Proceedings of the 2007 Winter Simulation Conference
S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.

A CO-DESIGN MODELING APPROACH FOR COMPUTER NETWORK SYSTEMS

Weilong Hu Hessam S. Sarjoughian

Arizona Center for Integrative Modeling & Simulation
Computer Science and Engineering Department
Arizona State University
Tempe, AZ 85281-8809, U.S.A.

ABSTRACT

Co-design modeling is considered key toward handling the complexity and scale of network systems. The ability to separately specify the software and hardware aspects of computer network systems offers new capabilities beyond what is supported in modeling frameworks and tools such as NS-2 and OPNET. The DEVS/DOC simulation environment supports logical co-design specification based on the Distributed Object Computing (DOC) abstract model. To overcome DEVS/DOC's lack of support for visual and persistent modeling, this paper presents SESM/DOC, a novel approach, which is based on the Scaleable Entity Structure Modeler (SESM), a component-based modeling framework. This approach supports logical, visual and persistent modeling. Modelers can develop software and hardware models separately and systematically integrate them to specify a family of computer network system designs. This paper details the SESM/DOC co-design modeling approach with the help of a search engine system example, and presents a discussion for future research directions.

1 INTRODUCTION

Simulation is commonly used for analysis and design of systems ranging from embedded devices to distributed network systems. In recent years, the modeling and simulating of combined discrete/continuous systems has witnessed major advances. Frameworks such as DEVS (Zeigler, Praehofer et al. 2000) and Ptolemy II (Eker, Janneck et al. 2003) offer capabilities to describe discrete and continuous aspects of a system and their integration. The underlying concept of these frameworks is to support co-design where a system is divided into software and hardware layers and their integration is specified through A/D and D/A conversions. However, the underlying abstractions of these frameworks are not well suited for modeling and simulating computer network systems.

Networked systems are important to be described in terms of software and hardware layers. Model abstractions are defined in terms of software applications executing on hardware nodes. Modeling and simulation approaches that are used for networked system modeling are generally monolithic – hardware and software components are not separately modeled and combined systematically. For example, to model and simulate a network system, NS-2 (ns-2 2004) can be used to describe the system's communication protocol and hardware resources. With this framework and its toolset, modelers use abstractions that have combined software and hardware aspects of computing nodes. This and others such as OPNET (OPNET 2004), however, do not support separate specification of software and hardware with the capability to synthesize them.

A framework that explicitly separates software and hardware layers of networked systems is known as Distributed Object Computing (Butler 1995). An abstract specification is provided to describe a system in terms of its software and hardware components and the mapping of the former to the latter. The Discrete Event System Specification (DEVS) modeling and simulation framework has been used to realize the Distributed Object Computing concept and abstractions (Hild 2000; Sarjoughian, Hild et al. 2000). The DEVS/DOC approach extends the generic DEVS modeling concepts and models to support co-design specification of network systems. It enables modelers, for example, to model a computer network in terms of separate sets of hardware and software components and software to hardware mappings. With this approach, the hardware and software layers can be independently specified and the software to hardware mapping be used to configure system designs. This level of flexibility is useful for evaluating alternative designs – i.e., hardware designs and topologies, software application capabilities, and how they are synthesized (Hild, Sarjoughian et al. 2002; Hu and Sarjoughian 2006).

Aside from logical specification of network systems using DEVS/DOC, it is also important to support visual and persistent modeling (Burmester 2005). Visual and persistent modeling need to have their own concepts and elements. A component-based modeling framework called Scaleable Entity Structure Modeling (SESM) offers a uni-

fied logical, visual, and persistent foundation for developing hierarchical simulation models (Sarjoughian 2001; Fu 2002; Sarjoughian 2007). A realization of this framework is developed which supports the semi-automatic translation of SESM logical models into the simulation code for execution in DEVSJAVA. However, the SESM framework does not support co-design modeling as described above. In this work, we have introduced the DOC co-design modeling into the SESM. This has resulted in the SESM/DOC approach which supports logical, visual, and persistent co-design modeling with support for generating partial simulation models that once completed can be executed using DEVS/DOC simulation environment.

2 BACKGROUND

2.1 DEVS/DOC

Distributed object computing (DOC) offers concepts and an abstract model for describing a network system in terms of a software layer mapped onto a hardware layer (see Figure 1). The software layer describes software components of the network system in terms of a Distributed Cooperative Object (DCO) model. The hardware layer describes the hardware components of the network system in terms of a loosely coupled network (LCN) model. The mapping of DCO onto LCN is described in terms of an Object System Mapping (OSM).

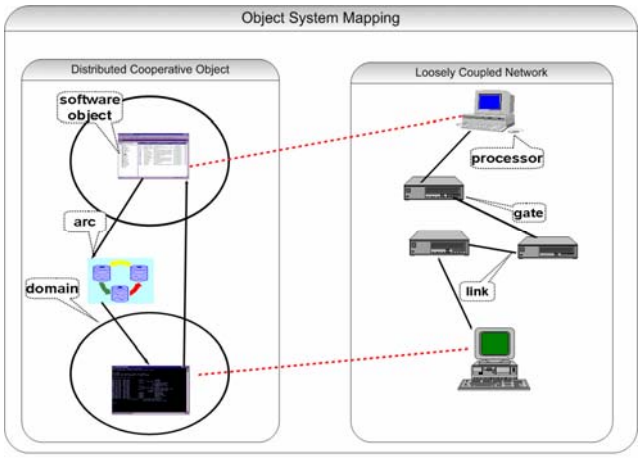

Figure 1. DOC Conceptual View

The Distributed Cooperative Object is for modeling software components. A *software object* is defined to have attributes (data members) and methods (function). The size of a software object is defined by the collective memory requirements of these attributes and methods. The attributes include size, thread mode, and initial method. The Loosely Coupled Network is for modeling the hardware components including the *CPU,* the *transport,* the *link,* the *router* (network router and routing unit), and the *network interface*. Another hardware component is *processor* which consists of the *CPU*, the *transport*, and the *routing unit* components. The Object System Mapping defines a set of axioms for assigning software components to hardware components.

The hardware provides computation and memory resource for software. It also presents the network topology (Hild, Sarjoughian et al. 2002). When a software object is invoked, its size loads into the memory of its assigned processor. Besides memory, another resource needed to run a software object is the processing mode which handles the workload of the software. Software objects are defined to be executed in one of three modes: *none, object, method*. In the *none* mode, the software can only process one job at a time, all other jobs need to be queued. In the *object* mode, a software object can have one job per defined method concurrently active. In the *method* mode, all the incoming jobs of a software object can be processed concurrently.

DEVS/DOC provides a mechanism for managing the workload of software objects. Each software object can generate, send/receive, and finish jobs. The software object workload is managed by classifying jobs and handling in different stages (i.e., need to be 'done', is in 'processing,' and 'finished'). The software object records its number of done jobs which is used to determine when the software has finished its tasks. Once all the software objects have completed their work, the simulation is completed and will be stopped. The control of the simulation execution is defined in terms of pre- or user-defined experimental frame models.

The Object System Mapping defines the assignment of the components in the DCO layer to the components in the LCN layer. The OSM mapping concept is formulated in terms of couplings in the DEVS/DOC. The software object loads itself into its assigned processor memory by sending a load software message to the processor. The selected method in the software object will be executed by the processor's CPU. When all the jobs are done, the software object will unload itself from the hardware and release the processor's memory.

The Distributed Object Computing abstract model components and their relationships were formalized in DEVS and implemented in the DEVSJAVA simulation environment. The DEVS/DOC environment supports modeling the components of the DCO and LCN layers with their OSM component. The structure and behavior of these components are specified as atomic and coupled DEVS models (see Table 1 for the partial Routing Unit model specification). Details of the transition and time advance function specifications for all models can be found in (Hild 2000). Compare to other environments such as DEVS, NS-

2, and OPNET, DEVS/DOC provides direct support for discrete-event co-design specification of network systems.

Table 1. A partial Routing Unit DEVS/DOC Model

Routing Unit	<X, Y, S, δ_{ext}, δ_{int}, δ_{conf}, λ, ta>
InPorts	{inLoop, inLink}
OutPorts	{outLoop, outLink}
X	{(import, pdu)}
pdu	(clientID, searching key word, size, request format)
Y	{(outPort, pdu)}
S	Phase × σ × OutLoopBuffer × OutLinkBuffer × OutLoopDelay × OutLinkDelay × SearchingIndex Phase ∈{passive, busy}, σ ∈ $R^{+}_{0,\infty}$

2.2 Scalable System Entity Structure Modeler

The Scalable System Entity Structure Modeler (SESM) (Sarjoughian 2001; Fu 2002; Bendre and Sarjoughian 2005; Sarjoughian and Flasher 2007) is a framework aimed at hierarchical component-based modeling. Its specification capabilities are derived from Entity-Relation (ER), System Entity Structure (SES) (Zeigler and Hammonds 2007) , and Object-Oriented (OO) modeling approaches. It introduces visual modeling and transforming logical models into simulation models. Its realization is a modeling engine for developing specifications that have formal logical syntax and semantics (see Figure 2).

In SESM, the logical models are defined to consist of primitive and composite model types. A composite model consists of one to many primitive and/or composite components. The composite model and its components have the same model type. The primitive and composite model types can be used to define different kinds of models. For example, SESM supports partial specification of atomic models and complete coupled models (Bendre and Sarjoughian 2005). Similarly, it supports specifying XML schemas (Sarjoughian and Flasher, 2007). .

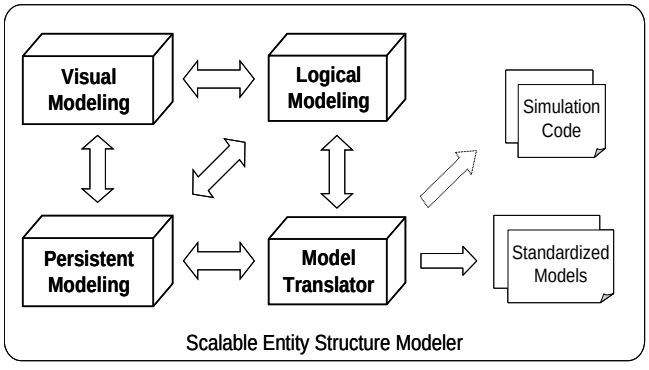

Figure 2. Logical, Visual, and Persistent Model Types

Each primitive and composite model type is defined in terms of Template, Instance Template, and Instance model types. A primitive Template Model can have input/output with ports and states. The collection of input and output ports for each component is defined as its interface. Input and output ports may be used to receive or send simple data or complex objects. A composite model can have input/outputs with ports, states, and a set of links connecting the components that are contained in it. Any two components can send and receive information using links. Every composite component for a Template Model or Instance Template Model has a unique name and tree structure. The allowed relationships among composite components are *whole-part*. Given a component, a sub-component and super-component composition relationship may exist only when no sub-component can be the same as its (immediate or higher) super-component. The sub-component is referred to as *part* and the component and super-component are referred to as *whole*. Composite components can be used in multiple composite Instance Template Models. The hierarchy depth of a composite component is equal or greater than two. All instances of a composite component (corresponding to the Instance Model) are distinguishable from one another using their assigned (or given) names. The primitive and composite Instance Models are instances of their respective Instance Template Models. The models satisfy the uniformity constraint which states two components having the same name must have identical structures. The *is-a* relationship is also defined for primitive and composite components. It can be used to specify a model (specialize) to be specialized to one or more other models (specialized). The specializee and specialized are defined such that the former is replaced with the latter when instance models are generated. Instance models can be generated from Instance Template models. An Instance model can be total or partial — i.e., a model hierarchy can be of any hierarchical depth depending on the model that is being transformed.

Two types of logical models are defined – simulatable ad non-simulatable (Sarjoughian and Flasher, 2007). This separation is introduced to differentiate between models that can be simulated in time from models whose behavior is not defined in terms of time. For example, models of a processor and queue have dynamical behaviors. However, a processor model, for example specified in DEVS, can process a job given a finite period of time. In contrast, a queue model, for example specified in UML, can return a job that is queued without taking any time. Given the specifications of the simulatable and non-simulatable models, the simulatable models are defined to contain non-simulatable models, but not vice versa. For example, the processor model can have a queue model.

SESM is visual modeling environment and therefore supports visual modeling of logical model. Visual models

are represented as hierarchical blocks and tree structures. These provide complementary visual models such that the block models depict coupling of ports and the tree structure shows multi-level model hierarchy. Coupling is presented as a line with an arrow showing the direction of information flow. A component in a composite model can be either a primitive model or a composite model. Three different kinds of couplings are supported. They are internal, external input and external output couplings. These couplings are defined according to the DEVS coupled specifications, but they may be changed to support other kinds of models (e.g., block models in Simulink).

Another key aspect of the SESM framework is its support for model persistence. Logical models are stored according to a set of relational schemas. Model persistence in a relational database supports retrieving information about any model component efficiently. This capability is particularly useful for large-scale and complex models.

3 COMBINED SOFTWARE/HARDWARE MODEL SPECIFICATION

The software and hardware parts of a system can be modeled based on the SESM's modeling framework. Users may develop models according to the distributed object computing framework. The modelers must apply the DOC concepts and methods manually. The use of SESM without direct support for logical co-design modeling concepts and constructs is ad-hoc and undesirable. Furthermore, the SESM's framework does not support visual and persistent co-design modeling. To overcome these limitations, it is desirable to introduce the DOC co-design concepts and methods to the SESM framework. The resulting SESM/DOC can support co-design from *logical* aspect as well as the *visual*, *persistence*, and *model transformation* aspects. For example, the logical models stored in the SESM database cannot distinguish among software and hardware model components. Similarly visual models cannot be differentiated to represent DCO and LCN models. Additionally, the concept of software to hardware mapping (OSM) is supported.

The main capability of the SESM/DOC is independent specifications for software and hardware model components. This separation is to support by database schemas that conform to the DOC abstract specification. Furthermore, visual model design is to support modeling of software, hardware, and their integration. SESM/DOC must also support integrating software and hardware layers by mapping the former to the latter based on the OSM specification. Therefore, to handle the separation and integration of software and hardware layers, new model types and constraints are introduced to the SESM framework.

3.1 Example Model

Before detailing the SESM/DOC, a server-client network system which includes two servers and two clients is considered (see Figure 3). The two servers can be used by clients to search for information. One server supports text file search and the other supports video file search. A client can search for mixed text and video information. From a co-design perspective, the network system shown in Figure 3 can be designed to consist of software and hardware components and their integration. The *Text File Server* and *Video File Server* are two software components. The *Link 1*, *Router 1*, and *Hub_ethernet_1* are examples of LCN components. In the following sections, this system network will be studied based on the separation of its software applications and hardware facilities.

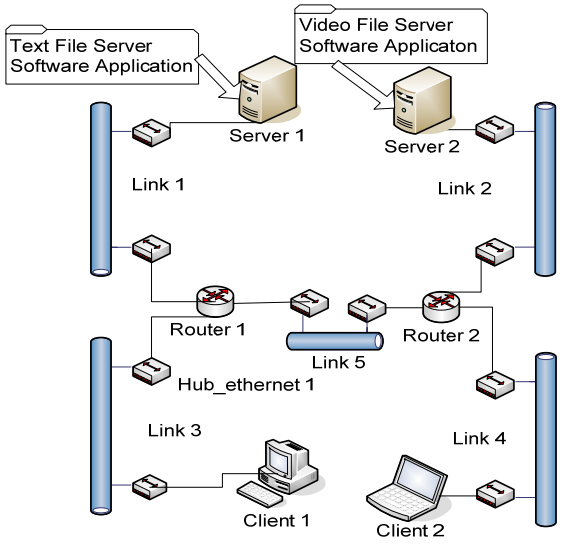

Figure 3. Network System Example

3.2 Co-Design Model Types

In order to support co-design model specification, the SESM/DOC approach defines primitive and composite model types for software and hardware model components. These models extend the syntax and semantics of the SESM models. Each of the software and hardware models can have whole/part and coupling relationships. Model may have specialization relationships to one another. For example, a model may not contain other models as components or a composite model may have specific whole/part relationships (see LCN models). The SESM/DOC models are defined as follows:

- Software layer (SESM/DCO): alternative software model specifications and configurations are supported using a predefined software model.
- Hardware layer (SESM/LCN): alternative hardware specifications and configurations are supported using a predefined collection of hardware models.

- Object System Mapping Layer (SESM/OSM): alternative assignments of software models to hardware models to define combined software/hardware models.

3.2.1 Software Layer Model Types

The software layer model in SESM/DOC specification corresponds to the Distributed Object Computing (DCO) layer. It is defined to have two model types. They are Software Layer Model (SLM) and Software Application Model (SAM). The SLM and SAM correspond to the Distributed Object Computing layer and the software object defined in DCO. The SLM is a composite model which is specified in terms of one or more SAMs. Every SAM is a primitive model which corresponds to a software application that is to be modeled. The SESM/DOC supports the following modeling constraints in the software layer:

1. SLM is a composite model that can only contain a finite number of SAMs;
2. SLM has to contain at least one SAM;
3. SAM is primitive model;
4. SAM can only be contained in a SLM;
5. SLM and SAM can only be coupled with one another.

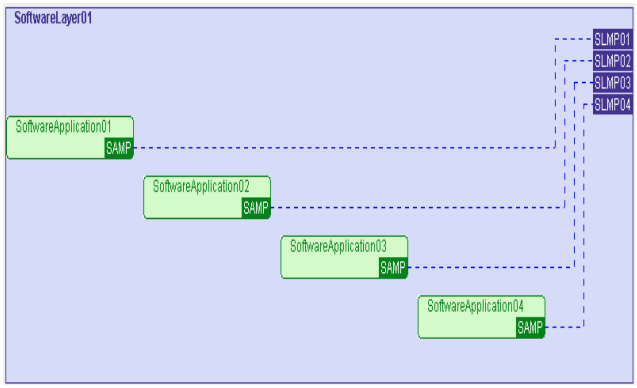

Figure 4. Server-Client Software Specification

In the example shown in Figure 3, the server-client network architecture has two server applications and two client applications in the software layer. These can be specified in the software modeling working section as shown in Figure 4. The software layer model has four software application models. The *SoftwareApplication01* and *SoftwareApplication02* are the server software applications; (these are called "Text File Server" and "Video File Server" in Figure 3). The *SoftwareApplication03* and *SoftwareApplication04* are the client software applications. The ports defined for SAM and SLM are bi-directional and therefore couplings between the software layer model and these software application models are bi-directional.

3.2.2 Hardware Layer Model Types

The hardware layer model in SESM/DOC specification corresponds to the Loosely Coupled Network (LCN) layer. It is defined to consist of Hardware Layer Model (HLM) and a set of primitive and composite hardware model types. The hardware model types are *Processor Model* (PM), *Network Interface Model* (NIM), *Link Model* (LM), and *Router Model* (RM). The composite models are *Processor Group Model* (PGM), *Network Interface Group Model* (NIGM), *Link Group Model* (LGM), *Router Group Model* (RGM), *Processor and Network Interface Unit Model* (PNM), and *Processor and Network Interface Unit Group Model* (PNGM).

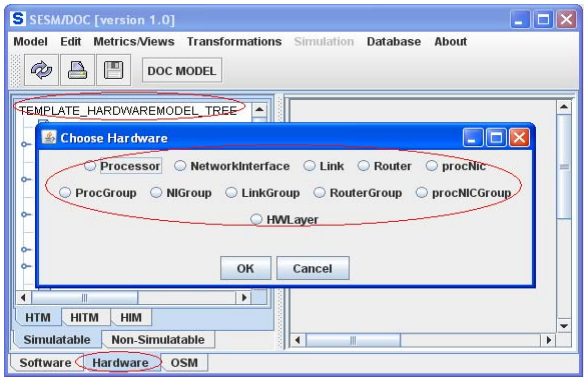

Figure 5. Model Selection in Hardware Model Working Section

Given the kinds of hardware models that can be defined based on the constraints that are defined for LCN, the following modeling constraints are supported in SESM/DOC.

1. HLM can only contain group models (PGM, NIGM, LGM, RGM);
2. PGM can only contain PM;
3. NIGM can only contain NIM;
4. LGM can only contain LM;
5. RGM can only contain RM;
6. PNGM can only contain PNM;
7. PNM can only contain one PM and one NIM;
8. RM, NIM, LM, RM are primitive models.

These model types and the composition relationships among them allow specifying different hardware network topologies. The coupling constraints between these different model types are defined as follows.

1. HLM can only be coupled with group models
2. RM can only be coupled with RGM;
3. NIM can only be coupled with NIGM;
4. LM can only be coupled with LGM;
5. PNM can only be coupled with PNGM;
6. RGM can only be coupled with RM and NIGM;

7. NIGM can only be coupled with NIM, RGM, LGM, PM, PNGM and RGM;
8. LGM can only be coupled with NIGM or PNGM;
9. RGM can only be coupled with NIGM.

Given the distinctions between the software and hardware models a modeler must choose the kind of a model that is to be developed. The "DOC Model" allows the modeler to choose either the Software Layer Model or Hardware Layer Model. For modeling the hardware layer of a network system, the selection of hardware provides a list of model components as listed above and shown in Figure 5.

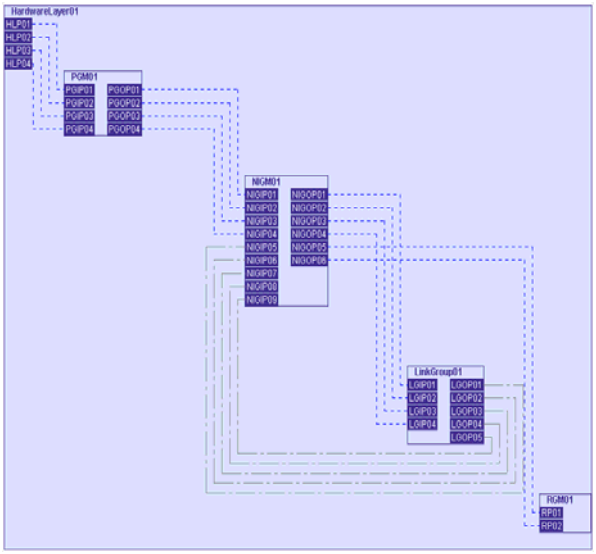

Figure 6. Server-Client Network System Architecture Hardware Design

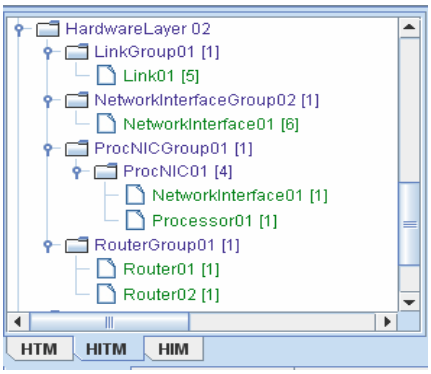

Figure 7: Instance Template Model for HardwareLayer01

Given the above model types and constraints, a hardware layer model such as *HardwareLayer01* can be specified as shown in Figure 6. Furthermore, SESM/DOC supports specifying alternative model specifications. For example, an alternative hardware network specification for the *HardwareLayer01* can be defined using PNGM. These different models are specified in terms of the Instance Template Model with additional information such as the number of processors, the number of links, and routers (see Figure 7). The topology of the hardware layer can be changed which results in different computer network system models.

3.2.3 Object System Mapping Layer

Unlike the DOC and LCN, the OSM specifies mapping software to hardware components. The OSM model layer allows specifying three model types: SLM and HLM models and how they may be synthesized. Similar to SLM and HLM, the OSM Layer model satisfies some constraints. These composition and coupling constraints are.

1. the OSM model contains only one SLM and one HLM;
2. the SLM to HLM can only be coupled with another.

In SESM/DOC, the choices of the SLM and HLM models that can be used in the OSM layer are specified in the simulatable software and hardware layers.

3.3 Multiple Working Sections for Software/Hardware Layer Model Design

As a co-design modeling approach, SESM/DOC offers three modeling working sections: *software modeling*, *hardware modeling*, and *system modeling*. Figure 4 depicts the user-interface of the SESM/DOC. This environment extends the software architecture of SESM and is implemented using Java™ and MS ACCESS database technologies. In each working section, the SESM concepts and functionalities that separates simulatable and non-simulatable modeling are used. The Template Model, Template Instance Model, and Instance Model are supported and thus modelers to create (or delete/modify) DCO, LCN, and OSM models. For software modeling, Software Template Model (STM), Software Instance Template Model (SITM), and Software Instance Model (SIM) are defined (see Figure 8). Similarly, Hardware Template Model (HTM), Hardware Instance Template Model (HITM), and Hardware Instance Model (HIM) and OSM Template Model (OTM), OSM Instance Template Model (OITM), and OSM Instance Model (OIM) are defined. All software and hardware component models (e.g., software template model) are specified as simulatable models that can have non-simulatable models as state variables or input and output values (see Section 2).

Each of the working sections has its own unique tree structure and block models and support model operations that are unique to DCO, LCN, and OSM. For example, Figure 8 provides the view of a software model working

section which has "Template_SoftwareModel_Tree" (a tree structure only for software models) and also a view for software models (see the "softwareApplication02" model in the right panel). Every simulatable model component in the software modeling working section can be a software application or a software layer (similarly, every simulatable model component in the hardware modeling working section can be a hardware application or a hardware layer). Alternative system models can be synthesized from the DCO and LCN layers.

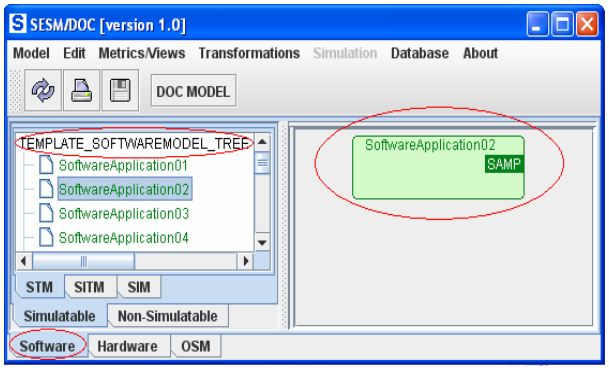

Figure 8. Software Model Working Section

The models in the software and hardware working sections are separated. The software model (or hardware model) can only be created, edited and viewed in its designated working section. In the OSM section, the modeling of software components mapped to hardware components is supported. Therefore, in OSM, there are three kinds of models – *software*, *hardware* and *object system mapping*. When an OSM model is created, a software layer model needs to be selected from the simulatable software model and a hardware layer model needs to be selected from the simulatable hardware model. Figure 9 shows a software layer model chosen for an OSM model in the OSM working section.

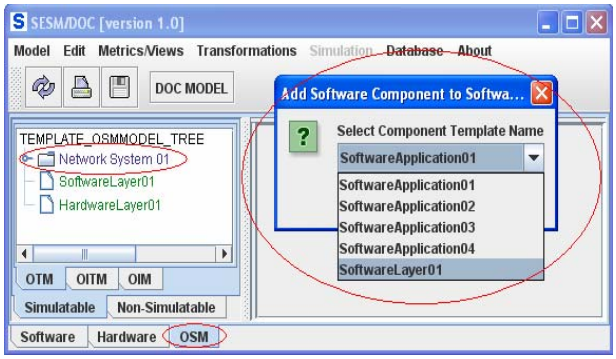

Figure9. Model Integration in OSM Working Section

3.4 Model Visualization and Persistent

As shown above, SESM/DOC supports logical model specifications. It also needs to support visual modeling. Furthermore, all logical models need to be stored in a relational database. The visual and persistent co-design modeling defined simplifies model development, reuse of models, separation of software and hardware, and alternative software/hardware specifications. The visual and persistent modeling capabilities are particularly important for large-scale, complex model systems. This because visual modeling using tree structure and block models reduces model development effort and supports maintaining consistency among a family of alternative models.

3.4.1 Model Visualization

The visual modeling separates software and hardware from one another by extending the SESM visuals for co-design. The distinction among model types defined for software and hardware components, software/hardware layers, and multiple mappings of software layers to hardware layers is important in the modeling of computer network systems.

As the scale of a model grows, it is important to reduce the number of components and their relationships. This is because visual models of large-scale systems is known to be NP hard problem (Young, Cook et al. 2003). Hierarchical modeling combined with a diagonal layout of model components of a composite model reduces significantly the difficulty of visualizing model couplings. However, since couplings between components (or layers) are bi-directional, it is important to use visual notations that distinguish between uni- and bi-directional visual notations. A uni-directional coupling is defined for coupling and shown as a dashed line with a single direction. A bi-directional coupling is defined for mapping and is shown as a directionless dashed line.

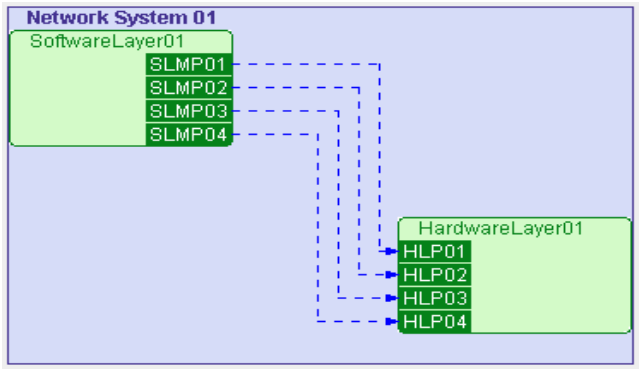

Figure 10. SoftwareLayer01 to HardwareLayer01 Mapping

The use of bi-directional coupling reduces visual clutter that can result even with small-scale models. For example, since software components have bi-directional couplings

with the software layer, the couplings in SESM/DOC are shown as a directionless dashed line (see Figure 4). This reduces significantly the number of couplings. For hardware components, they have bi-directional coupling and thus directionless links are also used (see Figure 6). Considering the OSM layer, the mapping is shown as a dotted line with an arrow (see Figure 10). This visual notation helps to separate compositions defined for software and hardware layers of a system from the assignments of the software layer model to hardware layer model.

3.4.2 Model Persistent

DCO, LCN, and OSM models are stored in a set of relational database tables. The SESM/DOC database schemas play a major role since it allows storing software and hardware co-design models systematically. The logical DCO, LCN, and OSM models specified in SESM/DOC are stored in three sets of tables which are extensions of those defined for SESM. The Entity-Relationship schemas are defined in accordance with the software/hardware co-design logical specification presented above. The different schemas for the DCO, LCN, and OSM conform to the DEVS/DOC logical model abstractions. The "ModelType" schema (i.e., a table in the SESM/DOC database) is introduced to distinguish among DCO, LCN, and OSM models. The software layer and hardware layer, and object system model types and their constraints are also defined as schemas.

3.4.3 Model Consistency

SESM/DOC is devised to ensure consistency among logical, visual, and persistent models. First, it uses the SESM's concepts and constructs to guarantee that all visual modeling operations are consistent with the logical model. Second, the consistency is extended based on the DEVS/DOC models. The SESM/DOC data schemas ensure that the visual models are developed and stored in accordance to their logical specifications. For example, the LCN link group model (LGM) can only contain link models (LM). When a component is added to the LGM, the type of the container model and the model to be added are checked to have proper model types. If the added component is a link model (LM), then the SQL query succeeds. Otherwise, the query fails and an error message is displayed. As another example, when a modeler wants to add a router model (RM) to a link group model, SESM/DOC displays the "A Link Group Model Can Not Contain a Router Model!" since this is an invalid operation. In order to allow only well-defined composite models, the SESM/DOC examines every coupling and only allows those that satisfy the DOC specification. For example, a router group model (RGM) can only be connected to a network interface group model (NIGM); if a RGM is connected to a processor group model (PGM), SESM/DOC displays the "A Router Group Model Can Not be Connected to a Processor Group Model!". Besides ensuring the above constraints, model types are important for object system mapping modeling. In the OSM working section, when a system model is created, only one software layer model and one hardware layer model can be added to the system model. The software layer model and the hardware layer model can be used to define system mapping assignment.

4 DISCUSSION

Since SESM/DOC supports independent software and hardware modeling with the capability to synthesize them, it can be extended with new model types for software and hardware components that are not defined in DOC. Given the existence of the models in a database, simulation models can be partially transformed to simulation code. For example, given DEVS/DOC, the automatically generated primitive and composite software and hardware simulation models have full structural specifications. Specification of primitive models are partial. For example, transition and time advance functions of a primitive model cannot be defined visually or stored in the database.

The separation of concern afforded by SESM/DOC co-design is important toward model Validation, Verification, and Accreditation (VV&A). As the scale and complexity of models increase, verification and validation becomes more difficult. To reduce the immense effort required for developing correct models, the SESM/DOC provides a basis for separately carrying out VV&A for software and hardware as well as their combination. The separation affords systematic verification and validation using the combined logical, visual, and persistent models – i.e., every model's structure is guaranteed to be consistent with the DOC abstract specification. To ensure compliance, the structural model specifications are examined for correctness given the logical software, hardware, and object system mapping model specifications. However, as with all other modeling approaches and tools, a modeler may specify models that are consistent with the DOC abstract model but unsuitable given some desired aspect and/or resolution of a network system structure and behavior. In terms of simulation modeling (and thus validation), even though SESM/DOC can generate partial simulation models, the amount of effort it takes could be significantly less, especially for large-scale and complex models.

5 CONCLUSION

A co-design modeling approach has been developed for describing computer network systems that are defined in terms of combined software and hardware models. It supports visual modeling according to the distributed object computing abstract model. With SESM/DOC, computer network system can be specified in terms of a set of soft-

ware model components (software model layer) mapped onto a set of hardware model components (hardware model layer). The co-design modeling approach is aimed at specifying families of models and also supports generating partial simulation code for DEVS/DOC. This modeling approach is attractive for systems that are being developed to execute using the Global Information Grid and Service Oriented Computing frameworks and technologies.

REFERENCES

ACIMS, Arizona Center for Integrative Modeling and Simulation, Available via <http://www.acims.arizona.edu> [accessed June 2007].

Bendre, S. and H. Sarjoughian 2005. Discrete-Event Behavioral Modeling in SESM: Software Design and Implementation. In *Proceeding of the 2005 Advanced Simulation Technology Conference*, pp. 23-28, San Diego, CA.

Burmester, H., S. Henkler 2005. Visual Model-Driven Development of Software Intensive Systems: A Survey of Available Techniques and Tools. In *Proceeding of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, Dallas, Texas, USA.

Butler, J. 1995. Quantum Modeling of Distributed Object Computing. *Simulation Digest* 24(2): 20-39.

Eker, J., W. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs and Y. Xiong 2003. Taming Heterogeneity--the Ptolemy Approach. In *Proceedings of the IEEE* 91(2): 127-144.

Fu, T. 2002. *Hierarchical Modeling of Large-Scale Systems Using Relational Databases*. Master thesis, Department of Electrical & Computer Engineering, University of Arizona, Tucson, AZ, USA.

Hild, D. 2000. *DEVS-Based Co-Design Modeling and Simulation Framework and Its Supporting Environment*. Ph.D. thesis, Electrical & Computer Engineering, University of Arizona, Tucson, AZ, USA.

Hild, D., H. Sarjoughian and B. Zeigler 2002. DEVS-DOC: A Modeling and Simulation Environment Enabling Distributed Codesign. *IEEE SMC Transactions* 32(1): 78-92.

Hu, W., H. Sarjoughian 2005. Discrete-Event Simulation of Network Systems Using Distributed Object Computing Hybrid Discrete, In *Proceeding of 2005 Symposium on Performance Evaluation of Computer and Telecommunication Systems*, pp. 884-893, Philadelphia, PA.

Mathworks. "MATLAB/SIMULINK." Available via <http://www.mathworks.com/> [accessed March 2007].

ns-2. 2004. "The Network Simulator - ns-2." Available via < http://www.isi.edu/nsnam/ns> [accessed March 2006].

OPNET. 2004. OPNET Modeler. Available via <http://www.opnet.com> [accessed December 2006].

Sarjoughian, H. 2001. An Approach for Scaleable Model Representation and Management. Internal Report, pp. 1-11, Arizona State University, Tempe, AZ.

Sarjoughian, H. 2005. A Scalable Component-based Modeling Environment Supporting Model Validation. In *Proceeding of 2005 Interserive/Industry Training, Simulation, and Education Conference*, pp. 1-11, Orlando, FL.

Sarjoughian, H., D. Hild and B. Zeigler 2000. DEVS-DOC: A Co-Design Modeling and Simulation Environment. *IEEE Computer* 3(33): 110-113.

Sarjoughian, H., R. Flasher 2007. System Modeling with Mixed Object and Data Models. In *Proceeding of the 2007 DEVS Symposium, Spring Simulation Multi-conference* pp. 199-206, Norfolk, VA, USA.

Young, M., J. Cook and L. Mahabadi. 2003. Stochastic Local Search Algorithms for Minimizing Edge Crossings in Complete Rectilinear Graphs. Available via <http://www.cs.unm edu/~young/final_report.pdf> [accessed June 2006].

Zeigler, B., P. E. Hammonds 2007. Modeling & Simulation-Based Data Engineering: Introducing Pragmatics into Ontologies for Net-Centric Information Exchange, in-press.

Zeigler, B., H. Praehofer and T. Kim 2000. *Theory of Modeling & Simulation*, New York: Academic.

AUTHOR BIOGRAPHIES

WEILONG HU is a PhD candidate in the Computer Science and Engineering department at ASU. His research is in simulation-based co-design and software engineering. He can be contacted at <weilong.hu@asu.edu>.

HESSAM S. SARJOUGHIAN is Assistant Professor of Computer Science & Engineering at ASU and Co-Director of the Arizona Center for Integrative Modeling and Simulation. His research focuses on multi-formalism modeling, collaborative modeling, distributed simulation, and software architecture. He can be contacted at <sarjoughian@asu.edu>.