Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

DEVSMO: an ontology for DEVS model with complex mathematical models

Yunping Hu
Jun Xiao
Hao Zhao
Gang Rong

Institute of Cyber-Systems and Control
State Key Laboratory of Industrial Control Technology, Zhejiang University
Hangzhou, Zhejiang 310027, P.R.C

ABSTRACT

There are numerous modeling and simulation environments based on the DEVS formalism. Due to the incompatible modeling grammars, it has been a challenge to reuse DEVS models in different modeling tools. Existing XML-based model representations lack general expressions of the model behavior and only support one type of DEVS. In this paper, a modeling ontology named DEVSMO is proposed. DEVSMO uses structured programming theory to express the programming logic and uses MathML to express the mathematical models in the model behavior. Structured programming theory and MathML provide a set of standard terminologies, so the generality of DEVSMO is improved. Furthermore, DEVSMO can express various DEVS formalisms and has good reusability for the extended applications. Three cases are developed to test DEVSMO in the usability of expressing the model structure and the model behavior and in the reusability for further extension.

1 INTRODUCTION

DEVS (Discrete Event Systems Specification) is a universal formalism for the modeling and analysis of discrete event systems. There are various DEVS-based simulators, such as CD++ (Wainer 2002), DEVS-Java (Sarjoughian and Zeigler 1998), DEVS/C++ (Zeigler et al. 1996), etc. Though different DEVS-based simulators support the same DEVS formalism, they have different modeling grammars. Thus a model developed in one simulator is difficult to use in others.

For the DEVS model reuse, many researchers have used XML (Extensible Markup Language) to represent DEVS models, such as XLSC (Meseth et al. 2009), DEVS-XML (Martín et al. 2007), DEVS Meta Language (DEVSML) (Janoušek, Polášek, and Slavíček 2006) and DEVS Modeling Language (DEVSML) (Mittal et al. 2007,Mittal and Douglass 2012). XML is a markup language for documents containing structured information (Bray et al. 1997). XML-based DEVS model representations are independent of specific modeling languages and the models represented by them can be shared in different simulators. However, XML is difficult to represent the system behavior of DEVS models because the programming logic and the mathematical models in the model behavior need complex XML markups. DEVS-XML only expresses the if-else logic through the XML elements with a Condition_ prefix and expresses the mathematical models through the name-value element attributes. XLSC provides a set of statements including expressions and commands to express the logic. The expressions provide some operation elements to build a mathematical model while the commands provide some logic elements to implement a programming structure. Both DEVS Meta Language and DEVS Modeling Language use parts of JavaML to express the model behavior. The existing model representations do not use a standard set of markups to express the programming logic and the mathematical models, so they are incompatible with each other. In addition, DEVS-XML refers to classic DEVS, while others refer to parallel DEVS. Different DEVS types have different model formalisms, so the existing model representations lack a unified format which is compatible with various DEVS types.

Compared with XML, OWL (Ontology Web Language) can express the sematic information. Recently, OWL has been widely used to represent simulation models for better model discovery, interoperability, integration and reuse (Silver et al. 2007, Turnitsa et al. 2010). PIMODELS (Lacy 2006) expresses discrete event simulation models in the aspect of process interaction world view. DeMO (Silver et al. 2011) expresses discrete event models in the aspects of state-oriented, time-oriented, activity-oriented and process-oriented. COSMO (Teo and Szabo 2008) is a component-oriented ontology, which is used to describe the components and compositions of a simulation model. DEVS Ontology (Peng et al. 2010) provides a black box description of DEVS models for the model maintenance and reuse. However, the first three model ontologies contain fewer terminologies corresponding to the DEVS formalism and DEVS Ontology lacks the description of the model behavior, so they are difficult to support the reuse of DEVS models.

In this paper, we propose a modeling ontology named DEVS Math Ontology (DEVSMO) to support the representation of various DEVS models. DEVSMO contains three sub ontologies including DEVS model ontology, model structure ontology and model behavior ontology. For improving the expression of the model behavior , we use structured programming to express the programming logic and use MathML to express the mathematical models. Structured programming provides clear semantics for a general programming logic and is independent of specific programming languages such as C++ and Java. MathML provides a standard set of XML markups to describe mathematical notions, and it is a recommendation of the W3C math working group. Though DEVSMO in this version only implement the representations of classic DEVS and parallel DEVS, it has good reusability to support the extension of representing the other kinds of DEVS models.

The rest of the paper is organized as follows. The background of developing DEVSMO is introduced in section 2. DEVSMO is presented in section 3 in detail. A software framework for the application of DEVSMO is provided in section 4. We then present three cases to test the usability and reusability of DEVSMO. Concluding remark and future work for DEVSMO are given in Section 6.

2 BACKGROUNDS

2.1 DEVS

A system modeled based on DEVS is composed of atomic and coupled components (Zeigler et al. 2000). In the following, classic DEVS is described in the mathematical formulation:
$Atomic\ DEVS = \{X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta\}$
Where X is the set of input
Y is the set of output
S is a set of states
$\delta_{int} : S \rightarrow S$ is the internal transition function
$\delta_{ext} : Q \times X \rightarrow S$ is the external transition function where $Q = \{(s,e)|s \in S, 0 \le e \le ta(s)\}$
$\lambda : S \rightarrow Y$ is the output function
$ta : S \rightarrow R^{+}_{0,\infty}$ is the set positive reals with 0 and ∞
$Coupled\ DEVS = \{X, Y, D, \{M_d|d \in D\}, EIC, EOC, IC, S\}$
Where $X = \{(p,v)|p \in IPorts, v \in X_p\}$ is the set of input ports and values
$Y = \{(p,v)|p \in OPorts, v \in Y_p\}$ is the set of output ports and values
D is the set of the component names
$M_d = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta)$ is a DEVS with $X_d = \{(p,v)|p \in IPorts_d, v \in X_p\}$
$Y_d = \{(p,v)|p \in OPorts_d, v \in Y_p\}$
$EIC \subseteq \{((N, ip_N), (d, ip_d))|ip_N \in IPorts, d \in D, ip_d \in IPorts_d\}$ is the external input coupling connect external inputs to component inputs

$EOC \subseteq \{((d, op_d), (N, op_N)) | op_N \in OPorts, d \in D, op_d \in OPorts_d\}$ is the external output coupling connect component outputs to external outputs

$IC \subseteq \{((a, op_a), (b, ip_b)) | a, b \in D, op_a \in OPorts_a, ip_d \in IPorts_b\}$ is the external output coupling connect component outputs to external outputs

$Select : 2^D - \{\} - D$, the tie-breaking function

Compared with classic DEVS, parallel DEVS adds the confluent transition function to atomic model and eliminated *Select* from coupled model. The confluent transition function can decide the next state in cases of collision between external and internal events, so it is not necessary to use *Select* to make a choice among imminent components. Classic DEVS and parallel DEVS are basic DEVS formalisms. There are some extensions of DEVS including dynamic DEVS (Barros 1995), symbolic DEVS (Chi 1997), real time DEVS (Cho and Kim 1998) and fuzzy DEVS (Kwon et al. 1996).

2.2 OWL

OWL is a language for publishing and sharing ontologies in the web and is part of the growing stack of W3C recommendations (Smith et al. 2009). OWL has three increasingly expressive sublanguages, called OWL Lite, OWL DL and OWL Full. Different types of OWL satisfy different application requirements. OWL Lite supports a classification hierarchy and simple constraints. OWL DL, based on description logic theory, supports strong expressiveness and has computational completeness and decidability for reasoning. OWL Full supports maximum expressiveness but has no computational guarantees for reasoning. DEVSMO is developed in OWL DL for the need of simple reasoning in the translation of the DEVSMO instances.

2.3 MathML

MathML (Mathematical Markup Language) is an XML application for describing mathematical notation and capturing its both structure and content (http://w3c.org/Math). MathML can express arbitrarily complex mathematical models including differential equations and mathematical model sets besides algebraic model (Carlisle et al. 2001). MathML has two kinds of markup sets including content markup and presentation markup. The former focuses on exposing the semantics of functions and the later focuses on describing an equation similarly to the way one read it. In DEVSMO, we choose content MathML to express the mathematical models because there are some correspondences between the content markups and the existing XML-based DEVS model representations.

2.4 Structured Programming

Structured programming is a classical result of program schematology (Böhm and Jacopini 1966), which states that any deterministic flowchart program is equivalent to a while program. Deterministic while programs are formed inductively from sequential composition (p; q), conditional tests (if b then p else q), and while loops (while b do p), where b is a test and p & q are programs (Kozen and Tseng 2008). Though C++ and Java are object-oriented programming (OOP) languages, the internal structure of class methods in OOP still follows the principle of structured programming.

In the executable DEVS modeling languages, many functions and class methods have been defined in advance, such as the *holdIn(state, time)* function and the *externalFunction(const ExternalMessage &msg)* method in the CD++ modeling language. The individualization process of class methods for a DEVS model is to call various functions according to the principle of structured programming.

3 DEVSMO

The overview of DEVSMO is illustrated in Figure 1. DEVSMO is composed of three sub ontologies: DEVS model ontology, model structure ontology and model behavior ontology. DEVS model ontology describes the classification of DEVS formalisms. Model structure ontology provides the terminologies to express the model structure, which consists of atomic model structure and coupled model structure. Model behavior ontology is used to express the behavior parts of DEVS models, including four parts of function, action, math model and control structure.

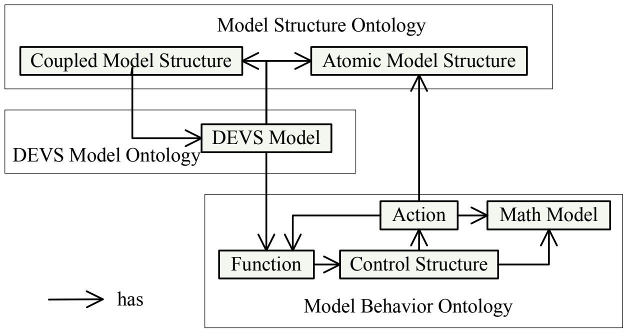


Figure 1: Overview of DEVSMO

In the following, the three sub ontologies of DEVSMO are described in detail.

3.1 DEVS model ontology

The UML diagram of DEVS model ontology is illustrated in Figure 2. DEVS *model* is divided into *atomic model* and *coupled model*. Each model refers to a *DEVS type* which may be *basic DEVS (classic DEVS and parallel DEVS)* or *extended DEVS (dynamic DEVS, symbolic DEVS, real time DEVS and fuzzy DEVS)*. Some concepts in model structure ontology and model behavior ontology, like *inputs, outputs* and *output function*, are used to express the composition of DEVS models.

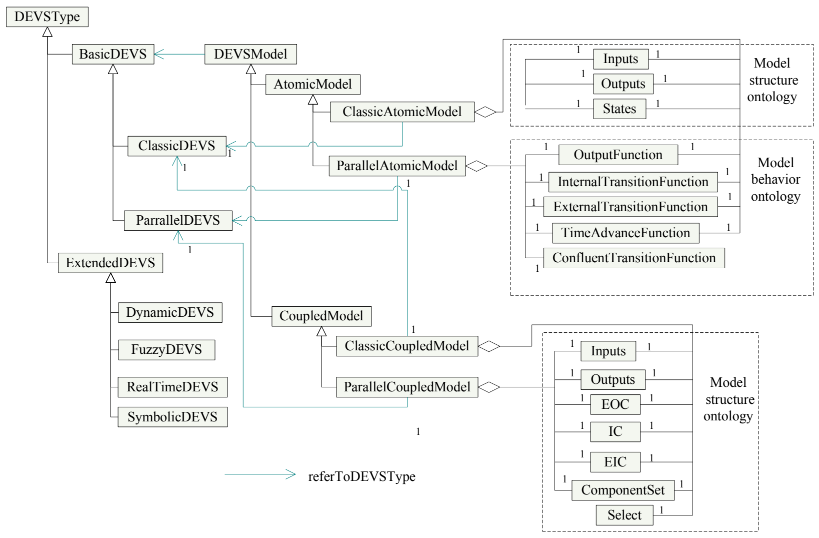


Figure 2: The UML diagram of DEVS model ontology

3.2 Model structure ontology

In DEVSMO, the model structure is described by model structure ontology which does not contain any concept about the programming logic and the mathematical models. The UML diagram of model structure ontology is illustrated in Figure 3.

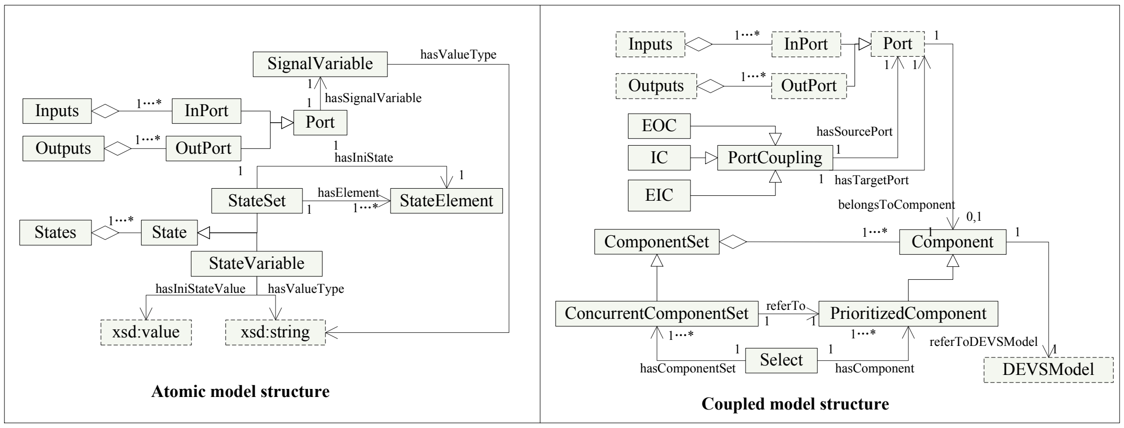


Figure 3: The UML diagram of model structure ontology

The *inputs* and *outputs* in atomic model are both composed of *port*. Each *port* corresponds to a *signal variable* which has a data property named *hasValueType*. *State* is divided into two kinds, one which is *state set* consisted of *state elements*, and the other which is *state variable* with the data properties of *initial value* and *value type*.

The coupled model structure part imports some concepts from the atomic model structure part, presented in the dashed boxes. The classes of *EOC, EIC* and *IC* are sub classes of *port coupling* which is composed of a source *port* and a target *port*. Each *port* belongs to a *component* which refers to a *DEVS model*. For the expression of concept *select*, we refer to its description in DEVS-XML. Each *select* has more than one two-tuples of a *concurrent component set* and a *prioritized component*.

3.3 Model behavior ontology

The mode behavior is the most complex part in the DEVS model representation. The four parts of function, action, control structure and math model in model behavior ontology are illustrated in Figure 4. The concepts in the dashed boxes are imported from the other parts. The concepts of *action* and *control structure* come from extended structured modeling theory (Lenard 1992, Lenard 1993) which extends structured modeling to express DEVS models.

In the following, we describe the four parts of model behavior ontology separately in detail:

- Function part. Functions are a part of an atomic model. The *function* concept is the super concept of *time advance function, output function* and *transition function* and each *function* has a *control structure* to express the programming logic. The *transition function* has three types: internal, external and confluent.
- Control structure part. We choose structured programming theory to express the control structure of the programming logic. There are three kinds of control structures: loop, selection and sequence. *Loop* is composed of *While* and *Do*, *Selection* is composed of *If*, *ThenDo* and *ElseDo* and *Sequence* is composed of *Do* and *ThenDo*. Each *Do* corresponds to a *control structure* or some *actions*. The concepts of *If* and *While* have a super concept named *condition*.
- Action part. The concept of *action* means an atomic statement in the programming logic. Each *action* belongs to a function in a DEVS model. For example, the *send* action belongs to *output function* and the *execution* action belongs to *confluent transition function*. Some actions have relationships with the concepts in model structure ontology or the function part. For example, each *signal value update* action has a *signal value* and each *execution* action may execute an *internal transition function* or an *external transition function*.

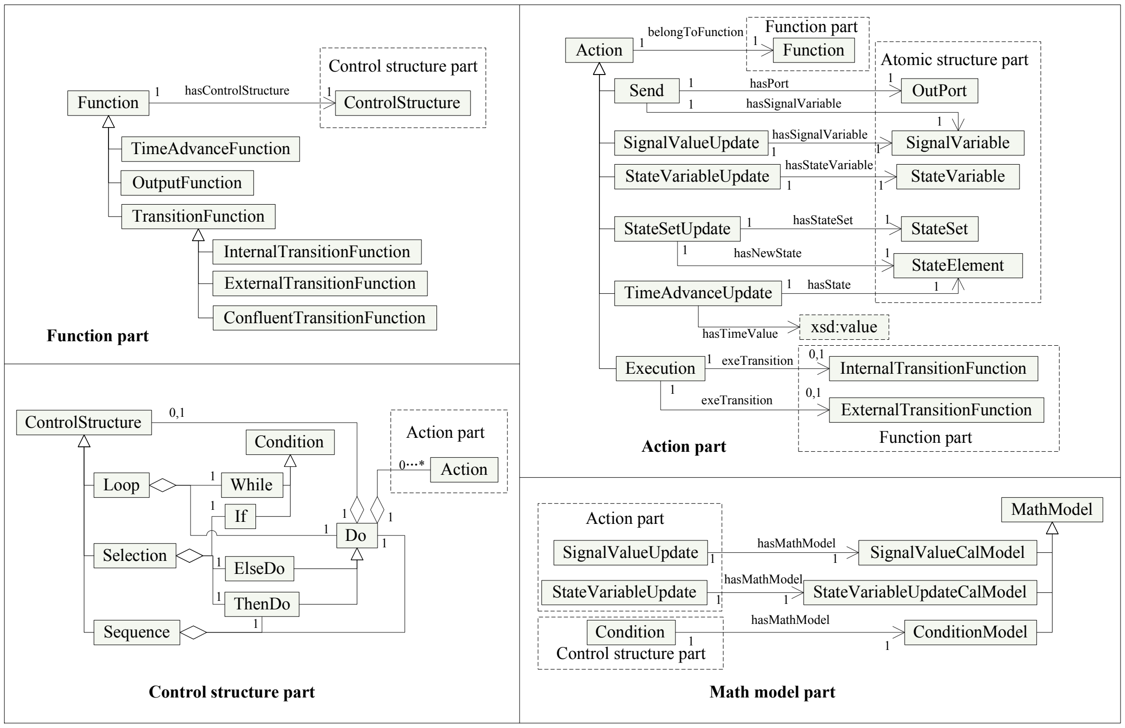


Figure 4: The UML diagram of model behavior ontology

- Math model part. The math model is a part of control structures or actions in the model behavior. The math model can express numerical operations like *signal value* calculation in the action and express logical calculations like *condition* in the control structure.

4 THE FRAMEWORK OF DEVSMO APPLICATION

The models represented by DEVSMO cannot be executed directly. The model instances in DEVSMO can be translated into the formats of executable modeling languages or XML-based model representations. The framework of translation is illustrated in Figure 5. The models stored in *.owl files can be read and written through the Jena component (McBride 2002) which provides the APIs for the ontology operation and the SPARQL (http://www.w3.org/TR/rdf-sparql-query) query. The MathML codes embedded in *.owl files can be operated through MathML DOM which extends the Core API of DOM to describe the objects and methods specific to MathML elements. For the translation to executable languages like CD++ or DEVS-Java, a template specific to the target language is needed. The template contains some constant contexts in different models, such as the reference library files in *.h files of CD++ language. For the translation to XML-based model representations like DEVS-XML or XLSC, the DOM interface provides class methods to interact with the objects in XML documents.

5 CASES

5.1 The translation of the model structure represented by DEVSMO

In this case, an atomic DEVS model named *model* has an input port named *in_1*. The model instance represented by DEVSMO is illustrated in Figure 6. The *AtomicModel* class has a *model* individual. The *hasInputs* property of *model* has a value of *model_inputs*. The *hasPort* property of *model_inputs* has a value of *in_1*. The instances of *model* and *in_1* can be translated separately as the values of ATOMIC_MODEL_NAME and PORT_NAME elements in DEVS-XML and can also be written into the *.h file of CD++ as illustrated in

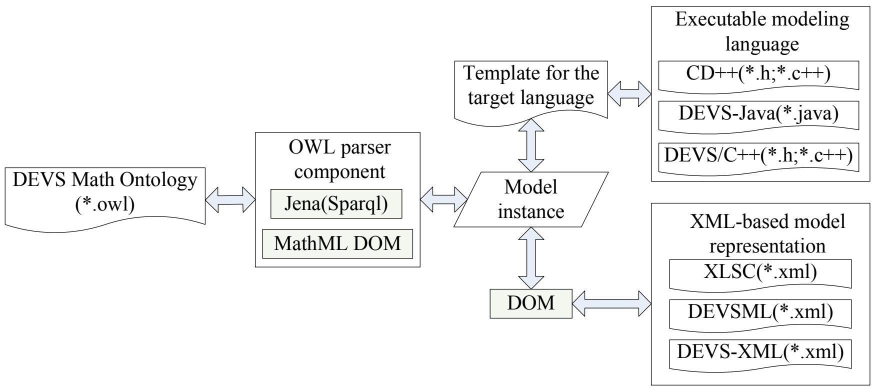
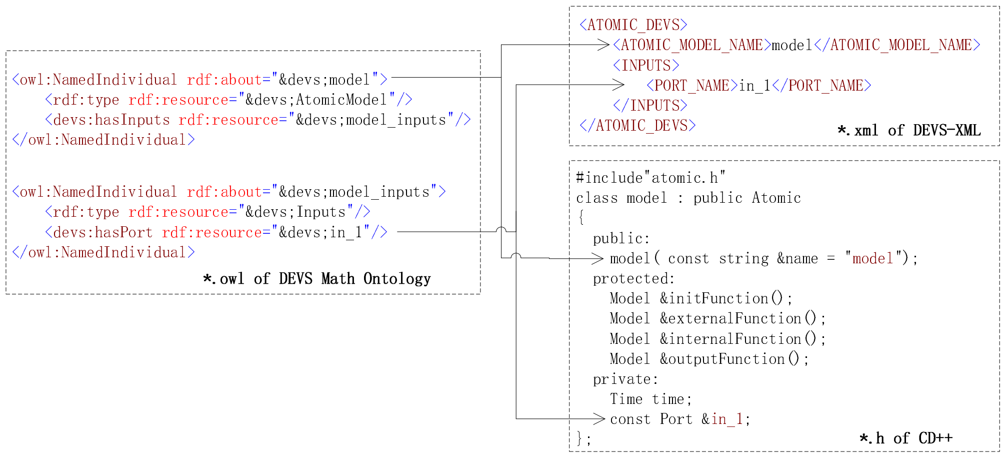


Figure 5: The prototype of model translation implementation

Figure 6. The other codes in the *.h file of CD++ are the constant contexts in the translation template. This case proves that the model structure represented by DEVSMO can be translated into executable modeling languages and XML-based model representations.



```
<owl:NamedIndividual rdf:about="&devs;model">
    <rdf:type rdf:resource="&devs;AtomicModel"/>
    <devs:hasInputs rdf:resource="&devs;model_inputs"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="&devs;model_inputs">
    <rdf:type rdf:resource="&devs;Inputs"/>
    <devs:hasPort rdf:resource="&devs;in_1"/>
</owl:NamedIndividual>
                              *.owl of DEVS Math Ontology
```

```
<ATOMIC_DEVS>
    <ATOMIC_MODEL_NAME>model</ATOMIC_MODEL_NAME>
    <INPUTS>
        <PORT_NAME>in_1</PORT_NAME>
    </INPUTS>
</ATOMIC_DEVS>                        *.xml of DEVS-XML
```

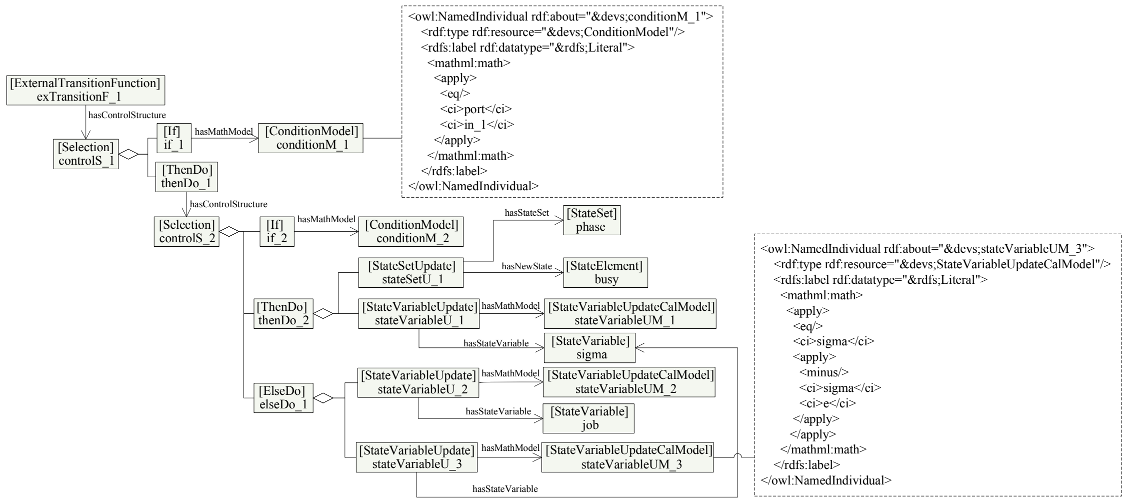
```
#include"atomic.h"
class model : public Atomic
{
  public:
    model( const string &name = "model");
  protected:
    Model &initFunction();
    Model &externalFunction();
    Model &internalFunction();
    Model &outputFunction();
  private:
    Time time;
    const Port &in_1;
};                                  *.h of CD++
```

Figure 6: The translation of DEVS model structure

5.2 The model behavior represented by DEVSMO

In this case, the external transition function of *Processor* (Zeigler et al. 2000) model represented by DEVSMO is illustrated in Figure 7. The *exTransitionF_1* instance is an individual of *ExternalTransitionFunction* class. The *exTransitionF_1* instance has a selection control structure named *controlS_1* which is composed of *if_1* and *thenDo_1*. The *if_1* instance has a math model named *condition_1* which has a *Literal* represented by MathML. The *thenDO_1* instance corresponds to a selection control structure named *controlS_2* which is composed of *if_2*, *thenDo_2* and *elseDo_1*. The *thenDo_2* instance is composed of *stateSetU_1* and *stateVariableU_1*. The *elseDo_1* instance is composed of *stateVariableU_2* and *stateVariableU_3* . The two object properties of *stateSetU_1* instance have values, which means the *phase* state set has a new state *busy*. Each individual of *StateVariableUpdateCalModel* class has a *Literal* represented by MathML.

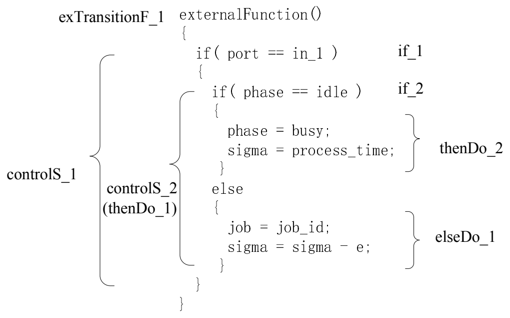
The correspondences between DEVSMO instances and the programming logic in the external transition function are illustrated in Figure 8.This case demonstrates that the programming logic and the mathematical models in the model behavior can be represented by DEVSMO completely.


[ExternalTransitionFunction] exTransitionF_1
hasControlStructure
[Selection] controlS_1
[If] if_1
hasMathModel
[ConditionModel] conditionM_1
[ThenDo] thenDo_1
hasControlStructure
[Selection] controlS_2
[If] if_2
hasMathModel
[ConditionModel] conditionM_2
hasStateSet
[StateSet] phase
[StateSetUpdate] stateSetU_1
hasNewState
[StateElement] busy
[ThenDo] thenDo_2
[StateVariableUpdate] stateVariableU_1
hasMathModel
[StateVariableUpdateCalModel] stateVariableUM_1
hasStateVariable
[StateVariable] sigma
[ElseDo] elseDo_1
[StateVariableUpdate] stateVariableU_2
hasMathModel
[StateVariableUpdateCalModel] stateVariableUM_2
hasStateVariable
[StateVariable] job
[StateVariableUpdate] stateVariableU_3
hasMathModel
[StateVariableUpdateCalModel] stateVariableUM_3
hasStateVariable

```
<owl:NamedIndividual rdf:about="&devs;conditionM_1">
  <rdf:type rdf:resource="&devs;ConditionModel"/>
  <rdfs:label rdf:datatype="&rdfs;Literal">
    <mathml:math>
      <apply>
        <eq/>
        <ci>port</ci>
        <ci>in_1</ci>
      </apply>
    </mathml:math>
  </rdfs:label>
</owl:NamedIndividual>
```

```
<owl:NamedIndividual rdf:about="&devs;stateVariableUM_3">
  <rdf:type rdf:resource="&devs;StateVariableUpdateCalModel"/>
  <rdfs:label rdf:datatype="&rdfs;Literal">
    <mathml:math>
      <apply>
        <eq/>
        <ci>sigma</ci>
        <apply>
          <minus/>
          <ci>sigma</ci>
          <ci>e</ci>
        </apply>
      </apply>
    </mathml:math>
  </rdfs:label>
</owl:NamedIndividual>
```

Figure 7: The external transition function of *Processor* model represented by DEVSMO



```
externalFunction()
{
  if( port == in_1 )
  {
    if( phase == idle )
    {
      phase = busy;
      sigma = process_time;
    }
    else
    {
      job = job_id;
      sigma = sigma - e;
    }
  }
}
```

Figure 8: The external transition function of *Processor* model

5.3 Reusability of DEVSMO

For the development of ontology, reusability is a key evaluating indicator. Ontology reusability is defined as the adaptation capability of an ontology to different application contexts, including those contexts that were not considered at the time of the creation of the ontology (Russ et al. 1999, Pâslaru-Bonta% cs 2007).

In this case, we extend DEVSMO to express fuzzy DEVS which applies fuzzy set theory to the set and function defined in classic DEVS, as illustrated in Figure 9. The concepts in dark blue are extended from DEVSMO. *FuzzyAtomicModel* is the sub class of *AtomicModel*. The functions in fuzzy DEVS have possibilistic features. The actions like *FuzzyStateVaribleUpdate* in model behavior ontology have a data property named *hasPossibilityRate*. Through this case, it is concluded that DEVSMO has good reusability and can be extended to express the other DEVS formalisms.

6 CONCLUSIONS AND FUTURE

The existing XML-based DEVS model representations lack general expressions of the model behavior and only support one kind of DEVS formalism. This paper proposes a DEVS modeling ontology named DEVSMO which improves the generality of the model behavior representation and can support various DEVS types. The programming logic of the model behavior is expressed based on structured programming theory which provides a conceptual description of various programming logics. The mathematical model of the model behavior is expressed by MathML which provides a set of standard mathematical markups. Furthermore, DEVSMO is divided into three sub ontologies and has good reusability for extension.

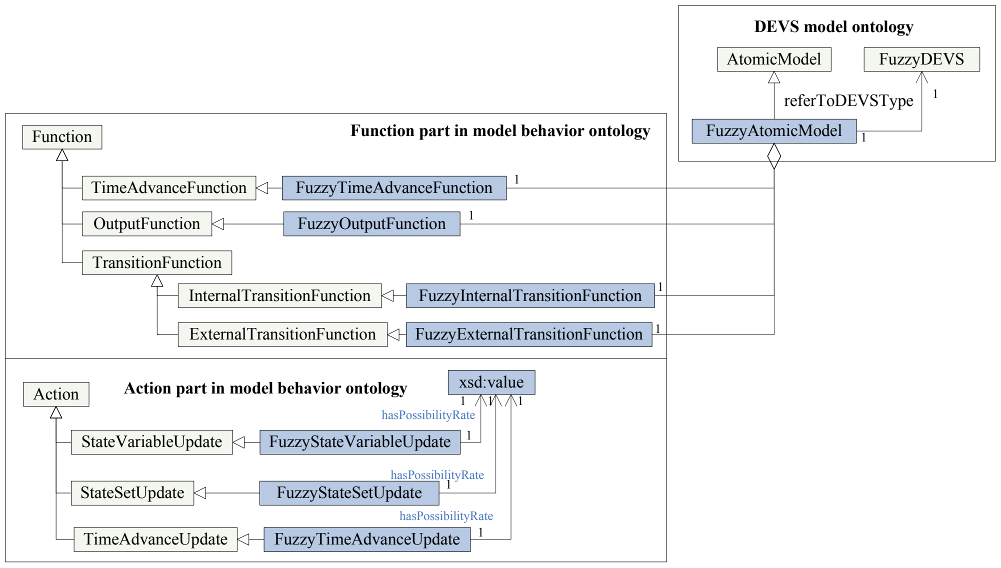


Figure 9: The extension of DEVSMO for fuzzy DEVS

Presently, though DEVSMO only implements the representations of classic DEVS and parallel DEVS, it can be extended to support the other DEVS formalisms. In the future, we can extend DEVSMO based on extended structured modeling theory to express optimization models for the multi-paradigm model management.

ACKNOWLEDGMENTS

In preparing this paper, the author has benefited from suggestions of Associate Professor Xiaolin Hu in Georgia State University. Furthermore, the financial supports from the National High Technology R&D programs of China (2012BAE05B03 & 2012BAE05B05) are also gratefully acknowledged.

REFERENCES

Barros, F. J. 1995. “Dynamic structure discrete event system specification: a new formalism for dynamic structure modeling and simulation”. In *Proceedings of the 27th conference on Winter simulation*, 781–785. IEEE Computer Society.

Böhm, C., and G. Jacopini. 1966. “Flow diagrams, turing machines and languages with only two formation rules”. *Communications of the ACM* 9 (5): 366–371.

Bray, T., J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. 1997. “Extensible markup language (XML)”. *World Wide Web Journal* 2 (4): 27–66.

Carlisle, D., P. Ion, R. Miner, N. Poppelier et al. 2001. “Mathematical markup language (mathml) version 2.0”. *W3C recommendation* 21.

Chi, S.-D. 1997. “Model-based reasoning methodology using the symbolic DEVS simulation”. *TRANSACTIONS of the Society for Computer Simulation International* 14 (3): 141–151.

Cho, S. M., and T. G. Kim. 1998. “Real-time DEVS simulation: Concurrent, time-selective execution of combined RT-DEVS model and interactive environment”. In *Proceeding of 1998 Summer Simulation Conference, Reno, Nevada*.

Janoušek, V., P. Polášek, and P. Slavíček. 2006. “Towards DEVS Meta Language”. *ISC 2006 Proceedings*:69–73.

Kozen, D., and W.-L. Tseng. 2008. “The BhmJacopini Theorem Is False, Propositionally”. In *Mathematics of Program Construction*, edited by P. Audebaud and C. Paulin-Mohring, Volume 5133 of *Lecture Notes in Computer Science*, 177–192. Springer Berlin Heidelberg.

Kwon, Y. W., H. Park, S. Jung, and T. G. Kim. 1996. “Fuzzy-DEVS formalism: concepts, realization and applications”. In *Proceedings Of The 1996 Conference On AI, Simulation and Planning In High Autonomy Systems*, 227–234. Citeseer.

Lacy, L. W. 2006. *Interchanging Discrete event simulation Process Interaction Models using the Web Ontology Language-OWL*. Ph. D. thesis, University of Central Florida Orlando, Florida.

Lenard, M. L. 1992. “Extending the structured modeling framework for discrete-event simulation”. In *System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on*, Volume 3, 494–503. IEEE.

Lenard, M. L. 1993. “A prototype implementation of a model management system for discrete-event simulation models”. In *Proceedings of the 1993 Winter Simulation Conference*, 560–568. ACM.

Martín, J., S. Mittal, M. López-Peña, and J. De la Cruz. 2007. “A W3C XML schema for DEVS scenarios”. In *Proceedings of the 2007 spring simulation multiconference-Volume 2*, 279–286. Society for Computer Simulation International.

McBride, B. 2002. “Jena: A semantic web toolkit”. *Internet Computing, IEEE* 6 (6): 55–59.

Meseth, N., P. Kirchhof, and T. Witte. 2009. “XML-based DEVS modeling and interpretation”. In *Proceedings of the 2009 Spring Simulation Multiconference on ZZZ*, 152. Society for Computer Simulation International.

Mittal, S., and S. A. Douglass. 2012. “DEVSML 2.0: The language and the stack”. In *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation-DEVS Integrative M&S Symposium*, 17. Society for Computer Simulation International.

Mittal, S., J. L. Risco-Martín, and B. P. Zeigler. 2007. “DEVSML: automating DEVS execution over SOA towards transparent simulators”. In *Proceedings of the 2007 spring simulation multiconference-Volume 2*, 287–295. Society for Computer Simulation International.

Pâslaru-Bontaş, E. 2007. *A contextual approach to ontology reuse: methodology, methods and tools for the semantic web*. Ph. D. thesis.

Peng, Y., J. Huang, and K. Huang. 2010. “Ontology-based DEVS model maintenance and reuse”. In *Computer Application and System Modeling (ICCASM), 2010 International Conference on*, Volume 11, V11–116. IEEE.

Russ, T., A. Valente, R. MacGregor, and W. Swartout. 1999. “Practical experiences in trading off ontology usability and reusability”. In *Proc. of the Knowledge Acquisition Workshop (KAW99)*, Volume 86.

Sarjoughian, H. S., and B. Zeigler. 1998. “DEVSJAVA: Basis for a DEVS-based collaborative M&S environment”. *SIMULATION SERIES* 30:29–36.

Silver, G. A., O.-H. Hassan, and J. A. Miller. 2007. “From domain ontologies to modeling ontologies to executable simulation models”. In *Proceedings of the 2007 Winter Simulation Conference*, 1108–1117. IEEE.

Silver, G. A., J. A. Miller, M. Hybinette, G. Baramidze, and W. S. York. 2011. “DeMO: an ontology for discrete-event modeling and simulation”. *Simulation* 87 (9): 747–773.

Smith, M., C. Welty, and D. L. McGuinness. 2009. “OWL web ontology language guide. W3C Recommendation (2004)”. *URL http://www. w3. org/TR/owl-guide*.

Teo, Y. M., and C. Szabo. 2008. “CoDES: An integrated approach to composable modeling and simulation”. In *Simulation Symposium, 2008. ANSS 2008. 41st Annual*, 103–110. IEEE.

Turnitsa, C., J. J. Padilla, and A. Tolk. 2010. “Ontology for modeling and simulation”. In *Proceedings of the 2010 Winter Simulation Conference*, 643–651. IEEE.

Wainer, G. 2002. “CD++: a toolkit to develop DEVS models”. *Software: Practice and Experience* 32 (13): 1261–1306.

Zeigler, B. P., Y. Moon, D. Kim, and J. G. Kim. 1996. “DEVS-C++: A high performance modelling and simulation environment”. In *System Sciences, 1996., Proceedings of the Twenty-Ninth Hawaii International Conference on,*, Volume 1, 350–359. IEEE.

Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. Academic press.

AUTHOR BIOGRAPHIES

Yunping Hu is a doctor candidate of Control Science and Engineering at Zhejiang University in Hangzhou, China. He holds a M.S. in Control Science and Engineering from Nanjing University of Technology in Nanjing, China. His research interests include modeling & simulation, model management and decision support system. His email address is yphu@iipc.zju.edu.cn.

Jun Xiao is a master candidate of Control Science and Engineering at Zhejiang University in Hangzhou, China. He earned his Bachelor’s degree in Automation from Wuhan University in Wuhan, China. His research interests include DEVS modeling & simulation and mathematical knowledge management. His email address is jxiao@iipc.zju.edu.cn.

Hao Zhao is a doctor candidate of Control Science and Engineering at Zhejiang University in Hangzhou, China. He earned his Bachelor’s degree in Automation from Zhejiang University in Hangzhou, China. His research interests include DDDAS, optimization via simulation and applications of simulation methods to production scheduling. His email address is hzhao@iipc.zju.edu.cn.

Gang Rong is Professor of Control Science and Engineering at Zhejiang University in Hangzhou, China. His research interests cover modeling, simulation & optimization, data-ming, data visualization and enterprise-control system integration in process industries. He holds a Ph.D. in Control Science and Engineering from Zhejiang University in Hangzhou, China. His email address is grong@iipc.zju.edu.cn and his web page is http://mypage.zju.edu.cn/en/rglab.

.