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

COMPOSING SIMULATION MODELS USING INTERFACE DEFINITIONS BASED ON WEB SERVICE DESCRIPTIONS

Mathias Röhl
Stefan Morgenstern

University of Rostock
Albert-Einstein-Str. 21
18059 Rostock, GERMANY

ABSTRACT

Using models in different contexts poses major integration challenges, ranging from technical to conceptual levels. Independently of each other developed model components cannot be expected to coincide in all description details, even if based on the same abstractions and assumptions. Variations in interface descriptions of model components have to be resolved. XML-based description languages from the area of web services provide standardized means for bridging diversities of implementations. This paper presents an adaption of the Web Services Description Language (WSDL) combined with XML Schema Definitions (XSD) to the specific requirements of model components in the area of discrete-event simulation. XML-based interface descriptions are integrated into a general model component architecture. Schema matching approaches provide the basis for syntactical compatibility checking of interfaces at the time of composition.

1 INTRODUCTION

To find model components and (re)use them in different contexts is still an issue of ongoing research in the area of modeling and simulation (Davis and Anderson 2004, Overstreet, Nance, and Balci 2002). To become interoperable, variations have to be addressed at different levels ranging from technical to conceptual ones (Tolk and Muguira 2003).

XML-based standards from the area of web services and the semantic web are recognized to play a major role for model integration (Tolk 2006). XML is particularly well suited for importing and exporting models in standard exchange formats and building repositories of model components. However, current XML-based standards are developed for describing software implementations not model components. An unaltered adoption of these standards to the domain of modeling and simulation hinders to tap the full potential of XML and its related technologies. Furthermore, integrating independently developed descriptions automatically remains a challenging task even in the domain of XML (Tolk and Diallo 2005).

The paper starts with an introduction of a basic architecture for model components based on XML and UML's Composite Structure Diagrams. It continues with a review of XML-based standards for describing interfaces of dynamic web resources. Afterward these standards are adapted to allow the description of model component interfaces. Compatibility checking of these interfaces is presented and related to schema matching approaches from the domain of distributed databases.

2 AN ARCHITECTURE FOR MODEL COMPONENTS

The Unified Modeling Language 2.0 (OMG 2005) provides standardized means to define components and composition structures. A slight adaption of UML constructs allows to use them within the domain of discrete-event simulation (Röhl 2006). Same as software components, model components can be described as comprising a set of ports, a set of parts, a set of connectors, and a behavior.

2.1 Component Model

Components need to publicize their provisions and context dependencies. UML 2.0 Composite Structure Diagram uses ports to declare an entity's points of interactions. Ports are typed by interface definitions. Interfaces typically contain method declarations in the realm of software engineering. Within discrete-event modeling and simulation, models generally do not call methods of other models but exchange events. By replacing method declaration of UML's interface definitions with event declarations, UML diagrams are suited for defining model components and compositions (Röhl 2006). A Port of a component refers to exactly one interface specification by means of the interface's identifier

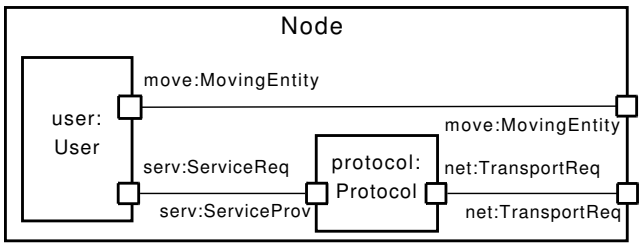

Figure 1: Composition of a node from a user and a protocol component.

and a name attached to it. Using ports instead of direct references to interfaces increases flexibility as multiple ports can be typed by the same interface while carrying different names, e.g. a component can exhibit a role multiple times to different peers. A special flag indicates whether a port is required for a component to function. If set to true the port has to be connected to a compatible interface of another component at the time of composition. If set to false, a port is interpreted as providing a functionality.

For the purpose of composition a component refers to other components as sub components and becomes itself a composite component. Composite components support the hierarchical, modular construction of models and thereby facilitate the development of large models from smaller ones.

Couplings connect ports of one model component to compatible ports of another component. Communication between models is only allowed along these connections. Components may be connected in two different ways. First it is possible to connect provided and required ports by a so called *assembly* connector. In contrast, the *delegation* connector connects two ports of the same type between a component and one of its sub components. Within compositions all required ports of a component need to be connected to a compatible counterpart, either by assembly or by delegation connectors.

Figure 1 shows a simple example for using UML's composite structure diagram for specifying compositions of model components. The example is taken from an ongoing simulation study in the area of mobile ad-hoc networks (Röhl, König-Ries, and Uhrmacher 2007). A network node is defined as the composite component `Node` that exhibits two ports and contains and connects two other components. The user has a port named *serv* that exhibits an interface of type `ServiceReq`. An assembly connector connects this port to the `serv` port of the protocol component, which is typed by `ServiceProv`. The composite component `Node` delegates the contextual dependency of the `protocol` sub component, indicated by a port of type `TransportReq`, to its own `net` port. The `move` port of the `user` is also delegated to an according port of `Node`.

Composite structure diagrams offer two kinds of abstractions. First, for composition within a certain context a component has to be connected via its ports. At the level of a node, the `User` and `Protocol` component are black boxes except their published ports. Besides the references to the published ports of user and protocol, the node makes no further assumptions about the implementation of both. Compositions abstract from internal details of the parts being composed. Second, component implementations of connected components are hidden to each other. The knowledge of components ends at its borders. A component can interact with its environment only via its ports. Thereby, direct dependencies between components and their contexts of use are eliminated. The user and the protocol component can be developed independently of each other.

2.2 XML Representation of Components

Decentralized development and usage of components in the large depends on a flexible representation format and integration with the world wide web. XML-based formats provide this flexibility. Storage of UML specifications is usually done with XMI. For the specification of model components and composition structures, we developed an XML Schema Definition (W3C 2004c) particularly suited for the purpose of discrete-event simulation (Röhl 2006). Schema Definitions mainly define the syntax of an XML document and thereby provide the basis for rendering XML documents syntactically valid or invalid.

An XML component definition, cf. Figure 2, comprises a unique identifier, a set of parameters, a reference to a parameter mapper, a set of ports, a set of sub components, a set of connections (between sub components), and a reference to a model definition. The actual model definition of a component has to be done with an additional XML document. Component definitions refer to models by unique identifiers, i.e. URIs. This makes component descriptions independent from a concrete modeling formalism. Of course, the modeling formalism used for defining the model implementations must be able to refine the abstract, public visible descriptions of a component. Elements of type binding associate the declarations of a component's published ports to "real" ports of the actual model definition.

For providing customizable model components, a component may exhibit a set of parameters. Internally, a component definition refers to a parameter mapper, which will evaluate set parameters and adapt the internal structure of a component at the time of composition.

We first defined interfaces by hard-coded types, i.e. by Java classes (Röhl 2006). Unfortunately, hard coded types limit a decentralized development of components dramatically. The programming language for executing the model is prescribed by the interface definitions. Only very limited variation is allowed for interfaces that become connected, e.g. types that inherit from the same base class of a particular programming language. Thereby, platform independence is

```
<?xml version="1.0" encoding="UTF-8" ?>
<component xmlns="http://www.../component">
  <id>unihro/com/node/v2</id>

  <param name="id" type="int" value="0"/>
  <param name="radiorange" type="double"
       value="250"/>
  <mapper>unihro.com.node.v2.Mapper</mapper>

  <port name="move" required="true">
    <type>MovingEntity/2.0</type>
  </port>
  <port name="net" required="true">
    <type>TransportReq/2.0</type>
  </port>

  <model>unihro/com/node/v2/model</model>

  <composition>
    <type>unihro/com/flooding/v2</type>
    <name>protocol</name>
  </composition>
  <composition>
    <type>unihro/com/user/v2</type>
    <name>user</name>
  </composition>
  <connection fromComponent="protocol"
       fromPort="serv" toComponent="user"
       toPort="serv"/>
  <connection fromComponent="this"
       fromPort="net" toComponent="protocol"
       toPort="net"/>
  <connection fromComponent="this"
       fromPort="move" toComponent="user"
       toPort="move"/>
</component>
```

Figure 2: The node component in XML.

lost. Furthermore, implemented types do not integrate well with data bases, whereas completely XML-based interface definitions can be queried flexibly.

3 XML-BASED DESCRIPTIONS OF WEB SERVICES

Web Services (W3C 2004b) provide a general architecture for distributed systems on the web. Central to the idea of web services is the decoupling of service provider and consumer. To this end, descriptions of services are encoded in a standard format and publicized.

Web Service Description Language, abbreviated WSDL, (W3C 2006b) is the standard to describe the syntactic part of a web service's interface. WSDL documents define the interface of a web service as a set of abstract operations. Each operation is intended to provide a certain functionality. Type definitions form the basis for publishing interfaces of a web service, i.e. they are used for declaring operation signatures and error values. WSDL allows to define simple and complex types by means of XML schema languages, e.g. W3C's XML Schema Definitions (W3C 2004c).

```
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.ex.../service/v2">

  <xs:complexType name="Call">
    <xs:choice>
      <xs:element name="call"
           type="ServiceSearchCall"/>
      <xs:element name="call"
           type="ServiceOfferCall"/>
      <xs:element name="call"
           type="ServiceInvokeCall"/>
      <xs:element name="call" type="LogInCall"/>
      <xs:element name="call"
           type="LogOutCall"/>
    </xs:choice>
  </xs:complexType>

  <xs:complexType name="ServiceSearchCall">
    <xs:sequence>
      <xs:element name="inquirer"
           type="Address"/>
      <xs:element name="serv" type="xs:string"/>
      <xs:element name="id" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
  ...
</xs:schema>
```

Figure 3: Types defined by XML schema definitions.

Technically, WSDL descriptions are XML documents consisting of four main elements. The kind of messages to be transmitted are defined within the *types* element. The *interface* element holds a set of operations that will be provided by the web service. The *binding* section defines how the messages, defined in the interface section, are exchanged. It assigns concrete message formats and transmission protocols such that the service can be accessed. The *service* element specifies where the service is located and can be accessed.

The attractiveness of WSDL stems from its compatibility to a number of related web standards that address technical issues for communications on the one hand and higher levels of interoperability on the other hand. With respect to transmission details, WSDL-descriptions bind to SOAP. As regards higher levels of interoperability, WSDL-descriptions may form the basis for semantic interface definitions of web services, e.g. by means of OWL-S (OWL Services Coalition 2003).

4 ADAPTING WSDL FOR MODEL COMPONENTS

The central idea of WSDL is suited for describing model components as well. WSDL specifies abstract interfaces by referring to type definitions made with a declarative and standardized schema language.

```
<description xmlns="http://www.../xmli"
 xmlns:ser="http://www.ex.../service/v2">

 <id>ServiceReq/2.0</id>

 <types>
  <xs:schema
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:import
        namespace="http://.../service/v2"/>
  </xs:schema>
 </types>

 <interface>
  <eventport name="call" isInput="false">
   <type element="ser:Call"/>
  </eventport>

  <eventport name="response" isInput="true">
   <type element="ser:Response"/>
  </eventport>
 </interface>
</description>
```

Figure 4: Definition of port type `ServiceReq/2.0` based on XSD imports.

4.1 Type Definitions

Defining types by means of XML schema languages is a very flexible approach for abstracting from platform specific details. Whether these types are used for messages transmitted between web services or they define events that can be exchanged between models, is irrelevant for the definitions as such.

Figure 3 illustrates the definition of the complex type *Call*. Such type definitions may be used by model definitions to announce the type of events they are able to send and receive.

While, the WSDL-approach for type definitions can be used for model components without modifications, this is not the case at the level of interface definitions.

4.2 Interface Definitions

Similar to UML, WSDL aims at describing the interface of software implementations. Both WSDL and UML announce points of interactions by means of interfaces. WSDL shares also the method-oriented view on components, which is not well suited for model components in the area of discrete-event simulation. Furthermore, WSDL's mechanisms for exception handling is not very meaningful for interacting model components. Models are preferably implemented in a certain kind of modeling formalism, which usually does not have facilities for exception handling.

Instead of letting an interface declare a set of operations, in our adaption of WSDL an interface definition comprises a set of event port declarations that announce the exchange of typed events. Events may flow in two directions. The special attribute *isInput* indicates whether a port declaration denotes a potential flow of events toward an entity, or it declares a port that is intended to emit events from an entity.

4.3 Usage of Interface Descriptions

UML and WSDL agree on the abstraction from implementations by means of interface specifications that are exhibited by communication endpoints. Compared to UML, WSDL benefits from rooting its interface definitions in XML Schema languages and URI. Thereby, a type system can be established decentralized on a global scope.

Nevertheless, UML's support for hierarchical composition structures, has no counterpart in WSDL. For modeling and simulation purposes, modular hierarchical modeling is an important feature. WSDL defines the syntactic part of an implementation's interface in a single document. In our component architecture based on UML's composite structures, communication endpoints are already contained in the top-level description document for a component (cf. Figure 2).

Model components are usually not directly executable pieces of software, but become simulated by a certain simulation engine. Therefor, our model components use binding elements not to define the technical details of transmission but relate abstract event ports of an interface definition to event ports of a model definition.

WSDL's types and interface concepts provide description elements complementary to the ones pertaining to UML's composite structures. Only the type of single ports has to be defined by WSDL-like descriptions. Thus, we use pruned WSDL-like descriptions for specifying interfaces that can be used within composite structures to type communication endpoints, i.e. ports. The glue between both types of documents are URIs. That is, a port declaration of a component definition refers to an interface declaration via a URI.

An example for the adaption of WSDL interfaces is listed in Figure 4. The document imports the XSD-types defined above. Based on these types the interface description declares two ports: one for receiving events of type *Call* and one for sending *Response* events.

5 COMPATIBILITY OF INTERFACES

The flexibility of XML-based interface descriptions poses a challenge for composition. Independently of each other developed interface definitions usually induce syntactic and semantic variations. Names of elements or attributes and the sequence of elements may vary between interface definitions. These variations have to be resolved at the time of composition.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/
      XMLSchema"
  xmlns:sawsdl="http://www.w3.org/2002/ws/sawsdl/
      spec/sawsdl#">

  <xs:complexType name="Adress"
    sawsdl:modelReference="http://www.example.org/
      IPAddress">
    <xs:attribute name="p1" type="xs:integer"/>
    <xs:attribute name="p2" type="xs:integer"/>
    <xs:attribute name="p3" type="xs:integer"/>
    <xs:attribute name="p4" type="xs:integer"/>
  </xs:complexType>
</xs:schema>
```

Figure 5: Type definition enriched with semantic annotations.

5.1 Matching

Within our component architecture a composition is considered to be sound, if all required ports of all sub components are connected to compatible counterparts. For each connection, compatibility has to be checked between two concrete interface types. As interface types are defined within schemas, compatibility checking can be based on schema matching approaches. Two types of two different schemes can be judged compatible, if a mapping between these two types can be established. Schema matching is a basic problem in many database and integration applications (Do, Melnik, and Rahm 2003). With schema matching, compatibility can be weaken from syntactic equivalence (Walsh 2004) of interface types to a less restrictive relation. It is usually carried out semi-automatically, since fully automatic solutions are usually not possible and manual matching of data by domain experts is time-consuming and tedious.

There exist different approaches to compare schemas in practice. Simple matchers compare two schema elements only with respect to one property. Examples are simple name matchers, phonetic matchers, structural similarity matcher, or synonym matchers based on dictionaries. Simple standalone matchers are too limited for practical matching tasks. Instead, powerful matching relies on a set of different (simple) matcher modules that can be combined on demand (Do and Rahm 2002). A modular matching architecture turns out to be especially beneficial for the modeling and simulation domain. As a simulation model may contain a huge number of different instances from a limited set of model component definitions, the integration of a matcher, which reuses previous matching results, saves a lot of checking effort.

5.2 Integrating Semantics

Syntactic descriptions provide information about the structure of input and output messages of an interface. Schema matching approaches try to deal with variations of type

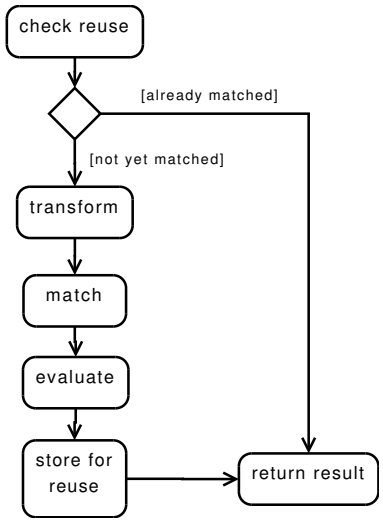

Figure 6: Matching activities.

definitions. To really disambiguate descriptions, e.g. in the case of homonyms, semantic information are needed.

Semantic Annotations for WSDL, abbreviated SAWSDL, (W3C 2006a) allow to add semantic information to WSDL and XML Schema definitions. Semantics are introduces by URI references to ontological definitions, e.g. made with the Web Ontology Language (W3C 2004a).

Figure 5 shows the definition of the type `Address` that was used in the definition of type `ServiceSearchCall` above (cf. Figure 3). Additionally to the schematic definition of an `Address`, the `modelReference` attribute states that an IP-Address is meant.

With SAWSDL syntactic differences in type definitions can be addressed explicitly by optionally referencing a schema mapping. However, the schema matching approach presented above requires no explicit mapping, as it is able to generate the mapping function from the matching process. Thereby, syntactic compatibility checking provides the details of how interfaces are made compatible, after semantic checkers have decided whether they are compatibility.

5.3 Tool Support

Figure 6 shows the activities as carried out by the match engine. First, it checks whether the types under consideration were already checked. If not, they are transformed to an internal representation, which basically is an attributed, directed acyclic graph (cf. Figure 7). Matching is then carried out on these type trees with a set of matchers made known to the tool. The overall outcome of matching is determined based on configuration. Table 1 shows a possible configuration for the match engine. It defines which matcher to use for which type information. Furthermore it assigns weights to the atomic matchings and how they are going to be combined. Finally, the result is compared to a constant

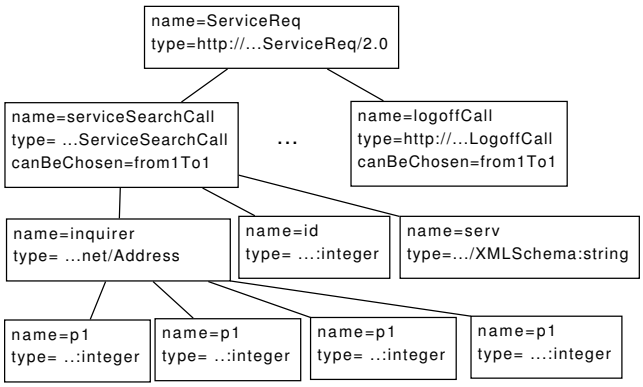

Figure 7: Internal representation of type `ServiceReq`.

Table 1: Configuration for matching and evaluation.

Information	Matcher	Weight	Mode
ModelRef	Equivalence	0.3	-
PathName	TriGram	0.15	-
PathName	EditDistance	0.15	-
TypeName	TriGram	0.1	-
TypeName	EditDistance	0.1	-
∀ leave types	EditDistance	0.1	avg
∀ children types	EditDistance	0.1	avg

value, expressing the minimum acceptable matching value, and stored for reuse.

Both the matcher *TriGram* and *EditDistance* referenced in Table 1 compare strings. *EditDistance* essentially calculates the Levenshtein distance. *TriGram* calculates the ratio between the number of equal character sequences of length three to the number of possible equal sequences.

Obviously, matching will only be successful if types refer to the same semantic concept and are syntactically similar. The kind of acceptable syntactic similarity can be varied by plugging different kind of matchers into the engine and using different combination modes. The degree of similarity may be configured with a global constant. Furthermore, the matching process can be supported by providing additional knowledge to it, i.e. known matches can be specified in a special configuration file.

Spin-off producst of the compatibility checking process are mappings between compatible types. For simulation-oriented composition approaches these mappings can be used to generated run-time adapters for communication between involved components. In the case of model-oriented composition approaches (Röhl and Uhrmacher 2006) the mappings can be used to generate message implemenations.

6 RELATED WORK

The presented approach for defining components and compositions orientates on UML's 2.0 Composite Structure Diagrams (OMG 2005). Interface type definitions draw their basic ingredients from WSDL (W3C 2006c). Both provide a method-oriented approach to interface definitions. Recently, UML's limitation of a purely operation-centric style was addressed by deriving a language, which aims at systems engineering. SysML (OMG 2006) provides flow ports in addition to standard method-oriented ports. In contrast to SysML, this paper combines elements of UML and WSDL for the purpose of defining model components and compositions. Selected concepts were adapted and integrated into a component framework for discrete-event simulation.

The adaption of web service standards and technologies to the domain of modeling and simulation has been done previously (Pullen et al. 2005, Möller and Dahlin 2006). These approaches adapt web service techniques in a rather direct manner at the software level. Models are wrapped by simulation engines and together treated as executable software entities, technically integrated via runtime infrastructure like HLA (IEEE 2000). We propose to adapt selected concepts from the area of web services technologies to *models*, not *simulations*.

Our approach is close to Gustavson and Chase (2004) in exploiting the flexibility of XML together with XSD and striving for platform independent specifications. Both approaches suggest to transform platform independent XML representations into platform specific models. While BOMs focus on HLA compliance, the approach presented here puts a strong emphasize on component definitions close to UML.

In the field of (distributed) data bases it is a known problem to find correspondences between elements of different data sources. Structure of data is prescribed by schema languages and the problem of finding correspondences between data schemas referred to as schema matching (Do and Rahm 2002). The challenge induced by differences in data type descriptions is also mentioned in the context of modeling and simulation (Tolk and Diallo 2005). This paper provides a concrete strategy for dealing with variations in interface descriptions based on schema matching and semantic annotations (W3C 2006a). It focusses on compatibility matching at the syntactical level and thereby complements ontology-based matchmaking approaches, (e.g. Yilmaz and Paspuleti 2005).

7 CONCLUSION

Component-based modeling faces the challenge to integrate descriptions that show syntactic and semantic deviations. Standards and technologies from the area of software development and web services provide a solid basis for the modeling and simulation domain, if adapted cautiously.

The presented approach combines complementary specification elements of UML and WSDL to allow a decentralized development of model components based on declarative interface definition. UML's hierarchical composition structures are enriched with type definitions inspired by WSDL.

The flexibility of the approach stems from the use of XML Schema languages for defining and referencing types and interfaces.

XML-based interface descriptions decouple the development of components. Models can be developed independently of each other and be equipped with their own interface definition. Component developers are not forced to build their component definitions on the base of central type repositories. Instead, references can be limited to (decentralized) ontology definitions.

At the time of composition syntactical variations have to be reflected in compatibility checking procedures. The important question is, when to judge differences in interface definitions as neglect-able and when as hazardous. Schema matching approaches and semantic annotations provide an automatable answer to this. Both integrate well with the proposed component architecture.

The presented component specifications are currently used for a simulation study in the area of mobile ad-hoc networks (Röhl, König-Ries, and Uhrmacher 2007). The until to now used Java-typed interface descriptions shall be replaced with schema-defined types. The focus of future work is to generate Java implementations for interface types from type mappings.

REFERENCES

Davis, P. K., and R. H. Anderson. 2004, April. Improving the composability of DoD models and simulations. *JDMS* 1 (1): 5–17.

Do, H. H., S. Melnik, and E. Rahm. 2003. Comparison of schema matching evaluations. In *Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*, 221–237. London, UK: Springer-Verlag.

Do, H. H., and E. Rahm. 2002. COMA - a system for flexible combination of schema matching approaches. In *VLDB 2002*, 610–621.

Gustavson, P., and T. Chase. 2004. Using XML and BOMs to rapidly compose simulations and simulation environments. In *Proceedings of the 2004 Winter Simulation Conference*, 1467–1475.

IEEE 2000, September. Standard for modeling and simulation (M& S) High Level Architecture (HLA) — Framework and Rules. Document 1516-2000.

Möller, B., and C. Dahlin. 2006, June. A first look at the HLA evolved web service api. In *Proceedings of 2006 Euro Simulation Interoperability Workshop*. Simulation Interoperability Standards Organization. 06E-SIW-061.

OMG 2005, July. UML superstructure specification version 2.0 (document formal/05-07-04). <www.omg.org/cgi-bin/doc?formal/05-07-04>.

OMG 2006, May. SysML final adopted specification. <www.omg.org/cgi-bin/doc?ptc/06-05-04>.

Overstreet, C. M., R. E. Nance, and O. Balci. 2002. Issues in enhancing model reuse. In *International Conference on Grand Challenges for Modeling and Simulation, Jan. 27-31*. San Antonio, Texas, USA.

OWL Services Coalition 2003. OWL-S: Semantic markup for web services. <www.daml.org/services/owl-s/1.0/owl-s.html>.

Pullen, J. M., R. Brunton, D. Brutzman, D. Drake, M. Hieb, K. L. Morse, and A. Tolk. 2005. Using web services to integrate heterogeneous simulations in a grid environment. *Future Generation Computer Systems* 21:97–106.

Röhl, M. 2006, May 28th-31th. Platform independent specification of simulation model components. In *ECMS 2006*, 220–225. Bonn, Sankt Augustin, Germany.

Röhl, M., B. König-Ries, and A. M. Uhrmacher. 2007. An experimental frame for evaluating service trading in mobile ad-hoc networks. In *Mobilität und Mobile Informationssysteme (MMS 2007)*, Volume 104 of *Lect. Notes Inform.*, 37–48.

Röhl, M., and A. M. Uhrmacher. 2006. Composing simulations from XML-specified model components. In *Proceedings of the Winter Simulation Conference*, 1083–1090: ACM.

Tolk, A. 2006. What comes after the semantic web – PADS implications for the dynamic web. In *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation (PADS'06)*.

Tolk, A., and S. Y. Diallo. 2005. Model-based data engineering for web services. *IEEE Internet Computing* 9 (4): 65–70.

Tolk, A., and J. Muguira. 2003, September. The level of conceptual interoperability model. Fall Simulation Interoperability Workshop (SISO), Orlando.

W3C 2004a. OWL Web Ontology Language: Overview. <www.w3.org/TR/2004/REC-owl-features-20040210/>. W3C Recommendation 10 February 2004.

W3C 2004b. Web services architecture. <www.w3.org/TR/2004/NOTE-ws-arch-20040211/>. W3C Working Group Note 11 February 2004.

W3C 2004c, October. XML Schema part 0: Primer second edition. <www.w3.org/TR/2004/REC-xmlschema-0-20041028/>. W3C Recommendation 28 October 2004.

W3C 2006a. Semantic annotations for WSDL. <www.w3.org/TR/2006/WD-sawsdl-20060928>. W3C Working Draft 28 September 2006.

W3C 2006b. Web services description language (WSDL) version 2.0 part 0: Primer. <www.w3.org/TR/2006/CR-wsdl20-primer-20060327>. W3C Candidate Recommendation 27 March 2006.

W3C 2006c. Web services description language (WSDL) version 2.0 part 1: Core language. <www.w3.org/

TR/2006/CR-wsdl20-20060327>. W3C Candidate Recommendation 27 March 2006.

Walsh, N. 2004, May. Infoset equality. <norman.walsh.name/2004/05/19/infoset-equal>. From the Technical Plenary, a URI that got lost: a quick "off-the-cuff" definition for XML chunk equality based on the Infoset.

Yilmaz, L., and S. Paspuleti. 2005, July. Toward a meta-level framework for agent-supported interoperation of defense simulations. *JDMS* 2 (3): 161–175.

ACKNOWLEDGMENTS

This research is supported by the DFG (German Research Foundation)

AUTHOR BIOGRAPHIES

MATHIAS RÖHL holds a MSc in Computer Science from the University of Rostock. His research interests are on component-based modeling and agent-oriented simulation. He is currently a research scientist at the Modeling and Simulation Group at the University of Rostock. His e-mail address is <mroehl@informatik.uni-rostock.de>. Web address of his homepage is <www.informatik.uni-rostock.de/~mroehl>.

STEFAN MORGENSTERN is currently finishing his Diploma in Computer Science at the University of Rostock. His email adress is <stefan.morgenstern@informatik.uni-rostock.de>