals. It visualizes its contents through aspects.
- Port: an atom that can connect to atoms in other models.
- Aspect: a visualization tool that allows for specific parts of a model to be viewable. The aspects of a model are defined when the model is defined.
- Attribute: can belong to an atom, model, reference, or connection. Usually a text field or menu that defines the role of the part in the solution.
- Containment/hierarchy: the notion of parts "belonging" to a model (can be many levels deep).
- Connections: an association between two (or more) parts. Currently in GME, connections can occur between atoms (but not models).
- Reference: succinctly, a pointer to a model or atom. References to references are also allowed.
- Conditional control/set: a way to group parts together at modeling time. This was designed as a way to have more specific aspects when designing, but also may serve the purpose of a set.

2.3 The Paradigm

A particular modeling solution class (or DSME) must obey the rules of that solution class. These rules are specified in something called a paradigm. The paradigm essentially defines the entities and relationships allowed in the given domain [7]. The paradigm, in this sense, defines the syntax of the modeling language. It is the paradigm that enables a generic, configurable model editor to become a DSME.

The paradigm is crafted for the solution class, and therefore describes the way that models and atoms, etc. are used in the solution. It does this by assigning a "kind" to the basic parts. A model may be called a "warehouse" or a "tire bay," and an atom may be called a "crane" or a "tire." The syntax of the modeling language as described in the paradigm is described in terms of these "kinds," not just in terms of the models and atoms themselves.

In general, the paradigm is represented by the specification of a metamodel. It formally defines the syntax, semantics, presentation, and translation specifications of a particular modeling environment [10].

The paradigm, called the metamodel, is described in terms of what is called the meta-metamodel. This meta-metamodel is the description of all of the possible syntax, semantics, presentation, and translation specifications of *any* metamodel (i.e. paradigm) [10].

2.4 Storage

Once a solution is crafted using the DSME, it is useful to store the solution. When the solution is stored, however, the paradigm is not stored with it. When the solution is loaded again, it must still conform to the paradigm that was used in its definition.

While at first it may not make sense to store the paradigm separate from the solution, consider the case where many solutions depend upon one paradigm. Any changes to the paradigm would then need to be replicated throughout all of the solutions. However, when the paradigm is stored separately and referenced by the models, then when the paradigm is changed it is relatively easy to validate whether the models still conform to the paradigm.

3 Standardizing Storage

As each DSME has a specific format for representing models, DSME's cannot exchange data because of nonstandard modeling representation. An analogy would be working with files in different formats, say opening a Microsoft Word file in the Unix-flavored vi. Although Word and vi are both tools in the same domain (word processing) they do not have a common storage format.

However, there exists in Word the option to "save as" (export the data) basic ASCII characters. This is a format that vi can read, and that Word can read. ASCII is the smallest set of concepts in the word processing world, and any document can be visually expressed with it.

Similarly to the word processing world, any DSME that accurately represents its domain shares concepts with any other accurate DSME, even if those concepts are not represented the same way. What is necessary in order to communicate between DSME's is the smallest set of concepts that they share.

3.1 The Meta-metamodel And Model Parts

When storing models, the models are actually stored in the language of the meta-metamodel. For example, a "Warehouse" model is actually known in storage as a "model of kind Warehouse." It is possible, therefore, to produce a listing of all models in a solution in terms of how they exist in the meta-meta sense, with attributes of how they exist in the meta sense. In other words, they *exist* in terms of the meta-metamodel, and *conform* in terms of the metamodel.

3.2 XML

The Extensible Markup Language (XML), developed by the World Wide Web Consortium (w3c) is a meta-markup language; a set of rules for creating semantic tags used to describe data [14]. XML is designed to structure data so that it can be easily transferred over a network and consistently processed by the receiver. Because XML is used to describe information as well as structure it, it can be thought of as a data description language. XML can be used to describe data components, records and other data structures, including complex ones.

An XML element is made up of a *start tag*, an *end tag*, and data in between. The start and end tags describe the data within the tags, which is considered the *value* of the element [14]. An element can optionally contain one or more attributes. An attribute is a name-value pair separated by an equal sign (=). XML is hierarchical – elements can contain other elements. Because XML is a highly structured language, it is important that all XML be well-formed. That is, the XML must have both a start tag and an end tag, and must be authored using the proper syntax [14].

There needs to be a way of making sure that data are structured in a particular way. Currently the most common way to define structured XML documents is to use a Document Type Definition (DTD). The DTD can be considered as the meta-XML. It defines the rules for the formation of a particular XML document. The validity of the XML file can be checked using the DTD. This will prevent transmission of incomplete or bad data. However, having a DTD is not enough: there must be something that performs the actual check. A software engine called a parser performs the actual check on the data to make sure it conforms to the DTD. This process is called validation. An XML document that conforms to a DTD is called a valid XML document. They can use freely available XML parsers to parse the XML document, which is made of the initial data.

In short XML is a universal, standardized language used to represent data such that it can be both processed independently and exchanged between programs and applications and between clients and servers [2].

3.3 XML and Model Representation

To translate models from one DSME to another, it is necessary to represent all of the domain-specific concepts. Likewise, to translate between general purpose modeling environments (ME's) it is necessary to understand the concepts of modeling in general. These concepts (briefly outlined in section 2.2) should be representative of all of the concepts of modeling, and are the concepts modeled by the meta-metamodel.

As mentioned in section 3.1 models are stored in terms of the meta-metamodel, with attributes of how the model relates to the DSME. In reviewing the concepts of section 2.2, XML is an excellent match for the requirements of a storage medium–

- XML inherently represents all data hierarchically
- Entities in DSME's may have attributes associated with them, which may be bound into elements in XML.
- XML has built-in support for references to other entities within the document, complete with enforcement rules to prevent null references.
- XML enforces the conformance of the document to a structured standard document type (DTD), which provides an easy litmus test for the validity of a model.
- These exists a plethora of parsers for XML documents, for virtually every operating system and programming language. Further, XML is a widely used published standard for data representation.

In addition to these reasons, XML is useful because it is viewable from the web. As far as specification goes, XML here is used only in a small way. Specification of models occurs in this paper only in terms of the modeling environment, not in terms of the definition of the paradigm. UML gives a way of graphically representing models, and it does this well. However, it does not specify a good textual syntax for model exchange. It was for this purpose that CDIF was developed. Although both XML and CDIF essentially only format data, XML is superior with its ability to test conformance with the DTD. This is where XML begins to stand out as a viable implementation.

3.4 Methodology

XML documents can be exchanged and used across dissimilar platforms and applications. Using XML, data from any DSME (for example, a paradigm within GME) can be wrapped inside a data structure, which can then be used by another DSME (e.g. Visio or any X Modeling Environment (XME)) that parses the data and uses it. Since the data structure and the data itself can be combined in a document object instance, the document "is" the database. This "database" can be processed by any application that can handle XML, even though that application may have no knowledge of the origin of the data [3].

As shown in Figure 1 , we use XML as the communication medium between DSME's.

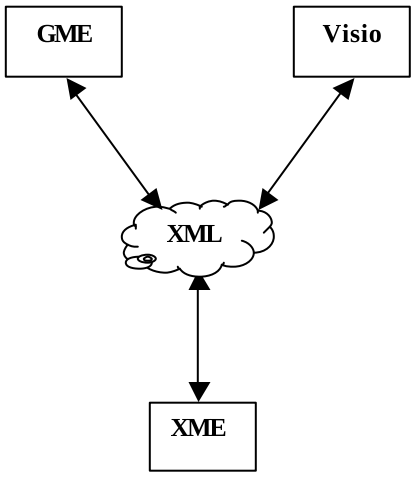

Figure 1 - Using XML as a communication language for exchanging information between DSME's.

In order to create the XML document from GME, XML data exporters were developed which traverse the model structures in the project, extract information from them, and translate the paradigm and model data into XML documents. These documents conform to DTD's written that define the models and the paradigm (see Figure 2).

The XML model files and the paradigm file are then sent to the other DSME along with the corresponding DTD's for loading. It is the responsibility of each modeling environment to develop XML export/import routines that can organize the necessary data in the format required by the XML files.

The models are then recreated with the XML loader in the destination DSME using the data that was exported to XML. Now, the same data that was represented in the GME version of the DSME is represented in whatever modeling editor has loaded the models. This is assuming of course that the semantics of the appearance of the data are maintained through the export/import process. Without the common storage format, this process can take a large amount of time, because the translation of model types is handled by human interaction, either through manually recreating the models, or writing a custom program for this specific migration. Using the common storage format approach, one mapping from the concepts in the DTD provides a solution for *any* static migration.

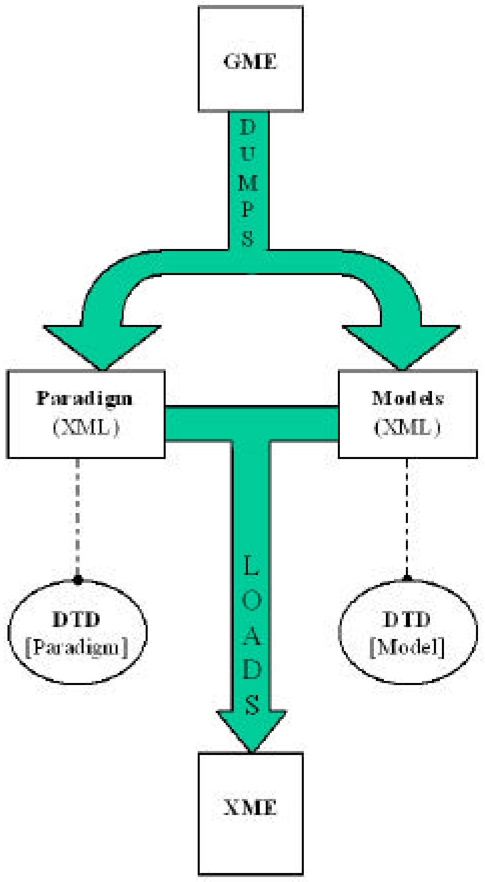

Figure 2 - Representation of Export/Import Solution

3.5 Smallest Set Of Modeling Concepts

With a standard form of expression for models, the next step is to define all of the possible concepts used for development of models. The overall goal of the standard is to set up a way to deliver all of the semantically important information. GME modeling concepts were described in section 2.2. However, we recognize that other modeling editors may have generalized other concepts of which the GME presents only the special case (e.g. GME currently represents all connections as binary with a source and destination, but other environments may allow for n-ary connections). The establishment of such a standard, as discussed earlier can exist only after experience and meeting with the maintainers of other similar modeling editors.

3.6 Current Success

A solution currently exists for the conversion from different versions of the GME at Vanderbilt. Currently, we are transitioning from one version of GME to another, and the addition of several concepts (not to mention the renaming of concepts from one version to another) is introducing the problem of model conversion. Here we have two distinct modeling environments, each of which cannot load the conventionally stored data of the other. Therefore, models defined under any paradigm using the old version will be different than under the new version, because the meta-metamodels are different, and thus the problem is solved by casting it as a migration between two DSME's.

Our current solution exports paradigms and model structures from the old GME to an XML file that the new version understands perfectly. The data does not change at all; what changes is the format in which the data are presented to the new version of GME.

4 Definition of the Standard

It would be premature of us to say that we have laid the groundwork for future work in the definition of a standard interchange format. In order to completely define the standard, the smallest set of general, graphical modeling concepts and components must be completely and formally defined. It is not possible to do this while looking at only one model editor, but instead a case study should be performed which would examine as many modeling editors as possible. However, we believe such a standard is achievable.

5 Future Work

The goal of this work (as described in section 1) is designed specifically for the translation of information between two modeling editors. However, it may be extended to include the translation of information between two paradigms. In this sense, although the two modeling editors may in effect be the same, the paradigms differ. This problem (of translating models defined under one paradigm to a quite similar, but significantly different, paradigm) is known as the model-migration problem. It is important to note that model-migration in this sense is more broad than mapping between two DSME's, because in model migration, the *paradigm* is significantly different. This means that new concepts are introduced, or existing concepts are deleted. It can also mean that new concepts are introduced to replace old concepts, but only in certain contexts. However, mapping between DSME's is certainly a special case of the model-migration problem.

When the model-migration problem is encountered, the most common solution is to rebuild the entire model database. By exporting the model database under one paradigm and subsequently importing it into the other, the same solution is inherently used: to recreate the models. However, automating the process reduces the time necessary and, more importantly, reduces errors that may occur.

The XML export/import solution has been used on several occasions in projects using the GME and has worked well. However, as more of the model-migration problem is examined, it is obvious that a total solution, even for one particular modeling editor, is quite a long way off.

6 Conclusions

It is certainly a problem that so many modeling environments cannot easily interchange data. A simple exchange format (similar to ASCII for word processors) is not only possible, but necessary for a computing world that leans more and more toward modeling as a solution to today's problems. We have shown that it is possible to produce XML documents that accurately contain all of the data of a model solution, and subsequently recreate those models in another modeling environment that uses the same paradigm. This was accomplished by delivering everything the modeling environment needs: a paradigm, and the models that conform to it.

By choosing XML, we took advantage of built in document validation. This means that one DTD could be defined to which all paradigms and models would have to conform. Then, any modeling environment could export documents that conformed to this DTD, and any modeling environment that could import the XML document could create those models. The advantage is that XML provides a common syntax and storage, and that there are literally dozens of free XML parsers available off the shelf [14].

Finally, by showing that models may be converted from one DSME to another using this technique, we have noted that an exploration of translation between many different modeling environments is the next step in developing a common DTD that could allow for any mature modeling environment to exchange data with another.

References

[1] The CDIF Homepage. Understanding between modeling tools. http://www.eia.org/eig/cdif/index.html

[2] A. Ceponkus, F. Hoodbhoy, *Applied XML: A Toolkit for Programmers*, John Wiley Inc. 1999.

[3] M. Fichtelman, "*Go Wireless*, XML Magazine", Summer 2000, Vol.1.

[4] I. Jacobson, G. Booch, J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

[5] G. Karsai, F. DeCaria, "Model-Integrated Online Problem-Solving Environment for Chemical Engineering," *IFAC Control Engineering Practice*, Vol. 5, No. 5, pp. 1-9, 1997.

[6] G. Karsai, J. Sztipanovits, S. Padalkar, C. Biegl, "Model Based Intelligent Process Control for Co-generator Plants," *Journal of Parallel and Distributed Systems*, pp. 90-103, 1992.

[7] A. Ledeczi, M. Maroti, G. Karsai, G. Nordstrom, "Metaprogrammable Toolkit for Model-Integrated Computing," Proceedings of the Engineering of Computer Based Systems (ECBS) Conference, pp. 311-317, Nashville, TN, March, 1999.

[8] E. Long, A. Misra, J. Sztipanovits, "Increasing Productivity at Saturn," *IEEE Computer Magazine*, August, 1998.

[9] A. Misra, G. Karsai, J. Sztipanovits, "Model-Integrated Development of Complex Applications," Proceedings of the Fifth International Symposium on Assessment of Software Tools, pp. 14-23, Pittsburgh, PA, June, 1997.

[10] G. Nordstrom, "Formalizing the Specification of Graphical Modeling Languages,**"** Proceedings of the IEEE Aerospace 2000 Conference, CD-ROM Reference 10.0402, Big Sky, MT, March, 2000.

[11] J. Rumbaugh, I. Jacobson, G. Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

[12] J. Sztipanovits, "Engineering of Computer-Based Systems: An Emerging Discipline," Proceedings of the IEEE ECBS 1998 Conference, 1998.

[13] J. Sztipanovits, G. Karsai, "Model-Integrated Computing," *IEEE Computer*, pp. 110-112, April, 1997.

[14] The XML Specification. World Wide Web Consortium: 1998. http://www.w3.org/TR/1998/REC-xml-19980210