A Scalable Component-based Modeling Environment Supporting Model Validation

Hessam S. Sarjoughian
Arizona Center for Integrative Modeling & Simulation
Computer Science & Engineering Department
Fulton School of Engineering
Arizona State University, Tempe, AZ
Email: sarjoughian@asu.edu

Keywords: Component-based Modeling, Model Development Process, Model Persistence, Scalability, Transformation, Validation, Visual Modeling.

ABSTRACT

Simulation validation remains a central and growing challenge, especially when models are large-scale and complex. To advance rigorous simulation validation, we have developed a component-based modeling environment called Scalable Entity Structure Modeler (SESM), which is based on the multi-aspect, multi-resolution component-based modeling paradigm. Using a multi-view graphical modeling interface, modelers can specify and compose large and complex models with arbitrary hierarchies. Models can be manipulated and stored directly in standardized relational databases—simplifying model specification and simulation experimentation tasks. A key advantage of a persistent model-base is that it enables modelers to measure a model's structural and some behavioral complexity metrics (e.g. number and types of components, inputs, outputs, and states of models).

A realization of this approach called Scalable Entity Structure Modeler with Complexity Metrics (SESM/CM) is developed. It supports syntactic and semantic correctness of domain-neutral properties of models given a set of well-defined relationships among model components and their interactions. The model components in the model-base can be transformed into XML and subsequently simulation model components suitable for simulation environments such as DEVSJAVA. The scalability and transformation features of the approach offer a sound basis for simulation at varying levels of abstraction, including arbitrary yet systematic validation of models in terms of their observable input and output relations and state transitions. Direct support for incremental configuration of simulation experimentations and executions is important for large-scale simulation validation. To this end, the model-base logically lends itself to the management of simulation experiments. This approach supports formal specification of models, design of experiments, and observation of simulation executions. The proposed modeling approach is illustrated using a computer network example. We discuss simulation validation and future research directions aimed at supporting integrative modeling and simulation of enterprise, supply-network systems.

ABOUT THE AUTHOR

Hessam S. Sarjoughian is Assistant Professor of Computer Science & Engineering department at Arizona State University (ASU), Tempe, Arizona. His research interests include multi-formalism model specification, collaborative modeling, agent-based simulation, and software architecture. He is Co-Director of the Arizona Center for Integrative Modeling and Simulation (ACIMS). He led the establishment of the Online Masters of Engineering in Modeling & Simulation in the Ira A. Fulton School of Engineering at ASU. His modeling & simulation educational activities are with the Society for Modeling and Simulation International (SCS) and the Modeling & Simulation Professional Certification Commission (M&SPCC).

INTRODUCTION

There is a consensus among researchers and practitioners that modeling and simulation activities are growing in scale and complexity (e.g., see (Fishwick 1995)). At the same time modeling has become one of the most important and challenging tasks in creating and managing simulation models that can serve a variety of needs (Davis and Anderson 2004). It is, therefore, useful and necessary to enable modelers and simulationists alike to build models using simple yet well-defined modeling techniques.

To develop models in a systematic fashion, it is important to separate a system from its models (Sarjoughian and Cellier 2001). As shown in Figure 1, a system can have any number of models depending on the objectives of the simulation modeling effort. Generally, models of a system can be differentiated depending on choices made about its aspect, resolution, and hierarchy. Models can specify distinct *aspects* of a system. For example, for a computer network system, a model may specify a network of computers and links to evaluate network topologies to provide service under high bandwidth media. Alternatively, a multi-stage computer network model may be developed to study schemes to protect a network against infected messages. For an aspect of a system, its model may be specified at *multiple levels of resolution*. One computer model may include stochastic or constant processing time, for example. Aside from these, a model specification may have some arbitrary *hierarchy*. An example is an intra network modeled as a single computer or as a collection of computers and switches interconnected with high-speed links.

Hierarchical models are defined using composition and specialization concepts. These modeling artifacts are necessary to specify multi-resolution and multi-aspect models of systems in terms of their structures and behaviors (Davis 1995).

As mentioned above, a system's models can be distinguished from one another depending on the desired resolution and the aspect of the system that is being modeled. In system-theoretic terms, an atomic model's resolution can vary based on its number of inputs, outputs, and state variables as well as the complexity of operations required for processing inputs and producing outputs. For a composite model, the resolution is determined by its components and their relationships.

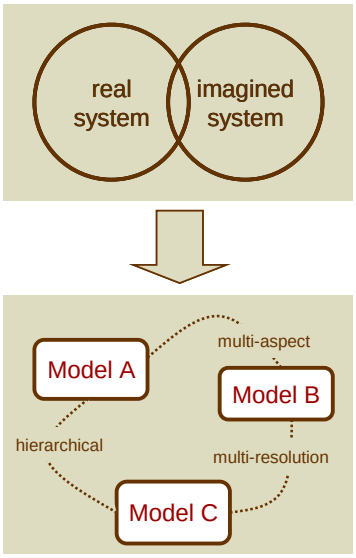


Figure 1: Systems and Model Relationships

A low-resolution model may be useful (and necessary) for concepts of operations whereas a high-resolution model may be desirable for design specification. A low-resolution model has fewer inputs, outputs, and states compared to its high-resolution counterpart. Aside from having simpler structural properties, the operational properties of a low-resolution model exclude details that are contained in a high-resolution model.

Given a system's different perspectives—e.g., conceptual analysis vs. engineering design—it is impractical to define a universal model that can satisfy what are often contradictory objectives. One model may be needed for designing a controller of a system while another may be needed for building the system. The complementary multi-resolution and multi-aspect model specifications show the importance of supporting *development and management of a family of models* for a system.

In the remainder of this paper, we describe the Scalable System Entity Structure Modeler (SESM) framework. In Sections 2 and 3, we present the SESM framework and its modeling paradigm. An example illustrating the features of SESM is described in Section 4. In Section 5, we describe related research and the role of the proposed environment as a foundation for model validation (Sargent 1994) which is founded on formal specification of models, design of experiments, and observation of simulation executions. We conclude with future research directions in Section 6.

A SCALABLE MODELING FRAMEWORK

A modeling framework should support specification of models using a general-purpose component-based modeling paradigm or one that can support multiple modeling specifications (e.g., continuous and discrete-time). In this framework, we consider system-theoretic models where every model component has inputs, outputs, states, and functions that operate on inputs and states to generate outputs. Advanced popular modeling and simulation tools are based on state and time-based component specifications (e.g., (Modelica 2000)).

As shown in Figure 2, it is important for the framework to provide database, visualization, and translation capabilities in order to handle complex and large-scale model specifications and their mapping to target simulation languages. The modeling module contains the syntax and semantics of the modeling paradigm—i.e., Scalable Entity Structure Modeler. The database module enables modelers to create, access, and manipulate persistent model specifications. A visualization module is important for ease of use (e.g., viewing complex relationships among model components and their properties). The translator module supports generation of simulation models from the stored specification models. Simulatable models are created by mapping model specifications into particular simulation code that can be executed by alternative simulation engines.

Collectively, the modeling engine, database, and translator allow modelers to develop complex models. Within such a framework, scalability and complexity related to validation and verification can be undertaken and managed in a principled manner. That is, modelers may specify a family of models in an iterative and incremental fashion where model validation and simulation verification impediments can be better overcome.

The Scalable Entity Structure Modeling (SESM) methodology provides a novel approach to representing a family of models that have well-defined relationships and can serve different purposes. For complex models, we can use alternative hierarchies, under which different models can be specified to represent different, unique aspects of a system's structure and/or behavior. The specifications of these models may be mutually exclusive depending on whether they share model components and what type of model hierarchy is used given decompositions and specializations within the system.

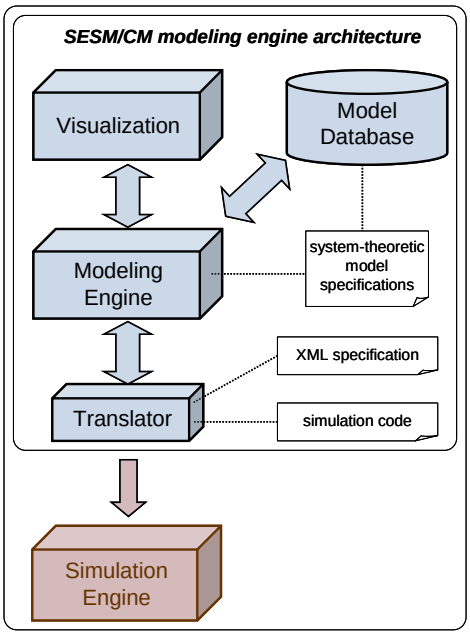


Figure 2: Conceptual Architecture Supporting System-Theoretic Model Specification and Simulation Execution

MODEL SPECIFICATION APPROACH

Systematic representation and manipulation of scalable multi-aspect and multi-resolution models depends on having multiple, complementary model types. Both logical and visual representation must be supported. An essential requirement for a modeling engine, therefore, is to represent models and guarantee their consistency (as stored in the database) as well as their dynamic visualization. That is, the modeling engine has to account for static and dynamic model creation and manipulation. Clearly, consistency is especially important as the scale and complexity of models increase. Consistency of a family of models depends on the uniqueness of model component compositions and specializations and the ability of the modeling engine to ensure that every change to a model is uniformly applied to the database.

The modeling approach proposed here is targeted for models that have well-defined external input/output interfaces and internal structures. We define three complementary types of models called **Template Model** (TM), **Instance Template Model** (ITM), and **Instance Model** (IM) (Sarjoughian 2001).

The separation of a model in terms of TM, ITM, and IM has three advantages. First, complex structural and

behavioral parts of a model can be specified using well-defined relationships and thus handled in a step-wise fashion. The Template Models capture individual model components where a component can have at most a hierarchy of length two and each model can be specialized into one or more specializations. The Instance Template Models are extended from the Template Models. They are devised for introducing multiple-levels of hierarchy using decomposition and specialization schemes. Instance models are instantiations are generated from the instance template models for target simulation environments.

Second, since the model specifications are stored in a relational database, a model can grow in size to thousands of components without any significant performance penalty. Third, graphical modeling can scale well for developing and manipulating large models. Given these capabilities, modelers can develop models at different, complementary levels of details (multi-resolution, multi-aspect) given the separation afforded by Template Model, Instance Template Model, and Instance Model.

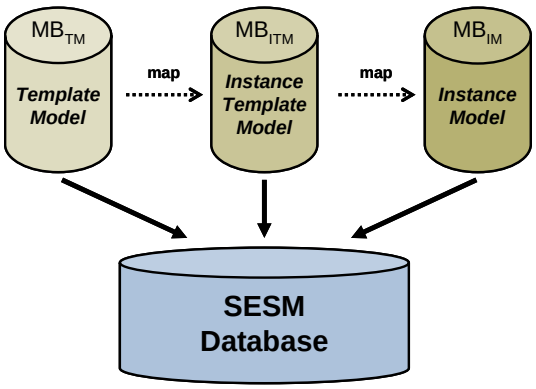


Figure 3: SESM Database with Conceptual TM, ITM, and IM Model Bases

Given these model types, we can conceptualize three distinct model repositories[1] (or model bases). These are called MB_{TM} and MB_{TIM}, MB_{IM}. The relationships (mappings) among these model types are characterized in terms of two concepts. First, the Instance Template Models extend the specifications included in the Template Models by allowing the former models to be configured hierarchically given alternative decompositions and/or specializations. Second, Instance Models are created from Instance Template Models by selecting among available alternative choices (i.e., specializations) therein.

Primitive and Composite Models

To specify Template Models, Instance Template Models, and Instance Models, it is useful to categorize each model as either *primitive* or *composite*. Primitive and composite models are synonymous with atomic and coupled models, concepts and techniques that are commonly used in simulation and software design.

Primitive models cannot have any components, but can be specialized. These, in general, are simple to conceptualize and generally straightforward to develop since each model stands alone. Composite models, however, have parts and can also have specialization relationships—i.e., a composite model can have has-part or kind-of relationships. We note that while the SESM modeling approach does not support the association, dependency, and realization relationships as commonly used in component-based software analysis and design modeling, the has-part and kind-of relationships support modeling rich and complex simulation models.

Primitive Template and Instance Template Models

A primitive *Template Model* in model base MB_{TM} has a finite number of input and output ports. It is not aware of having any parent/child or siblings relationships with any other (primitive or composite) template model. A primitive model can be specialized into one of a finite number of primitive template models where the former specializes the latter. Each model can be uniquely identified based on its name, state variables, functions, and/or input/output interface within its given model base. Furthermore, primitive template models can only be used as elements of composite template models.

A primitive *Instance Template Model* in model base MB_{ITM} is an instantiation of a primitive template model. Each instance template model is an exact copy of a template model. All instances of a primitive temple model are distinguishable from one another using their assigned (given) names. A primitive instance model can only be created when its corresponding primitive template model exists.

Primitive models have limited use by themselves since the kind-of and has-part relationships are not defined for them. Such relationships are essential in

[1] The separation of databases is conceptual. All models are represented in a single database, but can be distributed if necessary.

representing hierarchical complex model structures (parent-child, alternative choices, and their mixture of hierarchical forms).

Composite Template and Instance Template Models

A composite Template Model has an identical input/output interface as that in the primitive Template Model. It has a unique structure and name within its model base MB_{TM}. Every model can have as its elements a finite number of primitive and/or composite template models. The allowed relationships among composite template models are has-part and kind-of. A composite template model can only have a depth of length two. Exchange of information between component of a model and with the model itself is modeled via couplings which simplify specification of model interactions. The term has-part is used to emphasize that only composite model know of their children and not vice versa.

Similarly, kind-of relationships can exist between two model components. Composite template models are distinct in that they can be used in multiple composite instance template models. Composition and specialization relations are transitive. Given a component, a child and grandparent composition relationship may exist where no child can be the same as its immediate or super parent. Similarly, a specialization relationship is transitive.

A composite Instance Template Model contained in MB_{ITM}) is an instantiation of a composite template model. Each instance template model is an exact copy of a template model where the former can have multiple copies of a component and it can have arbitrary finite hierarchy. All instances of a template are distinguishable from one another using their assigned (given) names.

Instance Models

The primitive and composite Instance Models are instances of their respective Instance Template Models. With Instance Model available, we can translate them into simulation code. The instance models are generated in a two-step process. First, a model is selected from the Instance Template Model. This model can be total or partial—i.e., a model hierarchy can be of any hierarchical depth depending on the selected model. Second, for every model component that is specialized, one is selected. The resulting Instance Template Models, therefore, have no specializations. Therefore, in this process, any instance template model at any level of hierarchy is *reduced* to one that can be simulated. Every instance model represents an aspect of a model at a specific level of resolution.

Non-simulatable Models

In addition to the models defined above, it is also important to represent non-simulatable models which may be used as part of primitive and composite models. These models are distinct compared with the template models since they do not have input/output ports. Furthermore, these do not have time-based behavior as all template models do. Examples of non-simulatable models are object-based software components such as lists and sets.

Structural Modeling

All template model specifications have structural and behavioral parts. We highlight some of the main specifications of these models in terms of Entity-Relationships (ER) (Sarjoughian 2001; Fu 2002; Mohan 2003; Bendre and Sarjoughian 2005). This is appropriate because although these models can be manipulated visually (i.e., have object-orientation and visualization representation), their structure and behavior are considered to have only a simple relationship with the modelTemplate as shown in Figure 4.

An ER diagram depicting the main elements of the SESM modeling paradigm—the Template Model (modelTemplate), Instance Template Model (modelIT), Instance Model (modelInstance), and non-simulatable models (NSMTemplate)—is shown in Figure 4. Other elements of the ER diagram such as port template (portTemplate) and port instance template (portIT) models have similar ER specifications. The relational model database (repository) shown in Figure 2 contains all TM, ITM, IM, and NSM model components and their relationships.

Behavioral Modeling

In Systems Theory, the behavior of primitive models is described in terms of input and output interface (port names and their assigned values), states, and functions. The template model defined above is extended to support having state variables and port variables, each with name, type, and value. Since primitive and composite models have identical interfaces, the same ER diagram (specification) is used for both (see Figure 5). The state of primitive models is represented in terms

of name, type and value (see Figure 5(a)). The type and value choices are decided based on the choice of programming languages. Furthermore, the state variables can be arbitrarily defined by modelers. In particular, non-simulatable models can be used as state variables. The representation of functions is not supported in SESM since functions have arbitrarily structures—no generalized techniques exist to represent them as Entity-Relations (and thus tables in a standard relational database). Input and output port templates definitions are extended to include port variables as shown in Figure 5 (b).

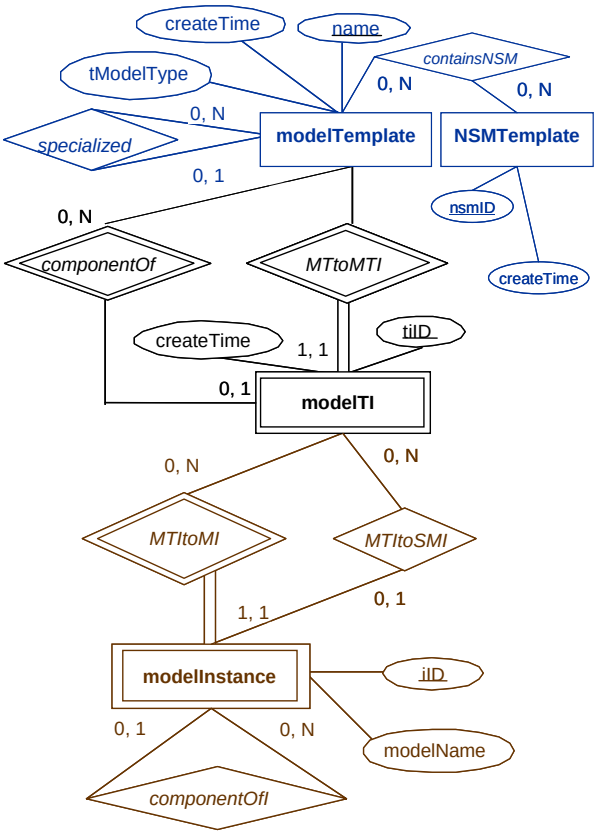
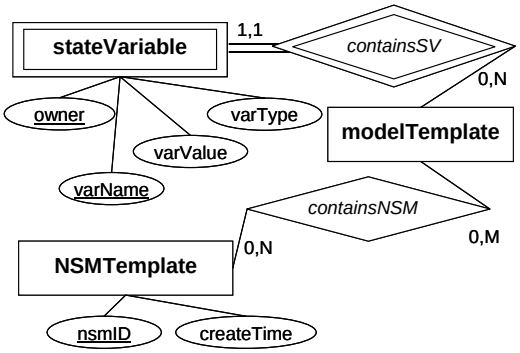


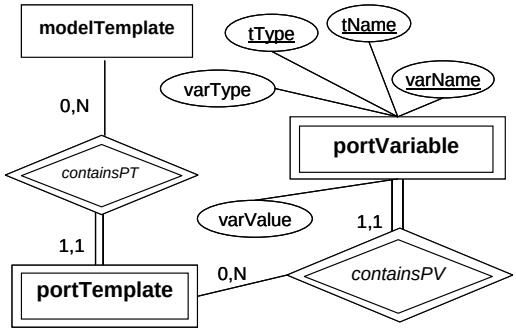
Figure 4: Snippet of the TM, ITM, IM, and NSM Models and Their Relationship

Complexity Metrics

For any model with a few tens of components, it is useful to have quantitative measure of their complexity. The complexity is defined as a set of metrics for primitive and composite models. For example, for a primitive model, we can determine its number of output ports (outP) and state variables (tSt). Similarly, complexity of a composite model can be measured in terms of its number of children (iChildren) and total number of couplings (tCo) at any level of hierarchy (see Figure 6). Such metrics provide quantitative analysis of the structural complexity of models.



(a) Model Template and State Variable



(b) Port Template and Port Variables

Figure 5: State Variable Model Template, Port Template, Port Variable, and Their Relationships

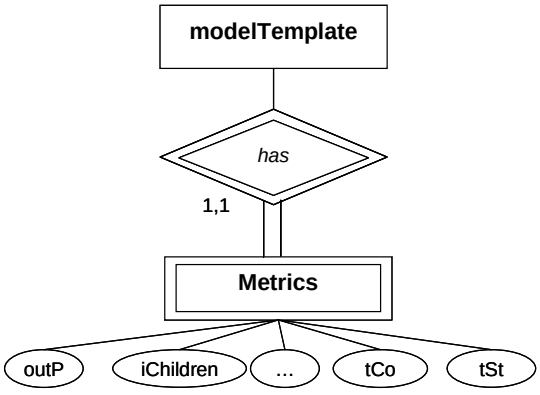


Figure 6: Structural Complexity Metrics

Measuring complexity of hierarchical models is useful, especially for large-scale models—e.g., it is impractical to manually measure the number of model components and their couplings or use any other software tool. Structural metrics can be computed based on the information captured in the Template and Instance Template models. Metrics can also be computed for Instance models.

Instance Model Translation

As suggested in Section 2, models specified in SESM/CM can be mapped (translated) into simulation code (see Figure 2). These simulation codes are not stored in a database since they can have arbitrary syntax; instead they are stored as flat files. Specifically, component-based like models can be completely translated into simulation code given the targeted simulation environment. The translation can be direct from database to simulation code or instead models in the database can be translated to standard languages (XML) and then translated to simulation code. For example, SESM/CM supports translation of primitive and composite instance models into a DEVSJAVA target simulation environment, which simulates Discrete Event System Specification (DEVS) models. All composite models can be automatically translated to XML and then to DEVSJAVA simulation models (DEVSJAVA 2002).

Primitive models, however, can be translated into simulation code only partially since generalized functions (e.g., external transition function or initialization of a model) cannot be stored in a relational database in a systematic fashion. The translated DEVSJAVA code includes "templates" (place holders) which must be completed via SESM editor or an IDE environment such as Eclipse. The input/output interfaces of atomic and coupled models (input and output ports, variables, and types) as well as state variables of atomic models are automatically mapped into DEVSJAVA simulation code.

SESM/CM ENVIRONMENT

A realization of the modeling approach presented in the previous section has been developed using Java and DBMS technologies. This environment called SESM/CM has client-server architecture. As shown in Figure 7, a modeler client has multiple views to graphically specify template, instance template, and instance models stored in a database and managed by a server. In the left-hand frame, there are two tabs: Simulatable and Non-Simulatable. The tree structures of the Template Model, Instance Template Model and Instance Model are available as three tabs (TM, ITM, IM) under the Simulatable tab. The Non-Simulatable tab shows the names of the models. Non-simulatable models may be stored (and viewed) in hierarchical directory structure, but they do not contain any modeling relationships as defined for simulatable models. The non-simulatable models are not stored in a database since they have arbitrary specifications.

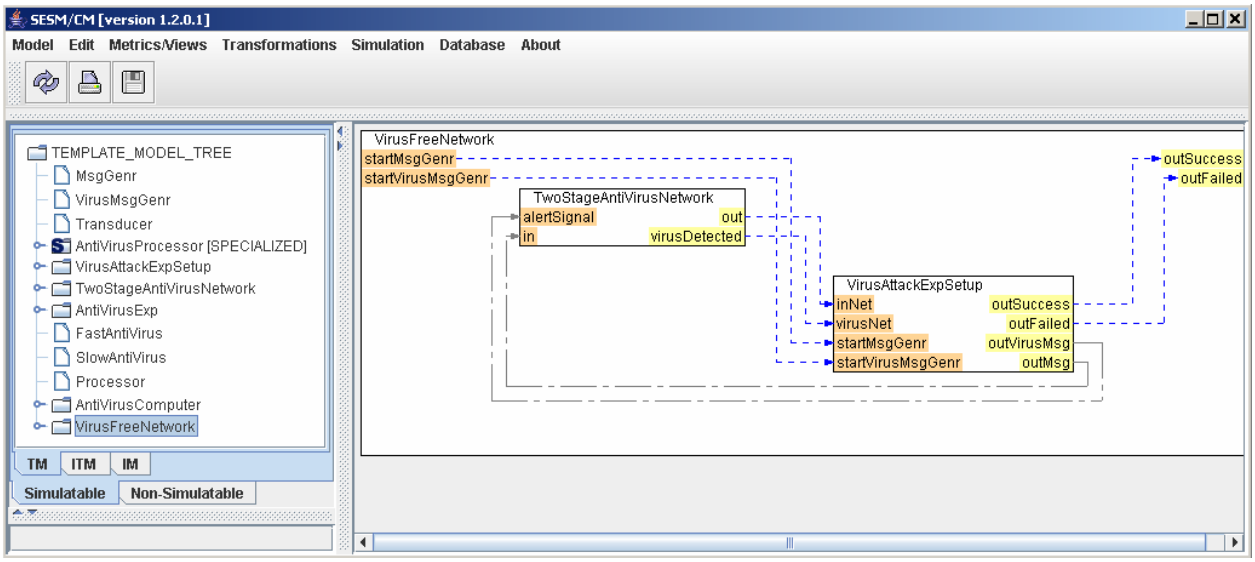


Figure 7: SESM/CM Environment

In the right panel, modelers can view the block representation of the TM, ITM, and IM models. Coupling relationships, specification of states (variables and types), and ports (port names, variables, and types) are supported in this panel. Complexity metrics and translation to XML and simulation code are also supported in this panel.

Model components and their ports are displayed in the right-hand frame and correspond to the model type (e.g., Instance Template Model) in the left panel. Under the Operation menu item, template models and instance models can be created. Some highlights of the model views shown in Figure 7 are described using a simple example below.

Demonstration of the SESM/CM Environment: VirusFreeNetwork

To help illustrate the Scalable Entity Structure Modeler with Complexity Metrics environment, we use as our example a computer network with the capability to detect and remove infected messages. One of the models in this system, called VirusFreeNetwork, is devised to destroy messages found to be infected. To model this and other types of computer networks (e.g., VulnerableNetwork), we have defined a set of primitive and composite models as shown in Figures 7 (template models) and 8 (instance template models).

There are five primitive template models. The Processor model's role is to process messages generated by the MsgGenr and send them to port out of its parent (AntiVirusComputer) if there are no messages alerting the processor that specific messages may be infected. If Processor receives a message from the MsgGenr and received an alert message from the VirusMsgGenr, the Processor sends the MsgGenr message to the AntiVirusProcessor.

The AntiVirusComputer determines (e.g., randomly) whether or not a message is infected. If it is infected, it is sent to the second AntiVirusComputer in the TwoStageAntiVirusNetwork for further processing. Otherwise, it is sent to the input port inNet of the VirusAttackExpSetp via specified couplings. The AntiVirusProcessor is specialized into FastAntiVirus and SlowAntiVirus where the latter takes more time to detect viruses compared to the former (e.g., using complex algorithms to detect difficult to find viruses). Each of these specialized models is also a primitive template model.

The MsgGenr and the VirusMsgGenr generate (safe, infected, and alert) messages for the TwoStageAntiVirusNetwork component. The Transducer keeps track of the TwoStageAntiVirusNetwork and messages generated by the MsgGenr and VirusMsgGenr to compute statistics such as ratio of safe vs. infected messages or percentage of infected messages that were disinfected.

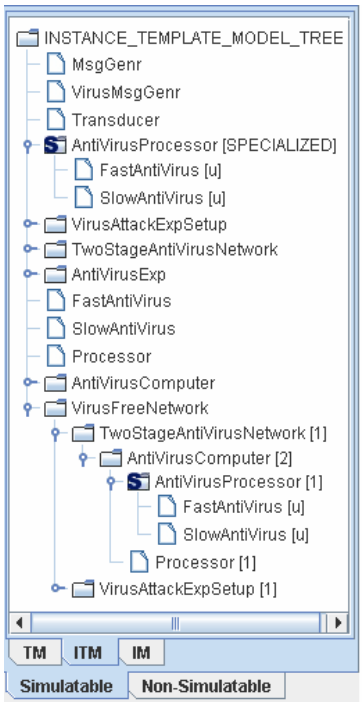


Figure 8: Hierarchical VirusFreeNetwork Instance Template Models

A number of composite instance template models are shown in Figure 8. For example, the AntiVirusComputer instance template model has Processor and AntiVirusProcessor primitive instance template models where the latter is specialized into FastAntiVirus and SlowAntiVirus template models. As shown in Figure 9, the AntiVirusComputer model has feed-forward and feedback couplings with external input and output couplings. Other composite template models such, as FourStageVulnerableNetwork, are defined using the primitive and composite template models available in the TemplateModel database (MB_{TM}).

Each instance composite instance template models that is part of parent model has a multiplicity number. For example, in the AntiVirusComputer model, there are two instances and shown as AntiVirusComputer [2]. In the case of specialization models such as

AntiVirusProcess, the multiplicity for the specialized models is undefined (shown as FastAntiVirus [u]). This is appropriate since the multiplicity is assigned to the model and not its specializations.

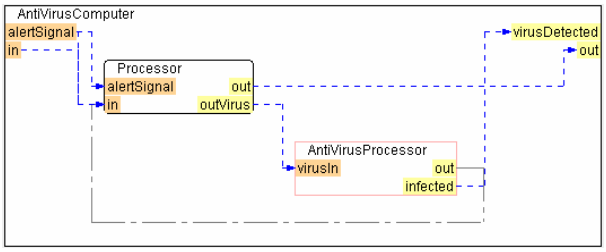


Figure 9: AntiVirusComputer Composite Instance Template Model

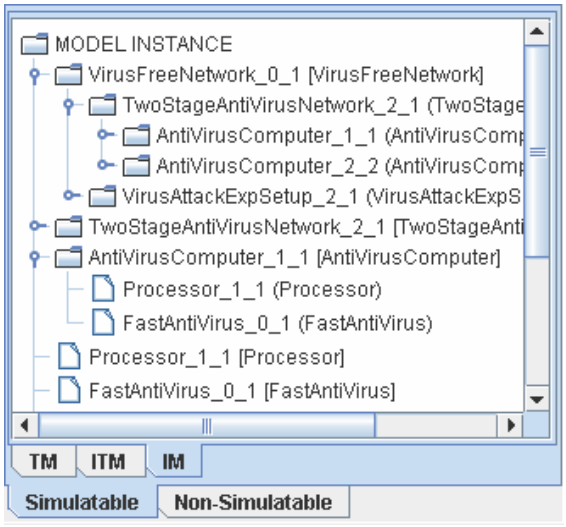
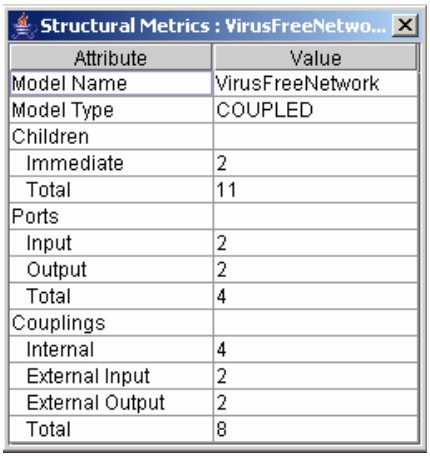


Figure 10: Instance Model Snippet for TwoStageAntiVirusNetwork

Figures 9 and 10 are visual representations of an instance template and instance models, respectively. The AntiVirusProcessor model is a specialized model—in creating an instance model, the modeler is requested to separate slow and fast types. The TwoStageAntiVirusNetwork model is instantiated from its instance template model. SESM/CM assigns instance model components unique names. As shown, the AntiVirusComputer_1_1 and AntiVirusComputer_2_2 instance models differ in terms of their AntiVirusProcessor primitive models (i.e., SlowAntiVirus_0_1 and FastAntiVirus_0_1) although they could be identical (see Figure 10).

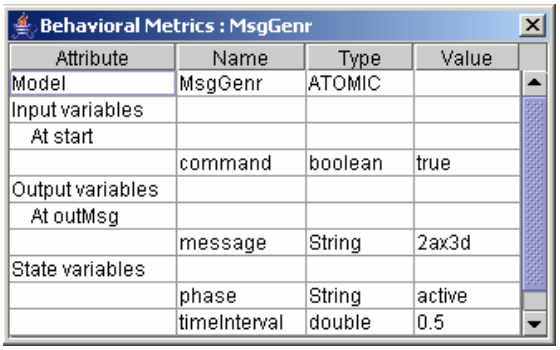
Complexity Metrics

Models specified in SESM/CM can be analyzed in terms of their structure. The structures of primitive and composite model components differ. We have defined a set of structural complexity metrics for instance template models to quantitatively determine static model complexity. The metrics for the composite instance template model VirusFreeNetwork is shown in Figure 11. The complexity metrics for primitive instance template models are a subset of those for composite instance template models. These structural complexity metrics offer information before any simulation experiments are undertaken. In this regard, system and simulation architects and designers can evaluate non-behavioral model specifications.



Attribute	Value
Model Name	VirusFreeNetwork
Model Type	COUPLED
Children	
Immediate	2
Total	11
Ports	
Input	2
Output	2
Total	4
Couplings	
Internal	4
External Input	2
External Output	2
Total	8

Figure 11: VirusFreeNetwork Structural Metrics



Attribute	Name	Type	Value
Model	MsgGenr	ATOMIC	
Input variables			
At start			
	command	boolean	true
Output variables			
At outMsg			
	message	String	2ax3d
State variables			
	phase	String	active
	timeInterval	double	0.5

Figure 12: MsgGenr Behavioral Metrics

In addition, we have defined a set of behavioral complexity metrics to complement structural complexity metrics. The input, output, and state variables of primitive model components show the

complexity of input and output message types and states (see Figure 12). The number of components, ports, etc allows modelers to quantitatively assess the complexity of individual primitive components as well as their composition. The aggregate metrics for the composite components are easy and efficient to compute since every primitive and composite model is stored in a relational database.

Model Validation

A key capability of any modeling methodology is to support model validation and simulation verification. A modeling environment not only needs to enable model specifications, but also facilitate the simulation development lifecycle. The SESM/CM framework described above supports generic component-based and incremental, iterative process for model specification, both of which are necessary to simulation modeling of contemporary systems. Given the Federation Development Process (FEDEP) lifecycle (IEEE 2003)—consisting of federation objectives formulation, conceptual model definition, federation design specification, federation integration and testing and experimentation)—SESM/CM can be used across all of its steps. In particular, the range of modeling capabilities is well suited for conceptual and design specifications. Its database feature can support FOM model persistence and SOM generation via XML-based simulation code generation. Furthermore, the framework supports validation and verification not only from a behavioral aspect but also provides structural aspect (i.e., structural and behavioral metrics depicted in Figures 11 and 12, respectively).

RELATED RESEARCH

Various approaches have been proposed to represent, use, and manage relationships such as has-part and kind-of among a set of entities (e.g., models). One such approach is known as the System Entity Structure (SES) (Zeigler 1984; Rozenblit and Zeigler 1993). It supports decomposition and specialization. This approach supports representing a family of models as a labeled tree with attached variable types with a set of axioms. The key axioms are uniformity (any two nodes which have the same label have identical attached variable types and isomorphic sub-trees), strict hierarchy (no label appears more than once down any path of the tree), alternating mode (each node has a mode which is either entity, aspect, or specialization) and inheritance (every entity in a specialization inherits all the variables, aspects, and specializations from the parent of the specialization). The remaining two axioms are valid brothers (no two brothers have the same label) and attached variables (no two variable types attached to the same item have the same name).

System Entity Structure employs a model base and an entity structure base, each of which corresponds to a directory-style file system. The model base contains primitive (atomic) as well as composite (coupled) models. An entity structure is similar to SESM in that it contains template models as well as instance models of a model base. However, it does not define how to distinguish between template and instance template models. The consequence is SESM does not use the alternating modes to specify a family of models. Instead, the Instance Template Model offers a new mechanism to model a family of models from template models—the key benefit of this simpler approach is its ability to handle complex model structures and scalability. Furthermore, models in SESM/CM are stored in a relational database that supports scalability and complexity metrics. Another capability is visual modeling.

In a separate work, a relational algebraic representation of SES was developed (Park, Lee et al. 1997). A relational database (ESQL/C, a SQL-compliant database supporting C language) was developed to capture the models. The work presented here is significantly different as the SESM/CM modeling approach offers new capabilities key for handling large-scale, complex systems (e.g., complexity and behavioral metrics and partial and total mapping of primitive and composite models into simulation code).

Aside from these, there exist UML and supporting software engineering tools for modeling. These offer specifying has-part and kind-of relationships, but the concepts and techniques introduced in SESM do not exist in UML and therefore are not supported by any software engineering modeling tool. Aside from these XML (XML 2005) and its variants offer syntactic representation that may be used with the SESM/CM specification.

FUTURE DIRECTIONS

The SESM modeling approach is generic and thus it is important to extend it to support specific domains of interest. This approach is especially relevant for managing knowledge domains and thus basic ingredients of models that exist for challenging domains such as wireless network systems, Joint Synthetic Battle Space, and supply-chain systems. The

capability for domain-specific model specification can facilitate application of the proposed approach for real-world case-studies.

A promising basic research direction is in extending the approach to support other types of model specifications such as continuous models, agent models, and observed (measured) data sources. This extends support for domain-specific model development. Additional model specifications in turn require developing corresponding translators for generating simulation code suitable for different simulation engines.

Another area of interest is support for collaborative modeling. In this direction, we are extending our work for collaborative use. The basic SESM/CM visual modeling features (e.g., automatic placement of model components in a diagonal layout and couplings with minimal crossing) provide scalable workspace for building models of real-world systems. The basic infrastructure supported by SESM including its database, client/server software design, and XML-based model translation, supports its extension to a collaborative environment. Finally, Grid Services and Service-Oriented Architecture are important technologies for further development of SESM/CM that could help extend it to industrial strength simulation development and testing.

CONCLUSIONS

In this paper we proposed a SESM/CM approach for developing models for a class of contemporary systems where they lend themselves to hierarchical simulation model specification. This framework supports development of models in a step-wise fashion where template models serve as a basis to specify instance template and instance models thus offering a new way of handling model scalability and complexity. The modeling approach enables structural and behavioral model specification of models that can be stored in databases. Aside from supporting model consistency, modelers can evaluate structural traits of their models using complexity metrics. Finally, the approach provides a basis for model validation via systematic model specification of large-scale, complex models and automatic translation into simulation code.

ACKNOWLEDGEMENTS

This research has been supported in part by NSF DMI-0075557 and Intel Research Council grants. I would like to thank Dr. Jim Wall of Texas A&M University for his initiative and support of this paper. Also thanks to T-S Fu, S. Mohan, S. Bendre, and R. Flasher who have contributed to the various aspect of the SESM software development.

REFERENCES

Bendre, S. and H. S. Sarjoughian (2005). Discrete-Event Behavioral Modeling in SESM: Software Design and Implementation. *Advanced Simulation Technology Symposium*, San Diego, CA.

Davis, P. K. (1995). Aggregation, Disagreegation, and the 3:1 Rule in Ground Combat, RAND.

Davis, P. K. and R. H. Anderson (2004). *Improving the Composability of Department of Defense Models and Simulations*. Santa Monica, CA, Rand.

DEVSJAVA. (2002). DEVSJAVA Modeling & Simulation, http://www.acims.arizona.edu.

Fishwick, P. A. (1995). *Simulation Model Design and Execution: Building Digital Worlds*, Prentice Hall.

Fu, T.-S. (2002). Hierarchical Modeling of Large-Scale Systems Using Relational Databases. Electrical and Computer Engineering. Tucson, Arizona, University of Arizona.

IEEE (2003). HLA Federation Development and Execution Process, *IEEE 1516.3*, IEEE.

Modelica (2000). Modelica Association. http://www.modelica.org/.

Mohan, S. (2003). Measuring Structural Complexities of Modular, Hierarchical Large-scale Models. Computer Science and Engineering. Tempe, Arizona, Arizona State University.

Park, H. C., W. B. Lee, et al. (1997). "RASES: A Database Supported Framework for Structured Model Base Management." Simulation Practice and Theory Vol. 5 (No. 4, May 1997): 289 - 313.

Rozenblit, J. R. and B. P. Zeigler (1993). Representing and Construction System Specifications Using the System Entity Structure Concepts. *Winter Simulation Conference*, Los Angeles.

Sargent, R. G. (1994). Verification and Validation of Simulation Models. *Winter Simulation Conference*.

Sarjoughian, H. S. (2001). An Approach for Scalable Model Representation and Management. Tempe, Arizona, Computer Science & Engr., Arizona State University.

Sarjoughian, H. S. and F. E. Cellier, Eds. (2001). *Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies*, Springer Verlag.

XML. (2005). http://www.w3.org/XML/.

Zeigler, B. P. (1984). *Multi-Facetted Modeling and Discrete Event Simulation*. New York, Academic Press.