Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

MODELS AND REPRESENTATION OF THEIR OWNERSHIP

Hessam S. Sarjoughian
Bernard P. Zeigler

Arizona Center for Integrative Modeling and Simulation
Electrical and Computer Engineering Department
University of Arizona
Tucson, AZ 85721-0104, U.S.A.

ABSTRACT

Models, similar to other intellectual properties, are increasingly being treated as commodities worthy of protection. Providing ownership for models is key for promoting model reusability, composability, and distributed simulation. However, to date, it appears no principled approach has been developed to support ownership of models. Instead, individuals such as modelers and legal personnel employ ad hoc means to obtain and (re)use models developed and owned by others. In this article, we briefly describe access control capabilities offered by computer languages, operating systems, and HLA ownership management services. The examinations of such methods suggest the need for formal ownership specification. The article discusses, in an informal setting, requirements for model ownership from the point of view of increasing demand and necessity for model reuse, distributed simulation, and future trends for collaborative model development. We develop concepts for model ownership suitable for collaborative model development and distributed execution. Based on the developed concepts, we present an approach, within the DEVS modeling & simulation framework, for specifying model ownership. The article closes with the consideration of the proposed approach for the Collaborative DEVS Modeling environment and a brief discussion of HLA services relevant to model ownership.

1 INTRODUCTION

For many years, research inquires and emphasis in distributed computing, and distributed simulation in particular, has been on advancing computational, communication, time management, and load-balancing techniques. Recently, other basic research inquiries in distributed simulation have focused on in interoperability and scalability issues. Examples of such inquires are middleware technologies such as High Level Architecture (HLA) (Dahmann, Kuhl et al. 1998, DoD 1998, DoD 1998, DoD 1999) and Common Object Request Broker Architecture (CORBA) (Orfali, Harkey et al. 1997; OMG 1998; O'Ryan, Levine et al. 1999). These have been employed to enable some degree of interoperability among distributed simulations (Fujimoto 1990; Fujimoto 1998). Similarly, advanced techniques have been proposed and implemented to reduce amount of data transmission among simulation nodes by a few orders of magnitude (Zeigler, Ball et al. 1998, Zeigler, Hall et al. 1999). While considerable research has been conducted in the areas of simulation interoperability and scalability, M&S research has yet to pursue research in the area of *model ownership*. Conceptual basis and modeling paradigms for model ownership is an essential need for distributed simulation, especially in multi-organizational settings.

Accountability for model ownership is integral to the existence and further advances in collaborative modeling as well as distributed simulation. There are numerous instances where ownership becomes the initial, and often the main, impediment to collaborative model development across multiple, sometime competing, organizations. Consider the case where two teams of modelers from two different organizations must collaborate with one another to develop a complex model comprised of two sets of models. One group of modelers is to devise a model of a communication system including its host hardware with many hundreds of components. The other group of modelers is to embark on building an E-commerce business application. The key observation is that while these two organizations are developing their own models, they must eventually interoperate with one another. This is to due to fact that only combined simulation of these two sets of models can reveal intertwined, complex concealed behavioral characteristic of the overall model. However, modeling of such large-scale, multi-organizational systems are plagued with numerous pitfalls. For example, due to lack of systematic model ownership and consequently lack of availability of other's models, the modeling teams must

make obscure assumptions. The undesirable consequence is that these independently developed sets of models are unlikely to interoperate. However, with well-defined model ownership, teams can use each other's models at varying levels of details throughout model development and execution phases and therefore minimize model interoperability related uncertainties. Furthermore, model ownership can provide orderly handling of proprietary issues.

As of this writing, HLA offers some limited capabilities to designate which set of simulation models can make available their attributes to other simulation models. For example, HLA supports data publishing/subscription and data management - i.e., it provides ownership in the sense of who may publish data as opposed to, for example, who the actual owner of a model is and what authorities the owner may posses. Similarly, in CORBA, the Electronic Commerce taskforce Object Management Group (OMG), has been proposed an specification to support alternative negotiation styles (e.g., bilateral and multilateral) and legal related issues (e.g., jurisdiction). The capabilities offered by HLA and CORBA indicates two distinct aspects of ownership: micro level (attributes of a model are considered) and macro level (entire organizations are considered).

Therefore, in this work we will provide a basis for model ownership concepts such as "modeler rights and privileges" in a collaborative setting. To devise an approach for formally representing model ownership, we will employ the Discrete-event System Specification (DEVS) modeling framework.

2 SCALEABLITY, INTEROPERABLITY, AND OWNERSHIP

Increasingly contemporary software systems are expected to operate in a distributed setting due to, in part, extensive interdependencies among various worlds' economies, short-lived collaborations, and their expected and accidental emerging complexities. Distributed computing technologies are expected to support scalability and interoperability requirements for systems exhibiting homogeneity and heterogeneity characteristics. While scaleabilty and interoperability are attracting much needed attention, in contrast, the necessity and role of distributed/multi-organization ownership is hardly recognized and reckoned with. This is unfortunate since the trio of scalability, interoperability and ownership must be collectively supported to enable distributed modeling and simulation.

The Authoritative Data Source (ADS) project, under the Defense Modeling and Simulation Office (DMSO) Master Plan 5000.59-P directive, has undertaken steps toward supporting data *credibility* and *composability*, and reduction in *cost* (Sheehan, McGlynn et al. 1999). Such efforts are directed toward a process whereby models (1) can be developed and stored according to standards, (2) identified, and (3) can be assigned authority (i.e., credibility) level. This and the HLA initiative are important in building a repository of "reusable" models. However, we believe *collaborative, distributed modeling* paradigm is needed based on the SCO principles as the underlying foundation to guide objectives such as those advocated by the ADS project. Within such a formal collaborative, distributed modeling paradigm, not only the data credibility, composability, and cost objectives are more likely be achieved, the broader needs (e.g., distributed simulation) of the M&S community can be supported as well.

2.1 Scalability

An examination of existing modeling and simulation environments reveals that ever more larger models are needed to represent an array of systems – environmental models (e.g., climatology), enterprise resource planning (e.g., supply chain), computer networks (e.g., FAA), and system of systems (e.g., C4I). Models capable of representing dynamics of such systems can be characterized along two dimensions: (1) number of sub-models and (2) data size and exchange frequency. From the modeling perspective, these dimensions, are interdependent since data size and exchange frequency are in part directly due to the number of sub-models and the number of state variables. The other key factors responsible for increased interaction among submodels can be attributed to model resolution, accuracy, cost, and execution speed.

2.2 Interoperability

Simulation models of a variety of systems have been developed over the span on many decades, many of which in isolation and therefore without the goal of satisfying model interoperability. Consequently, interoperability, generally, is considered at the simulation model execution level. Absence of formal interoperability concepts at the modeling level can be attributed to model development across literally all scientific disciplines without adhering to any formal, comprehensive modeling framework. This may explain why DoD's High Level Architecture (HLA) and its Object Model Template (OMT) proposed standards are primarily specified at the simulation level. HLA/OMT can support "interoperability" among simulations using the object orientation concepts. For example, it appears that interoperability across DoD's training, analysis, and acquisition mandates can be quite hard to achieve without relying on the old ad hoc means. In software engineering, CORBA has been instrumental in supporting heterogeneous software components to interoperate. The

CORBA specification (CORBA 1995) provides a host of services such as events, persistence, naming, time, concurrency, and licensing. These services, however, do not have their foundation in dynamical system modeling and therefore cannot in their present form enable "model interoperability." Nevertheless, the underlying concepts of CORBA provide a sound basis for modeling and simulation interoperability.

2.3 Ownership

We will compare model ownership with access control to show the required additions. Data and specifically component access control, from the object oriented world-view, is through field declarations. For example, "ownership" of simple and higher-order objects (derived from inheritance and composition relationships) can be enforced through field declarations. However, relying solely on such accessibility mechanisms for providing access control offer limited ownership capabilities for model development, model (re)use, and execution. Furthermore, in a distributed setting, such access controls are weak since they cannot provide a systematic approach to assigning, maintaining, and controlling ownership at alternative levels of granularities (e.g., attribute vs. class ownership).

Other means for access control can be enforced through security check and more generally private networks and user authentication. These approaches have been employed for users and their applicability and adaptation for model ownership is an open research inquiry. Next we discuss access control from the OO perspective as well as publish/subscribe in terms of data exchange among distributed nodes and model bases.

Other capabilities related to ownership (e.g., access control) have been available for many years. For example, SCMS Software Configuration Management Systems (SCMS) (Bersoff, Henderson et al. 1980) and configuration programming (Shaw, DeLine et al. 1995, Bishop and Faria 1996) support some types of access to data primarily based on the operating systems' services.

2.3.1 Access Control in Object Orientation

Object orientation provides a rich set of access control fields for each of classes, interfaces, attributes, and methods. A class/object is typically has attributes, methods, as well as inheritance and composition relationships. Attributes generally can be first-class objects. Therefore, an object's attributes can be thought of as component having its own attributes, methods, and inheritance relationship. Object orientation provides a uniform access control policy via field declarations for objects, attributes, and methods. *Class* field modifier choices are public, abstract, and final. *Interface* modifier choices are public and abstract. The access control of a class/interface (object) with an inheritance relationship is determined by its inherited parent field declaration.

In Java, for example, public, protected, private, and static field modifiers can be assigned to *attributes*. Other, less relevant attribute field modifiers are final, transient, and volatile. The final modifier can be used for providing one incarnation of the field, transient modifier provides non-persistence fields, and volatile modifier makes private copies for threads. The field declarations for *methods* are public, protected, private, abstract, static, final, synchronized, and native.

Field declarations and their abilities to control read, write, and execution can be seen in two settings: non-distributed and distributed. The field declarations have well-defined characteristics in non-distributed setting – that is a class hierarchy of objects contained in a single memory workspace. In contrast, field declarations in a distributed setting are primarily through packages. While attributes, methods, and inheritance field declaration provides some level of mutual exclusion, they cannot support ownership since ownership can be enforced through file ownership offered by operating system. Moreover, the field declarations are too weak for control access control for individual use, especially outside of a given hierarchy of classes or a package. For example, consider an attribute for a class `clsA`. By declaring this attribute to have no field modifier, the attribute is available within its own package and to any other child class (e.g., `clsB extends clsA`) that resides in the same package.

2.3.2 Publish/Subscribe in Distributed Computing

In the world of distributed computing, the basic concepts of object-orientation are insufficient due to the fact that there does not exist a single-address space, but instead make it possible to communicate across a network. Accordingly, "distributed object", an extend form of object, make it possible for it to be used in a seamless fashion. Such distributed objects not only have their encapsulated knowledge, but also can be manipulated across a network or send their data to others.

The concepts of publish and subscribe allow a component to publish its own data or subscribe to data from other components. The components capabilities to interchange data, makes it possible for them to be heterogeneous and therefore support portability and scalability. Of course, components should be able to interoperate with inhomogeneous data types under the condition that they are able to transform receiving data to their expected types.

Data exchange can be either through ports or not. Components can have unidirectional input and output ports through which various kinds of data types can be sent or received. Alternatively data can be sent or received through "virtual" bi-directional ports.

2.3.3 Distributed Model Bases

Our underlying assumption is that persistence models exist in an independent fashion in a distributed setting. Moreover, our treatment of the ownership is independent of form of storage – that is models located in a node can be either in a database or not. The basic requirement that must be satisfied is that the model can be treated as an object and independent of any specific programming language.

3 INFUSING OWNERSHIP INTO DEVS MODELING

A rather extensive variety of models, methodologies, and applications have been introduced. For example, models are represented using laws of physics, chemistry, statistics using a variety of mathematical formalisms such as a systems theory, neural networks, fuzzy logic, situation calculus. Aside from these types of models, many of today's contemporary computer systems (e.g., commerce) are modeled using UML (UML 2000) and other computer language-based schemas such as System Entity Structure (SES) (Zeigler 1984; Rozenblit 1992).

Model design and simulation of distributed systems, in comparison to non-distributed (singular) systems, is much more complex. From the standpoint of collaborative model development and distributed simulation, scalability, interoperability, and ownership considerations are of fundamental importance. In particular, a viable modeling paradigm must enable its users to represent and formalize interoperability and ownership concepts as well as lending itself as blueprint for alternative forms of simulation execution strategies.

Given the existence of numerous modeling approaches, we focus our attention on systems theory as the basis to formalize interoperability and ownership concepts discuss above. From the three alternative system theoretic formalisms, discrete event system specification has been shown to lend itself quite well in characterizing many kinds of systems exhibiting causal, time-invariant (varying), and (non) deterministic behaviors. More importantly, distributed systems, generally, are event-oriented and are generally best represented in discrete event form.

Given modeling paradigms founded on the principles of System Theory, the Discrete-event System Specification (DEVS) is believed to be the most appropriate candidate for incorporating distributed model ownership. The DEVS modeling formalism enables characterization of systems in terms of hierarchical modules with well-defined interfaces. Due to its system-theoretic foundations, DEVS modeling paradigm naturally maps into object-orientation implementation and consequently has been implemented in sequential, parallel, and distributed environments (AIS 1998; Zeigler, Praehofer et al. 2000). In addition to DEVS Modeling constructs, the System Entity Structure (SES) in conjunction with rule-based system have been proposed and employed to deal with design choices. The combination of DEVS and SES can provide a suite of modeling capabilities for representing model spaces and dynamics containing intelligent features (e.g., rule-based model synthesis, fuzzy logic, and neural networks based dynamics.)

3.1 Discrete Event System Specification

The Discrete Event System Specification (DEVS) (Zeigler, Praehofer et al. 2000)modeling approach supports capturing a system's structure in terms of *atomic* and/or *coupled* models where coupled models are hierarchical satisfying closure under coupling (Zeigler 1984). While a part of a system can be represented as an atomic model with well-defined input/output interfaces, a system represented as a DEVS coupled model designates how systems can be coupled together to form system of systems. Such coupled models also have the same input/output interfaces. Given atomic models, DEVS coupled models can be formed in a straightforward manner. Both atomic and coupled models can be simulated using sequential and/or various forms of parallel, distributed computational techniques (Zeigler and Sarjoughian 1997).

3.2 Atomic Model with Ownership

Aside from OO declaration fields such as public, we employ the broader concepts of publish and subscribe. These concepts are used with distributed systems (e.g., client/server applications and distributed simulations). We define DEVS atomic model with ownership as a mathematical structure:

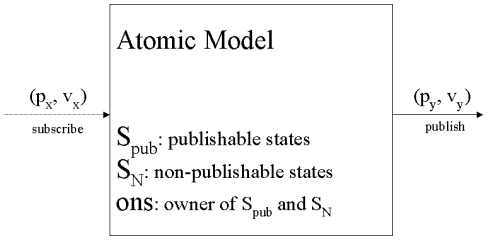
$AM = \langle X, Y, S, \delta, \lambda, ta, ons \rangle$ where

- X set of *input events*,
- S set of *sequential states*,
- Y set of *output events,*
- δ *state transitions* due to internal changes and external stimuli
- λ *output function* generating external events as outputs,
- *ta* *time advance* function,

3.2.1 Ons Ownership

Sets S and Y represent *publishable* and *non-publishable* data of an atomic model. Similarly, set X represents *subscribable* data. The element *ons* is used to assign/identify the model owner. The model ownership is w.r.t. the data contained in the model and specifically what

can be made available to other models. The remaining elements of the atomic model structure represent its dynamics. In terms of modular, hierarchical distributed modeling, internal functionality of an atomic model is of interest only to the extent of its states and input/output events (data) associated with their input/output ports. Therefore, we can characterize an atomic model's ownership as follows.



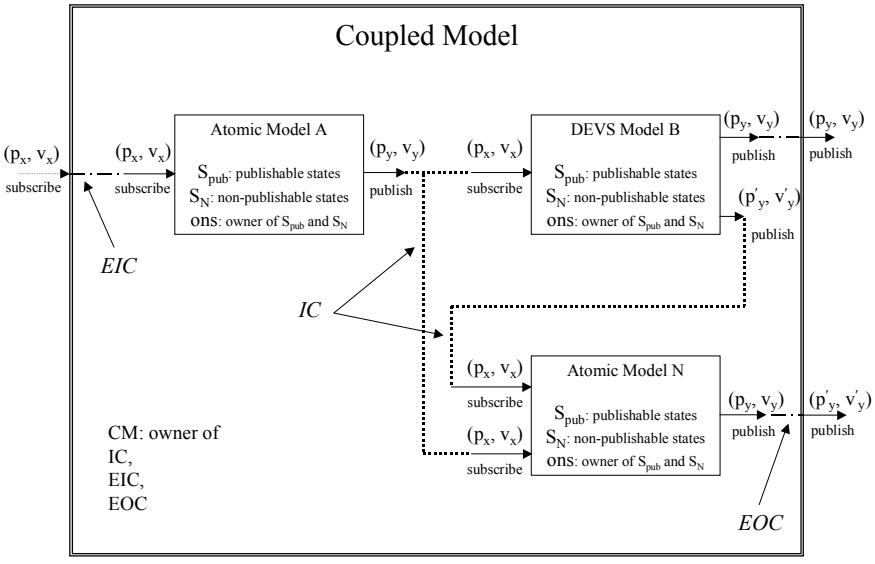
X = $\{(p_x, v_x) \mid p_x \in P_x$, *input port names*, $v_x \in V_x$, *input values*, $S_{sub} \rightarrow V_x$, $S_{sub} = \overline{S}_{pub} \cup \overline{P}_{pub}$ where $\overline{S}_{pub}$ and $\overline{P}_{pub}$ are other DEVS models publishable *states* and *parameters*, respectively.$\}$

Y = $\{(p_y, v_y) \mid p_y \in P_y$, arbitrary *input port names*, $v_y \in V_y$, arbitrary *input values*, $S_{pub} \rightarrow V_y$, $S_{pub} \subseteq S$.$\}$

S = $S_{pub} \cup S_N$ where S_{pub} and S_N are the set of states that are *publishable* and *non-publishable* states, respectively.

ons is an arbitrary name with an associated authentication (passwd.) The model owner is able to specify publishable vs. non-publishable states and parameters. Owner can be a singular or not. If multiple owners are permitted or desired, appropriate levels of authorities (e.g., master/slave) mechanisms are needed in addition to support for resolving conflicts for inconsistent concurrent assignments of (un)publishable states.

The existence of input/output ports can result in separate ownership of states/parameters, thus allowing for distinct ownership for ports and states/parameters. Under the supposition that both ports and states/parameters of an atomic model can be assigned ownership, ownership can be of three types: (a) single owner for ports and states/parameters, (b) a single owner for each port and the data (i.e., states/parameters) that is made available via it, (c) multiple owners for ports and a single owner for states/parameters. Input and output ports have opposite scope w.r.t. states/parameters. An output port can sanction what subset of publishable states can be made available to other models. That is, output port ownership is superior to data ownership. The date owner, however, would be able to sanction what input values (states published by other models) it may choose to use, thus asserting control (superiority) over input port ownership. These characterizations, along with the concept of modularity and object orientation, point to case (a) as being the most ideal. In this case, state/parameters and output ports ownership are the same. Similarly, ownership of input ports would be the same as accepting/rejecting other model's published states/parameters. Cases (b) and (c) provide no fundamental advantage since supporting such finer grain ownership will complicate needlessly ownership characterization of coupled models. In the next section, we consider ownership of ports for the coupled model case in view of our present discussion.



Since DEVS, Differential Equations System Specification (DESS), and Discrete-Time System Specification (DTSS) are founded on the basis of systems theory concept [19], the ownership characterization of the DEVS atomic model is equally applicable to continuous systems that can be represented as differential equations. (Treatment for the discrete-time systems is the same as DEVS.) That is, the specification of DESS is $AM_{cont} = \langle X, Q, Y, f, \lambda, ons \rangle$ where all of its elements are the same as those of DEVS atomic model except *f* which is the rate of change function.

3.3 Coupled Model with Ownership

Given a coupled model (CM), its representation is concise and reusable since any coupled model has a corresponding basic DEVS model due to the closure-under-coupling property. The couplings among components can be systematically captured using output to input mappings. In particular, there exist three different types of coupling: *internal coupling, external input coupling,* and *external output coupling*. Internal coupling interconnects components of a coupled model. External input coupling interconnects input ports of a coupled model to input ports of its components. Similarly, external output coupling interconnect component output ports of a coupled model to the output ports of the coupled model itself. Note that the ports provide a generic pipe through which a variety of messages (data/objects) can be transferred from one component to another

$CM = \langle X, Y, D, \{M_d \mid d \in D\}, IC, EIC, EOC, ons \rangle$

where

- *X* set of *input port* and *value pairs;*
- *Y* set of *output port* and *value pairs;*
- *D* set of the *component names;*
- *IC* set of *internal coupling* connecting component outputs to component inputs;
- *EIC* set of *External input couplings* connecting external inputs to component inputs;
- *EOC* set of *external output coupling* connecting component outputs to external outputs;
- *ons* an arbitrary name with an associated authentication (passwd.) Publishable vs. non-publishable states and parameters are controlled by MO via the internal, external input, and external output couplings ownership.

Subject to:

$X = \{(p_x, v_x) \mid p_x \in InPorts,\ v_x \in V_x, , S_{sub} \rightarrow V_x\ ,\ S_{sub} = \bar{S}_{pub} \cup \bar{P}_{pub}\ ,\ \bar{S}_{pub} = S^1{}_{pub} \times \ldots \times S^m{}_{pub}$ *such that* $S^i{}_{pub} \neq S_{M_d}$ *for i = 1, ..., m for* $d \in D.\}$

$Y = \{(p_y, v_y) \mid p_y \in OutPorts,\ v_y \in V_y, , S_{pub} \cup P_{pub} \rightarrow V_y\ ,\ S_{pub} \subseteq S^1{}_{pub} \times \ldots \times S^n{}_{pub}$ *such that* $S^j{}_{pub} = S_{M_d}$ *for j = 1, ..., n for* $d \in D.\}$

Each component is a DEVS model. That is, for each $d \in D$,

$M_d = \langle X, Y, S, \delta, \lambda, ta, OM \rangle$ is a DEVS as defined above.

Coupling and ownership specification:

$IC \subseteq \{((a, (p_{y,a}\ ,\ v_{y,a})), (b, (p_{x,b}\ ,\ v_{x,b}))) \mid a, b \in D,$

$p_{y,a} \in OutPorts_a\ ,\ S^a{}_{pub} \subseteq S^a\ ,\ S^a{}_{pub} \rightarrow V_{y,a}\ ,\ v_{y,a} \in V_{y,a}\ ,$

$p_{x,b} \in InPorts_b\ ,\ v_{x,b} \in V_{x,b}\ , V_{x,b} \subseteq V_{y,a}\ .\}$

Interpretation: model owner can (1) ascertain components couplings and (2) restrict child component *a* output values $(V_{y,a})$ that can be made available as input values $(V_{x,b})$ to sibling component *b.}*

$EIC \subseteq \{((CM, (p_{x,CM}\ ,\ v_{x,CM})), (d, (p_{x,d}\ ,\ v_{x,d}))) \mid d \in D,\ p_{x,CM} \in InPorts_{CM},\ p_{x,d} \in InPorts_d,\ \bar{S}_{pub} \rightarrow V_{x,CM}\ ,\ \bar{S}_{pub} = S^1{}_{pub} \times \ldots \times S^k{}_{pub}$ *such that* $S^i{}_{pub} \neq S_{M_d}$ *for i = 1, ..., k for* $d \in D,\ V_{x,d} \subseteq V_{x,CM}$ *for* $d \in D.\}$

Interpretation: model owner can (1) ascertain *EIC* couplings and (2) restrict component *CM* input values $(V_{x,CM})$ that can be made available as input values $(V_{x,d})$ to child component *d.}*

$EOC \subseteq \{((d, (p_{y,d}\ ,\ v_{y,d})), (CM, (p_{y,CM}\ ,\ v_{y,CM}))) \mid d \in D,\ p_{y,d} \in OutPorts_d\ ,\ p_{y,CM} \in OutPorts_{CM},\ S^d{}_{pub} \rightarrow V_{y,d}\ ,\ S^{CM}{}_{pub} \subseteq S^1{}_{pub} \times \ldots \times S^h{}_{pub}$ *such that* $S^i{}_{pub} = S_{M_d}$ *for i = 1, ..., h for* $d \in D,\ V_{x,d} \subseteq V_{x,CM}$ *for* $d \in D.\}$

Interpretation: model owner can (1) own *EOC* couplings and (2) restrict child component *d* output values $(V_{x,d})$ that can be made available as output values $(V_{x,CM})$ to component *CM.*

No direct feedback loops between the output and input ports of any DEVS model is allowed. That is, no output port of a component may be connected to an input port of the same component i.e., *((a, ($p_{y,a}$, $v_{y,a}$)), (b, ($p_{x,b}$, $v_{x,b}$))) ∈ IC implies a ≠ b.* For simplicity, without loss of generality, our formulation does not explicitly account for parameters.

Coupled models are defined to own couplings alone. As discussed in Section 3.2, the ports of any atomic model can have ownership associated with them. A simple approach is to let a coupled model own its input/output ports with the implication that the EIC (subscription) and EOC (publishing) ownership are also applicable to input and output ports. This approach lends itself to modular ownership of ports for any DEVS model in a simple manner. Of course, it is possible to grant multiple ownership in a coupled model – let each of IC, EIC, and EOC be owned by separate owners. Or indeed, designate ownership at a finer grain level – have multiple owners for multiple couplings for each set of declared IC, EIC, and EOC coupling. These finer grain ownership strategies require complex ownership strategies and are not discussed here.

3.4 Role of Ownership in System Entity Structure

A System Entity Structure (SES) provides the means to represent a family of models as a labeled tree (Rozenblit 1992). Two of its key features are support for decomposition and specialization. The former allows representing a large system in terms of smaller systems. The latter supports representation of alternative choices. Specialization enables representing a generic model (e.g., a computer display model) and its specialized variations (a flat panel display or a CRT display.) Based on SES axiomatic specifications, a family of models can be represented and pruned to study and experiment with design choices (alternatives.) An important, salient feature of SES is its ability to represent models not only in terms of their decomposition and specialization, but also aspects (SES represents alternative decompositions via aspects.)

4 SUPPORTING ENVIRONMENTS

To demonstrate the applicability and usefulness of ownership concepts, we propose the Collaborative DEVS Modeler (CDM) (Sarjoughian, Nutaro et al. 1999). Such an environment is attractive since it can support dispersed modelers to develop models in a collaborative setting. In collaborative settings, ownership related concerns are always present if each modeler owns her model (i.e., models are stored, maintained, accessed from the modeler's physical location).

The collaborative DEVS modeler is based on the integration of two disciplines: *Modeling* and *Distributed Object Computing* (Sarjoughian, Nutaro et al. 1999). The Collaborative DEVS Modeler approach to modeling is based on the concept of a session which is a "loosely bounded workspace" within which a group of knowledge modelers develop a model as a team. This conceptual view of CDM illustrates two basic issues: distribution of modelers (knowledge workers) and their resources across time and space. It supports synchronous and asynchronous synthesis of models. Synchronous collaboration can be supported by complementary capabilities such as text-based and video-teleconferencing tools in order to support rich collaboration among modelers. These capabilities can be tailored for modeling by providing representations of common modeling primitives and enforcing correct modeling constructs. Asynchronous collaboration also benefits from the availability of modeling primitives and semantics enforcement.

We can distinguish two types of modeling activities: *model construction* and *model synthesis* (Zeigler, Sarjoughian et al. 1997). Model construction mostly deals with identifying dynamics of models while model synthesis concerned with synthesizing coupled models given existing atomic and/or coupled models. Realizing that the boundary between construction and synthesis can be imprecise due to the iterative nature of modeling, the development of ownership concepts, methods, and methodologies demand caution.

5 RELATED WORK

5.1 High Level Architecture

The Department of Defense has instituted the HLA standard (HLA 1999)across its branches to support reusable, plug and play heterogeneous distributed simulation. Its overarching objective is to enable simulation interoperability and reuse, thus enabling various simulations to interoperate with one another in logical and/or real-time. HLA/OMT is founded on the Federation Development Process following software engineering principles. As such, HLA/OMT offers some support for distributed "model design" by following the software engineering principles, but intentionally does not incorporate any specific modeling paradigm and methodology (Sarjoughian and Zeigler 1999; Sarjoughian and Zeigler 1999).

The HLA standard is a modeling and simulation interface specification. The HLA/OMT specification, one of the three pieces of the HLA standard, defines primarily HLA object models as opposed to model dynamics. Users are able to develop different classes of simulations across alternative domains using HLA Interface Specification (IS) and Object Model Template (OMT) for defining data

exchange interfaces among *federates* (simulation components) and rules for constructing *federations* (composite simulations). HLA/IS provides *federation, time, data distribution, declaration, object,* and *ownership management* services. Alternative combinations of the latter three services can be used to enable (a) object class registration and discovery, instance attribute updating and reflection, (b) parameter sending or receiving belonging to interaction classes. HLA/OMT supports object and interaction class hierarchies – these class hierarchies are interchangeable. Publish/subscribe are used with object class attributes and send/receive are used with interaction class parameters. Of significant relevance in terms of ownership are the declaration, object, and ownership management services provided by the Run Time Infrastructure (RTI) to federates. A great majority of these services are in support of simulation execution while others are for simulation model (i.e., federate) declaration and initialization.

Declaration Management services are used by federates to declare their intention to generate or receive information. Object and interaction classes are the focus of declaration management since they must be available prior to, for example, registering object instances, updating of attributes and sending of interactions. The Declaration Management services are start(stop) registration of object classes, turning on(off) interaction classes, (un)publishing object/interaction classes and (un)subscribing object class attributes, and (un)subscribing interaction classes. Such services rely on some form of ownership of the object and interaction classes and their content (attributes and parameters).

Object Management services such as object instances registration and delete object instance allow the federates of a federation to transfer ownership of object instance attribute with on another. The RTI and federate "update attribute values" and "reflect attribute values" services together provide the basic mechanism for data exchange. Other Object Management services support sending and receiving of interactions. The send and receive interaction services enable sending of interactions to federation and receipt of interactions by a federate.

Ownership Management services support to grant and transfer ownership of one or more attributes of an object instance. A federate is an owner of an attribute and not the object instance. This allows various attributes of an object instance to be "owned" by different federates. Using these services, a federate can invoke services such as "update attribute value." Some of the main RTI's Ownership Management services are "unconditional/negotiated attribute ownership divestiture", "attribute ownership acquisition," "attribute ownership release response," and "is attribute owned by federate". The complementary services provided by a federate are "inform attribute ownership" and "attribute ownership unavailable."

Attributes may or may not have ownership. Owner of an instance attribute is the federate responsible for publishing it. Likewise, if a federate seizes to publish an attribute the attribute will have no owner. That is, there exist interdependencies between ownership and publication. We observe that this work presented allows ownership assignment using well-defined comprehensive mechanisms. It further supports ownership persistence, thus supporting dynamic, run-time ownership modifications and run time execution.

6 CONCLUSIONS

We have described the ownership for models and the necessity of models having distinct owners. We discussed a variety of existing means by which models may be controlled. In particular, the concepts of access control and HLA publish/subscribe were discussed in relation to model ownership. After analyzing the constituents of ownership from a formal point of view, we extended the DEVS modeling framework to support ownership assignment. Ownership was defined for atomic and coupled models. We further described how the system entity structure knowledge representation scheme can be extended to support ownership as well.

ACKNOWLEDGMENTS

This research has been supported in part by NSF Next Generation Software (NGS) grant #EIA-9975050 and DARPA Advanced Simulation Technology Thrust (ASTT) Contract #N6133997K-0007.

REFERENCES

AIS (1998). AI & Simulation Research Group. <http://www.acims.arizona.edu>.

Bersoff, E., V. Henderson, et al. 1980. *Software Configuration Management*, Prentice-Hall.

Bishop, J. and R. Faria 1996. Connectors in configuration programming languages: are they necessary? *3rd International Conference on Configurable Distributed Systems*, Annapolis, MD.

CORBA 1995. CORBA: Architecture and Specification, OMG.

Dahmann, J. S., F. Kuhl, et al. 1998. Standards for simulation: as simple as possible but not simpler the high level architecture for simulation. *Simulation* **71**(6): 378-387.

DoD, U. S. 1998. High-Level Architecture Interface Specification (Version 1.3).

DoD, U. S. 1998. High-Level Architecture Rules (Version 1.3).

DoD, U. S. 1999. High-Level Architecture Object Model Specification (Version 1.4).

Fujimoto, R. 1990. Distributed simulation. *Communications of the ACM* **33**(10): 30-53.

Fujimoto, R. 1998. Time management in the high-level architecture. *Simulation* **71**(6): 388-400.

HLA 1999. High Level Architecture, Defense Modeling and Simulation Office.

OMG 1998. CORBA/IIOP 2.2 Specification, http://www.omg.org/corba/corbaiiop.html.

Orfali, R., D. Harkey, et al. 1997. *The Essential Client/Server Survival Guide*, John Wiley & Sons.

Orfali, R., D. Harkey, et al. 1995. *The Essential Distributed Objects Survival Guide*, John Wiley & Sons.

O'Ryan, C., D. L. Levine, et al. 1999. Applying a scaleable corba events service to large-scale distributed interactive simulations. *5th Workshop on Object-oriented Real-time Dependable Systems*.

Rozenblit, J. R., J.F. Hu 1992. Integrated knowledge representation and management in simulation based design generation. *IMACS Journal of Mathematics and Computers in Simulation* **34**(3-4): 262-282.

Sarjoughian, H. S., J. Nutaro, et al. 1999. Collaborative DEVS modeler. *International Conference on Web-Based Modeling and Simulation*, San Francisco, SCS.

Sarjoughian, H. S. and B. P. Zeigler 1999. Collaborative modeling: the missing piece of distributed simulation. enabling technology for simulation science, *13th SPIE*, Orlando, FL.

Sarjoughian, H. S. and B. P. Zeigler 1999. The role of collaborative DEVS modeler in federation development. *Simulation Interoperability Workshop*, Orlando, FL.

Shaw, M., R. DeLine, et al. 1995. Abstractions for software architecture and tools to support them. *IEEE Transactions on Software Engineering* **21**(4): 314-335.

Sheehan, J., L. McGlynn, et al. 1999. Authoritative data sources: how do i efficiently find the knowledge i require? *PHALANX* **32**(1): 13-16.

UML 2000. Unified Modeling Language. <http://www.rational.com/uml/index.jtmpl>.

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

Zeigler, B. P., G. Ball, H. S. Sarjoughian 1998. The DEVS/HLA Distributed Simulation Environment And Its Support for Predictive Filtering, ECE, The University of Arizona.

Zeigler, B. P., S. B. Hall, et al. 1999. Exploiting HLA and DEVS to promote interoperability and reuse in lockheed's corporate environment. *Simulation* **73**(5): 288-295.

Zeigler, B. P., H. Praehofer, et al. 2000. *Theory of Modeling and Simulation, 2nd Edition*, Academic Press.

Zeigler, B. P. and H. S. Sarjoughian 1997. Object-oriented DEVS. *11th SPIE*, Orlando, Florida.

AUTHOR BIOGRAPHIES

HESSAM S. SARJOUGHIAN is Assistant Research Professor of Electrical and Computer Engineering at the University of Arizona. His current research interests are in theory, methodology, and practice of distributed/ collaborative modeling & simulation. His other research interests are in AI and Software Engineering. His email and web addresses are <hessam@ece.arizona.edu> and <www.acims.arizona.edu >.

BERNARD P. ZEIGLER is Professor of Electrical and Computer Engineering at the University of Arizona, Tucson. He has written several foundational books on modeling and simulation theory and methodology. He is currently leading a DARPA sponsored project on DEVS framework for HLA and predictive contracts. He is a Fellow of the IEEE. His email and web addresses are <zeigler@ece.arizona.edu> and <www.acims.arizona.edu >.