A Domain Parser for a Generic Models Library

F. Bernardi, J.F. Santucci
University of Corsica, SPE, UMR CNRS 6134
Quartier Grossetti, B.P. 52
20250 Corte, France
{bernardi, santucci}@univ-corse.fr

Abstract

The computer modeling and simulation field is very large and complex. For a long time, a lot of formalisms and techniques have been developed, and this multiplicity has led to the development of many modeling and simulation environments incompatible with each others, even if they are based on the same formalism. Otherwise, since the studied systems are more and more complex, model reusability has become a major concern. In view of these notings, we decided to propose a contribution to the design of reusable models libraries that can be integrated in different modeling and simulation environments. We present in this article our approach to allow a genericity of use of such libraries using the notion of Domain Parser.

1 Introduction

For H. Vangheluwe, “depending on the context, modeling and simulation is often seen as a subset of Systems Theory, Control Theory, Numerical Analysis, Computer Science, Artificial Intelligence or Operations Research” [16]. Numerous Science fields have specific modeling and simulation techniques or formalisms but, on the other hand, a set of these ones are considered as “generic”. A good example of this kind of generic formalisms is DEVs (Discrete eVent System specification) introduced by B.P. Zeigler in the early eighties [18, 19]. However, some current researches aim to use simultaneously different formalisms. This kind of modeling approach is called the “multi-paradigm modeling” [17, 15].

The members of a modeling team often use different specific modeling formalisms, thus different specific modeling and simulation environments that are often pieces of software they themselves designed. They want also to be the most efficient possible in decomposing a given problem in sub-problems that can be solved using reusable models.

This concept of reusability is very important, and has been studied in a lot of researches [1, 2, 6, 14]. It is of course not specific to the modeling and simulation field, and can be found, for instance, in the software design field. The reusability model described in [11] claims that in this domain, “one of the most important activity is the acquisition, the instantiation and/or the modification of existing reusable components”. Nowadays, the concept of software component is widely accepted. One of the definition that can be found in the literature is that a software component is a software object presenting well-known interfaces [13]. Since the frontier between the software and the modeling and simulation fields is very thin, an analog notion has been developed: the notion of modeling component. This notion is often defined in the same way than the notion of software component. The difference is that the represented object is a model described in a given formalism, and not necessarily a software component.

A complementary notion to reusability is the notion of storage of component. A storage architecture (also called library) is a tool allowing researchers to store and to retrieve reusable components. The development of different modeling and simulation environments that use different formats for the description of the models raises a real storage problem. As we said earlier, these environments are often developed by the researchers that use them, and building different storage architectures for each of them can take a long time.

We think that it is necessary to have a federative tool allowing the researchers to store and to retrieve models described in different formats.

Thus, our goal is to propose a contribution to the design of a models library that can be integrated in various modeling and simulation environments. Some work has already been done, (see [3, 4, 5] for more explanations) and we present only in this article our approach concerning the genericity of use, using what we call a “Domain Parser”.

55
The paper is organized as follows: the two first sections are devoted to the presentation of the basics of our approach, the notions of context and of elements, while the Domain Parser is explicated in a third section. We finish the paper with a conclusion and some perspectives of work.

2 Context-in and Context-out Models

A storage architecture and a modeling and simulation environment perceive a same model in a totally different way. In the first case, the model is passive. It can not handle external entreaties. In the second case, the model is active since it is placed in a context of use. It can handle external entreaties arriving on its communication ports. Building a model storage architecture appears then as building an architecture allowing the representation of models out of their context of use.

In order to manage these two points of view, we introduce two notions, context-in and context-out models:

A context-out model is an abstraction of a model. It is defined or encapsulated in a special format allowing it to be stored in a library. All the stored models share this special format.

A context-in model is a context-out model extracted from its library and directly usable in its modeling and simulation environment. This extraction is in fact a format conversion.

The context-out format must be able to be adapted to all the model forms that the library designer could have to manage. This point implies very precise formal descriptions of the elements able to be stored in a library.

The notion of context is very important in our approach since it dictates all our design process, allowing us for instance to introduce different storage elements totally independent from any modeling and simulation environment. These elements are described in the following sub-section.

3 Elements of a Library

We call Element an object that can be stored in a models library (defined itself as an element). We define six different and complementary elements: libraries, domains, application domains, classification and inheritance intermediate models and model files. They can be classified in two categories: the classification elements and the storage elements.

3.1 Classification Elements

The classification elements are the simplest ones. They are used by the library designer in order to guarantee a good organization of his library, and so, to make easier the navigation in it. These elements are: the libraries, the domains, the application domains (also called ADs) and the classification intermediate models (also called CIMs). See [3, 4, 5] for more informations on the subject.

3.2 Storage Elements

The storage elements are probably the most important ones. They allow the persistence and the crossing between the two contexts previously defined. They are: the inheritance intermediate models (also called IIMs) and the model files (also called MFs).

Very briefly, IIMs define a properties sharing between two or more models that can be model files or other IIMs. The two storage elements have in common that they can encapsulate a part or a complete description of a model. “Full” model files (that do not depend on an IIM) are the most correct representation of context-out models.

Model files can represent two different types of context-out models. The first one is composed by the non-decomposable model files. These models are structurally the simplest ones since they are self-sufficient in terms of description. The second type is composed by the decomposable model files. These models are composed by an aggregation of other models and describe the couplings between them. Each of these decomposable models can be themselves reused in another one, constituting a hierarchy of models. A decomposable model is defined as a composed parent model in relation to its composite children models.

In a modeling and simulation environment, context-in models are described using a special format. Thus, following our approach, this format depends on its domain. In order to translate the format of a context-in model file into a context-out one, we use the notion of Domain Parser described in the next section.

4 Domain Parser

We chose to store our context-out models in a standard format using the XML language following a spe-
specific Document Type Definition (DTD, [12]).

In our approach, a context-out non-decomposable model file is composed of variables, methods and ports (in [19], B.P. Zeigler shows that any kind of complex system can be represented using this characteristics). The genericity of our approach relies upon the transformation of the context-in format in the context-out one. We use for that the notion of Domain Parser.

We call Domain Parser an object able to be used in two distinct modes, and able to analyze or to create a file that describes a context-in model. A Domain Parser relies upon a separation methodology of the extent of the model from its description format. Thus, the selected approach consists in defining a separation methodology for each domain in a library. Using such a methodology, a Domain Parser allows the user to transform a context-in model to a context-out one. The main advantage of this approach is that, never mind the model is, it can be placed context-out. Our implementation of a Domain Analyzer is based on the Builder Design Pattern [10] used in order to “separate the construction of a complex object from its representation so that the same construction process can create different representations” (Figure 1).

A Domain Parser is represented by a class (DomainParser) that uses three main objects: a parser specialized in decomposable models (DecomposableModelFileParser), one specialized in non-decomposable models (NonDecomposableModelFile), and another one for the automatic determination of the model type (TypeSelector). These parsers are called specific parsers. So, the addition of a separation methodol-
ogy for a given domain implies the implementation of four abstract classes (the fourth one, Parser, contains common methods to the two real parsers).

Space is missing in this article to show a real example, but the reader can found more precisions in the references previously given.

5 Conclusion and Perspectives

We presented in this article the management of the genericity of a models library using the notion of Domain Parser. We can note that this work has been implemented and validated using the JDEVS modeling and simulation environment in the ecological modeling field [9, 8].

We have several perspectives for this work. First, we want to add more domains to the Domain Parser (we are studying the VHDL test in a current research). Another point is that we want to add a graphical representation to the models using the SVG language [7]. Though, our main perspective is to develop an approach allowing the context-in models to be translated from a domain to another using our context-out representation.

References


[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, 2002.


