Model Design Using Hierarchical Web-Based Libraries

Fabrice Bernardi
University of Corsica
UMR CNRS 6134
Corte, France
bernardi@univ-corse.fr

Jean-François Santucci
University of Corsica
UMR CNRS 6134
Corte, France
santucci@univ-corse.fr

ABSTRACT

Design tools can be profitably associated with libraries of reusable modeling components that will make the description and also the validation of the models much easier. Furthermore, applications of today and tomorrow will be increasingly based on three fundamental technologies: Object Orientation, Client/Server and Internet. We propose in this article an object-oriented architecture for the definition of Web-based hierarchical models libraries. The originality of our approach lies in the facts that it is based on : (i) a notion of genericity of use, (ii) notions like inheritance and abstraction links between the stored models and (iii) Web-based storing and consulting libraries procedures.

Categories and Subject Descriptors

J.6 [**Computer Applications**]: Computer-Aided Design; D.2.11 [**Software**]: Software Engineering—*Software Architectures*; D.2.13 [**Software**]: Software Engineering—*Reusable Software*

General Terms

Design, Management

Keywords

models reuse, models libraries, Web-based access, abstraction hierarchy

1. INTRODUCTION

The design of complex manufactured systems is a task requiring a lot of time to be achieved. One way to speed up this task is to develop methodologies for deriving and for using reusable design components. Recently a set of research work has been oriented towards this direction. We can highlight for example : (i) that components represent over 70% of product cost in the EDA industry[15] ; (ii) a multi-company project called ECIX (Electronic Component Information Exchange) and presented in [5]. In the same time, computer simulations have been becoming increasingly complex and require efficient system simulation framework [14]. Furthermore, increasing size and geographical separation of design data and teams has created a need for network-based design environments [6]. Usually, Design tools are profitably associated with libraries of reusable modeling components that will make the description of the models and also their validation much easier [4]. Storing models in a common generic library has several benefits. First, the genericity of this storage service can be offered to various modeling and simulation environments. Second, a common library allows environments to share information so they can interact each other, and third, modeling components can be shared by several users. This last point is the most important since it allows a design team enabling an efficient collaborative work.

Applications of today and tomorrow will be increasingly based on three fundamental technologies [13]: Object orientation allowing applications to be viewed in terms of natural objects; Client/Server allowing application components to behave as service consumers (client) and service providers (server); The Web allowing to access resources located all around the world. A Web-based access is a very interesting perspective for a generic modeling components library, since it allows an user-friendly remote access using a simple Web browser. We describe in this article an object-oriented architecture for the definition of Web-based hierarchical models libraries. The libraries are based on a hierarchical organization of concepts relating to the class of applications. The leaves of the hierarchical structure represent the elements which can be used in model descriptions. Information concerning the elements can be placed at the appropriate level in the hierarchy, thus, eliminating needless repetition when the elements are being defined. The originality of our approach lies in the facts that it is based on a strong notion of genericity of use, and notions like the inheritance and abstraction links between the stored models. We describe how managing inheritance between stored models can improve their classification and their accuracy, and how the abstraction hierarchy allows to describe a same model at various detail levels.

The paper is organized as follows. Section 2 presents the foundation of our work: the component-oriented hierarchical modeling and simulation process. Section3 introduces the notion of models library with all its inherent specificities. Section 4 discusses our approach for the construction of a Web-based access to a models library. Finally, section 5 concludes this paper and provides some perspectives of work.

DAC 2002 June 10-14, 2002, New Orleans, Louisiana, USA

2. COMPONENT ORIENTED MODELING AND SIMULATION PROCESS

We introduce in this section the basic concepts of the hierarchical components-oriented modeling and simulation process : (i) the abstraction hierarchy, (ii) the notion of modeling component and (iii) a complete modeling and simulation process.

2.1 Abstraction Hierarchy

One of the most difficult tasks in the field of modeling and simulation of complex systems is to choose a good level of detail. In all Domains, models are built at a precise abstraction level. The abstraction level of a model determines the amount of information which is contained in the model. The quantity of information in a model decreases with the abstraction levels: a model described at a low abstraction level will contain more information than a model described at a higher abstraction level [2].

Determining the correct abstraction level refers to selecting the quantum of information that must be included in the model to help address the modeling goals. So, well defining the abstraction level is an important step in modeling, and is often done very early in the modeling process described in section 2. A model described according to several abstraction levels is said a "hierarchical model".

2.2 Notion of Modeling Component

Since many years, reusable modeling components appear to be the ideal paradigm for implementing simulation models [15, 11]. We define a modeling component as a model that can be described following various abstraction levels, presenting well-defined communication interfaces (called ports) and that can be reused in various contexts.

2.3 Component-Oriented Process

We introduce in this part a theoretical modeling process that can be applied to all component-oriented modeling processes. This process can be divided in eight steps given in Figure 1. The first step (box 1) is to formalize the problem: presentation, objectives to be reached, validation criterions,... The second step (box 2) is to identify the components to be used. Starting from this identification, components are : (i) selected in a models library, or (ii) built from scratch (box 3). Once all the components are well defined, the newly created ones are stored in a models library for further retrieval (box 4). The next step is to build the global model with the previously selected or defined components (box 5). The main advantage of this modeling process lies in this step, since it consists only in interconnecting components. Once the global model is well-defined and seems acceptable, the simulator is build and the simulation is performed (box 6). The next step (box 7) concerns the validation of the model defined in box 5. If the validation step presents good results, the model can be used (box 8), otherwise it is necessary to go back and perform changes either in the components identification (box 2), in the components selection (box 3) or in the global model building (box 5).

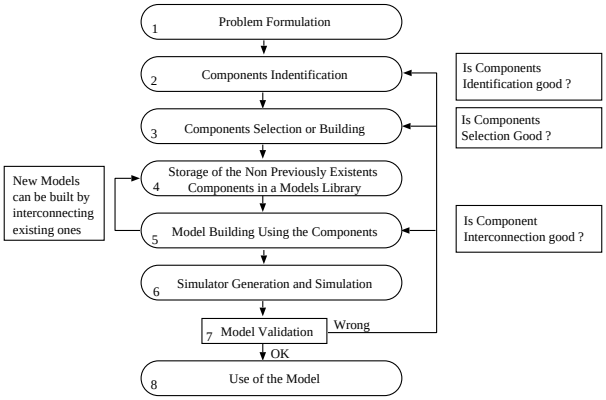

Figure 1: Components Oriented Process

3. DESCRIPTION OF A MODELS LIBRARY

When we deal with CAD (Computer Aided Design), a modeling component is most of the time a software component written in a given language and presenting some input and output ports. This component is used in a software framework (called modeling and simulation environment) able to handle it. A persistent model is a model that can exists before being used in the context of the environment, and that can exists even after its use. We define a models library as an object-oriented architecture, allowing a model designer to store and retrieve persistent models, directly reusable in their own modeling and simulation environment.

3.1 Basic Concepts

A models library is structured according to two paradigms: the application domains and the abstraction levels. Furthermore the three following requirements should be taken into account : a models library must allow the storage independence from the considered application domain, must manage an inheritance hierarchy between the stored models, and must manage the abstraction links between the stored models. The notion of library we introduce here is different from the one commonly used in the Computer Science. We can compare it to the notion of Object Database [9], since we want to store directly reusable models rather than software functions or objects used for well-defined tasks. One of the most important objectives of a models library, as defined in this paper, is to be independent from the storage mode. That means that a library must be able to store fundamentally different models coming from fundamentally different modelling and simulation environments. This independence implies that we must dissociate the contents from the format of the stored models, and also that the communication interfaces of a library with the external applications must be identical for all kinds of models.

We define, in order to be as precise as possible, two notions: "context-in" and "context-out" models. A context-out model is an abstraction of a model. It presents a structure allowing it to be stored in a models library. A context-in model is a context-out model extracted from a Library and formatted so as to be directly reusable in its environment.

In order to avoid a repetition of properties inside same kinds of models, a models library must deal with an inheritance between the stored models. This inheritance between a parent model and its children allows to store these shared properties inside special models (called parent models), dramatically simplifies the children models, and facilitates the maintenance of the whole Library. Furthermore,

this inheritance hierarchy inside a library provides all the classical benefits of object inheritance: automatic properties transmission and methods overloading if components are described using algorithmic functions. The importance of a good models library is that the model designer can be supplied with reasonable alternatives [12]. The choice between these alternatives can be strongly facilitated if the inheritance hierarchy is structured in a smart way.

3.2 Elements of a Models Library

Building a Models Library is as creating a high level representation of the models and their relations. In order to meet the requirements stated above, we define five kinds of objects to be stored in a Library object.

Domain elements allow performing a classification between theoretical domains of the stored models (Ex: DEVS Simulation [16], High Level Synthesis, VHDL Test).

Application Domain elements allow performing a classification between the application domain inside a given domain (Ex: Microelectronic, Energetic,...).

Classification Intermediate Model elements, belonging to an Application Domain, allow the creation of a classification hierarchy between the storage objects. This kind of model is not a storage model.

Inheritance Intermediate Model elements, which are storage models, allow the share of characteristics with its children through an inheritance mechanism.

Finally, Model File elements represent context-out models. It is the basic storage element of the Library.

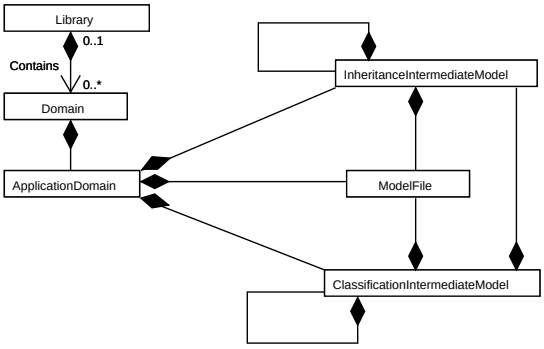

Figure 2: Links Between Elements

Figure 2 presents how are organized the links between these elements.

4. WEB-BASED ARCHITECTURE

We present in this section our approach for the definition of an architecture allowing to access the storage engine through a network or the Web. In the first part we introduce the basics of the Web-based architecture while the second part exposes the Remote Access package, the key of Web-based approach we have defined .

4.1 Basics of the Approach

The Web can serve as an operating system, and as a distribution channel for applications [7]. The main characteristics of the Web are: ease of navigation and use, ease of publishing content, new distribution models and enabling of a network-centric computing paradigm. Our Libraries Architecture is built on a core storage engine, and provides a set of interfaces enabling a remote access. This is one of the main features of our approach since it allows a design team to work on the same models stored on a storage server.

The basic idea is that the storage engine is running on a server and that the client access the models using a Web browser or directly from the environment upon a local network or even over the Internet, following the classical 3-tiers architecture. Clients do not need to know how the models are stored on the server, they should only access them for consulting, adding or removing them. We wanted to provide to a final user the easiest way to access the models libraries remotely. We found that the best approach was to use a Web browser since people are more and more aware of this kind of software even if they are not computer specialists.

This approach but appears viable only in the cases where the user wants to consult or manage a library. This thought led us to develop another complementary approach: if the final user wants to use the models library inside a modeling and simulation environment, we provide the computer specialist in charge with the environment the capacity to connect it with the storage engine using a set of simple APIs. The reader can note that we built these APIs in order to hide all the complexity of the storage engine.

4.2 The Remote Access Package

One of the main advantages of our libraries architecture is that it is independent from the format of the model. The storage engine included in the package Models Library contains two servers, a server acting as a file transfer server, and another one able to process instantiated and compiled objects. The Remote Access package uses a combination of these two servers in order to allow a remote access to the storage engine through interfaces defined in the Storage Engine package. These two servers use very simple communication protocols and two different connectors between the storage engine and the modeling and simulation environment.

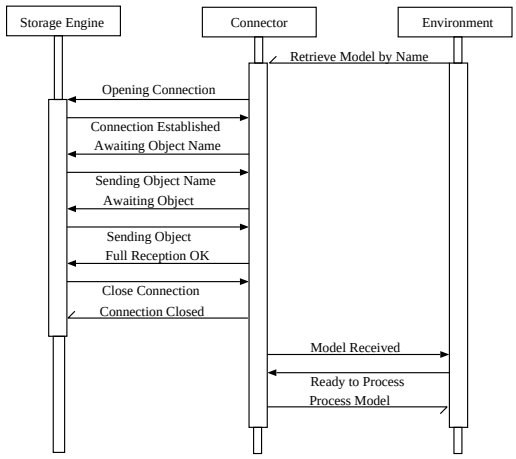

Figure 3: The Simple Model Object Transfer Protocol, SMOTP

The Simple Model Object Transfer Protocol (SMOTP, Figure 3) is used by the server in charge with the processing of previously instantiated and compiled models. Since these kinds of objects already contain all the characteristics of the associated modeling component, the transfer protocol

is very simple. When the connection is opened, the dialog between the connector and the storage engine consists only in transferring the name of the model and the associated object.

The Simple Model File Transfer Protocol (SMFTP) is used by the server in charge with models provided as description or sources files. The main difference with the SMOTP lies in the fact that the abstraction level must be specified, since it usually does not appear in the model file. However, these protocols are very similar and very simple. That was our objectives since we had in mind to keep the efficiency of the storage engine.

4.3 Software Realization

The implementation of the whole software packages has been performed using the Java language, and we used the servlet/Java Web Start approach for the remote part of our work. There are many advantages in adopting such an approach [8]: first, servlets are persistent since they are loaded once by the Web server and can maintain services between requests; Secondly, servlets are fast since they are loaded once, they offer much better performance than over CGI approaches; Lastly, servlets are platform-independent and extensible since they are written in Java and since they can take advantage of all the Java benefits.

We use a set of interfaces contained in the Storage Access package as a kind of "connector" between them. The main advantage of this approach is that modifications on the servlets or on the storage engine can be performed independently, since the Storage Access will not change.

In our current implementation, context-out models can be of three types: Compiled Java objects, Java sources files or C++ sources files. The Library Storage engine can deal with these three types and their own specificities. If the modeller wants to store compiled Java objects, the engine creates a Proxy object for each model to be stored. This Proxy is built from the ".class" file using a de-serialization mechanism. In order to avoid replication, the engine uses a single-instance architecture: there is only one instance of a model that lives in the server (this was inspired by the J2EE EJB model [1]). A modeller can also store source files in the Library. Our engine is able to deal with Java and C++ source files using some mechanisms based on the XML language, a text-based language allowing to dissociate the contents from the format of data [10]. The basic idea is to transform the source code of a model in XML following predefined DTDs (Document type Definition), and to instantiate and to associate an object in the engine with this file. This approach has many advantages: we can produce DTDs for any kind of programming languages in order to extend the possibilities, we can work independently on the file without performing tasks on the associated object, and we can modify directly from the Proxy object some parameters of the model.

5. CONCLUSION

We introduced in this paper our approach for the definition of models libraries enabling the reusability of modeling components. These models libraries are built on the concepts of modeling components, abstraction hierarchy and genericity of use. They allow the model designer to perform its model design very quickly, once models have been introduced in such a library. We provided also this user with a set of APIs allowing him to process his models through some Web-based APIs and graphical interfaces accessible using a simple Java-enabled Web browser. Our approach has been validated in two different domains: VHDL test and DEVS-based environmental modeling and simulation [3]. We have three main perspectives of work. We saw that we are currently able to store instantiated Java objects. The next step will be to develop the storage engine in order to enabling it to store C++ instantiated objects. This will be done with a massive use of XML-based techniques. The second perspective is to define a meta-language in order to help the models design. Finally, we want to study how to manage and maintain with the most efficiency a distribution of storage engines over a network and over multiple hosts.

6. REFERENCES

[1] K. Ahmed and C. Umrysh. *Developing Enterprise Java Applications with J2EE(TM) and UML*. Addison-Wesley Pub Co, 2001.

[2] P. Benjamin, J. Erraguntla, D. Delen, and R. Mayer. Simulation modeling at multiple levels of abstraction. In *Proceedings of the 1998 Winter Simulation Conference*, 1998.

[3] F. Bernardi, E. de Gentili, and J. Santucci. Reusable models integration in a devs-based modelling and simulation environment. In *Proceedings of ESS2001*, 2001. Marseille, France.

[4] A. Breneuse, J. Top, J. Broenink, and J. Akkermans. Libraries of reusable models: Theory and application. *Simulation*, (71), 1998.

[5] D. Cottrell. Electronic component information exchange (ecix). In *Proceedings of the DAC 1997*, 1997. session 35.2.

[6] G. Konduri and A. Chandrakasabn. A framework for collaborative and distributed web-design. In *Proceedings of the DAC 1999*, 1999.

[7] J. Kuljis and R. Paul. A review of web-based simulation: Whither we wander ? In *Proceedings of the 2000 Winter Simulation Conference*, 2000.

[8] K. Moss. *Java Servlets, second edition*. McGraw-Hill, 1999.

[9] ODMG. *ODMG 3.0 Specifications*. Morgan Kaufmann Publishers, 2000.

[10] W. Pardi. *XML in Action*. Microsoft Press, 1999.

[11] H. Praehofer, J. Sametinger, and A. Stritzinger. Building reusable simulation components. In *Proceedings of WEBSIM2000, Web-Based Modelling & Simulation*, 2000. San Diego, CA, USA.

[12] R. Rosenberg. The bond graph as an unified database for engineering system design. *Journal of Engineering for Industry*, 97, 1975.

[13] A. Umar. *Cient/Server Internet Environments*. Prentice Hall, 1997.

[14] P. van den Hamer, W. van der Linden, P. Bingley, and N. Schellingerhout. A system simulation framework. In *Proceedings of the DAC 2000*, pages 699–705, 2000.

[15] R. Wadhwani. Component and library management tutorial. In *Proceedings of the DAC 1991*, 1991. session 1.1.

[16] B. Zeigler. *Theory of Modeling and Simulation*. Academic Press, 1976.