Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

EXTEND SRML SCHEMA BASED ON DEVS: AN EXECUTABLE DEVS LANGUAGE

Chen Liu
Qun Li
Wei-ping Wang
Yi-fan Zhu

School of Information System and Management
National University of Defense Technology
Changsha 410073 , Hunan, CHINA

ABSTRACT

This paper analyzes the significance of the representation and reusability of SRML when being used in simulation models as well as its drawbacks. The paper also discusses the ways to extend SRML schema based on DEVS. The emphasis is placed on the elaboration of mapping DEVS onto SRML schema to formulate SRML's basic syntax and semantics for the structure and behavior representation of atomic model and coupled model. The model structure, such as property, input interface, output interface and sub-model composition, are described by a group of XML marks. The model behavior, such as external transition, internal transition, output and time-advance functions are described by script language and a group of standard interface offered by SRML simulator in Script marks. The paper then reviews the SRML element architecture and finally gives a simulation demo of using SRML to build differential equation model.

1 SRML

SRML (Simulation Reference Markup Language) (Reichenthal 2002 and Reichenthal 2003) was first raised by Steven. W. Reichenthal from Boeing Co. and was submitted to W3C and subsequently released as a standard draft in 2002. W3C has now set up a formal SRML research team to study SRML schema and design standard simulator interface.

The emergence of SRML is the result of modeling and simulation development. The process of modeling and simulation in large-scale simulation project development consists of many complicated activities catering multi-fields. Its accomplishment and development require the cooperation of different professional staff in different fields and different geological locations. The simulation technology is also moving from single field application to collaborative simulation development. The basic problem in collaborative simulation is the model integration and model reusability. HLA resolved the simulation interoperability in execution time but not in design time (Tolk 2002). The hot research topic in Pentagon is CMSE (Composable Mission Space Environment) (DMSO 2004, Tolk 2004 and Davis 2004), which is focused on the normalization of simulation models to achieve syntactic and semantic composition of simulation models.

SRML is aiming to solve the normalization problem in simulation models in order for the models to be easily developed and reused. SRML is based on XML data exchange standard and declare a group of less quantity but relatively complete elements and elements properties to describe abstract characteristics, structures, and behavior to support building system model. It can support modeler to maximum extent to build simulation entity with the XML elements. At the same time, SRML attempts to create a flexible reference standard to represent simulation in order to make simulation model integration and reuse convenient. Different simulation fields can also apply standard XML to describe different simulation methods, such as Petri-Net, Finite state automata and block models.

SRML draft is still improving and some elements and properties have deficiencies, especially *simulation* elements, *ItemClass* and *Item* element definition. SRML needs to be amended and extended to incorporate existing simulation theories and methods. In addition, SRML requires complex application examples to test and improve.

2 DEVS

DEVS(Discrete Event System Specification) (Zeigler 2000) was developed based on the research of system theory by Professor B. P. Zeigler. DEVS considers each system as a model with independent internal structure, behavior and explicit I/O interface. Some of atomic models can form coupled model by means of certain connection relationship and coupled model can subsequently be used as element of

bigger coupled model. Therefore the modular and hierarchical modeling pattern would be achieved.

An atomic model of DEVS is a structure：

$M=<X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta>$

X: the set of input values

S: the set of states;

Y: the set of output values

δ_{int}: $S \rightarrow S$, is the internal transition function

δ_{ext}: $Q \times X \rightarrow S$, is the external transition function, where $Q=\{(s, e) \mid s \in S, 0 \le e \le ta(s)\}$: is the total states set

e: the time elapsed since last transition

λ: $S \rightarrow Y$, is the output function

ta: $S \rightarrow R^{+}_{0,\infty}$, is the time advance function.

The execution process of atomic model is illustrated in Figure 1. When atomic model received input event from input ports, the external transition function is triggered, and the states are changed. Time-advance function control the time of internal transition, when the elapsed time $e= ta(s)$, the model output the value $\lambda(s)$ to output ports, and change to state δ_{int} (s), then the next internal transition time is advanced to time now + $ta(s)$.

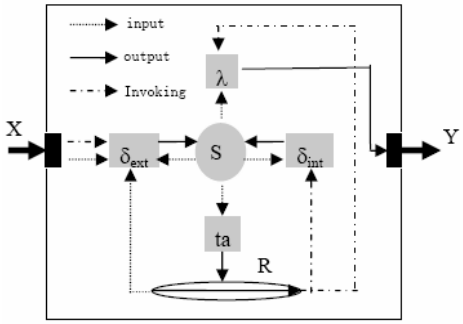

Figure 1: The execution mechanism of atomic model.

A coupled model of DEVS is a structure as follow：

$N=<X, Y, D, \{M_d\}, \{I_d\}, \{Z_{id}\}, Select>$

X：the set of input values

Y：the set of output values

D：the set of components included in coupled model of DEVS

$\{M_d\}$：$\forall d \in D$，M_d is an Atomic model of DEVS

$\{I_d\}$：$\forall d \in D \cup \{N\}$，$I_d$ is the influencer set of d，$I_d \subseteq D \cup \{N\}, d \notin I_d$

$\{Z_{id}\}$：$\forall i \in I_d$，Z_{id} is a function, the i to d output translation with

Z_{id} ：$X \rightarrow X_d$，if $i = N$

Z_{id} ：$Y_i \rightarrow Y$，if $d = N$

Z_{id} ：$Y_i \rightarrow X_d$，if $d \neq N \wedge i \neq N$.

Coupled models result from composing atomic models or “lower level” coupled models. They comprise the set of components, hence building models, their own set of input and output ports, and a set of coupling specifications between the models, which includes transition functions between the ports and the models as well as between the models.

DEVS provides a modular and hierarchical simulation modeling methodology and unified model description framework on top of the system theory. It also applies set language to conduct rigorous mathematical approval and offers theoretical basis for system modeling and simulation. However, DEVS uses formal set language, can only specify system abstractly, and is not able to be used directly in model representation and simulation. XML is now becoming the standard of data exchange in software application. Hence incorporating DEVS to design SRML schema is the solution of standardizing simulation model.

3 MAPPING DEVS ONTO SRML

The mapping relationship between DEVS and SRML is summarized in Table 1.

Table 1: Mapping relationship between DEVS and SRML

Atomic model	*ItemClass* element
$X=\{(x, px)\}$	*EventSink* elements set
$Y=\{(y, py)\}$	*EventDispatcher* elements set
x, y	*EventClass* element
S	*Property* elements set
δ_{int}	*Script* element with *function eventSinkName*(*Event* event)
δ_{ext}	*Script* element with *function eventSinkName*(*Event* event)
λ	*Script* element with simulator API *SendEvent*(*String* eventDispatcherName, Event event) *PostEvent*(*String* eventDispatcherName, *Event* event) *BroadcastEvent*(*Event* event, *Boolean* direction)
ta	*Script* element with simulator API *ScheduleEvent*(*String* eventSinkName, *Event* event) *PostEvent*() and *BroadcastEvent*()
Coupled model	*ItemClass* element
$X=\{(x, px)\}$	*EventSink* elements set
$Y=\{(y, py)\}$	*EventDispatcher* elements set
$D,\{M_d\},\{I_d\},\{Z_{id}\}$	*Item* elements set with *Link* element
Select	Simultaneous events will be processed according as the priority of model instances

3.1 Mapping Atomic Model of DEVS onto SRML

$M=<X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta>$

M: Represents atomic model of DEVS. In SRML, atomic model is declared with the *ItemClass* element. The attributes of ItemClass element include *Name*, *Language*, *SuperClass* and so on. *Name* attribute specifies the atomic model's unique name. *Language* attribute specifies which kind of action language is being used to develop model's behavior. The action language can be "JavaScript", "Java", "C" in SRML. *SuperClass* attribute specifies the super atomic model of this model, SRML support single root inheritance.

X: The input set of atomic model. In SRML, the input interface is described by *EventSink* element. One *ItemClass* element can consist of multi *EventSink* elements. The attributes of *EventSink* include *Name*, *EventClass*, *LinkFixed* and so on. *Name* attribute specifies the unique name of an input interface. *EventClass* attribute specifies the event type which the model can receive from the input interface. All event types used for model interaction are defined by the element *EventClass* In SRML. *LinkFixed* attribute specifies the event receiving mode from the input interface. *LinkFixed* is a boolean type attribute, "true" is the "link" mode, which implies that the input interface can only receive the event sent by the model explicitly linking with it, "false" is the "SubscribeAndPublish" mode, which implies that the input interface can receive all events sent by other models with the same event type specified by the *EventClass* attribute.

Y: The output set of atomic model. In SRML, the output interface is described by *EventDispatcher* element. One *ItemClass* element can include multi *EventDispatcher* elements. The attributes of *EventDispatcher* include *Name*, *EventClass*, *LinkFixed* and so on. *Name* attribute specifies the unique name of an output interface. *EventClass* attribute specifies the event type which the model may send to the output interface. *LinkFixed* is a boolean type attribute, "true" is the "link" mode, which implies that the output interface can only send the event to the model explicitly linking with it, "false" is the "SubscribeAndPublish" mode, which implies that the output interface can send all events to other models with the same event type specified by the *EventClass* attribute.

S: The state set of atomic model. In SRML, the state is described by *Property* element. One *ItemClass* element can consist of multi *Property* elements. The attributes of *Property* include *Name*, *DataType* and so on. *Name* attribute specifies the unique name of a state. *DataType* attribute specifies the data type of a state.

δ_{ext}, δ_{int}, λ, *ta*: The external transition function, internal transition function, send function and time advance function of atomic model. In SRML, the dynamic behavior of DEVS model is specified in element *Script*. The attributes of *Script* include *Name*, *Type* and so on. *Name* attribute specifies the unique name of a *Script*. *Type* attribute specifies the script language type, such as "javascript". Script language can define variable and function. It is easy to build model's behavior.

δ_{ext}: External transition function is used to define the atomic model's behavior triggered by external event received from input interface. The implementation formalism in javascript language is as the following code. *Function* is the javascript function declaration. *EventSinkName* is the external function name which must be identical with the name of one of *EventSink* elements defined in *ItemCla*ss. *Event* is the parameter of the external function, which is the received external event. External function can update model's states by the received external events.

```
function EventSinkName(Event event){
    var x=event.x;
    …
}
```

δ_{int}: Internal transition function is used to define the atomic model's behavior triggered by internal time advance event scheduled by the model. The implementation formalism in javascript language is the same with external transition function. The difference is that external event is sent by other model and consists of the information used to interact between models, otherwise internal event is scheduled by the model itself and consists only the current time information for time advance.

```
function EventSinkName(Event event){
    var x=event.time;
    …
}
```

λ: Output function is used to define the output behavior of the atomic model. The output function is implemented with the following standardized functions supported by the SRML simulator.

SendEvent(*String* eventDispatcherName, *Event* event). The function is used to send synchronous event to the output interface. Once the synchronous event is sent, the received model will invoke the external function at once. The eventDispatcherName parameter is the name of output interface; the event parameter is the object of the event to be sent.

PostEvent(*String* eventDispatcherName, *Event* event). The function is used to send asynchronous event to the output interface. When the asynchronous event is sent, it will be buffered in the event queue until the event happen time is fulfilled. Then the event destination models will receive the event and invoke the external function. The meanings of the parameters are the same as above.

BroadcastEvent(*Event* event, *Boolean* direction). The function is used to send synchronous or asynchronous event among the models at different levels. The direction parameter is the direction of event sending, "true" is the upwards sending, "false" is the downwards sending.

ta: Time-advance function is used to define the time advancing behavior of atomic model. The time advance function is implemented with the following standardized functions supported by the SRML simulator.

ScheduleEvent(*String* eventSinkName, *Event* event). The function is used to schedule the next event by the model itself. The next event's time will advance the global clock. When the global clock time is identical with the time of the next event, the model will invoke the internal transition function. The eventSinkName parameter is the name of clock tick *EventSink* element defined by the model.

The functions *PostEvent*() and *BroadcastEvent*() can also perform the time-advance behavior when they send asynchronous events.

3.2 Mapping Coupled Model of DEVS onto SRML

$N=<X, Y, D, \{M_d\}, \{I_d\}, \{Z_{id}\}, Select>$

N: Represents coupled model of DEVS. In SRML, coupled model is described by the *ItemClass* element too.

X: The input set of coupled model. In SRML, the input interface is described by *EventSink* element. The meaning is the same as that in atomic model.

Y: The output set of coupled model. In SRML, the output interface is described by *EventDispatcher* element. The meaning is the same as that in atomic model.

$D, \{M_d\}, \{I_d\}, \{Z_{id}\}$: Represents the sub-components set and the construction relation. In SRML, that is specified by the set of *Item* element. *Item* element is used to specify the instance of the DEVS model. One *ItemClass* element can declare multi *Item* elements. The attributes of *Item* include *ItemClass*, *Priotity*, *ItemID* and so on. *ItemClass* attribute specifies the name of the model from which the instance will be instantiated. *Priority* attribute specifies the priority of the instance, the event produced by the high priority will be processed firstly. *ItemID* attribute specifies the unique identifier of the instance. *Item* element also includes zero or more sub-elements such as *Script*, *PropertyValue*, *Link, and Event* and so on. *Script* element is used to define special behavior of the instance. *PropertyValue* element is used to define the initial property value of the instance. *Link* element is used to define the coupling relationship between the instances in coupled model. The model instances with the explicit linking will interact events each other as the "link" mode. *Event* element is used to define special event of the instance.

Select: The tie-breaking rule, which is used to process simultaneous events. In SRML, simultaneous events will be processed according to their priority.

4 THE ARCHITECTURE OF EXTENDED SRML SCHEMA

Mapping DEVS onto SRML constructs the core of the SRML schema. In addition, SRML also comprises *SRML* root element, *Simulation* element, *EventClass* and *ItemPrototype* elements, which construct the whole SRML schema architecture. It is illustrated in the following Figure 2.

The extended SRML schema is composed of *SRML*, *EventClass*, *ItemClass*, *Simulation*, *Script*, *Property*, *EventDispatcher*, *EventSink*, *Item*, *ItemPrototype*, *PropertyValue*, *Link* and *Event*, 13 elements in total.

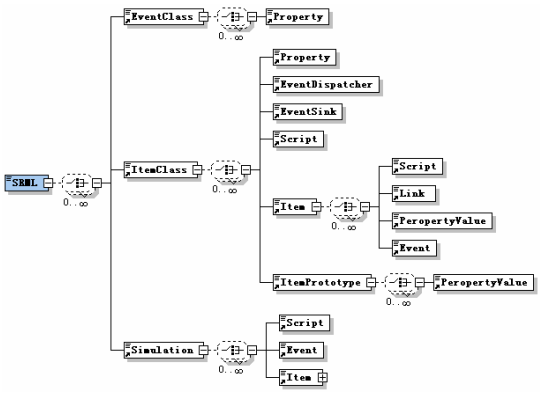

Figure 2: The architecture of extended SRML schema

SRML is the root element of SRML, which can comprise zero or more *ItemClass*, *EventClass* and *Simulation* as its sub-elements. SRML simulator can load SRML file and parse the definition of *ItemClass*, *EventClass* and *Simulation*, then generate model instances and events, drive the simulation. *EventClass* element is used to define the event class which is the data format used for interaction among model instances. All events used in the SRML must be one type of defined event class. *Script* element is used to define the behavior of the element such as *ItemClass*, Item and Simulation. *ItemClass* is the core element, which can define one type of model with the same properties, structure, input and output interfaces and behavior. *ItemClass* element is defined according to DEVS syntax and semantics, and can be used to specify atomic model or composite model. *Property* element can define the properties of *ItemClass* or *EventClass*. *EventSink* and *EventDispatcher* are used to define the input and output interface respectively. *Item* element is used to define model instance. *Item* can be the sub-element of *ItemClass* or *Simulation* to define the instance construction of coupled model or a simulation scenario. *ItemPrototype* element is used to define the prototype of one *ItemClass*. A prototype is one kind of *ItemClass* with the same property value. *Item* can also instantiated from an *ItemPrototype*. *PropertyValue* element is used to initialize the property value. *Link* element is used to define the interaction relationship between model instances. *Event* can define the

event instance in a simulation. *Simulation* element is used to define a simulation scenario which includes the information about the construction of model instances, execution batches, simulation time of begin and end conditions and so on.

5 SRML SIMULATION CASE

The extended SRML Schema is based on DEVS. It is therefore apparent that SRML can be used to build all DEVS models. To prove the completeness of the mapping from DEVS to SRML, we give a typical DEVS model demo built with SRML. Here is a very simple simulation demo to solve the differential equation $x'' - 0.8x' + x = 2$ (Fishwick 1995) expressed as DEVS (Bolduc 2002 and Kofman 2003) with SRML. To solve the equation, we should build the add model, subtract model, constant model, integrator model and chart model. We also have to define the interaction data formats and then couple these model instances to coupled model according to the equation and define the simulation scenario to begin simulation eventually. Before building SRML models, we give the DEVS specification of integrator model for convenient understanding of the mapping from DEVS to SRML.

DEVS specification of integrator atomic model is:

Inports={"dValue"}, X_{dValue}=Number

$X=\{(p,v) \mid p \in \text{Inports}, v \in X_p\}$

Outports={"CurrentValue"}, $Y_{CurrentValue}$=Number

$Y=\{(p,v) \mid p \in \text{Outports}, v \in Y_p\}$

S={"InitValue", "DeltaValue", "Step", "Result"}

δ_{ext}(phase, σ, e, (p,v))=(DeltaValue=v.Number, σ-e,0)

δ_{int}(phase, σ)=(Result = Result + DeltaValue * Step, σ=step)

λ(phase, σ)=("CurrentValue", Result)

ta(phase, σ)= σ

DEVS specification of the differential equation coupled model will be omitted. Some SRML code fragments are offered as follow.

The code for interaction data formats definition:

```
<EventClass Name = "Number">
    <Property Name = "Number" DataType = "Double"/>
</EventClass>
```

The code for integrator model definition:

```
<ItemClass Name="IntegratorBlock">
    <Property Name="InitValue" DataType ="Double"/>
    <Property Name="DeltaValue" DataType ="Double"/>
    <Property Name="Step" DataType ="Double"/>
    <Property Name="Result" DataType ="Double"/>
    <EventSink Name = "dValue" EventClass="Number" LinkFixed = "True"/>
    <EventSink Name = "Tick" EventClass="Tick" LinkFixed = "True"/>
    <EventDispatcher Name = "CurrentValue" EventClass="Number"/>
    <Script Type="text/javascript"><![CDATA[
        This.Step = 0.1;
        var event = new Event("Tick");
        event.Time = This.Step;
        Simulation.ScheduleEvent("Tick",event);
        function OndValue(e){
            This.DeltaValue = e.Number;
        }
        function OnTick(e){
            This.Result = This.Result + This.DeltaValue * This.Step;
            var event = new Event("Number");
            event.Number = This.Result;
            event.Time = Simulation.CurrentTime;
            Simulation.SendEvent("CurrentValue",event);

            event = new Event("Tick");
            event.Time = Simulation.CurrentTime + This.Step;
            Simulation.ScheduleEvent("Tick",event);
        }]]>
    </Script>
</ItemClass>
```

The code for coupled model definition of the differential equation:

```
<ItemClass Name="FishwickModel">
    <Item ItemClass="ConstBlock" ItemID="0">
        <PropertyValue Name="Value" Value="0.8"/>
    </Item>
    <Item ItemClass="ConstBlock" ItemID="1">
        <PropertyValue Name="Value" Value="2"/>
    </Item>
    <Item ItemClass="MultiplyBlock" ItemID="2">
        <Link Name = "0" Target = "0" EventSinkName = "InputNumber1" EventDispatcherName ="OutputValue"/>
        <Link Name = "6" Target = "5" EventSinkName = "InputNumber2" EventDispatcherName ="CurrentValue"/>
    </Item>
    <Item ItemClass="SubBlock" ItemID="3">
        <Link Name="1" Target = "2" EventSinkName = "InputNumber2" EventDispatcherName ="Result"/>
        <Link Name="2" Target = "1" EventSinkName = "InputNumber1" EventDispatcherName ="OutputValue"/>
    </Item>
    <Item ItemClass="SubBlock" ItemID="4">
        <Link Name="3" Target = "3" EventSinkName = "InputNumber1" EventDispatcherName ="Result"/>
        <Link Name="7" Target = "6" EventSinkName = "InputNumber2" EventDispatcherName ="CurrentValue"/>
    </Item>
    <Item ItemClass="IntegratorBlock" ItemID="5">
        <PropertyValue Name="InitValue" Value="1"/>
        <Link Name="4" Target ="4" EventSinkName = "dValue" EventDispatcherName ="Result"/>
    </Item>
    <Item ItemClass="IntegratorBlock" ItemID="6">
        <PropertyValue Name="InitValue" Value="0.46"/>
        <Link Name="5" Target = "5" EventSinkName = "dValue" EventDispatcherName ="CurrentValue"/>
    </Item>
    <Item ItemClass="DynamicChart" ItemID="7">
        <Link Name="8" Target ="6" EventSinkName = "xInput" EventDispatcherName ="CurrentValue"/>
    </Item>
</ItemClass>
```

The code for simulation scenario definition:

```
<Simulation Name = "Fishwick" StartTime = "0" EndTime = "30" nBatches = "3">
    <Item ItemClass = "FishwickModel" ItemID = "11">
    </Item>
</Simulation>
```

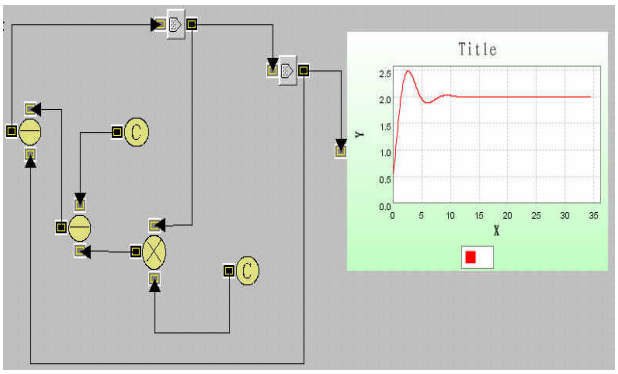

Figure 3: Solving differential equation with SRML

We have designed and implemented the SRML simulator. There are other well-known DEVS simulators such as DEVSJava at University of Arizona and DEVSim++ at KAIST. SRML simulator is identical with these simulators in simulation mechanism in that they all obey the DEVS abstract simulator algorithm. The differences are the DEVS model representation formats which can be understood by the simulators. DEVS model built by SRML is easier and more platform-independent.

The simulation process is illustrated in Figure 3.With the time advancing, the variable x converging to the result value 2. Although the case is very simple, it can demonstrate the modeling and simulation ability of SRML. It can be used to build very complex system simulation.

6 CONCLUSION

Extended SRML based on DEVS inherits all characteristics of DEVS and is able to build discrete event system model, continuous time system and mixed system model. Moreover, SRML is a platform-independent simulation language based on XML and script language, which is capable of describing system model structure and behavior. It can also be used as model representation standard and loaded by simulator to conduct simulation. SRML makes the model development simple and fast and it improves model reusability. Extended SRML, which already established relatively complete elements architecture, can build complex system models and meta-models in different domains. It could be used more widely in many fields and would accelerate the development of simulation model specification standardization.

ACKNOWLEDGMENTS

I would like to appreciate Ms Luo Ying and the WSC reviewers for reviewing the paper and giving lots of valuable advices.

REFERENCES

Bolduc, J. S., Vangheluwe, Hans. 2002. Expressing ODE Models as DEVS: Quantization Approaches. In *Proceedings of the AIS'2002 Conference* (*AI, Simulation and Planning in High Autonomy Systems*), pages 163-169.

Davis, Paul K., Robert H. Anderson. 2004. Improving the Composability of Department of Defense Models and Simulations. Santa Monica, CA: RAND. Available via <http://www.rand.org/publications/MG/MG101/> [accessed July 6, 2005].

Defense Modeling and Simulation Office(DMSO).2004. Composable Mission Spaces Environments [online]. Available via <http://www.dmso.mil/public/warfighter/cmse/> [accessed July 6, 2005].

Fishwick, P. A. 1995. Simulation Model Design and Execution, Prentice-Hall Inc.

Kofman, E. 2003. Discrete Event Based Simulation and Control of Continuous Systems. PhD thesis.

Reichenthal,Steven.W. 2003. SRML-Simulation Reference Markup Language [online]. W3C Note. Available via <http://www.w3.org/TR/SRML/> [accessed July 6, 2005].

Reichenthal, Steven. W. 2002. SRML: A Foundation for Representing BOMs and Supporting Reuse. In *Proceedings of the 2002 Fall Simulation Interoperability Workshop,* Orlando, Florida.

Tolk, Andreas. 2002. Avoiding another Green Elephant – A Proposal for the Next Generation HLA based on the Model Driven Architecture. In *Proceedings of the 2002 Fall Simulation Interoperability Workshop*, Orlando, Florida.

Tolk, Andreas. 2004. Composable Mission Spaces and M&S Repositories - Applicability of Open Standards. In *Proceedings of the 2004 Spring Simulation Interoperability Workshop*, Washington, D.C.

Zeigler, B. P. 2000. Theory of System Modeling and Simulation . New York: Academic Press.

AUTHOR BIOGRAPHIES

CHEN LIU is a PH.D student in the Institute of System Engineering, Information System and Management School of National University of Defense Technology in China. His research interests focus on simulation model specification and integration, multi-modeling, model driven simulation architecture, design of experiment and simulation evaluation. His e-mail address is <liuchenchangsha@hotmail.com>.

QUN LI is an associate professor in the Institute of System Engineering, Information System and Management School of National University of Defense Technology in China. His research interests focus on theory of modeling and

simulation, design and implementation of simulation environment, weapon system Effectiveness simulation evaluation. His e-mail address is <liqun_nudt@sina.com>.

WEIPING WANG is a professor in the Institute of System Engineering, Information System and Management School of National University of Defense Technology in China. His research interests focus on theory of modeling and simulation, weapon system simulation and SBA. His e-mail address is < wwpi@public.cs.hn.cn>.

YIFAN ZHU is a professor in the Institute of System Engineering, Information System and Management School of National University of Defense Technology in China. His research interests focus on theory of modeling and simulation, weapon system simulation and SBA. His e-mail address is <yifanzhu_nudt@sina.com>.