Discrete-Event Behavioral Modeling in SESM: Software Design and Implementation

Shridhar Bendre & Hessam S. Sarjoughian[1]
Arizona Center for Integrative Modeling & Simulation
Computer Science and Engineering Department
Fulton School of Engineering
Arizona State University, Tempe, Arizona
{shridhar.bendre | sarjoughian}@asu.edu

Keywords: Behavior Specification, Graphical Modeling, Model Persistence, Simulation Model Generation.

ABSTRACT

The analysis and design of many contemporary large-scale and complex systems such as agent based systems can benefit from an environment which can support component-based hierarchical modeling. To examine these systems via simulation and in particular in terms of model validation, it is important to use model repositories. This supports designers to systematically study alternative system structures and behaviors. The Scalable System Entity Structure Modeler with Complexity Measures (SESM/CM) offers a basis for modeling modular simulatable and non-simulatable models with complexity metrics. This SESM/CM is extended to allow modelers capture and store some aspects of a system's behavior. This extended environment allows transformation of models to partial DEVSJAVA models which need to be extended for execution in the DEVSJAVA simulation environment.

1 INTRODUCTION

Architecture of a system is mainly influenced by its requirements and in particular its quality attributes [Bass98]. The quality attributes of a system and software are categorized as runtime (e.g., performance and communication patterns) and non-runtime (e.g., maintainability, availability and reusability). Generally, it is easier to achieve desired quality attributes in smaller systems by means of traditional software or system engineering processes such as design, coding and testing. But as these systems grow in size and complexity, many existing modeling approaches in support of analysis and design become inadequate or impractical. The structures of such systems can be defined using composition and specialization techniques. In particular, a system can be described as a set of atomic and composite components which can be combined using ports and couplings to form increasingly larger models. To aid the modeling of large-scale systems, it is important to employ repositories like Relational Database Management Systems (RDBMS), since they provide systematic, scalable and efficient medium for storing and accessing models [Sar02, Fu02]. One of the most important benefits of using a database repository is its support for reusability. Having a scalable, reusable model repository further supports simulation and consequently model validation, which is the key in the system development life-cycle.

Simulation is an imitation of the operation of a real-world or imaginary system over time. Simulation modeling can be used both as an analysis tool for predicting the effect of the changes to existing systems and/or as a design tool to predict the performance of a new system under varying circumstances. In order to validate a system, the modeler needs to simulate and compare a model and its behavior against a system under different experimental conditions (subjecting the model to input scenarios and observing output scenarios under specific initial conditions). Input and output scenarios are mainly characterized by the input and output variables, their data types and their values while model initialization includes the specification of state of the model, which is characterized by the initial values of the state variables of the model. Large-scale hierarchical models can be built by reusing persistent models. This in turn requires the storage of structural and behavioral aspects of models in a permanent repository such as database. This storage helps the modeler to conduct simulation and take steps toward model validation.

The goal of this paper is to capture the behavioral aspects of atomic models and to support generation of their simulatable counterparts toward simulation validation. This involves the design and techniques to capture dynamic characteristics of atomic models. This includes representation and manipulation of input, output and state variables in the relational databases. To aid the modelers, it is also necessary to develop an interface to support the capturing of these be-

[1] Corresponding author.

Accepted for Publication

havioral aspects of atomic models. Since a model may have parts that themselves do not qualify as “simulatable models”, it is important to support representation and storage of non-simulatable models (NSM) [Sar05, Ben04]. List, bag or set are examples of non-simulatable model and can be used as components of simulatable atomic models.

2 BACKGROUND

We use the Scalable Entity Structure Modeler with Complexity Measures (SESM/CM) [Sar02, Fu02, Smo03] as the underlying modeling environment. SESM/CM is suitable for developing component-based hierarchical models. It offers a basis for modeling behavioral aspect of atomic models by providing the structural specification and storage of the model. This environment shares some basic concepts from component-based model specification [Boo99], systems theory [Wym93], and System Entity Structure (SES) [Zei84, Roz93]. These offer different schemes to organize alternative model or system structures. The fundamental object of the SES formalism is an entity (model) which can represent an object having identification, attached variables and a range set. This range set is an enumeration of values that the variable can assume. This entity can be of two types, atomic entity and composite entity. Atomic entities cannot be broken down into sub entities, while composite entities are decomposed into other entities, either atomic or composite. SES provides three types of relationships among the entities, namely aspect (alternative representation of the system or model), decomposition (Part-Whole relationship) and specialization (Parent-Child relationship).

2.1 System Entity Structure Modeler with Complexity Metrics

SESM/CM is based on a new approach to modeling large-scale systems [Sar02]. The approach supports modeling of a system using three complementary types of models called Template Model, Instance Template Model, and Instance Model. The basic approach is component-based modeling where a system is viewed as a collection of components which are composed using input and output ports and couplings. A template model specifies atomic and composite models as components with input and output ports and values. An atomic template model specification contains state variables and a name. The components of each of these three model types are restricted to have the same type – a template model can have other template models and not models of instance template or instance models. A composite template model specification has couplings and a name. Composite template models are restricted to have atomic and/or composite template models as children. Furthermore, the name assigned to atomic and composite models must be unique such that any composite model can be uniquely identified within its hierarchical decomposition. A composite template model is defined to have a hierarchy of length two. An instance template model is the same as a template model. This type of model is defined to have a finite hierarchy of length greater than two. Furthermore, this model does not specify multiplicity of a model component within any composite model – a model can have one to a finite number of copies of the same instance template model. An instance model is an instantiation of an instance template model where the multiplicity of model instances is specified.

A Template Model can be specialized into one or more specialized components. The ability to specialize complements composition. Composition and specialization together support different types of models depending on the intent of the modeler. These types of models need to be constructed in three stages - template model, instance template model, and instance model developments - as described above. A modeler first creates Template Models, Instance Template Models, and then Instance Model in a sequential manner. In the stage of instance model generation, a modeler decides which specialized model component is to be used. Of course, it is possible to iterate among these stages. As noted above, one essential advantage of this modeling approach is the ability to create alternative models depending on desired alternative resolutions and aspects.

It has a hybrid client-server type of architecture, which combines the features of different flavors of client-server architectures [Fu02]. It is composed of four main constituents – Client (user interface), Network Environment (communication medium), Server (modeling engine) and Database Management System (DBMS).

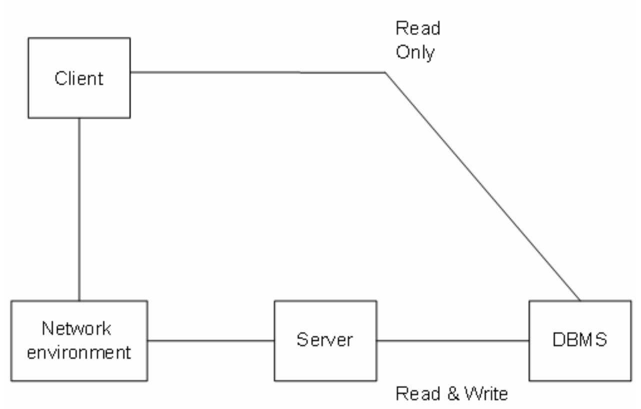


Figure 1. SESM/CM Client-Server Architecture

DBMS stores the model data in hierarchical manner, Server initializes and manipulates the model database as per the users request, Client allows users to display and modify the models in the database while the Network Environment acts as a channel between client, server and database.

Client and Server independently initialize and maintain their connectivity with the database. Client has “Read-only” access which means it can independently read the model data from the database while for writing to the database, it has to communicate with the server which writes the data to the database. The server has both “Read and write” access which means it can read and write data from and to the database. In this way, SESM/CM allows multiple readers but only single writer of the data. In this architecture, client and server are loosely coupled and hence result in better design and implementation.

3 RELATED WORK

Large scale systems are increasingly developed by using model-based analysis and design techniques. As these systems grow in size and complexity, a methodical approach is required to have a repository, which can provide the capabilities like usability, scalability, modifiability and storage of models. Relational database is an appropriate option for these repositories as it provides functionalities like creation, modification, storage and most importantly reuse of the stored models. It offers modular hierarchical representation of the models in the database by providing the relationships like composition (Part-Of relationship) and specialization (IS-A relationship). It allows user to enforce the constraints on the models stored in the relations set. It also provides scalability and flexibility by providing the data independence where data is decoupled from the application development. And finally, it uses Structured Query Language (SQL) as an interaction medium, a standard language for the relational databases; which is important for application portability.

An approach for modeling a system founded on the principles of object-orientation and system theory is the Unified Modeling Language Real-Time (UML-RT) which extends UML [Boo99, Dou04]. The modeling approach is appropriate for modeling systems ranging from enterprise information systems to distributed web-applications to real-time embedded systems. UML-RT and SESM/CM are similar in how they represent a system’s structure (i.e., component and relationships) as both of them support is-a and part-of relationships. But they are different as UML-RT is intended for software specification [Fow99] while SESM/CM is targeted for representing simulation models. A major difference that is relevant to this work is how UML tools and SESM/CM store models. Tools such as Rational Rose store models as flat files whereas SESM/CM stores models in a relational database. This allows SESM/CM to offer better scalability and reusability.

4 MODELING BEHAVIORAL SPECIFICATION AND SIMULATION

The modeling approach shown in Figure 2 supports modeling of a system using SECM/CM template instance template, and instance model types [Ben04,Sar05].

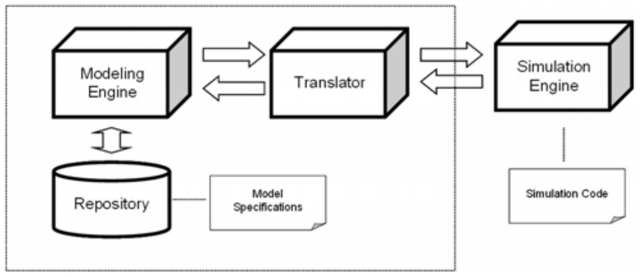


Figure 2. Modeling Framework

Model specification defines a system in terms of its structure and behavior. Structure of the system is defined in terms of name, ports and couplings while the behavior of the system is defined in terms of the behavior of atomic models. Hence, to specify the behavior of the system, it is necessary to specify the behaviors of all the atomic models inside the system. The behavior of an atomic model is defined in terms of dynamic characteristics of the model such as input variables, output variables, state variables and state transition functions as shown in Figure 3.

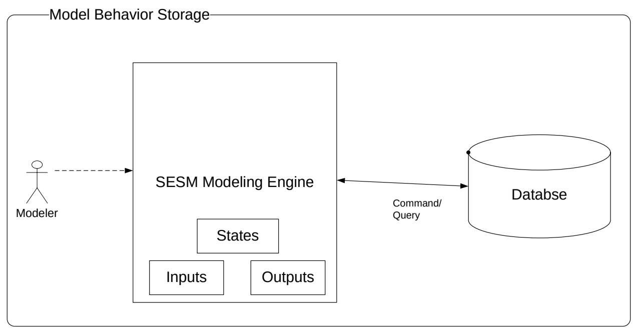


Figure 3. Behavioral features of SESM/CM

Behavior of the model is defined as the change in the state of the model. Discrete Event System Specification defines the change in the state of the model as a consequence of some event occurred to the system or occurred within the system. These events are mainly categorized into inputs arrived at the system, outputs sent out from the system and change in the internal state of the system. Every model defined in DEVS formalism [Zei03] is provided with input as well as output ports for the communication with the other atomic and/or coupled models which are connected to each other by means of couplings. The inputs and outputs are in the form of variables which has defined name, data type and value(s). Every input/output port is associated with zero or more variables while every input/output variable must be associated with either input or output port.

State of the system at a particular point of time is defined in terms of all of the state variables associated to its atomic models. State variables are associated directly to atomic model unlike port variables, which are associated to the model through ports. Similar to port variables, state variable are also defined in terms of name, data type and value(s). As coupled model doesn't have a defined state, there are no state variables associated with it, while each atomic model is associated with one or more state variables. Values of all the state variables collectively define the state of the model.

In addition to these models, it is also important to represent non-simulatable models which may be used as part of atomic models. These models are distinct compared with the template models since they do not have input/output ports. Such non-atomic models are referred to as non-simulatable since their behavior is not time-dependent. Examples of these models are object-based user defined complex data structures such as a list or a queue, which are useful to hold multiple values. As stated above, input-output-state variables are defined in terms of name, data type and value(s). The data type of these variables is an important aspect. This data type can be divided two types; either primitive data type (supported by the programming language such as integer, character, string, etc) or non-simulatable (NSM) models.

4.1 Database Schema for Atomic Model Dynamics

Models developed in SESM/CM are primarily structural. They are described and stored in a relational database in terms of structural features of the model components such as identity (i.e., model name), hierarchy (i.e., decomposition), input/output interface (i.e., port names) and their creation time. In order to execute (simulate) these models to observe their behavior in response to input stimulus, they need to be extended in terms of behavioral aspects of the model. In particular, it is important for an atomic model specification to support modeling of input and output variables, state variables, and functions. Reusability of structural and behavioral aspect of these models can be achieved by storing them in a database [Sar05]. These extensions, in terms of new entities, relationships and constraints, to support the behavioral aspects are shown in Entity-Relationship diagram in Figure 4 [Ben04].

5 GENERATION OF SIMULATION MODELS

Once the model is specified in terms of variables, the specification of a model needs to be transformed into simulation code for execution by one or more simulation engines. This is a two step process as shown in Figure 5.

5.1 Database to XML Transformation

There are various choices for the storage type of the transformed models, but we choose to convert and store them as a well-formed XML document as XML is considered as the best option to handle structured or semi-structured data/documents. The XML document contains the information about the structure of the model such as model name, input port number and names, output port number and names, information about the sub-components and the couplings between them and behavior of the models in terms of inputs, outputs, state variables and non-simulatable models. This will facilitate the component based approach for model validation.

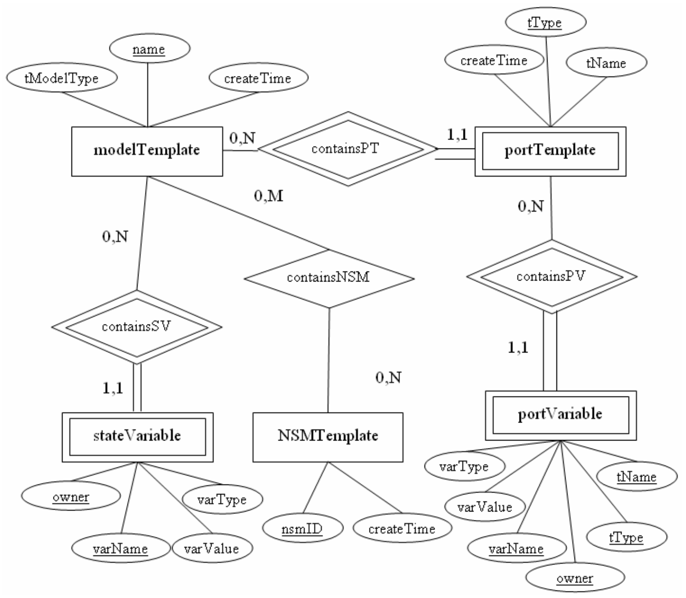


Figure 4. Partial SESM/CM E-R Diagram

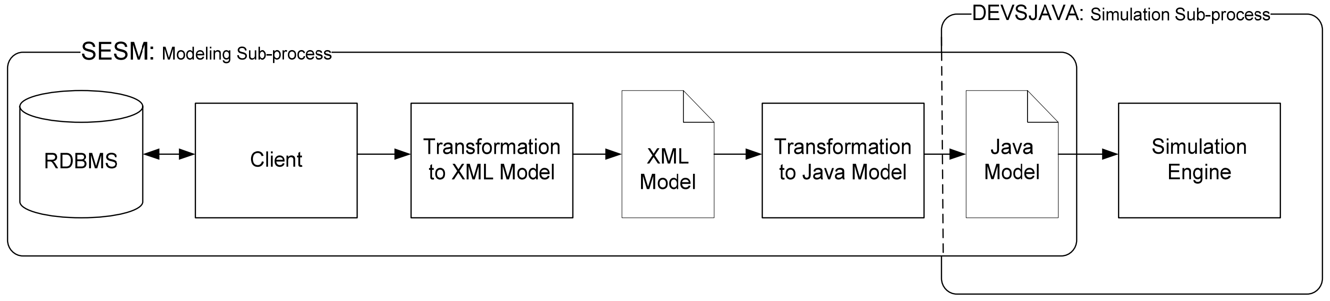

DEVSJAVA: Simulation Sub-process
SESM: Modeling Sub-process
RDBMS
Client
Transformation to XML Model
XML Model
Transformation to Java Model
Java Model
Simulation Engine

Figure 5: Transformation from SESM/CM to XML to DEVSJAVA

5.2 XML to Java Model Transformation

Once created, these XML models with structural and behavioral capabilities need to be simulated to test their completeness and correctness. To demonstrate these capabilities, we employed DEVSJAVA [ACIMS04, Sar03, Zei03] which support execution of models written in the Java Programming language. Therefore, in order to simulate models stored in SESM, they need to be transformed into Java syntax which can be compiled and executed in DEVSJAVA. Hence, it is necessary to develop a modeling-to-simulation mapping to transform atomic, coupled, and non-simulatable models into forms that can be compiled using the Java compiler and executed using the DEVSJAVA simulation engine.

6 SESM/CM DESIGN OVERVIEW

As discussed earlier, SESM/CM uses hybrid architecture as shown in Figure 1. By design, the SESM/CM system includes the SESM package, the Network Environment package, SESM client, and SESM server as shown in Figure 6.

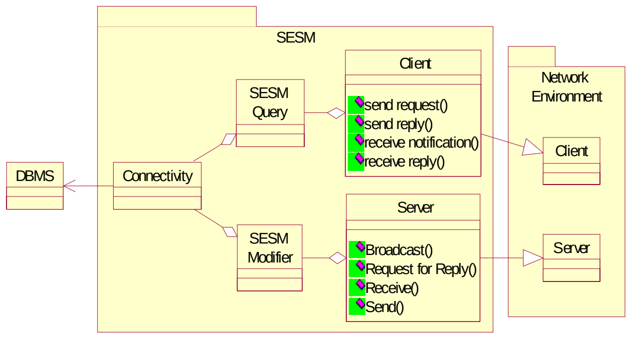


Figure 6. SESM/CM System Components

The SESM package should serve as an API used to access the SESM representation model data stored on the DBMS. There are three main components in the SESM package; Connectivity, SESM Query, and SESM Modifier. The Connectivity component is used to connect to the DBMS. It handles all the communication between the SESM system and the DBMS. The SESM Query component retrieves data from the DBMS using SQL and maps the data into object-oriented SESM models. The SESM Modifier component modifies the SESM representation models on the DBMS. The component translates the requested modification into appropriate SQL statements. The SESM Server extends the Server provided by the Network environment package. Messages received by the SESM Server are processed, and modifications are performed accordingly. The SESM Client utilizes the SESM Query component to retrieve and display the SESM representation model visually on its graphical user interface (GUI). The user also modifies the model through the SESM clients GUI. The network Environment package manages the communication between the SESM Client and the SESM Server by providing the components that can be extended by the SESM Client and SESM Server.

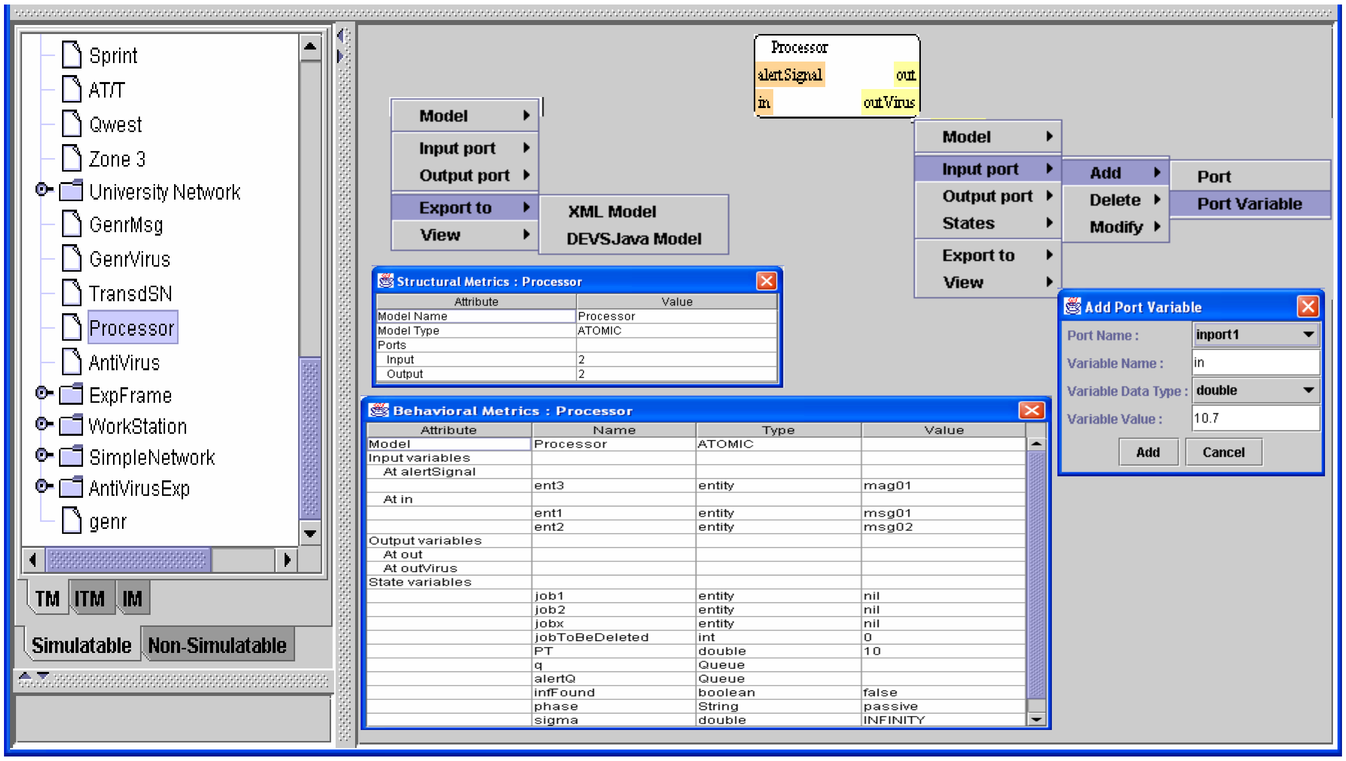
6.1 USER INTERFACE DESIGN

Graphical User Interface of SESM/CM is also extended to support the modeler with features to specify the behavioral aspects of atomic model and to transform the models to achieve simulation. These extensions involve extensions to support Non-Simulatable Model Tree (NSM) as shown on the left-hand side in Figure 7. Also, simulatable model trees and non-simulatable model tree are separated from each other. It also supports the creation of new NSM models. A new pop-up menu is provided for NSM trees non-leaf node, which allows a user to add new NSM models on a server. On server, NSM model name is stored in the database along with the creation time, while the model source code is stored on the server as a flat file at a specified location.

Visual model command menu is restructured from the previous version of modeling environment i.e., SESM/CM. The rationale for changing this is to keep the consistency between the System View (i.e., mathematical representation of model), SESM/CM views/GUI (i.e., graphical or logical representation of model) and Database (i.e., structural and relational representation of the model). User interfaces are provided to capture behavioral aspects. A sample is shown in figure 7 for “Add Port Variable”. Menu items are added to support the exporting SESM/CM models to XML and Java models and to show the structural metrics and behavioral information of the model.

7 CONCLUSION

Modeling and Simulation approach is useful for analysis, design and development of many types of systems including agent-based systems. Since there is increasing need for agent-based systems and their inherent representations as a collection of objects, the extended SESM/CM environment supports their analysis and design. Simulation model of a system can be developed in a systematic fashion to study the structural and behavioral aspects of the system over time. For this, modeler needs to specify the model in terms of its structure and behavior and to make them persistent to achieve model reusability. This paper has concentrated on specifying the behavioral aspects of atomic models in terms of their input, output and state variables and their storage in the relational database to achieve the reusability and further transformation of these models into simulation compatible format. This approach can help with validation of simulation models.



Structural Metrics : Processor

Attribute	Value
Model Name	Processor
Model Type	ATOMIC
Ports	
Input	2
Output	2

Behavioral Metrics : Processor

Attribute	Name	Type	Value
Model	Processor	ATOMIC	
Input variables			
At alertSignal			
	ent3	entity	mag01
At in			
	ent1	entity	msg01
	ent2	entity	msg02
Output variables			
At out			
At outVirus			
State variables			
	job1	entity	nil
	job2	entity	nil
	jobx	entity	nil
	jobToBeDeleted	int	0
	PT	double	10
	q	Queue	
	alertQ	Queue	
	infFound	boolean	false
	phase	String	passive
	sigma	double	INFINITY

Add Port Variable

Port Name :	inport1
Variable Name :	in
Variable Data Type :	double
Variable Value :	10.7

Add Cancel

REFERENCES

[ACIMS04] ACIMS, http://www.acims.arizona.edu/.
[Bass98] Bass, L., Clemens P., Kazman R., *Software Architecture in Practice*, The SEI series in Software Engineering, 1998, Addison-Wesley.
[Ben04] Bendre, S., "Behavioral Model Specification Towards Simulation Validation Using Relational Databases," Master Thesis, 2004, Computer Science and Engineering Department, Arizona State University, Tempe, Arizona.
[Boo99] Booch G., Rumbaugh J., Jacobson I., *The Unified Modeling Language Use Guide*, 1st Edition, Pearson Education, 1999.
[Dou04] Douglass, B.P., *Real Time UML: Advances in the UML for Real-Time Systems*, 3rd Ed., 2004.
[Fu02] Fu, T. S., "Hierarchical Modeling of Large-Scale Systems using Relational Databases", 2002, Electrical and Computer Engineering Department, University of Arizona, Tucson, Arizona.
[Roz93] Rozenblit, J.R. and B.P. Zeigler. Representing and Construction System Specifications Using the System Entity Structure Concepts, Winter Simulation Conference, 1993, Los Angeles, CA.
[Sar02] Sarjoughian, H.S., "A Model for Design of Scalable Modeling of Modular, Hierarchical Systems," Internal Report, 2002, Computer Science & Engineering Deptartment, Arizona State University, Tempe, Arizona.
[Sar03] Sarjoughian, H.S., Singh, R.K, "Building Simulation Modeling Environments Using Systems Theory and Software Architecture Principles," Advanced Simulation Technology Conference, p. 99-104, April, Washington DC.
[Sar05] Sarjoughian, H.S., "A Methodology for Component-based Modeling of Large-scale and Complex Systems," 2005, in preparation.
[Smo03] Mohan S., "Measuring Structural Complexities of Modular Hierarchical Large Scale Models," 2003, Computer Science and Engineering Department, Arizona State University, Tempe, Arizona.
[Wym93] Wymore, W.A., *Model-based Systems Engineering: An Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design*, 1993, Boca Raton, CRC.
[Zei84] Zeigler, B.P., *Multifacetted Modelling And Discrete Event Simulation*, 1984, Academic Press.
[Zei03] Zeigler, B.P. and Sarjoughian H. S., Introduction to DEVS Modeling and Simulation with JAVA: Developing Component-based Simulation Models, 2003, http://acims.eas.asu.edu/PUBLICATIONS/publications.shtml.