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

A METAMODEL-BASED REPRESENTATION METHOD FOR REUSABLE SIMULATION MODEL

Yonglin Lei
Lili Song
Weiping Wang
Caiyun Jiang

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

ABSTRACT

The composition and reuse of simulation model is currently a hot research topic in the area of complex system modeling and simulation. A unified simulation model representation is one of the key techniques to facilitate reuse. The model representation concept and its role in simulation are present firstly. The reusable simulation models, when compared to common simulation models, give some special requirements to the specification and method of model representation, which are summarized in the second part. Three traditional simulation model representation methods, i.e. representation by programming language, by simulation language and by generic model language, and their respective shortcomings when representing reusable simulation models are analyzed next. The advantages of metamodel in software representation are widely acknowledged in recent years. With this in mind this paper presents a metamodel based method to represent reusable simulation model, and the design considerations of metamodel-based representation specification is also discussed.

1 INTRODUCTION

Nowadays simulation composability is one of the most challenging topics in the military simulation domain. Many literatures (Overstreet, Nance, and Balci 2002, Davis and Anderson 2003, Hofmann 2004, Yilmaz and Oren 2006) have figured out that model composition and reuse is the key issues of simulation composability research. Model reuse is a systematic process that emphasizes using existing models as much as possible and avoiding repeating modeling in order to reduce simulation development costs (Lei et al. 2007). The key to simulation model reuse is to enhance the reusability of model itself, which can be achieved by fostering reuse-oriented modeling idea and providing supporting techniques for reuse. An important technique is to build a unified simulation model representation, which can be used: (1) to support the communication among various users of the same model about its functionality and semantics, (2) as the collaborative analysis base of different modeling formalisms used to specify the different facets of complex system, and (3) as the target representation to which existing models can transform for use in a reuse architecture.

The simulation model representation methods are the methods of designing simulation model specification. According to the specific requirements implied by *REusable simulation Model (REM)* over model representation, the shortages of several current representation approaches are analyzed firstly. Inspired by Dr Tolk's idea of metamodel's key role in improving simulation interoperability and composability (Tolk 2004), we develop a meta-model based model representation method which can meet the requirements of REM.

2 REM REPRESENTATION REQUIREMENTS

2.1 Simulation Model Representation

Simulation modeling is a process in which behavioral logic of dynamic systems is described and simulation models are produced. To represent a simulation model means to describe and store it. One of model representation's basic features is multi-user oriented. The target users of model representation include modelers, model integrators, simulation analysts, simulation environment and so on. It should provide consistent model contents for all its users and is the core and base of simulation study, as shown in Figure 1. Among them, Modeler chooses appropriate modeling techniques according to the system's features to model it. The result model will be stored as a model file in terms of a certain representation approach. The very model file is the model representation. Various participants in a simulation system can work with different parts of same model representation according to specific interests by such techniques as projection, transformation, visualization etc. Modelers are mainly concerned with the correctness of the model representation, i.e. whether the model representation accurately describes the structural and behavioral logic features of the real system. Model integrators are more concerned with capabilities and interface information within model representation, and do their integration work based on those information. Simulation analysts' mainly focuses are on the run-time input and output interface information and running environment requirements within model representation. Simulation environment is concerned with executable service framework and available simulation services information in model representation. It is obvious that different users have different requirements for model representation. The key to select a model representation method is to make the model representation capable of meeting all users' requirements.

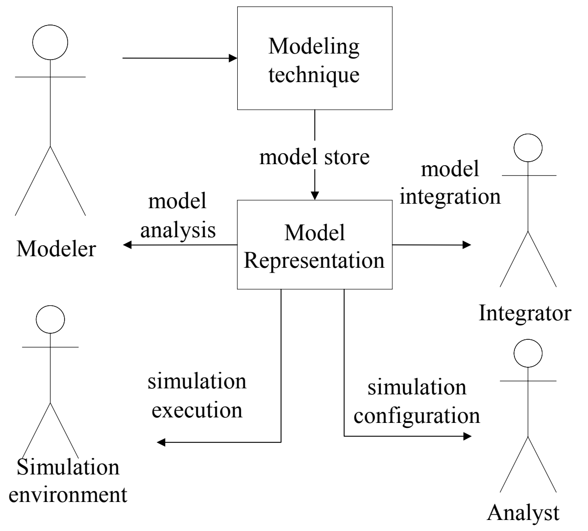

Figure 1: Model representation in simulation.

2.2 REM Representation Requirements

As a common simulation model, the REM representation should be both user-readable and machine-executable. As a reusable simulation model, REM representation's main characteristic is its support for reusability. The reusability imposes some new requirements to REM representation, which can be summarized as follows:

2.2.1 Representation Requirements Towards Modeling

Well modeling techniques are often based on certain math formalisms and use domain-oriented modeling concepts so as to make modeling as easy as possible. For a large-scale complex system, a single formalism is often not adequate to describe all aspects of the system. Various disciplines or subsystems of a complex system are usually based on different formalisms and described with domain-specific concepts. For this reason, REM representation should meet following basic requirements:

R1: can represent models described using multiple formalisms;

R2: can represent models described using different domain-specific concepts;

For those models based on existing modeling languages and modeling tools, REM representation should:

R3: support integration of these models by transformation;

In order to provide some implementation selectiveness to REM, REM representation should not be restricted to specific programming languages and technical platforms. So REM representation should:

R4: be independent of languages and platforms;

It's favorable that model representation can be automatically mapped to various programming languages and platforms, which is the core idea of MDA. That is to say, it is necessary that model representation is compatible with MDA.

2.2.2 Representation Requirements Towards Reusability

In order to support the analysis and judgment of model reusability, REM representation is required to support computer-aided analysis of behavioral logic equivalence. Therefore, behavioral logic in REM representation should be formalized, which requires REM representation to:

R5: support formalized representation and analysis of REM behavioral logic;

2.2.3 Representation Requirements Towards Simulation

A difference between REM simulation and common simulation is that REM is independent of a specific simulation framework. So REM representation should:

R6: support portability across different simulation frameworks.

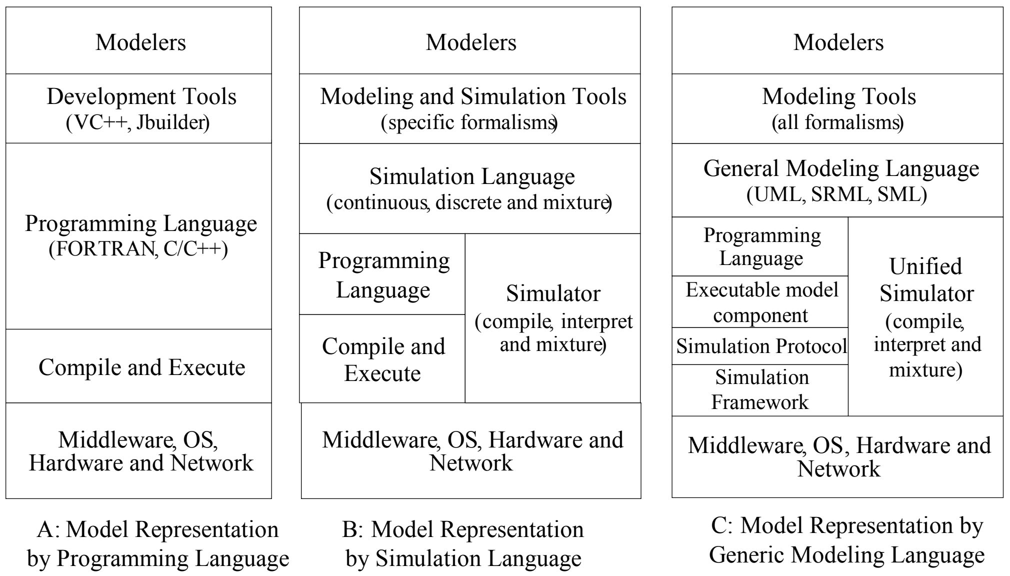
Modelers
Development Tools (VC++, Jbuilder)
Programming Language (FORTRAN, C/C++)
Compile and Execute
Middleware, OS, Hardware and Network
A: Model Representation by Programming Language
Modelers
Modeling and Simulation Tools (specific formalisms)
Simulation Language (continuous, discrete and mixture)
Programming Language
Compile and Execute
Simulator (compile, interpret and mixture)
Middleware, OS, Hardware and Network
B: Model Representation by Simulation Language
Modelers
Modeling Tools (all formalisms)
General Modeling Language (UML, SRML, SML)
Programming Language
Executable model component
Simulation Protocol
Simulation Framework
Unified Simulator (compile, interpret and mixture)
Middleware, OS, Hardware and Network
C: Model Representation by Generic Modeling Language

Figure 2: Typical model representation methods.

3 META-MODEL BASED MODEL REPRESENTATION

3.1 Current Simulation Model Representation Methods

Model representation is a bridge between user and computer. Model representation specification is a protocol between modeling and simulation. All simulation models must use some representation method which can be evaluated by friendliness and generality. Obviously, the way by which humans think and that by which computers process are extremely reverse for model representation. General speaking, the higher abstract degree of model representation is, the nearer to modeler's thinking way, and the better friendliness. On the design of representation method, therefore, a core tenet is to make the degree of abstract as high as possible and ignore the technical details required by implementation. Nevertheless, different domains have different abstract characteristics and representation requirements, and with the degree of abstract increasing, the generality of representation method falls down, and the most friendly representation method for certain domain is often difficult to be applied in other domains. There is trade-off between friendliness and generality for the representation requirements of models involving multi-domains.

Simulation model representation methods mainly go through three stages, including representation by programming language, representation by Simulation Language and representation by generic modeling language (Figure 2). In various domains all three representation methods are still in use to different degree.

3.1.1 Representation by Programming Language

Early simulations have mostly aimed at continuous systems, whose simulation model representations are often based on Programming Languages such as FORTRAN and following C, C++, Java and Ada. The model representations of later arisen Discrete Event Simulation, which became the most important domain of simulation research, are largely based on Programming Languages.

Programming Language is characterized with generality, and can represent almost all simulation models. The main shortcoming is its low degree of abstract, poor friendliness and high modeling difficulty. So additional model documents are needed to support user to perceive model information, and powerful visual program tools and common functional module library are built to facilitate modeling. Another shortcoming of Representation by Programming Language is that it doesn't prescribe a standardized simulator framework. In a common simulation application, models and their simulator are not explicitly distinguished and are mostly compiled together, which makes the models not portable and hard to reuse. For this reason, many communities or organizations have standardized interfaces between simulator framework and simulation models. Two well-known such interface standards are HLA/RTI maintained by first DOD then IEEE and SMP/SMI (Argüello et al. 2000) maintained by ESA. Currently Representation by Programming Language is mainly used in co-operational simulations of multidisciplinary and highly changeable complex systems.

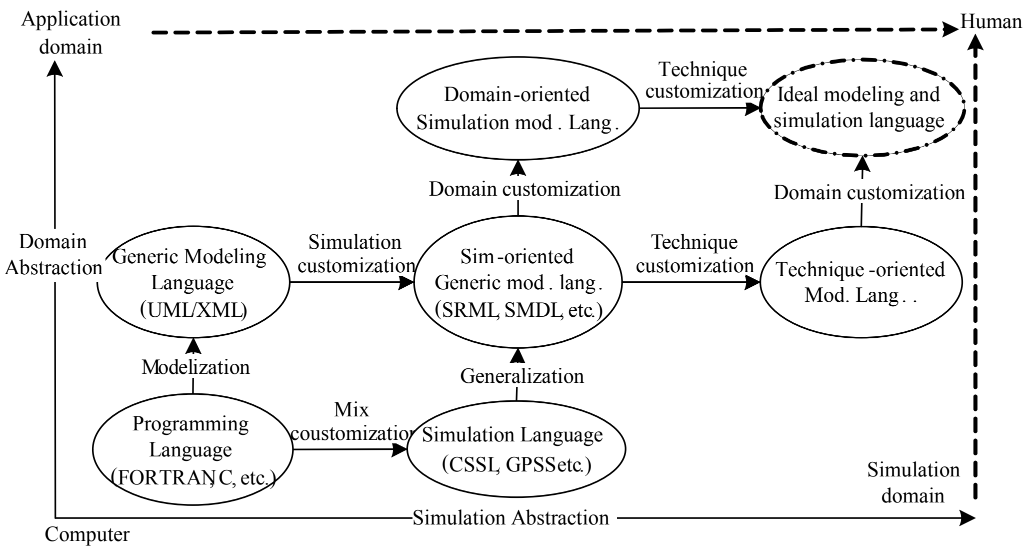
Application domain
Human
Technique customization
Domain-oriented Simulation mod. Lang.
Ideal modeling and simulation language
Domain customization
Domain customization
Domain Abstraction
Generic Modeling Language (UML/XML)
Simulation customization
Sim-oriented Generic mod. lang. (SRML, SMDL, etc.)
Technique customization
Technique-oriented Mod. Lang..
Modelization
Generalization
Programming Language (FORTRAN, C, etc.)
Mix coustomization
Simulation Language (CSSL, GPSS etc.)
Simulation domain
Computer
Simulation Abstraction

Figure 3: The evolution of simulation model representation.

3.1.2 Representation by Simulation Language

Simulation Languages are generally derived from the module library of some Programming Language. Said differently, the modeling primitives of a Simulation Language are devised from the common modules that are generic for simulation modeling in general or specific domains. And the common modules to support simulation control and schedule are encapsulated into simulator or simulation primitives of the Simulation Language. If the simulator is produced, the models represented by the Simulation Language are generally interpret-then-executed, and the behavior of modeling primitives is usually generated by executing the executable module that has been compiled into simulator. If the simulation primitives are produced, the modeling and simulation primitives can be automatically translated into codes of some Programming Language. For example, ACSL models can generate FORTRAN representation, Simulink models can generate C representation, and ROSE (Williams, St. Pierre, and Nergaard 1999) models can generate FORTRAN representation. Then all model codes and simulation codes can be compiled into a whole executable simulation program.

Simulation Language is usually used for certain simulation domain, such as CSSL (including ACSL and ICSL and other specific languages conform to CSSL) for continuous system simulation, GPSS for discrete event system simulation, SLAM for network system simulation, and Modelica for multi-domain physical simulation. Simulation Language is specially designed for simulation, so it can embed specific modeling techniques, which is an advantage over Programming Language (e.g. GPSS and SLAM embeds process-oriented modeling technique), or it can be directly designed in term of some modeling techniques, such as Euler network modeling language (Wang et al. 2003), XML-based Statecharts modeling language – PcML (Amaral et al. 2004) and Petri network modeling language – PNML (Billington et al. 2004). To simplify modeling, the support tools of Simulation Languages generally support graph-based modeling, and the languages embedding some modeling techniques can model using relevant graph-based modeling environments.

Compared with Programming Language, Simulation Language has greatly improved its degree of abstract, for example the continuous and discrete system simulation are distinguished at least. Coarse-grained modeling and simulation primitives and the separation of models and simulator all reduce the difficulty of modeling and enhance the reusability of simulation. When confronted with multi-domain co-operation simulation application, however, Simulation Language is insufficient inevitably.

3.1.3 Representation by Generic Modeling Language

Generic modeling language is a kind of software system modeling language with object-oriented concept in mind. It is the platform-independent abstract of various Programming Languages and can be automatically mapped into various Programming Language and generate platform-dependent codes. Generic modeling language further provides modeling elements to describe the structure, behavior and function of general software systems, so as to support modeler to define system specification and establish system models independent from specific implementation techniques. Currently UML/XML is the in-fact standard of generic modeling language. UML is used for visual expression and XML for model storage.

Simulation model (i.e. computer simulation models without exception) is a special software system that can also be described using generic modeling language. In fact many simulation models are modeled using UML, and

Table 1: Applicability of methods for REM representation.

Requirements \ Methods	Representation by Programming Language	Representation by Simulation Language	Representation by Generic modeling language
Friendliness	Poor	Good	Moderate
Generality	Excellent	Moderate	Excellent
R1: Multi-Formalisms	Indirect support	Support one or few	No support limitation by extensible mechanisms
R2: Multi-Domain Concept	No support	Only support specified domain(s)	No support limitation By extensible mechanisms
R3: Model Integration	Hardly support	Hardly support	**Support partly**
R4: Language- and Platform-Independent	No support	No support	Support fully
R5: Formalization and Analyzability of Behavior Logic	No support	Support by the embedded formalisms	Support by the embedded formalisms
R6: Portability	Models are compile-then-execute, so can support by adding standardized model interfaces.	If models are interpret-then-executed, it's difficult to define the models interfaces. So it depends.	Model codes are auto generated and compile-then-executed, Can support by standardized model interfaces.

there are some generic modeling languages towards simulation domain, such as UML-based simulation model definition language (SMDL) (ESA 2005) in SMP2, XML Schema-based SRML and SML (Kilgore 2001). According to basic features of simulation and based on the extensible mechanism of generic software modeling languages, these languages define their modeling and simulation elements and are still very generic in simulation domain. They also inherit the extensibility of generic software modeling language, which makes it capable of generating sub-languages towards specific simulation domains.

Generic modeling languages are competent enough for expressing simulation models and can support various modeling techniques. When utilizing its extensible mechanisms, a generic modeling language can be customized as a technique-oriented modeling language. It is certainly that the technique-oriented modeling language can be further customized into an application domain-oriented modeling language.

3.2 Applicability of Representation Methods

The applicability of previous three methods when representing REM is shown in Table 1.

The table shows that Programming Languages are not suitable to represent REM although satisfying some requirements, because they are too low in degree of abstract when viewed as simulation languages. Simulation Languages are designed towards some modeling technique and application domain, and can hardly meet the REM's multi-formalism and multi-domain representation requirements. Generic modeling language introduced the advantages of both Programming Language and Simulation Language to some degree, but is insufficient to integrate the existing heterogeneous models. These insufficiencies can be made up by meta-modeling technique, for that the integration of heterogeneous models can be implemented on the meta-level, which is more efficient to deal with the heterogeneous problems.

3.3 Metamodel-based Representation

Programming Languages and Simulation Languages, as well as some generic modeling languages, are usually defined using Extensible Backus-Naur Form (EBNF) (Xia and Martin 2003), which is broadly used to define the grammar of languages. The EBNF grammar's main limitation is that it doesn't support some advanced language features, such as refining, generalization, namespace and modularization. As an instead, multi-hierarchical meta-modeling architecture supports these features and enable meta-model to describe the inherit-hierarchy of concept. Generic modeling languages like UML employ meta-model to define themselves (Marcus and Porres 2003).

Meta-model (Gregory 1999) is the model of models. It defines how to create models, semantics of models as well as how to make them integrated and interoperable. It is the specification of modeling environment for certain domain, and defines the syntax and semantics of the domain and can represent all systems in the domain. Meta-model has higher degree of abstraction than model, and can deal with the models integration problem better.

Meta-model based representation is also a kind of representation by generic modeling language. The difference with others is that it defines generic simulation modeling language explicit based on meta-model. When designing the simulation-oriented metamodel, the legacy models integration requirement is well taken into account, so as to make the metamodel the integration base of various modeling techniques, modeling tools and modeling languages. This metamodel is called Common Meta Model (CMM) in this paper. The CMM based REM representation is shown in Figure 4.

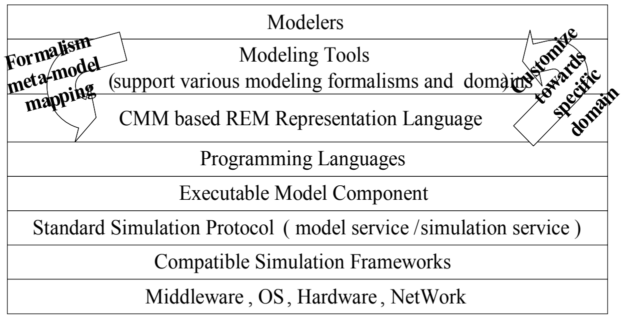
Modelers
Formalism meta-model mapping
Modeling Tools (support various modeling formalisms and domains)
Customize towards specific domain
CMM based REM Representation Language
Programming Languages
Executable Model Component
Standard Simulation Protocol (model service /simulation service)
Compatible Simulation Frameworks
Middleware , OS , Hardware , NetWork

Figure 4: CMM based REM representation.

Figure 4 is different from figure 2-C in two points:

1. The generic modeling language in figure 4, i.e. the REM representation language, is based on CMM, which will support the integration of existing modeling techniques, tools and languages by meta-model mapping.
2. In metamodel based REM representation approach the models' running style is unified as compile-then-execute, so as to design the interface standard between REM implementation components and simulation framework. The interface standard acts as a simulation protocol of REM, and includes model service protocol and simulation service protocol two parts.

The CMM's support to existing languages, tools and techniques as well as application domains is illustrated in Figure 5, in which the meta-modeling and meta-model mappings are the core concepts and techniques. Meta-modeling will ensure the syntax consistency between other meta-models and CMM; and the purpose of meta-model mapping is to construct the semantic correspondence of metamodels, so as to do works like model transformation and codes generation. In Figure 5, the middle column "formalisms/CMM/application domains" serves to design the standard REM modeling tools, and the middle row "modeling language/CMM/modeling tools" serves for the integration of legacy models.

3.4 CMM Modeling Considerations

3.4.1 Selection of Metamodeling Language

Just like modeling, meta-modeling is also a kind of modeling, and in essence meta-modeling languages belong to modeling languages. However, the modeling languages in common use have not made considerations to support meta modeling in their design time, and are not suited to be meta-modeling languages. General speaking, the modeling languages used for meta-modeling can be divided into two categories: the first category includes special-designed meta-modeling languages, such as CASE Data Exchange Format (CDIF), Meta Object Facility (MOF); the other category includes common modeling language with meta-modeling capability, such as UML, EXPRESS and ER diagram. Since UML2.0 and MOF2.0 have been consistent in the meta-modeling core, and both are defined using the UML 2.0 Infrastructure, CMM should adopt a combination of MOF, UML, OCL, XMI and QVT, because this combination:

1. Has enough capability of meta-modeling. MOF can be used to define the abstract syntax of meta-model, and UML can represent the meta-model diagrammatically, and OCL can define the static semantic of meta-model, and XMI is used as the storage specification of meta-model. To sum up, the modeling requirements of CMM are satisfied sufficiently.
2. Is in fact standard of meta-modeling. MOF and UML are the actual meta-modeling standards in software domain, which makes all MOF-based meta-models of various modeling languages and modeling tools exchangeable and interoperable easily.
3. Many modeling tools, after extending their meta-models towards simulation direction, can be used to create simulation models. Since CMM is based on UML's meta-modeling core, current UML tools can be extended expediently so as to support CMM and be used to create REM models.
4. Query/View/Translation, published by OMG recently, can describe the mapping between CMM and meta-models of models to be integrated. QVT not only can implement the transformation from legacy models to REM, but also can automate the converse transformation from REM to formalisms like DEVS, Statecharts and Petri net, which are easier when doing model analysis.

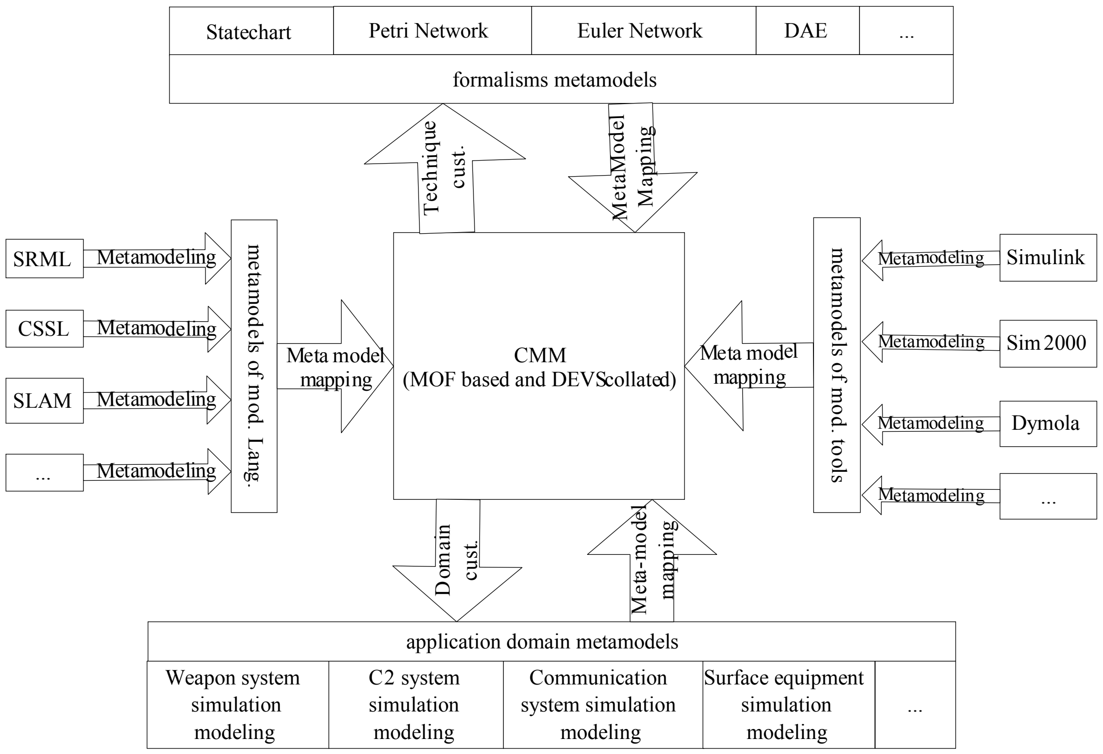

Figure 5: Common meta model (CMM).

3.4.2 Selection of Abstraction Level

Modeling languages can inhabit in different abstraction levels. Since mapping of various modeling formalisms is one of CMM's basic design requirements, CMM's abstraction level should not be too high; otherwise the mapping from lower abstraction formalisms to CMM will lose semantic information. Zeigler has proved that DEVS can represent almost all kinds of system models, including continuous time models, discrete time and discrete event models, and is the super formalism of various formalisms (Zeigler, Kim, and Praehofer 2000). In the Formalisms Transformation Graph (Hans, Lara, and Mosterman 2002) constructed by Professor Hans Vangheluwe, he points out that DEVS's level of abstract is lower than any other formalism, and all other formalisms therefore can be map to DEVS not only losing none semantic, but also making the semantic of more abstract formalisms explicit. For this reason, CMM should view the lossless mapping from DEVS as its design aim.

4 CONCLUSIONS

We proposed a metamodel-based simulation model representation approach, which can well overcome the shortcomings of traditional simulation model representation methods when representing the Reusable Simulation Models (REM). The design considerations of a specific metamodel REM representation specification, i.e. Common Metamodel (CMM), are also discussed. We have developed a initial CMM implementation inspired by Simulation Model Definition Language (SMDL) of Simulation Model Portability 2 (SMP2) standards, which is a self-bootstrapped metamodel but not based on OMG's MOF suite. As a partial validation to CMM, we created a MOF-based DEVS metamodel, and finished the mapping from this DEVS metamodel to CMM and hereby the QVT-based transformation from DEVS models to REM (Lei 2006). The development of a CMM-based modeling and simulation environment of REM is our main work of next step.

REFERENCES

Amaral, A. S., R. R. Veloso, N. L. Vijaykumar, C. R. L. Frances, E. Oliveira. 2004. On proposing a markup language for statecharts to be used in performance evaluation. *International Journal of Computational Intelligence* 1:3.

Argüello, L. et al. 2000. SMP: a step towards model reuse in simulation, ESA Bulletin number 103.

Billington, J., S. Christensen, K. Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno, and M. Weber. 2004. The Petri net markup language: concepts, technology, and tools. *Meeting on XML/SGML based Interchange Formats for Petri Nets*, Arhus, Denmark.

Davis, P.K. and R. H. Anderson. 2003. Improving the composability of DoD models and simulations.

RAND, National Defense Research Institute, Santa Monica, CA.

ESA. 2005. SMP 2.0 Metamodel, Technical Report No EGOS-SIM-GEN-TN-0100.

Gregory, G. N. 1999. *Metamodeling - Rapid Design and Evolution of Domain-Specific Modeling Environments*. Ph.D. Thesis, Vanderbilt University.

Hans, V., J. D. Lara, and P. J. Mosterman. 2002. An introduction to multi-paradigm modelling and simulation. In *Proceedings of AI, Simulation and Planning–AIS'2002*. Lisbon. SCS International.

Hofmann, M. 2004. Challenges of model interoperation in military simulations. *SIMULATION* 80: 659-667

Kilgore, R. A. 2001. Open source simulation modeling language (SML). In *Proceedings of the 2001 Winter Simulation Conference*, ed. B. A. Peters, J. S. Smith, D. J. Medeiros, M. W. Rohrer, 607-613. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Lei, Y. L. 2006. *Simulation Model Reuse Theory and Approaches with Heterogeneous Integration Support.* Ph.D. Thesis, National University of Defense Technology.

Lei, Y. L., W. Zhao, W. P. Wang, and Y. F. Zhu. 2007. A simulation model reuse approach for virtual prototyping. *Computer Integrated Manufacture System* 13:287-294

Marcus, A. and I. Porres. 2003. A relation between context-free grammars and meta object facility metamodels. TUCS Technical Report No 606, ISBN 952-12-1337-X, ISSN 1239-1891.

Overstreet, C. M., R. E. Nance, O. Balci. 2002. Issues in enhancing model reuse, *International Conference on Grand Challenges for Modeling and Simulation*.

Tolk, A. 2004. Metamodels and mappings - ending the interoperability war. 04F-SIW-105, *Fall Simulation Interoperability Workshop*, Orlando, Florida.

Wang, W. P., Q. Li, Y. F. Zhu, and F. Yang. 2003. *Flexible Simulation Theory and Applications*. Changsha: National University of Defense Technology Press.

Williams, A. D., J. St. Pierre, and K. Nergaard. 1999. ROSE SIMSAT Harmonisation, *SESP'99*.

Xia, Y. and G. Martin. 2003. Rigorous EBNF-based definition for a graphic modeling language. In *Proceedings of the Tenth Asia-Pacific Software Engineering Conference (APSEC'03)*.

Yilmaz, L. and T. I. Oren. 2006. Prospective issues in simulation model composability: basic concepts to advance theory, methodology, and technology. *The MSIAC's M&S Journal Online*, 2:1-7.

Zeigler, B. P., T. G. Kim and H. Praehofer. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. 2nd ed., Academic Press.

AUTHOR BIOGRAPHIES

YONGLIN LEI is a Ph.D. student in the School of Information System and Management at National University of Defense Technology. His research interest lies in the area of composable simulation and exploratory analysis. His e-mail address is <samuelyonglin@gmail.com>.

LILI SONG is a Ph.D. student in the School of Information System and Management at National University of Defense Technology. Her research interest lies in the area of simulation semantic composability and ontology. Her e-mail address is <sllwendy@gmail.com>.

WEIPING WANG is a professor at National University of Defense Technology of China. He got his Ph.D. in 1997 from National University of Defense Technology. His research interests are system of systems engineering, simulation composability and simulation-based evaluation. His e-mail address is <wangwp@nudt.edu.cn>.

CAIYUN JIANG is a Ph.D. student in the School of Information System and Management at National University of Defense Technology. Her research interest lies in the area of simulation composability and concept modeling. Her e-mail address is <jcy2006@126.com>.