SiMA: a discrete event system specification-based modelling and simulation framework to support model composability

M Nedim Alpdemir

The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology 2012 9: 147
DOI: 10.1177/0037549712441742

Original article

JDMS

Journal of Defense Modeling and Simulation: Applications, Methodology, Technology
9(2) 147–160

DOI: 10.1177/0037549712441742
dms.sagepub.com
SAGE

SiMA: a discrete event system specification-based modelling and simulation framework to support model composability

M Nedim Alpdemir

Abstract

In this paper we briefly present our views on composability of models and interoperation of simulations; we introduce our simulation software framework, namely Simulation Modelling Architecture (SiMA), and we present our contributions to the underlying formal model (i.e. discrete event system specification). We also present our simulation construction environment, which supports composability of simulation models through a simulation construction tool chain. Finally, we provide a case study where a sample simulation scenario is used to demonstrate the features of our SiMA environment that facilitate composability of simulation models.

Keywords

modeling and simulation, discrete event system specification, re-usability, simulation model composition, simulation interoperability

1. Introduction

Component-based simulation frameworks have long been investigated in the simulation research community. Modelica[1] language, DSBlock[2] and Simulink[3] systems are examples of component-based simulation systems with emphasis on Continuous Simulation. The discrete event system specification (DEVS)[4] approach, COST[5] and SIMKIT[6] systems, on the other hand, are examples that are more discrete event simulation centric. A common characteristic of these systems and approaches is that they try to establish mechanisms, rules and specifications that would enable the construction of complex simulation models from smaller sub-models in a systematic way. The main motivation behind these efforts was, of course, to facilitate easier and robust development of complex simulations through the re-use of previously developed and tested reliable building blocks.

1.1 Challenges for composition and re-use of simulation components

Development of simulation components for a highly complex system within a single project and a single administrative domain (i.e. a single institution) is a challenge in itself. When the components are developed within different administrative domains and their re-use history is to span the lifetime of multiple projects, the challenge grows exponentially. The reason is obvious: as the specification and development context of the components diversifies, the likelihood of syntactic, semantic and behavioural incompatibilities grows accordingly. A considerable number of contributions have already been made to study multiple aspects of composability and interoperability in the context of simulation applications. Notably, Page and Opper[7] discuss the horizontal and vertical dimensions of composition, where the horizontal dimension signifies the interoperability (via couplings) and multiresolution (via

TÜBITAK BILGEM UEKAE - İLTAREN, Ankara, Turkey

Corresponding author:
M Nedim Alpdemir, TÜBITAK BILGEM UEKAE - İLTAREN, 06800 Ümitköy, Ankara, Turkey.
Email: nedima@iltaren.tubitak.gov.tr

composition), and the vertical dimension signifies aggregation. Kasputis and Ng[8] list six major steps to outline a methodology for developing a composable simulation. Tolk et al.[9] provide a useful categorization of levels of interoperability and devise a well-defined model for Levels of Conceptual Interoperability (i.e. LCIM). Yilmaz[10] provides an account of the importance of contextual information to support composability. Sarjoughian[11] attempts to devise a conceptual framework for model composability in terms of the composability of modelling formalisms leading to the concepts of mono, super, meta and poly model composability. Davis and Tolk[12] address multi-facets of model composability, including syntax, semantics, pragmatics, assumptions and validity. Notably, they elaborate on multiresolution modelling and draw the attention of the reader to the relevancy and impact of it on composability.

One fundamental issue seems to be to establish an overarching theoretical framework that is, on one hand, sufficiently wide and deep to cover multiple aspects of composability and interoperability requirements, and at the same time is not too restrictive to over-constraint the conceptual, behavioural and performance characteristics of individual units of composition.

Exploring the relevant literature and our own experience brings us closer to the view stated by Davis and Tolk[12] that 'pure composability' in the form of strict plug-in/plug-out is unreasonable due to substantive subtleties about the component models and the assumptions that underlie them. Another important postulation stated by many researchers, including Davis and Tolk, which we find assertive is that composability and interoperability need to be considered distinctly (but not in isolation), since the requirements implied by each of them have considerable differences. Interoperability primarily targets the orchestration of potentially heterogeneous simulation components, thus prioritizing run-time oriented aspects. Composability, on the other hand, implies aggregation of models, both along horizontal and vertical (hierarchical) dimensions, prioritizing robust and coherent coupling of components, including their conceptual alignment, thus calling for more formal and theoretical approaches. As such, we tend to abstract the interoperation of components as a 'communicative and collaborative *action*' of heterogeneous entities to present a '*collective emergent behaviour*' to the outside observers. We also want to abstract composition in terms of 'coupling of components' and 'alignment of concepts' to present an opaque, unified view. The work presented in this paper primarily focuses on the latter (i.e. composability). Our primary contribution is Simulation Modelling Architecture (SiMA)-DEVS, which defines a specialized form of DEVS formalism and a software framework, SiMA, that implements it. We also describe our simulation construction tool chain that facilitates easier plumbing of simulation components. Our motivation is to share our experiences and the results of our efforts to develop a simulation environment with inherent support for composable simulation models.

The rest of the paper is structured as follows: Section 2 provides an account of multi-aspects of composability from the particular point of view of the authors; Section 3 provides a short background on DEVS formalism and a discussion of our extensions to DEVS; Section 4 introduces SiMA software architecture; Section 5 provides a case study where a sample simulation scenario is used to demonstrate the features of our SiMA environment that facilitate composability of simulation models; and finally Section 6 provides some concluding remarks.

2. Multi-aspects of composability

The views presented in this paper build upon the relevant literature, but aim at opening another window that perhaps offers a slightly different angle. To summarize the various dimensions of the composability and interoperability from that particular perspective, we propose to consider the following aspects in a complementary manner.

2.1 The modelling formalism (the philosophy)

The modelling formalism specifies in a formal way the semantics of the modelling approach used to represent an entity or a process. In that way, the modelling formalism determines the abstraction used by the modeller and therefore may introduce certain implicit or explicit constraints on the resulting model. As such it delineates the *philosophical dimension* of the modelling enterprise. It is very important for the modelling approach to allow for multi-formalism support, since the complexity of the real world does not lend itself to using one basic abstraction at a time.

2.2 The type system and the grammar (the language)

The fundamental purpose of a type system is to prevent run-time errors in a computation.[13] The grammar, on the other hand, provides the *foundations* for specifying all syntactic structures. Since interoperability implies a meaningful emergent behaviour as a result of interactions between distinct behavioural units, a well-defined scheme both for type conversions and for creating mappings between the world views of each behavioural unit is essential. An extensible markup language (XML) schema-type system emerged as a good candidate for a domain- and language-independent-type system. As illustrated in Figure 1, the XML schema-type system can function as an inter-lingua to mediate (i.e. map and reverse-map, with the reservation

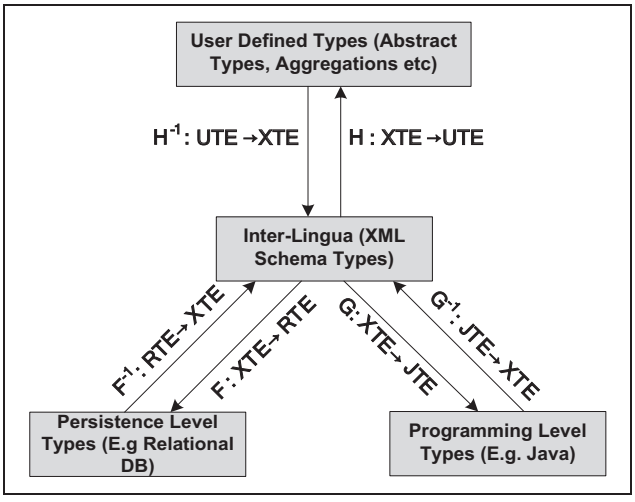

Figure 1. Mediation for syntactic mapping via an inter-lingua.

that inverse mapping may not always exist) between different bindings of essentially the same world description.

In addition, XML schema can be used to define a grammar for higher-level definitions, such as the conceptual model and semantic rules. However, for a highly effective contextual alignment of individual models, it needs to be supported with semantic ingredients such as ontologies.

2.3 The conceptual model (the vocabulary and the (grand) story)

The conceptual model is very important, as it delineates the *context* for specification, composition and interoperation of components. The notion of context is attracting more and more attention of the simulation community. In fact, the utility of contextual information in computing applications where the definition, development and execution of individual software components span multiple administrative domains has already been acknowledged in many other communities. In particular, service-oriented computing and pervasive computing communities have focused on making the contextual information a first class citizen of the overall environment. The proposal to introduce Context Oriented Programming[14] as a new programming approach primarily for pervasive computing applications is such an attempt. The need for contextual information for successful interoperation of simulation components is also well addressed in the simulation community. Yilmaz,[15] for instance, provides an in-depth discussion of the possible use of the contextual information to support a component-based simulation environment.

Defining a single conceptual model to represent a large set of simulation models, however, is practically impossible. A pragmatic solution would be to define domain-specific information models, ontologies, etc. Community-based information model development efforts must be supported. This issue is closely related to the notion of a Common Reference Model (CRM) proposed by Tolk and Diallo[16] as part of the Model Based Data Engineering (MBDE) process. The CRM becomes, in fact, the common language spoken and understood by all members of a federation. Tolk and Diallo emphasize that a reference model such as the CRM facilitates information exchange needs of participating systems, but does not impose its world view on all systems.

2.4 The coupling specification (the legal contract for union or collaboration)

Coupling is a concept sometimes somewhat teeming with negative connotations (consider, for instance, its meaning in the Object Oriented theory where its scarcity is a good thing together with high cohesion). We argue that the approach used in formal abstraction of the coupling relationships between models is an essential ingredient of the overall formalization of model composition in the modelling and simulation domain. Further, we observe that the notion of 'input and output ports' provides a convenient abstraction for *model composition*, serving as the focal point of aggregation. We find the DEVS approach very convenient for an abstraction of model composition in terms of port couplings. As will be introduced in more detail in Section 3, we specialize parallel DEVS formal definition by developing a more strictly qualified notion of input and output ports. As such, we effectively emphasize couplings between model ports to imply 'data coupling' (as opposed to control coupling, content coupling, external coupling, etc.), with a firm reference to type-system compliance.

2.5 Run-time interoperation application programming interfaces (the protocol for communication, collaboration and co-existence)

As opposed to static model coupling, the run-time coupling of components has to be driven via an interaction protocol. The interaction protocol could simply dictate compatibility at the syntactic level or it could involve some semantic elements, such as compliance to a behavioural abstraction (such as that of DEVS model execution protocol). It is self-evident that successful interoperation is likely to become a reality only through wide adoption of international standards. High-level architecture (HLA) is a good example of such an interoperation standard. Having said that, interoperation is not a straightforward concept. As Tolk et al.[9] suggest, various different forms of interoperability are conceivable, each addressing a distinct set of requirements, with a growing degree of complexity.

With these considerations in mind, research at our research institute (the National Research Institute of Electronics and Cryptology (UEKAE) under the Turkish Scientific and Technological Research Council (TUBITAK)) has focused on composability, re-usability and interoperability, with a relatively higher priority on composability for supporting the construction of complex simulations primarily through the re-use of in-house models. The DEVS approach was of particular interest for us for a number of reasons: (1) it provides a set theoretic formal basis for both structural and behavioural specification and hierarchical composition of simulation models with arbitrary complexity levels; (2) it specifies a protocol for model execution; (3) it promises a unifying theoretical frame for hybrid systems[17] with both continuous and discrete event properties (see Kofman et al.[18] for an in-depth discussion on representing continuous systems with DEVS via quantization of the state base). These properties have contributed to our decision to use DEVS as the formal basis to implement our simulation framework (i.e. SiMA). To elaborate more on this, SiMA was developed to meet requirements for model composability and re-usability, high performance and robustness, and direct support for industrial quality simulation application development. It is a known fact that architectural flexibility and run-time efficiency are often conflicting goals for software frameworks. To overcome the challenge, we took a systematic approach to reconcile the tension between these conflicting goals:

- we benefited from the sound formal specification offered by DEVS with some constraints to establish strict semantics for model definition, composition and execution;
- we designed our framework as a layered architecture where central processing unit (CPU)-demanding simulators can be written in an efficient programming language (i.e. C++), whereas the framework layer is written in .NET platform to support easier component composition, distribution and management.

As a result, SiMA comes to possess the following distinct characteristics:

- it is based on the DEVS approach, which provides a solid formal basis for hierarchical model composition;
- its Simulation Execution Engine implements the parallel DEVS protocol, which provides a well-defined and robust mechanism for model execution;
- it extends the DEVS formalism with a specific port type and transition function;
- it comes with a variety of tools, such as a Graphical Model Editor, an automated model construction tool chain that aims at providing support for Model Driven Engineering, an efficient and configurable trace generation framework, and a Code Generation Tool (KODO) that supports the message-processing infrastructure through automated marshalling/unmarshalling across the simulation layers.

3. SiMA-DEVS: our approach

Our modelling and simulation framework SiMA is based on the DEVS approach as a solid formal basis for complex model construction. DEVS is a formalism introduced by Zeigler[19] to describe discrete event systems. In DEVS formalism, a basic model (atomic model), is defined formally as[4] $M = < X, S, Y, \delta_{\text{int}}, \delta_{ext}, \lambda, ta >$, where X is the set of input events; S is the set of states; Y is the set of output events; $\delta_{\text{int}} : S \rightarrow S$ is the internal transition function; $\delta_{\text{ext}} : Qx \times \rightarrow S$ is the external transition function, where $Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$ is the total state set and e is elapsed time since last transition; $\lambda : S \rightarrow Y$ is the output function; $ta : S \rightarrow R^{+}_{0, \infty}$ is the time advance function.

A DEVS-coupled model is defined formally as $N = < X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,j}\}, SELECT >$, where X is the set of inputs; Y is the set of outputs; D is the set of component names; M_d is the model of component d, $\forall d \in D$; I_d is the set of component influencers of d, $\forall d \in D \cup \{N\}$; $Z_{i,j}$ is the i-to-j output-to-input function, $\forall j \in I_i$; $SELECT$ is the tie-breaking function to arbitrate the occurrence of simultaneous events. A complete description of DEVS semantics can be found in Zeigler et al.[4]

SiMA[20] Simulation Execution Engine implements the parallel DEVS protocol, which provides a well-defined and robust mechanism for model execution. SiMA builds upon a specialized form of DEVS formalism that achieves the following.

1. Formalizes the notion of 'port types', leading to a strongly typed (and therefore type-safe) model composition environment. In this respect we specialize the basic DEVS formalism by introducing more strict syntactic constraints on the port definitions.
2. Introduces a new transition function to account for model interactions involving state inquiries with possible algebraic transformations (but no state change), without simulation time advance. In this respect we extend the basic DEVS formalism. This is similar to the notion of zero-lookahead in HLA[21] from a time-management point of view.

Our SiMA-DEVS formalism is given below:

$SiMA-DEVS = < X, S, Y, \delta_{\text{int}}, \delta_{ext}, \delta_{con}, \delta_{df}, \lambda, ta >$ where,
$S, \delta_{\text{int}}, \delta_{ext}, \delta_{con}, \lambda, ta$ are as defined in the parallel DEVS formalism,
$X = \{(p, v) | p \in InPorts, v \in X\}$
is the set of input ports and values,
$Y = \{(p, v) | p \in OutPorts, v \in Y\}$
is the set of output ports and values,
$InPorts, OutPorts$: sets of input and output ports such that :
$InPorts = \{(\Gamma, \tau) | \Gamma \cdot v : \tau, v \in X_p\}$,
$OutPorts = \{(\Gamma, \rho) | \Gamma \cdot v : \rho, v \in Y_p\}$,
Γ : XMLSchema type system,
τ, ρ : data types valid w.r.t. XMLSchema type system,
$\delta_{df} : PDFT_{in} \times S \rightarrow OutPorts'$ is the time invariant
direct feed through function where:
$PDFT_{in} \subseteq InPorts$ and $OutPorts' \subseteq OutPorts$.

Note that the set of input ports, *InPorts*, is formally defined as a set of pairs where each pair defines one input port of a model uniquely. The first element of each pair, Γ, is a typing environment (in our case the XML schema-type system) and the second element of the pair (τ) is a data type that is valid in Γ, where each input value v conforms to data type τ. This is formally denoted by the typing judgment $\Gamma \cdot v : \tau$, which asserts that a term v has a type τ with respect to a static typing environment Γ for the free variables of v (or, in short, 'v has type τ in Γ').[13] Similar semantics apply to output ports, too. Thus, we make strong typing and type-system dependency of the ports explicit in the formal model.

It may be argued that strong typing and type-system dependency are essentially run-time properties of the execution environment, and that making those explicit in the formal model may ambiguate the abstraction level of the formal specification. Although this argument is generally valid, we counter-argue that there are a number of merits in following our proposed route, in particular the following.

1. We introduce a type discipline to the definition of the externally visible model interfaces (i.e. ports), leading to a semi-formal basis for the specification of the overall information model of the system being modelled. Ensuring that the overall information model is specified using a well-defined type system, we support syntactic compatibility at the modelling level.
2. We facilitate Model-Driven Engineering through well-typed and type-system dependent external plugs to enable automated port matching and model composition. In fact, the Model-Driven simulation construction tool chain for SiMA is successfully implemented, via a number of tools such as a code generator, a model builder and a model linker. This simulation construction tool chain is discussed in detail in Section 4.1.
3. By making strong typing and type-system dependency explicitly visible in the formalism, we reduce the gap between modelling-level logical composability constraints and run-time-level pluggability constraints.

Note also that, in addition, we introduce a new transition function, δ_{df}, that enables models to access the state of other models through a specific *type* of port, without advancing the simulation time. As such, it is possible to establish a path of connected models along which models can share parts of their state, using state variables to compute derived values instantly within the same simulation time step. As stated earlier, this is similar to the notion of zero-lookahead found in HLA.[21] One may argue that the zero-lookahead behaviour could be modelled by adjusting the time advance function of an atomic model such that the model causes the simulation to stop for a while, do any state inquiry via existing couplings, then re-adjusting the time advance to go back to the normal simulation cycle. Although this is possible, we argue that by introducing a transition function and a specific port type that is tied (through run-time constraints imposed by the framework) to that particular transition function we gain several advantages:

1. the models can communicate and share states with each other without the intervention of the simulation engine, thus providing a very efficient run-time infrastructure;
2. allowing such communications only to occur through a specific port type (compile-time and run-time checks are carried out), the framework is able to apply application-independent loop-breaking logic at the ports to prevent algebraic loops, thereby ensuring model legitimacy.

Intuitively, we can state that the behaviour of SiMA-DEVS is equivalent to that of parallel DEVS from a Language Equivalence point of view.[22] Hwang[22] shows that two atomic DEVS models can be equivalent if the languages they generate are the same. The total language a DEVS atomic model generates is simply defined in terms of transition trajectory, output trajectory and the total trajectory generated by that DEVS model. Since the new function we introduce does not change the model's state, the state trajectory would be the same as classical DEVS. Although the output trajectory of a SiMA-DEVS model is directly affected by δ_{df}, the same output behaviour can be generated by a standard DEVS model via the *next time* calculation logic. In other words, the DEVS models can be forced to stop advancing at a specific point in time to send data to other models. As such, equivalence is also true

from a timed language[23] point of view. So, the benefit SiMA-DEVS offers in this respect is that the ability to model zero-lookahead behaviour is encapsulated into a well-defined mechanism and this mechanism is introduced into the model definition as a first class entity, rather than relying on custom implementations. We leave the formal discussion of the equivalence out of the scope of this paper and refer the reader to Hwang,[22,23] and Hwang and Cho.[24]

3.1 Support for dynamic changes in model composition relationships

A relatively more demanding requirement in the context of model composition is the ability to change the coupling relationships between the simulation models (or simulators representing the models) at run time, while the simulation execution is under progress. This requirement mostly stems from the fact that analysing the behaviour of complex and adaptive systems through simulation often requires the underlying modelling and simulation approach to support structural and behavioural changes. This implies that models may switch between different behavioural specifications (e.g. fidelity levels), as well as various coupling configurations dynamically at run time, depending on various triggering events. To cope with such structural and behavioural change requirements, simulation environments must incorporate d*ynamism* into their fabric.

We observe that several approaches to dynamism in simulation environments are already proposed in the relevant literature.[25–28] We note that three distinct categories of change are discussed in those existing approaches: (1) a change in the overall compositional state of models (i.e. some models may cease to exist to be part of the composition, or new models can join the simulation); (2) a change in the connectivity relationships (coupling) among the existing models; (3) a change in internal functional behaviour of one or more of the existing models. We find two of the formal approaches to the variable structure models in the DEVS environment particularly relevant to our work. The first one is DSDEVS (Dynamic Structure DEVS), introduced by Barros.[27] The second one is Dynamic DEVS (dynDEVS), introduced by Uhrmacher.[25] Our approach to add dynamism to our basic SiMA-DEVS model is similar to that of dynDEVS and aims at supporting all three categories of change mentioned above. To be more precise, we conform to both dynDEVS (for atomic models) and dynNDEVS (for coupled models defined as composition of atomic models), as the underlying formal specifications, with some trivial extensions. In particular, we introduce a state synchronization mechanism between networks of connected models, to be performed at the end of a structural change phase, in case a model wants to update the values of such state variables that are within the common set of pre-and post-change models (i.e. they are not introduced newly after the model's structural transition), but have values that stayed unchanged during the pre-change simulation period. This mechanism is instrumental in cases where a model A initializes some of its state variables at the beginning of the simulation but does not receive updates for those variables until some influencer model B goes through a structural change that causes those variables to be updated, or in cases where a new model B is added that introduces a new coupling influencing one of the input ports of model A. Further information on our dynamism extensions to SiMA-DEVS can be found in Deniz et al.[29] This paper does not delve into the details of those extensions. Rather, the focus of this paper is to illustrate how SiMA supports static composability.

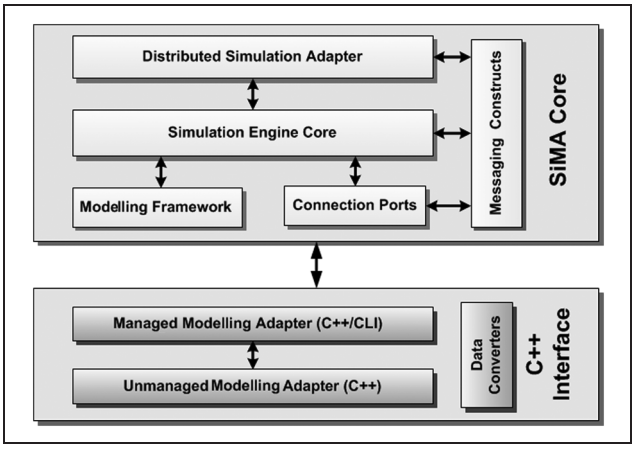

Figure 2. SiMA software architecture.

4. SiMA software architecture

SiMA is a software framework for developing simulation models and executing simulations (Figure 2). It has two main layers: *SiMA Core* and *C++ Interface*.

SiMA Core consists of five sub-components that are used for modelling and simulation. *Modelling Framework* is a set of classes and data types to use in model development. Atomic models and all their subclasses are defined in this framework. *Connection Ports* is the transportation component that contains classes for defining ports and their connection limits. *Simulation Engine Core* uses these two components, *Modelling Framework* and *Connection Ports*, to execute a simulation. In other words, it implements the DEVS simulation protocol and in addition exposes administrative interfaces to manage and track the simulations at run time. The *Distributed Simulation Adapter* is the interface for connecting SiMA simulations with external distributed simulation infrastructures. Currently, a HLA adapter is implemented for a specific run-time interface (RTI). However, we believe details of

this issue are complicated enough to be considered in the context of another paper. *Messaging Constructs* is the component that presents all base data-type classes and rules for the inter-communication of atomic models and *SiMA Core* components.

The *C++ Interface* layer is implemented to allow C++ to be used as a model implementation language for SiMA atomic models. All core SiMA components are developed in .NET, but we support models implemented in both .NET and C++ to co-exist in the same run-time environment during a simulation. However, since there is a strict boundary between their coding environments, various adapters and components that manage the interoperability between .NET and C++ atomic models and SiMA components are implemented in the *C++ Interface* layer.

The *C++ Interface* layer consists of three sub-components. *Unmanaged Modelling Adapter* has the same interface and class hierarchy as the .NET *Modelling Framework*, except it is developed in pure C++ language. *Managed Modelling Adapter* is developed in C++/CLI, which is a special edition of C++ language in .NET, that allows access to both C++ and .NET methods and data types. *Managed Modelling Adapter* handles the interoperability management and delegates all simulation commands to C++ models, and provides all information required by them from the simulation environment. *Data Converters* are special adapters that perform marshalling of all values between .NET and C++ data types in both ways. A model developer can use the KODO tool (described in Section 4.1) to auto-generate these data converters for his/her data types.

4.1 Simulation construction tool chain for SiMA

SiMA is supported with a number of tools that collectively allow automatic construction of the run-time simulation environment using previously defined SiMA models. The simulation construction tool chain consists of four main steps and involves the use of five tools (Figure 3).

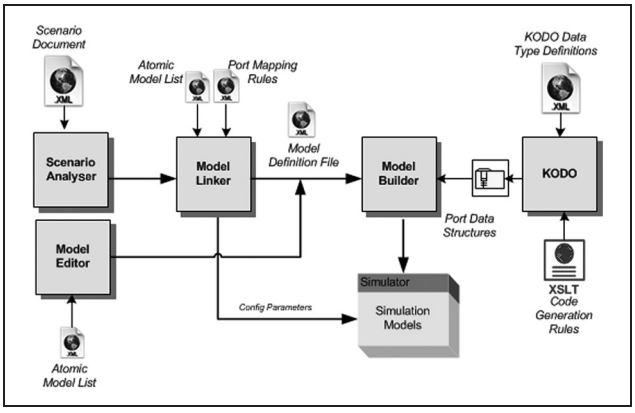

Figure 3. Simulation construction tool chain

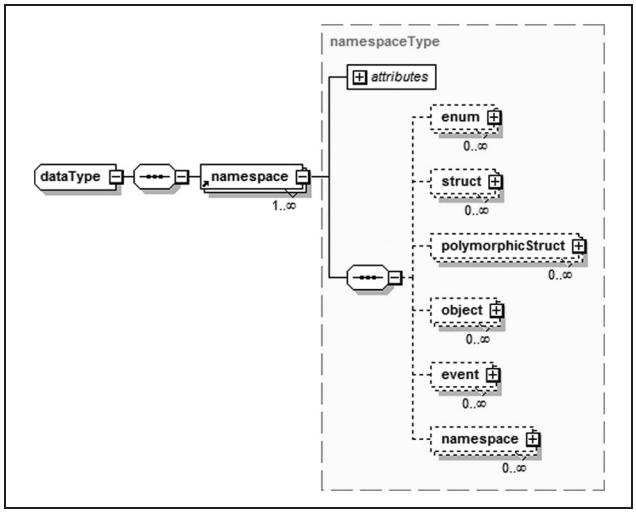

Figure 4. XML schema representation for KODO type definitions

KODO: a model developer using SiMA needs to define data types to be used for model initialization and inter-port communication. Due to .NET/C++ interoperability requirements, SiMA needs to apply special handling on port data types for serialization/deserialization of data values flowing between atomic models. Similarly, the model initialization phase requires access to input configuration data types. KODO aids the developer by auto-generating not only data beans but also additional processing logic for marshalling and type conversion, for access to model input configuration files for state initialization, and for trace generation. KODO allows definition of data types with simple XML definitions and generates all classes and methods required for the model developer. KODO type definitions have to conform to the KODO XML schema that defines the grammar for the type definitions. Figure 4 illustrates a graphical representation of top-level constructs of this schema. Note that events and objects are defined as distinct types so as to distinguish between uniquely identifiable variables (objects) and transient port data (events). Objects and events consist of data 'members', each with either a primitive type (e.g. int, double, etc.) or referring to a complex 'struct' definition (or a polymorphic struct with support for inheritance).

Scenario Analyser: there are two starting points (or entry points) in our simulation construction tool chain. One of those has a more constrained initial specification, where a simulation study is defined by a *scenario* (the term 'scenario' is not used to connote a domain-specific experimentation procedure (e.g. those defined by Military Scenario Definition Language (MSDL; see

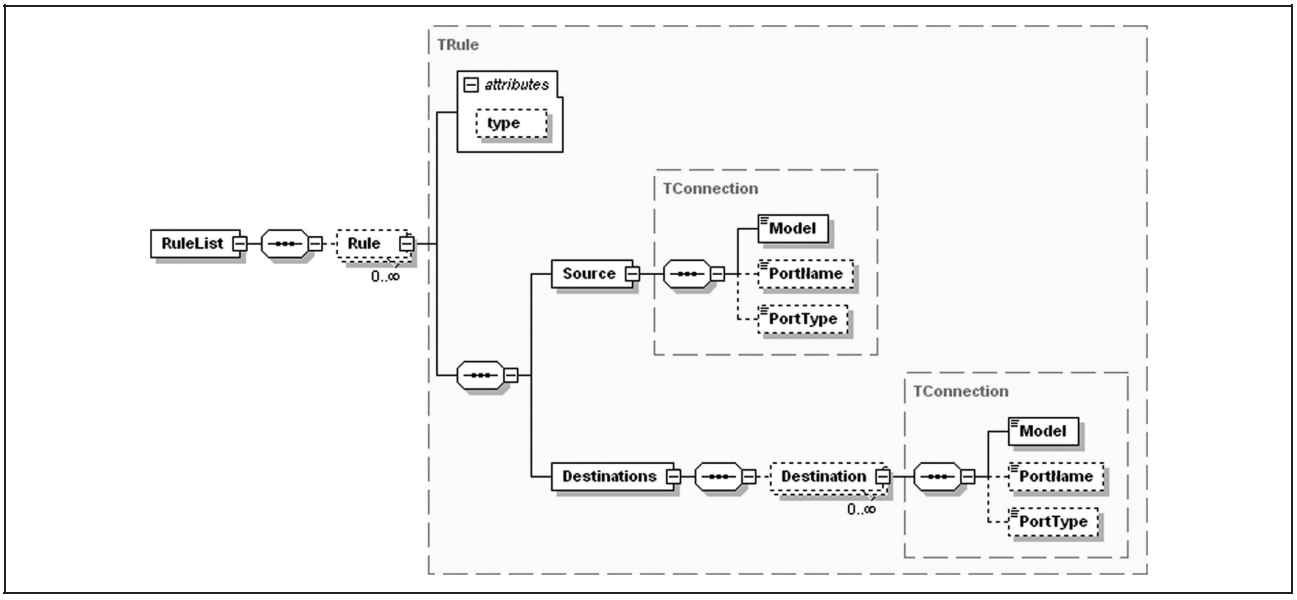

Figure 5. XML schema representation for port mapping rules.

http://sstc-online.org/2009/pdfs/RLW2375.pdf) – it only indicates the bag of models included in the simulation where model hierarchy information is also provided) that describes all required models, model hierarchy information and model initialization data. In this case our Scenario Analyser interprets a scenario definition and identifies all atomic models (from the component library) and defines their composition hierarchy in an intermediate form to be used by the Model Linker. It is generally an application-dependent analyser that converts the application-dependent scenario file to be used by application-independent Model Linker.

Model Editor: the second route to the pipeline is the SiMA Model Editor. The Editor provides a context-independent model composition environment where both atomic and coupled models can be created and organized as libraries, thereby facilitating reusability of simulation models. SiMA allows defining atomic models either in C++ or in C# programming languages. Once these models are compiled into binary components (i.e. dynamic link libraries (DLLs)), they can then be graphically coupled using SiMA Model Editor in hierarchical block diagrams to construct more complex composite models.

Model Linker: Model Linker reads its special scenario definition file and the port mapping rules. Port mapping rules define additional user-defined constraints where data-type driven port mapping based on exact matching of port types is not sufficient to define the complete coupling relationships in a non-ambiguous and valid way. In a way, these rules constitute a semantic layer on what is essentially a syntactic plumbing scheme. The grammar of the rules is defined by yet another XML schema, which is given in Figure 5. The Linker produces two output files: the first file, the *model definition file*, includes all the port connections, names and locations of atomic models and their hierarchy. A top-level graphical view of the XML schema representing the structure of the model definition file is given in Figure 6. The second output file, *configuration parameters*, includes all initialization data for each atomic model in the scenario.

Model Builder: Model Builder reads the model definition output of the Model Linker, then creates and combines all atomic model object instances in the memory of the host computer. As a result, a single coupled model object instance is returned to the caller, which can be used by the SiMA simulation engine to run a simulation.

5. Case study

In this section, we present a sample case study to illustrate how SiMA and our simulation construction environment are used to support the simulation application developer.

Our case study involves a typical intrusion detection scenario where an intruder is detected and tracked while it navigates through a field deployed with a number of tiny wireless sensors with arbitrary layout. The simulation models required to implement this study are as follows.

1. **Sensors**: sensors sense the movement activities in the environment and can communicate with other sensors within their range. Each sensor consists of four sub-components:
   a. antenna;
   b. sensing unit;
   c. processor;
   d. battery.

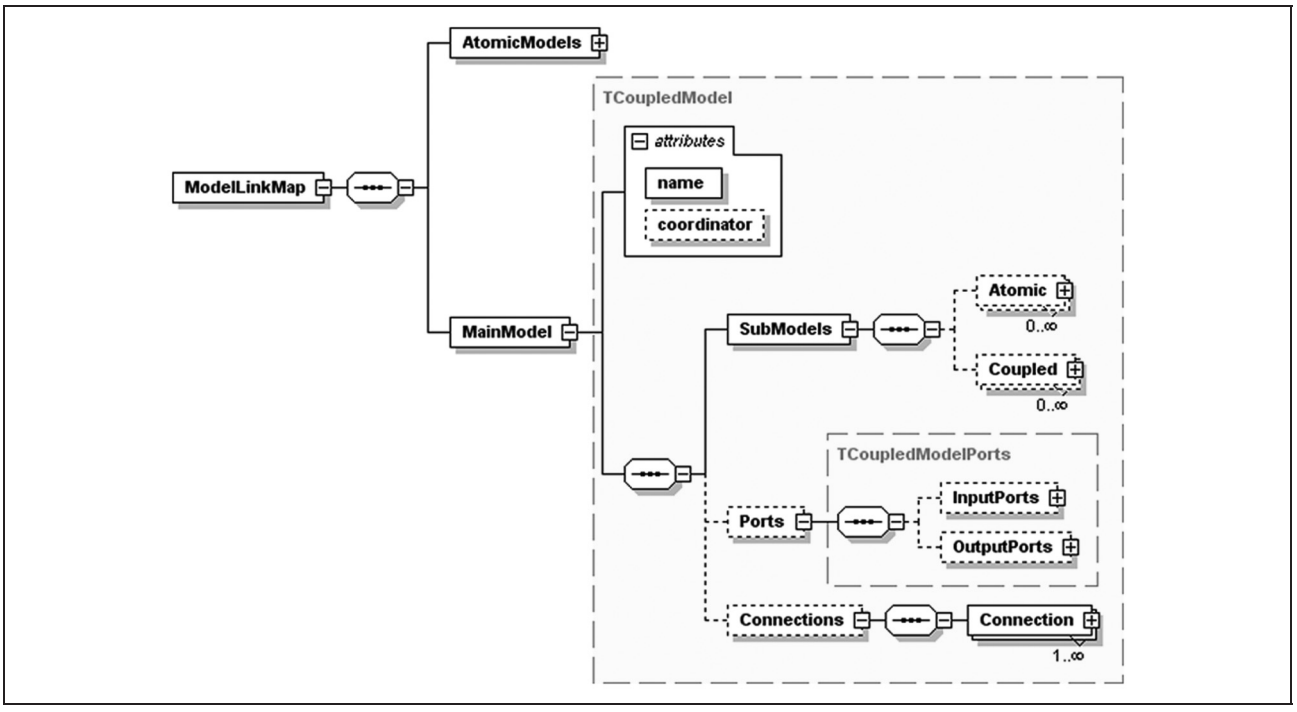

Figure 6. XML schema representation for Model Linker output.

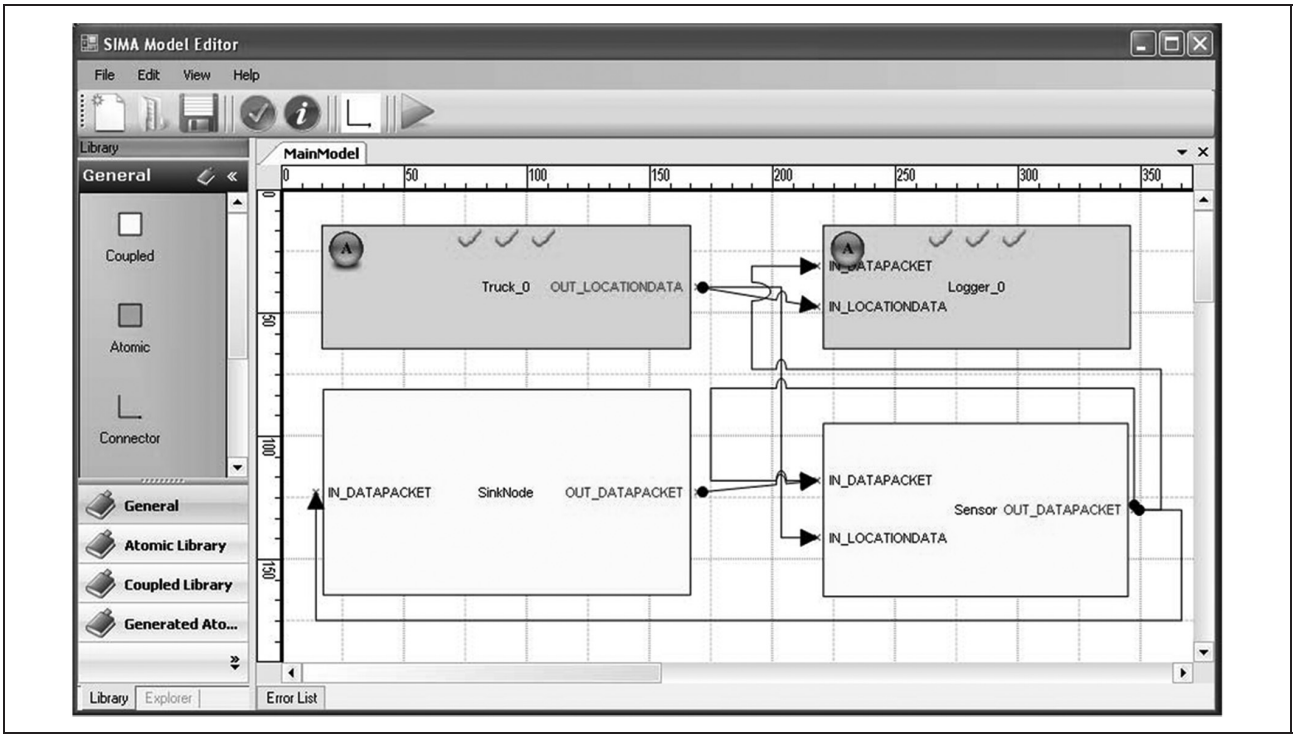

Figure 7. Top-level model structure.

Each sensor is represented by a coupled model and its sub-components are atomic models.

2. **Main sensor (or sink unit)**: the main sensor can communicate with the sensors within its range. It sends queries to the sensors and waits for the response messages. The main sensor contains two sub-components: the antenna and processor. Unlike other sensors, the main sensor does not sense the environment and it does not contain a battery.
3. **Truck**: the truck represents the intruder. It has a predefined circular path and it follows this path during the simulation. Sensors that get close enough to the truck can understand the location of the truck and prepare a message to be delivered to the main sensor.
4. **Logger**: the logger saves all the location and data packages, created by sensors and the truck, in a structured way.

A top-level visual representation of SiMA-DEVS models defined for this simulation is given in Figure 7, where model structure and couplings can be seen. Dark rectangular shapes with an 'A' sign on the top-left corners indicate atomic models and lighter rectangular shapes with no signs on the top-left corner indicate coupled models. Couplings are shown with directed arrows between model ports. The direction of the arrow connotes the direction of the data flow. Port names are denoted as labels beneath the ports. A second-level view of the model layout where models and port connections inside the sensor-coupled model is given in Figure 8. When the modeller saves the model layout, the Model Editor produces the model definition file, an intermediate form to be used by the Model Linker.

5.1 Elements of the conceptual model: KODO data types

We now introduce some of the entity definitions that constitute the elements of the conceptual model pertaining to the overall simulation exercise. As stated earlier, KODO type definitions specify the structure and relationships of entities that constitute the conceptual model. There are four main types of messages circulating in the intrusion detection simulation. We provide KODO data-type definitions for these messages below to illustrate the usage of our framework.

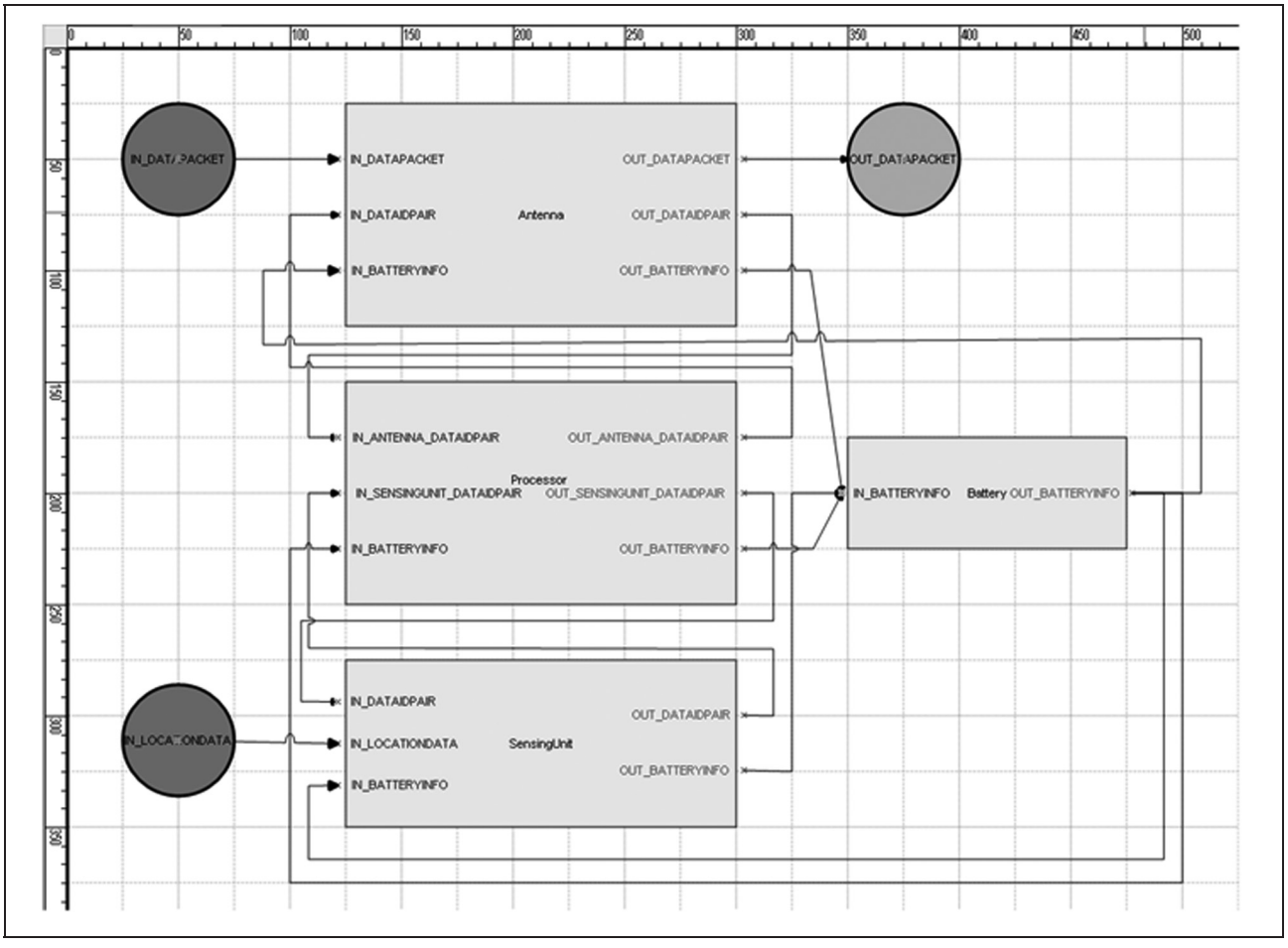

Figure 8. Models and port connections inside the sensor-coupled model.

DataPacket: sensors use this message type to communicate with each other. Each data packet contains information about the data gathered by sensing the movement activities in the environment.

```
<<event name="DataPacket">
<member typeName="SensingData" name="sensingData"
    type="struct" />
</event>
```

LocationData: this message type contains a vector. The truck model periodically updates its location and sends a LocationData message to the sensor models.

```
<<event name="LocationData">
<member typeName="double" name="X" type="primitive" />
<member typeName="double" name="Y" type="primitive" />
<member typeName="double" name="Z" type="primitive" />
 </event>
```

DataIDPair: this message is used between sensors inner models, which are the antenna, processor and sensing unit. Each message contains sensing information and an ID section. Since each sensor model can simulate more than one physical sensor, the ID field is used for identifying the sensor that actually sent the message.

```
<<event name="DataIDPair">
<member typeName="SensingData" name=" sensingData"
     type="struct" />
<member typeName="int" name="ID" type="primitive" />
 </event>
```

BatteryInfo: this message is used by the battery model to manage the life time of the battery.

```
<<event name="BatteryInfo">
<member typeName="int" name="amount" type="primitive" />
<member typeName="bool" name="isActive" type="primitive" />
<member typeName="long" name="ID" type="primitive" />
  </event>
```

SensingData: this complex type is used in both DataPacket and DataIDPair complex types. This type consists of several fields, including where the message is created, what is its destination, what type of sensing data it contains, etc.

```
<struct name="SensingData">
<member typeName="int" name="ID" type="primitive" />
<member typeName="long"name="originID" type="primitive" />
<member typeName="long"name="targetID"type="primitive" />
<member typeName="int" name="type" type="primitive" />
<member typeName="int"name="hopCount" type="primitive" />
<member typeName="int"name="temperatur"type="primitive"/>
<member typeName="int" name="light" type="primitive" />
<member typeName="int" name="sound" type="primitive" />
<member typeName="int" name="tempVar" type="list" />
</struct>
```

5.2 Model composition

As stated earlier and illustrated in Figure 4, model composition specification can be formed either through the plumbing performed via the Model Editor environment, or via a higher-level scenario definition normally created by a higher-level application layer. Figures 7 and 8 illustrate the first case. Assuming we start with a scenario definition, then the Linker automatically constructs the couplings using its type-matching algorithms (for syntactic plumbing) and, in addition, a user-defined rule set. A sample rule set used for the case study is given in Figure 9. Each rule in the figure has a type that is either 'allow' or 'deny'. If a rule is of type 'allow', then the specified connections between a given source and its destination(s) are allowed, but not other syntactically valid connections. On the other hand, if a rule type is 'deny', the specified connections between given source and destinations are marked illegal for the plugging algorithm (even if the connection is syntactically valid). Each rule contains only one source, but has multiple destinations. Each source and destination is identified by a model field and optionally by a port field. If the port is not defined, all the ports of the specified model are taken into consideration.

The sample rules given above state that no connection between sensing unit and antenna is allowed. Therefore, even if their port types match, the linker will not add any coupling between the sensing unit and the antenna.

5.3 Discussion

To relate what we have presented in Section 4 and in this section to the discussion on multi-aspects of composability given in Section 2, we provide a summary of important properties, constructs and mechanisms of our approach in relation to those multi-aspects.

1. *The modelling formalism*: SiMA-DEVS ensures that an established modelling formalism is followed as a basis for model composition. Thus, the notion of a 'model' is defined formally in a non-ambiguous way, including its structural and behavioural aspects.
2. *The type system and the grammar:* through enhancements to port definitions of DEVS, SiMA-DEVS supports strongly typed port definitions to aid automated port mapping. As illustrated via KODO types, all model ports have a well-defined type in conformance to an XML schema-type system as the basis. Various XML schema definitions further specify the grammar for expressing the conceptual model and also the model-coupling relationships.

```
<RuleList xsi:noNamespaceSchemaLocation="LinkerRules.xsd">
<Rule type="Deny">
        <Source><Model>Antenna</Model></Source>
        <Destinations>
            <Destination><Model>SensingUnit</Model></Destination>
            <Destination><Model>Processor</Model></Destination>
         </Destinations>
</Rule>
<Rule type="Allow">
         <Source><Model>Antenna</Model><PortType>PacketID</Port></Source>
         <Destinations>
              <Destination>
                  <Model>Processor</Model>
                  <PortName>In_Antenna_DataIDPair</PortName>
               </Destination>
          </Destinations>
</Rule>
<Rule type="Allow">
          <Source><Model>Processor</Model>
          <Port>Out_Antenna_DataIDPair</Port></Source>
          <Destinations>
              <Destination><Model>Antenna</Model>
                    <PortType>PacketID</PortType>
                </Destination>
           </Destinations>
</Rule>
</RuleList>
```

Figure 9. Sample rules for model connections.

3. *The conceptual model:* KODO data-type definitions provide elements of the overall conceptual model. The structure of the messages flowing through the port connections, as well as some state variables, can be specified using KODO definitions. This provides a mechanism similar to that provided by the Federation Object Model (FOM) found in HLA.
4. *The coupling specification:* the Model Linker generates the coupling specification based on syntactic port-type mappings and also adhering to the additional mapping rules. Thus the Linker makes extensive use of the strong typing introduced by SiMA-DEVS.
5. *The run-time interoperation application programming interfaces (APIs):* the Model Builder cerates the top-level coupled model, which implements an interface (called *IDEVSModel*) compliant to the DEVS parallel simulation protocol. Thus SiMA simulation engine utilizes that interface as a run-time API to advance the simulation models along their trajectories. SiMA also provides a HLA adapter for interoperation among other distributed simulators, but the details of this adapter are left out of the scope of this paper.

Presently, we do not address various other relevant aspects, such as validation of the composition (e.g. see the work done by Szabo and Teo[30]) or selection of individual models for composition from a repository based on some semantically driven methodology (e.g. see the work done by Bartholet et al.[31]). We assume that model development and composition environment is relatively well-controlled, where the model repository is populated with verified models and maintained within the same administrative domain.

Ahn et al.[32] propose a composition algorithm for HLA compatible distributed simulation environments that aims at finding an optimal set of Composite Federations (CFs) that provides a balanced level between access control (information hiding) and reduced inter-communication overhead. Their methodology is somewhat orthogonal and could be complementary to our approach, since our current work is more relevant to intra-federate model composability. The work presented by Zoubi and Wainer[33] is particularly interesting for us, since the RESTful Interoperability Simulation Environment (RISE) they propose can be considered as a higher-level interoperability middleware that would enable SiMA models and simulators to interoperate with other DEVS-based or non-DEVS simulators in a distributed execution environment. The idea of specifying inter-model connection configurations via port couplings is quite similar to ours, except that in our case port mappings are done on the basis of syntactic type matching automatically (subject to some exception rules). Due to their focus on distributed simulation they also address time synchronization issues (both conservative and optimistic),

whereas in our case we assume conservative time management in a less liberal composition environment.

6. Conclusions and further work

In this paper we have discussed issues regarding model composability and simulation interoperability, introduced our approach to dealing with some of the challenges that are particularly related to composability and presented our work on developing a DEVS-based Modelling and Simulation Framework that addresses those challenges. We have elaborated as follows.

- Firstly on our extensions on formal semantics of DEVS. In this respect we make *strong typing* and *type-system dependency* of the ports explicit in the formal model, thereby introducing a type discipline to the definition of the externally visible model ports. As such we relate to the requirements mentioned earlier in our preliminary discussion for:
  - a sound formal model;
  - type-system compliance;
  - a port-centric, consistent coupling specification.
- Secondly, on the architectural properties that makes SiMA a flexible software framework.
- Thirdly, on SiMA tool support to facilitate relatively low-cost (in terms of development effort) simulation construction for developers.

We have also provided a case study where we exemplified our claims. As stated in Section 5.3, type-qualified model port definitions introduced by SiMA-DEVS, together with further mapping constraints defined via rules, are used by our Model Linker to generate the coupling specification for a model composition in an automated way.

SiMA has been used in a number of simulation applications successfully, with strict requirements targeting high performance and ease of model development for the simulation of complex systems. We plan to improve our simulation construction tool chain, in particular by focusing on the SiMA Model Editor to make it a fully fledged layout editing and code generation environment, as a plug-in to an Integrated Development Environment (IDE) (e.g. Microsoft Visual Studio). There is also ongoing work within our team to improve the dynamism support to SiMA. Our objective for dynamism support is to support real world applications in the near future, anticipating that there will be room for improvement to the mechanisms we devised. In particular, scenarios where supporting dynamic fidelity-level adjustments of multiple models in a coordinated way is a requirement are potential use cases where we hope dynamic SiMA would provide a viable solution for application developers.

Funding

This work was partially supported by the R&D office of the Turkish Ministry of Defense.

References

1. Modelica Association, http://www.modelica.org. [Accessed, Novemer 2011].
2. Otter M and Elmqvist H. The Dsblock model interface for exchanging model components. In: *proceedings of EUROSIM'95*, 1995, pp.505–510.
3. The Mathworks Inc. *Using Simulink, version 5.0.2*. Natick, MA: The Mathworks Inc., April 2003.
4. Zeigler BP, Praehofer H and Kim TG. *Theory of modeling and simulation*. 2nd ed. Academic Press, 2000. San Diego, USA.
5. Chen G and Szymanski BK. COST: a component-oriented discrete event simulator. In: *proceedings of the 2002 winter simulation conference*, 2002, pp.776–782.
6. Buss A. component-based simulation modelling with SIMKIT. In: *proceedings of the 2002 winter simulation conference*, 2002, pp.243–249.
7. Page EH and Opper JM. Observations on the complexity of composable simulations. In: *proceedings of the 1999 winter simulation conference*, 1999, pp.553–560.
8. Kasputis S and Ng HC. Composable simulations. In: Joines JA, Barton RR, Kang K, et al. (eds) *proceedings of the 2000 winter simulation conference*, 2000, pp.1577–1584.
9. Tolk A, Turnitsa CD and Dallo SY. Composable M&S web services for net-centric applications. *J Defense Model Simulat* 2006; 3: 27–44.
10. Yilmaz L. Models to improve reuse and composability of defense simulations. *J Defense Model Simulat* 2004; 1: 141–151.
11. Sarjoughian HS. Model composability. In: Felipe Perrone L, Lawson BG, Liu J, et al. (eds) *proceedings of the 2006 winter simulation conference (WSC '06)*, 2006, pp.149–158.
12. Davis PK and Tolk A. Observations on new developments in composability and multiresolution modeling. In: *proceedings of the 2007 winter simulation conference*, IEEE Press, 2007, pp.859–870.
13. Cardelli L. Type systems. In: *CRC handbook of science and engineering*. 2nd ed. 2004, Ch. 97. Chapman and Hall/CRC, Bowdoin College, Brunswick, Maine, USA
14. Hirschfeld R, Costanza P and Nierstrasz O. Context-oriented programming. *J Object Technol* 2008; 7: 125–151.
15. Yilmaz L. On the need for contextualized introspective simulation models to improve reuse and composability of defense simulations. *J Defense Model Simulat* 2004; 1: 135–145.
16. Tolk A and Diallo S. Model-based data engineering for web services. In: *Evolution of the web in artificial intelligence environments*. 2008, pp.37–161. Springer-Verlag, Berlin, Heidelberg

17. Praehofer H. *System theoretic foundations for combined discrete – continuous system simulation*. PhD Thesis, J Keper University of Linz, 1991.
18. Kofman E, Lapadula M and Pagliero E. PowerDEVS: a DEVS–based environment for hybrid system modeling and simulation. Technical Report LSD0306, Universidad Nacional de Rosario, 2003.
19. Zeigler, BP, Theory of Modeling and Simulation, First ed., 1976, Wiley Interscience, New York.
20. Kara A, Deniz F, Bozagac CD, et al. Simulation modeling architecture (SiMA), a DEVS based modeling and simulation framework. In: *proceedings of summer computer simulation conference (SCSC'09)*, SCS, 2009, pp.315–321.
21. Fujimoto RM and Weatherley RM. Time management in the DoD high level architecture. In: *proceedings of PADS'96*, 1996, pp.60–67.
22. Hwang MH. Identifying equivalence of DEVSs: language approach. In: *proceedings of 2003 summer computer simulation conference*, Society for Computer Simulation International, 2003, pp.319–324.
23. Hwang MH. Equivalence and minimization of output augmented DEVS. In: *proceedings of 2003 IEEE conference on system, man, and cybernetics*, IEEE, 2003, pp.409–414.
24. Hwang MH and Cho SK. Timed behavior analysis of schedule preserved DEVS. In: *proceedings of 2004 summer computer simulation conference*, Society for Computer Simulation International, 2004, pp.173–178.
25. Uhrmacher AM. Dynamic structures in modeling and simulation: a reflective approach. *ACM Trans Model Comput Simulat* 2001; 11: 206–232.
26. Baati L, Frydman C and Giambiasi N. LSIS_DME M&S environment extended by dynamic hierarchical structure DEVS modeling approach. In: *proceedings of the 2007 spring simulation multiconference*, Norfolk, VA, Volume 2, 25–29 March 2007, pp.227–234.
27. Barros FJ. Dynamic structure discrete event system specification: formalism, abstract simulators and applications. *Trans Soc Comput Simulat* 1996; 13: 35–46.
28. Barros FJ. Modeling formalisms for dynamic structure systems. *ACM Trans Model Comput Simulat* 1997; 7: 501–515.
29. Deniz F, Kara A, Alpdemir MN, et al. Variable structure and dynamism extensions to SiMA, a DEVS based modeling - and simulation framework. In: *proceedings of summer computer simulation conference (SCSC'09)*, SCS, 2009, pp.117–124.
30. Szabo C and Teo YM. An Approach for validation of semantic composability in simulation models. In: *proceedings of the 2009 ACM/IEEE/SCS 23rd workshop on principles of advanced and distributed simulation (PADS '09)*, IEEE Computer Society, Washington, DC, 2009, pp.3–10.
31. Bartholet RG, Brogan DC and Reynolds PF. The computational complexity of component selection in simulation reuse. In: Kuhl ME, Steiger NM, Armstrong FB, et al. (eds) *proceedings of the 2005 winter simulation conference*, 2005, pp.2472–2481.
32. Ahn JH, Seok MG, Sung CH, et al. Hierarchical federation composition for information hiding in HLA-based distributed simulation. In: *proceedings of the 2010 IEEE/ACM 14th international symposium on distributed simulation and real time applications (DS-RT '10)*, IEEE Computer Society, Washington, DC, 2010, pp.223–226.
33. Al-Zoubi K and Wainer GA. RISE: REST-ing heterogeneous simulations interoperability. In: *proceedings of the winter simulation conference 2010*, 2010, pp.2968–2980.

Author Biography

M Nedim Alpdemir received his MSc (1996) in Advanced Computer Science and his PhD (2000) in Component-Based Simulation Environments from the Department Computer Science, University of Manchester, UK. He worked as a Research Associate, and later as a Research Fellow, in the Information Management Group (IMG) at the Department Computer Science of University of Manchester, UK, until 2005. Currently, he is the head of the Software Infrastructures Group and supervises the Simulation Software Frameworks team at TÜBITAK BILGEM UEKAE - İLTAREN, Ankara, Turkey.