Journal of System Simulation

Volume 27 | Issue 12 Article 2

7-30-2020

Ontology Based DEVS Modeling Method

Rong Gang
Institute of Cyber-Systems and Control, State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou 310027, China;

Xiao Jun
Institute of Cyber-Systems and Control, State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou 310027, China;

Yunping Hu
Institute of Cyber-Systems and Control, State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou 310027, China;

Yiping Feng
Institute of Cyber-Systems and Control, State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou 310027, China;

Ontology Based DEVS Modeling Method

Abstract

Abstract: In the field of modeling and simulation, it's difficult to reuse the models because the models and simulation system are coupled seriously. The Ontology-driven modeling theory is a simulation modeling methodology for model reuse. A new DEVS modeling method was proposed based on this theory. *The DEVS modeling ontology DEVSMO was built and the DEVS model could be described as the instance of the DEVSMO*. This form of DEVS model was independent of the DEVS simulation environment, which improved the model reusability. *A tool was developed to translate the DEVSMO instance to DEVS executable models* to verify the effectiveness of the method.

Keywords

ontology-driven modeling, DEVS, ontology, modeling ontology

Recommended Citation

Rong Gang, Xiao Jun, Hu Yunping, Feng Yiping. Ontology Based DEVS Modeling Method[J]. Journal of System Simulation, 2015, 27(12): 2878-2890.

第 27 卷第 12 期 系统仿真学报© Vol. 27 No. 12
2015 年 12 月 Journal of System Simulation Dec., 2015

一种基于本体的 DEVS 建模方法

荣冈，肖俊，胡云苹，冯毅萍
（浙江大学工业控制技术国家重点实验室智能系统与控制研究所，杭州 310027）

摘要：在建模和仿真领域，很多模型因为与仿真系统结合过于紧密，导致模型难以重用。本体驱动建模理论是一种为模型重用而生的仿真建模方法论，基于该理论提出了一种新的 DEVS(Discrete Event System Specification)建模方法。*该方法建立了 DEVS 建模本体 DEVSMO(DEVS Modeling Ontology)，将具体的 DEVS 模型描述为 DEVSMO 的实例，这种形式的 DEVS 模型脱离的具体的仿真环境，在一定程度上提高了模型的复用性。利用 DEVS 规范和具体仿真环境下的可执行模型的对应关系，开发了 DEVSMO 实例映射工具 DEVSMO_Mapper，*将这种 DEVSMO 实例映射为 DEVS 可执行模型，验证了该方法的有效性。

关键词：本体驱动建模理论；离散事件仿真规范；本体；建模本体
中图分类号：TP391.9 文献标识码：A 文章编号：1004-731X (2015) 12-2878-13

Ontology Based DEVS Modeling Method

Rong Gang, Xiao Jun, Hu Yunping, Feng Yiping
(Institute of Cyber-Systems and Control, State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou 310027, China)

Abstract: In the field of modeling and simulation, it's difficult to reuse the models because the models and simulation system are coupled seriously. The Ontology-driven modeling theory is a simulation modeling methodology for model reuse. A new DEVS modeling method was proposed based on this theory. *The DEVS modeling ontology DEVSMO was built and the DEVS model could be described as the instance of the DEVSMO*. This form of DEVS model was independent of the DEVS simulation environment, which improved the model reusability. *A tool was developed to translate the DEVSMO instance to DEVS executable models* to verify the effectiveness of the method.

Keywords: ontology-driven modeling; DEVS; ontology; modeling ontology

引言

多年来，建模和仿真(M&S)在系统设计、原型验证、现场优化等领域有非常广泛、成熟的应用。工业界以及学术界建立了大量的模型和仿真系统。随着模型和仿真系统数量的增长，建模成本也逐步受到关注。在模型开发中，大量的模型和仿真系统结合得非常紧密，导致模型重用性低，让建模成本迅速增长。开发这种低重用性的仿真模型，不仅造成大量的人力、物力浪费，也在一定程度上限制了模型及仿真引擎的发展，严重影响了整个仿真系统的可维护性及可扩展性。

Zeigler 在 1976 年提出的离散事件系统规范(Discrete Event System Specification，DEVS)[1]是一种用于系统建模和仿真的模块化、层次化和形式化的机制。DEVS 从系统理论的角度，定义了一种抽

收稿日期：2014-06-11 修回日期：2014-12-11；
基金项目：国家科技支撑计划(2013AA040701)；
作者简介：荣冈(1963-)，男，陕西西安，博士，教授，研究方向智能工厂；肖俊(1991-)，男，湖北孝感，硕士生，研究方向数学知识管理；胡云苹(1984-)，男，山东济南，博士生，研究方向为知识管理。

象地表达模型的规范，将实验框架、模型和仿真器分离，在一定程度上提高了模型的复用性。DEVS 只是一种仿真建模规范，存在很多不同的仿真环境的实现，主要有 CD++，DEVS/C++，DEVSJAVA 等。在这些仿真环境的实现的基础上，许多研究又扩展出了 DEVS/HLA，DEVS/CORBA，cell-DEVS 等。这些不同的 DEVS 仿真环境在平台实现语言，建模语言上有所不同，正是由于这些不同，在其中一种 DEVS 仿真环境下的建立的模型并不能在其他的仿真环境中使用。为了解决这种不一致，很多研究利用 XML 技术将 DEVS 模型表达出来，然后映射成不同仿真环境下的可执行仿真模型。这种基于 XML 技术的模型表达有 XFD-DEVS[2]，XLSC[3]，DEVS-XML[4]。这种基于 XML 的技术仅仅是一种建模语言表层的转换，而仿真环境的不同以及建模语言不同所带来的问题，是和仿真平台关联较为紧密的。基于 XML 技术的表达模型并不能很好解决仿真平台深层次上不同所带来的问题。

近年来，本体论因能较好的描述领域知识而被引入到 M&S 领域。本体论使一些和领域知识相关的概念得到更加清晰的分类，并在逻辑上相互联系。本体可以使领域知识可以得到更好的重复利用，让一些领域知识方面的假设更加明确。本体不仅可以让知识在领域专家互相理解，而且可以通过机器来分析处理领域知识，这也为模型的集成、交互和重用提供了一个新的方向。

本体驱动建模理论(Ontology Driven Modeling)[5]是一种为模型重用而生的仿真建模方法论，仿真模型是模型本体的表达的实例，仿真模型在使用时，映射为与仿真环境相对应的可执行模型。这种基于本体的模型表达独立于具体的仿真环境，提高了模型的重用性。本体驱动建模的效率很大程度上取决于如何用本体来表达模型。Giancarlo Guizzardi 等人[6]建立了 DESO 用来评价离散事件仿真语言。Gregory A Silver 等人[7]从离散事件仿真的理论角度，建立了 DeMO 本体，分别从状态、事件、活动、过程这 4 个角度来表达离散事件仿真。Lee W. Lacy[8]从过程交互的观点建立了离散事件仿真本体 PIMODES。这些本体并不是为 DEVS 所设计，故无法直接应用与 DEVS 仿真。

因为本体驱动建模理论致力于使仿真模型脱离于具体仿真环境，故本文将本体驱动建模理论应用到 DEVS 规范上，以解决不同 DEVS 仿真环境下的模型重用问题。本文首先建立了基于 DEVS 规范的建模本体 DEVSMO，然后将该建模本体应用于实际建模过程，并开发了模型映射程序 DEVSMO_Mapper 将建模本体实例映射成可执行模型，并运行可执行仿真模型，验证了本文所提出的基于本体驱动建模理论的 DEVS 仿真建模方法的正确性。

1 背景

1.1 DEVS 仿真规范

DEVS 仿真范式是一种利用抽象数学集合形式来表达的离散事件仿真框架。DEVS 模型有分为 2 种：原子模型(atomic model)和耦合模型(coupled model)。原子模型是不可再进行拆分的最基本的模型，它定义了被建模实体的自治行为。耦合模型是将原子模型和耦合模型的以一种清晰的方式聚合在一起的模型。

原子模型的定义如下[1]：

$$\text{AtomicDEVS} = < IP, OP, X, Y, s_0, S, \delta_{\text{int}}, \delta_{ext}, \lambda, ta > \quad (1)$$

其中：IP，OP 为输入端口和输出端口集合；$X = \{(p,v) \mid p \in IP, v \in X_p\}$ 为输入事件的集合；p 为各输入端口；X_p 是各输入端口的事件集合；$Y = \{(p,v) \mid p \in OP, v \in Y_p\}$ 为输出事件的集合；p 是各输出端口；Y_p 是各输出端口的事件集合；s_0 为系统的初始状态；S为状态序列的集合；δ_{int}：$S \to S$，模型的内部状态转移函数；δ_{ext}：$Q \times X \to S$，外部状态转移函数，其中 $Q = \{(s,e) \mid s \in S, 0 \leqslant e \leqslant ta(s)\}$；$\lambda$：$S \to Y$，输出函数；$ta$：$S \to R^+_{0,\infty}$，时间推进函数。

耦合模型定义如下：

$$\begin{aligned}&\text{coupledDEVS} = \\ &< IP, OP, X, Y, D, \{M_d \mid d \in D\}, \\ &\text{EIC}, \text{EOC}, \text{IC}, \text{Select} >\end{aligned} \tag{2}$$

其中：IP, OP 为外部输入端口和输出端口；$X = \{(p,v) \mid p \in IP, v \in X_p\}$ 为输入事件的集合；p 是各输入端口；X_p 是各输入端口的事件集合；$Y = \{(p,v) \mid p \in OP, v \in Y_p\}$ 为输出事件的集合；p 是各输出端口；Y_p 是各输出端口的事件集合；D 为子模块的集合；M_d 为所有子模型的集合；EIC 为 External Input Coupling，外部输入耦合关系；EOC 为 External Output Coupling，外部输出耦合关系；IC 为 Internal Coupling，子模块的耦合关系；Select 为 Select 冲突解决函数。

1.2 本体及 OWL

本体有很多不同的定义，其中在人工智能以及计算机领域较为精炼的定义是“本体是用于描述概念的明确规范说明”。本体描述了某个知识领域内的概念、这些概念之间的关系以及这些关系的规则，这些术语、关系以及规则一起表达了这个领域之中的所有概念及术语。人和机器之间以及机器之间可以本体为信息载体来交流。因为本体能较好的表达知识，所以本体在人工智能、语义网、软件工程、生物信息学、信息交互等方面有重要的作用。

为了更好的建立、表达和表达本体，已经有很多研究提出了各种建立本体的理论，如一阶逻辑、描述逻辑等。在本体编码语言上也有很多选择 Ontoligua[9]，DAML[10]，LOOM[11]，Cyc[12]以及 OKBC[11]，如其中最常用的有资源描述框架(Resource Description Framework，RDF)[13]和网络本体语言(Web Ontology Language，OWL)[14]。RDF 是一种用于描述 Web 资源的标记语言，常被用来定义资源的类别属性等结构化数据。OWL 是 W3C 组织推荐的一种建立在 RDF 之上的网络本体语言，用于对本体进行语义描述。OWL 与 RDF 有很多相似之处，但是比 RDF 有着更大的词汇表及更强大的语言。

OWL 具有 XML 形式的语法，并且对知识概念有着极强的表达能力，故能较好的描述模型的结构和行为，并脱离具体的建模工具。Protégé 是一个使用 OWL 的可视化跨平台本体编辑器。Protégé 不仅提供了可视化的编辑视图，还建立了插件机制，有很多插件可以增强 protégé 的功能。

M&S 是一个非常综合的研究领域，不仅需要对建模对象有很深刻的理解，而且需要能熟练使用各种建模方法和技巧，其本质上就是两种不同的知识领域的交叉和结合。本体论因其研究的是某个执行领域内的概念以及这些概念之间的联系，故在 M&S 中有广阔的应用前景。除了本文前面所说的本体驱动建模理论，还有研究提出了 POPE[15-16]用于指导制药领域的建模。在 Pradeep Suresh 等人的研究中[17-18]使用 POPE 作为领域本体，将 POPE 本体映射到 OntoMODEL 中，完成了整个建模过程。在张童等人的研究中[19]，使用基于本体的信息模型来解决大规模分布式仿真系统中的信息异构问题。彭春光等人[20]归纳总结了本体在一般模型开发过程中的作用，提出本体能建立一种领域间交流对话的机制，可以有效的促进仿真建模、仿真的自动组合等方面研究的发展。

2 本体驱动的 DEVS 仿真建模方法

在 M&S 领域中，仿真一般分为 2 个过程：首先需要将实际对象抽象成概念模型，并在一定程度上描述这些概念的联系和概念所拥有的一些行为；然后需要将这个抽象的概念模型以一种计算机可以求解的形式表达出来。但正如本文前面所讨论的，大量的模型和仿真系统结合得非常紧密，建模人员在考虑建模的时候需要对仿真软件和仿真的细节需要很好的了解，这对仿真建模人员提出了较高的要求，同时也可能导致建立的模型通用性及重用性较差。而本体驱动建模的思想是将仿真模型表示为建模本体的实例，再利用映射软件映射到和仿真环境相适应的可执行模型。这大大的提高了模型的重用性，并极大程度减轻了仿真建模人员的负担。

基于集合理论的 DEVS 仿真范式提供了清晰的语义来描述仿真模型的结构和行为，而 DEVS 的各种仿真环境都是以面向对象的方式实现，提供了完整的逻辑结构来实现模型的行为。通过建立 DEVS 建模本体 DEVSMO，可以将 DEVS 模型的结构和行为以一种独立于仿真环境的形式描述出来，并且这种方式能够被机器和建模人员所理解。

2.1 本体驱动 DEVS 建模框架

本体驱动建模理论中，有 2 个元素不可缺少，一个是建模本体，另外一个是本体映射工具。本文提出的基于本体驱动思想的 DEVS 建模框架如图 1 所示，所有的 DEVS 仿真模型都是 DEVS 建模本体(DEVSMO)的实例，建立 DEVS 模型的过程也是为 DEVSMO 添加实例的过程。DEVSMO 实例与可以直接在仿真环境中执行的可执行模型不同，这种模型与仿真环境没有直接的关联，可以被称为中立的 DEVS 模型。这种中立的 DEVS 模型并不能直接在仿真环境中执行，需要转化到相应仿真环境中可执行的模型代码。这种转化过程可以通过本体映射工具自动完成，本体映射工具根据 DEVSMO 本体以及目标执行环境所需要的可执行模型之间的映射关系，生成相应的可执行代码，完成整个建模过程。

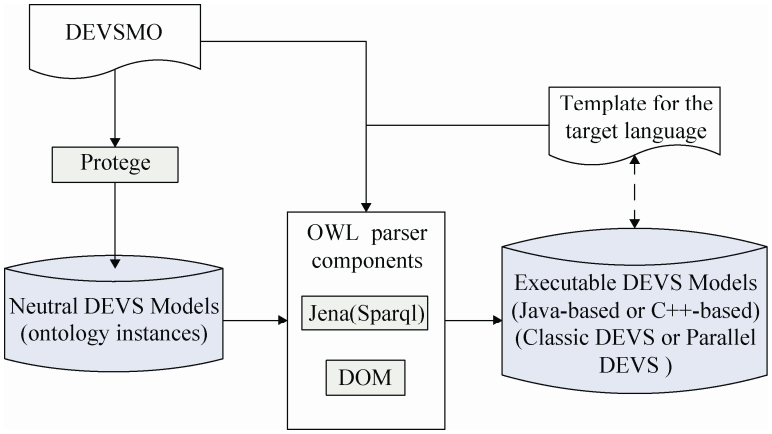

图 1 本体驱动下的 DEVS 建模框架

2.2 DEVSMO

在 DEVS 中，有很多概念都是从数学集合的角度来描述，但是这种数学集合形式的描述较难转换成可执行模型。同时，DEVS 可执行模型的代码对建模者来说也很难理解。这样，DEVSMO 一方面需要考虑 DEVS 所描述的模型结构，另一方面也需要考虑在面向对象的编程环境中所实现的模型行为。如果 DEVSMO 只考虑 DEVS 中所使用的概念，则很难将 DEVSMO 映射成 DEVS 模型；如果 DEVSMO 只考虑模型的实现，则所得到的 DEVSMO 缺少 DEVS 中的概念，造成其他的建模者很难使用这样的模型。为了解决这个问题，我们在 DEVSMO 中引入面向对象的思想，力图使 DEVSMO 既容易理解也能方便的转化为可执行模型。

图 2 是 DEVEMO 的结构图。图中描述了 DEVS 中的一些基本的概念以及这些概念之间的关系。DEVSMO 和 DEVS 一样也将模型分为原子模型和耦合模型，分别对应着图中的 AtomicModel 和 CoupledModel。在 DEVS 中，还有一些扩展的 DEVS 模型，例如 Parallel DEVS，Cell DEVS 等，故在这里使 ClassicAtomic 和 ClassicCoupledModel 分别从 AtomicModel 和 CoupledModel 继承。Classic-AtomicModel 包含了 InputSet(输入)、OutputSet(输出)、States(状态变量)、OutputFunction(输出函数)、InternalTransitionFunction(内部状态转移函数)、ExternalTransitionFunction(外部状态转移函数)，这些都和 DEVS 中的概念相对应。

图 2 描述了 DEVSMO 的基本结构，其中每个概念都有相关的一些属性和一些附属概念。图 3 是 InputSet 相关的概念示意图，每个 InputSet 都有一些 InputVariable 以及端口集合 InputPortSet，端口集合 InputPortSet 含有许多 InputPort，这些 InputPort 都有一个 InputVariable 与之关联。这样就将 DEVS 的输入相关的概念解释清楚了。

在 DEVS 中，状态变量是非常重要的概念，表现了这个模型的动态特性。一个 DEVS 模型至少包含有名为“Phase”的特殊状态变量 PhaseVariable 和时间推进变量 TimeElapsed Variable。其中，PhaseVariable 继承自 StateVariable。如图 4 所示，在 DEVSMO 中，每个原子模型都有一个状态变量集合 TotalStateSet，包含了 TimeElapsedVariable 和 PhaseVariable 这两个特殊变量以及用户定义的其他普通状态变量。

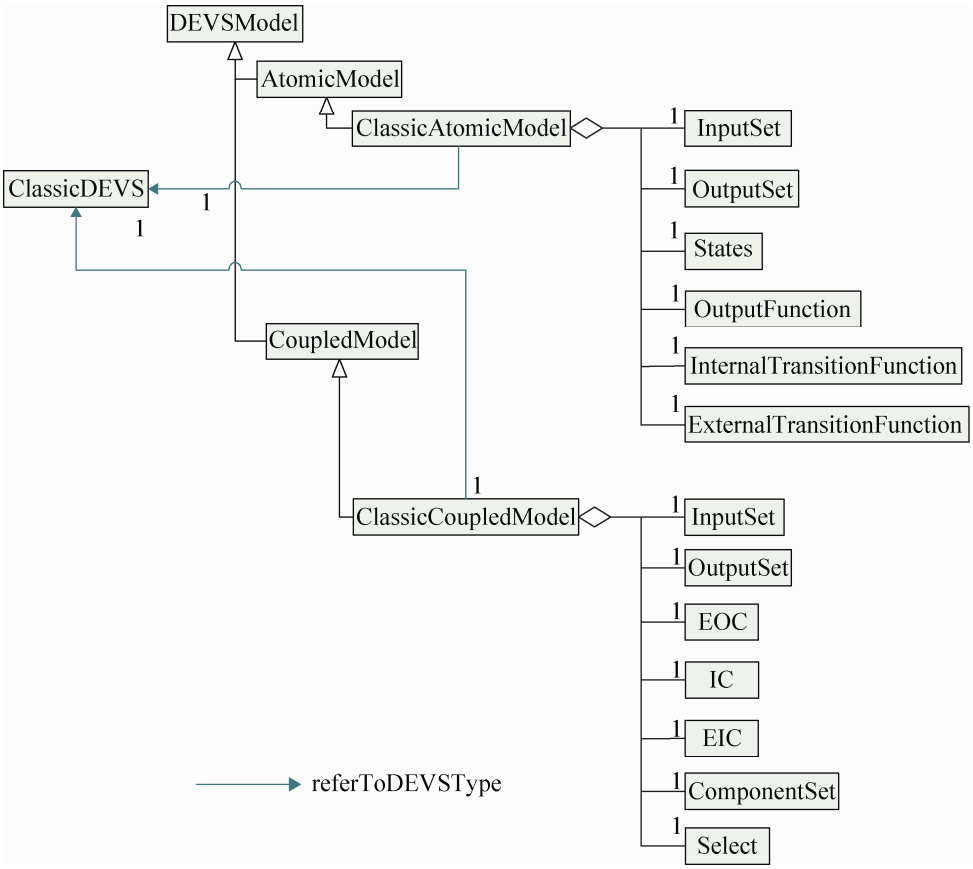

图 2 DEVSMO 基本结构图

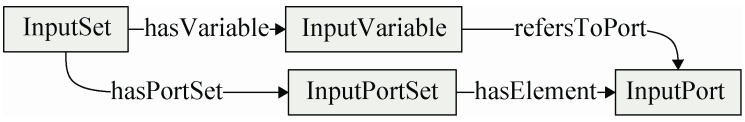

图 3 DEVSMO 输入端口概念示意图

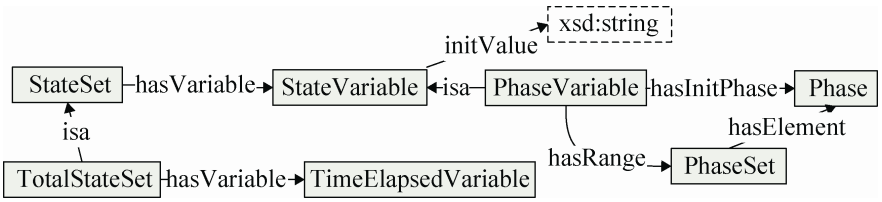

图 4 DEVSMO 状态变量定义

在 DEVSMO 中，模型的行为主要存在于内部状态转移函数，外部状态转移函数和输出函数中和一些初始化函数中。在 DEVS 中，模型的状态转移表现为多个状态变量的迁移，从一个状态变为另外一个状态；而在 DEVS 的可执行模型中，DEVS 模型的状态被分解为多个状态变量，在状态转移函数的一次执行过程中被分别赋予不同的值。这种赋值的行为可以看成是一次状态转移，成为 DEVSMO 中的一个 Action。每个 Action 都可能有一个 Condition，用于表达 Action 执行条件。从图 5 中，我们可以看到 DEVS 模型的行为可以分解为状态变量初始化(StateInitialization)、状态变量更新(StateUpdate)、输入输出变量赋值(InputVariable Assignment，OutputVariableAssignment)、端口输出(SendOutput) 以及模型参数赋值(Parameter Assigment)中。DEVSMO 将复杂的模型行为描述按照 DEVS 理论中的概念分解为一个个 Action。在这些 Action 中，模型行为的复杂程度已经降低，可以用 JAVAML、CPPML 等代码描述语言描述。这样也可以较好的转换为可执行模型代码。

在 DEVSMO 中，耦合模型的模块集合和选择函数的定义如图 6 所示。每一个模块集合都至少有一个模块。每个模块都对应着一个 DEVS 模型，多个模块也可以对应着同一个 DEVS 模型。在不同的模块中，模型参数是不一样的。

select 函数是一个特别的函数，其定义域是耦合模型中的模块集合，而其领域是也是模块集合。在 DEVSMO 中，选择函数被定义为一个有序对集合，集合中的每一个元素都含有一个优先选择的模块以及一系列可能同时有冲突的模块。

3 DEVSMO_Mapper

如前文所述，利用本体表达的 DEVS 模型并不能直接在仿真环境中使用，需要利用本体映射工具将中立的 DEVS 模型映射为和仿真环境相对应的可执行模型代码。本文开发了基于 JAVA 的 DEVSMO_Mapper 本体映射工具可以将上一节中的 DEVSMO 实例映射为 CD++环境下的 DEVS 仿真代码。软件整体框架如图 7 所示。

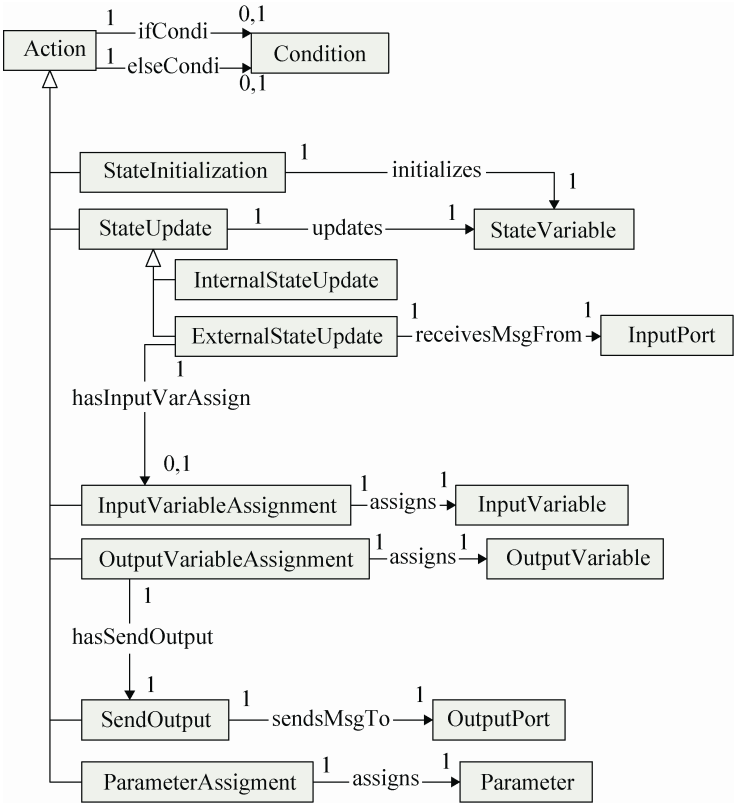

图 5 DEVSMO 中模型行为相关概念

第 27 卷第 12 期 系统仿真学报 Vol. 27 No. 12
2015 年 12 月 Journal of System Simulation Dec., 2015

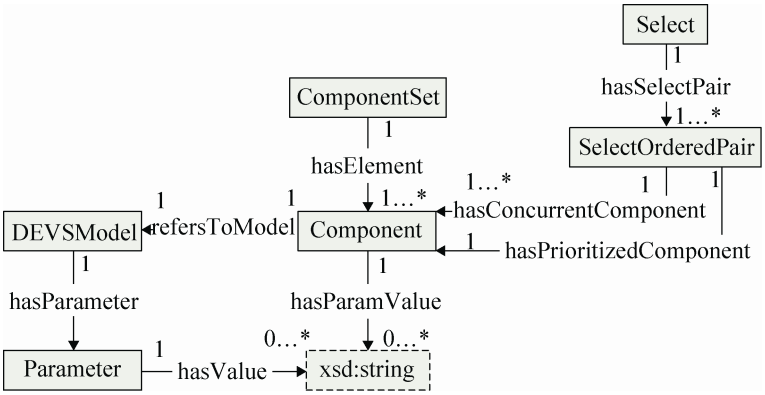

图 6 耦合模型模块集合定义

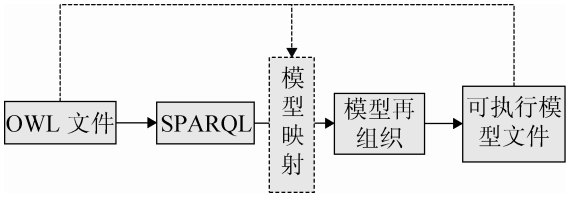

图 7 DEVSMO_Mapper 整体框架

DEVSMO_Mapper 首先对 OWL 文件解析，使用 SPARQL(Simple Protocal and RDF Query Language)将 OWL 文件中的模型信息抽取出来。然后根据可执行模型文件所需要的信息，提取相应的信息并组合为完整的可执行模型。

如图 8 所示，DEVSMO_Mapper 主要有模型管理、DEVS 基本信息模块和模型映射模块三部分。其中 DEVS 基本信息模块包含有组成 DEVS 模型的各种基本元素，这些元素组成了一个完整的 DEVS 模型。DEVSMO_Mapper 在解析 OWL 文件过程中提取的 DEVS 模型就以这种方式存在于程序中。模型映射模块从 DEVS 基本信息模块中提取信息，并且重新组合为 CD++下可以直接运行的*.cpp 和*.h 文件。模型管理模块装载*.owl 模型文件、提取 DEVS 模型信息、调用模型映射模块等功能。

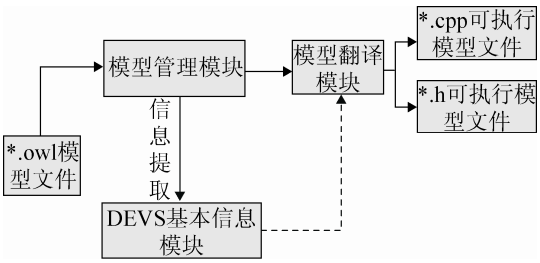

图 8 DEVSMO_Mapper 各模块

3.1 DEVS 基本信息模块

DEVS 基本信息模块主要包含了 DEVS 所对应的一些模型元素，如 ActomicModel，InputSet，OutputSet，StateSet 等类，这些类在一起构成了 DEVS 模型的结构和模型的行为。和图的组织结构类似，ActomicModel 对应着每个 DEVS 原子模型，其包含了输入集合 InputSet、输出集合 Output Set、参数列表 paraList、状态变量集合 StateSet、内部状态转移函数 InternalTransitionFunction、外部状态转移函数 ExternalTransitionFunction、输出函数 OutputFunction 等。其中 InitialFunction，Internal TransitionFunction，ExternalFunction，OutputFunction 行为类似，都用于保存一系列的模型操作模块，他们有着共同的基类 MultiVariableFunction。每个模型操作模块有一个操作条件和一系列具体的操作。DEVS 模型输入集合 InputSet 和模型输出集合 OutputSet 都有着一个 Port 列表，每个 Port 就代表 DEVS 模型的一个输入或输出端口，这个端口 Port 有自己的端口名 portName 和端口所对应的变量。详细的类和解释如表 1 所示。

3.2 OWL 文件信息提取

SPARQL 是为 RDF 开发的一种查询语言和数据获取协议，可以用于任何可以用 RDF 来表示的信息资源。而 OWL 就是建立在 RDF 的顶端之上，所以在解析 OWL 文件的过程中，本文使用 SPARQL 将 OWL 文件中的信息提取出来。相比较

原始的利用 DOM 方式来解析 OWL，利用 SPARQL 可以节省较多的代码，并可以将 SPARQL 的查询语句写成模板形式。

SPARQL 的语法和 SQL(Structured Query Language)类似，详细语法可以参见文献[21]，表 2 便是本程序所使用的部分 SPARQL 查询语句。

表 1 DEVS 基本信息类

类名	相关解释
AtomicModel	对应每个 DEVS 原子模型，其包含了输入、输出、参数列表、状态集合、内部状态转移函数等
InitialFunction	初始化函数
InputSet	输入集合，包含了一个输入端口的列表 LinkedList<Port>
OutputSet	输出集合，包含了一个输出端口的列表 LinkdedList<Port>
InternalTransitionFunction	内部状态转移函数
ExternalTransitionFunction	外部状态转移函数
OutputFunction	输出函数
TimeAdvanceFunction	时间推进函数
StateSet	模型状态集合，包含有默认的 Phase 和 Sigma 状态。
MultiVariableFunction	InitialFunction，InternalTransitionFunction，ExternalFunction，OutputFunction 的基类，包含有一个 ActionSet 列表。
ActionSet	对应着函数中的一个操作模块，这些操作模块决定了函数的作用，每个 ActionSet 都有包含操作模块的执行条件 Condition 和一个 Action 列表。
Action	表示模型的一个具体的简单的行为
Condition	每个 ActionSet 中都有一个 Condition。代表这个模型操作模块执行的条件。
Port	代表着 DEVS 模型中的一个端口，包含有端口名 portName 和对应的端口变量 signalVar

表 2 信息提取 SPARQL 语句

查询语句目标	SPARQL
查询 atomicModel 实例	SELECT ?model WHERE {?model rdf:type devsmo:ClassicAtomicModel}
查询模型参数列表 paramterList 及参数类型	SELECT ?parameter ?valueType WHERE {?parameter rdf:type devsmo:ParameterVariable ?parameter . devsmo:valueType ?valueType}
查询模型输入集合	SELECT ?inputSet WHERE {?inputSet rdf:type devsmo:InputSet . devsmo:processor devsmo:hasInputSet ?inputSet }
查询模型输入端口列表	SELECT ?inPort ?portVar ?varType Where {?inPort rdf:type devsmo:InputPort. ?portVar devsmo:referToPort ?inPort. ?portVar devsmo:valueType ?varType}
查询状态变量	SELECT ?state ?varType ?initValue Where {?state rdf:type devsmo:StateVariable. ?state devsmo:initValue ?initValue. ?state devsmo:valueType ?varType}
查询内部状态转移函数的 actionSet	SELECT ?Internal_actionSet WHERE { ?InternalFun rdf:type devsmo:InternalTransitionFunction . ?InternalFun devsmo:hasActionSet ?Internal_actionSet }
查询 actionSet 对应的 Condition	SELECT ?Cond where { devsmo:action_set_1 devsmo:hasCondition ?Cond}
查询 Port	SELECT ?port where { devsmo:action_set_1 devsmo:receivesEventFrom ?port}
查询 OutputSet	SELECT ?outCalSet where { ?outputFunc rdf:type devsmo:OutputFunction . ?outputFunc devsmo:has ActionSet ?outCalSet}

第 27 卷第 12 期 系统仿真学报 Vol. 27 No. 12
2015 年 12 月 Journal of System Simulation Dec., 2015

3.3 模型映射模块

CD++是一种基于 C++语言的 DEVS 仿真器，为了将 OWL 表达的 DEVS 模型转化为 CD++环境可直接使用的模型，DEVSMO_Mapper 需要输出符合 CD++规范的*.cpp 和*.h 文件。*.h 文件主要是模型的输入端口、输出端口、状态变量的申明；*.cpp 文件包含有模型的初始化、内部状态转移函数、外部状态转移函数、输出函数等模型行为方面的定义。由于 DEVS 模型的结构较为固定，不同模型的*.h 文件相差并不大，生成也比较简单，在此不再熬述，而*.cpp 文件可以按照如下几个步骤生成：

1. 引入头文件。需要将 atomic.h、message.h、mainsimu.h 以及模型头文件引入。这些头文件包含了 CD++中模型正常运行所需要的一些类。

2. 添加输入输出端口。在生成模型构造函数的时候需要添加模型输入输出端口。通过遍历 AtomicModel 中的输入集合 InputSet 和输出集合 OutputSet，并使用 addInputPort 或 addOutputPort 方法将端口添加到模型中。如在下图中，如需要添加输入端口 input_1，则需要使用 this->addInputPort(“input_1”)。

3. 初始化模型参数：在模型构造函数内需要初始化模型的参数。这个只需要遍历 AtomicModel 中的参数列表 paramterList，并将参数的初始值赋值给变量即可。

4. 生成模型初始化函数 initFunction。此过程中需要对 phase 和 sigma 等变量赋初值。

5. 生成外部状态转移函数 externalFunction。外部状态转移函数都是由输入端口的外部事件信号而触发的，所以需要判断输入端口，然后根据端口的不同，遍历对应的 ActionSet，将 ActionSet 中的操作转换成 c++语言形式，即可形成一系列的对状态变量的操作。

6. 生成内部状态转移函数 internalFunction。内部状态转移函数只需要遍历 internalFunction 中包含的 ActionSet，将 ActionSet 中的操作转化为 C++语言形式即可。

7. 生成输出函数 outputFunction。输出函数需要一个输出端口，而在生成输出函数的过程中，除了处理 ActionSet，还必须找到对应的输出端口，并在最后将输出端口变量的值通过此端口输出。

4 应用案例

为了验证本文提出的基于 DEVS 本体驱动的建模方法和 DEVSMO_mapper。本文利用本体驱动理论建立了一个炼油企业的仿真模型。

4.1 模型建立

炼油厂的生产是一个对物料进行处理、存储、运输的过程，在整个过程中有许多变量可以影响最终的产物的特性。而仿真建模就是对这样的一个炼厂进行抽象地过程，在这个过程中，需要了解很多相关的领域知识。OntoCAPE[22-23]是一个计算机辅助过程工程(computer-aided process engineering, CAPE)领域的一个非常完备的本体。为了便于将这些知识分类管理，OntoCAPE 根据所描述的知识的作用以及抽象程度，行成许多子本体和不同的抽象层次。其中抽象程度最高的是元模型层，主要包含了一些基本建模概念和基本设计原则；接下来是上层(Upper Layer)，主要包含了一些系统层面的相关概念；接下来的层次都比较具体，例如一些基本的装置操作、设备、物料以及物理化学属性等。由于 OntoCAPE 所包含的领域知识非常完备，故本文将 OntoCAPE 作为领域本体，建立一个炼油厂的简单物流模型。

一个炼油厂主要包含有 CDU(crude oil distillation unit，常减压蒸馏装置)，FCCU(fluid catalytic cracking unit，催化裂化装置)，汽油调和，柴油调和等生产装置以及罐，它们都是由管道相连接。而在 OntoCAPE 中，炼油厂被描述为一个网状系统，炼油厂的装置就是这个系统的装置节点，装置的侧线便是这些节点的端口。炼油厂的管线也对应着这些端口之间的连接。

在 OntoCAPE 的领域本体的驱动下，可以建立如下模型。整个模型可以分为加工设备、容积设备、分流点以及汇流点 4 类模型，其中加工设备和容积设备在 OntoCAPE 中都是装置节点。分流点和汇流点便是 2 种不同的连接点。

1) 加工设备：加工设备是指能改变物料物理化学性质的设备，涵盖了 CDU，FCC 等石化企业常见的生产装置。加工设备可以简单的抽象成由进料量、出料量以及加工方案组成的“产率模型”。不同的加工方案有不同的产率，而这就会影响出料的产物以及出料的流量。

2) 容积设备：容积设备是指储存物料的装置，涵盖了原料罐、中间罐以及成品罐等。容积设备一般都有进料口以及出料口，进料口和出料口的流量变化会影响物料的罐存。

3) 分流点：分流点指的是管道上的一些分叉节点，物料经过分流点时，以一定的比例分流到不同的地方去。

4) 汇流点：汇流点指的是管道上的一些交汇点，物料经过汇流点时，多个进料汇成一股出料。

因本文主要为了展现本体驱动建模的过程，故没有在模型建立上做过多研究，在此采用文献[24]的模型。

在这里，我们给出一个简单的加工设备模型。该加工装置模型有 2 个输入端口 port_In_1 和 scheduleNo，以及 4 个输出端口 port_out_1, port_out_2, port_out_3, port_out_4。其中，port_In_1 是物料输入的流量，scheduleNo 是生产方案控制信号输入。该装置模型中有两套生产方案，scheduleNo 输入值为 1，则切换到 1 号生产方案，如果 shcduleNo 输入值为 2，则切换到 2 号生产方案。port_out_1、port_out_2、port_out_3、port_out_4 为物料输出的流量，物料输出受到物料输入和生产方案影响。

加工设备的 DEVS 原子模型定义为：

$\text{Unit} =< IP, OP, X, Y, s_0, S, \delta_{\text{int}}, \delta_{ext}, \lambda, ta >$

IP：2 个输入端口，Port_In_1 和 scheduleNo 端口；

OP：port_out_1~port_out_4 四个输出端口；

$X = \{(p,v) \mid p \in IP, v \in X_p\}$：输入事件分为 2 类，一类是从物料输入端口进来的流量变化，值域是正实数；另一类是从方案控制信号输入端口进来的生产方案切换，值域是正整数。

$Y = \{(p,v) \mid p \in OP, v \in Y_p\}$：输出事件的集合，4 个输出端口的输出流量变化，值域是正实数。

S：系统状态集，s_0 是初始状态。

$S = \{Q(\text{in}), Q(\text{out}), B\}, s_0 = \{Q_0(\text{in}), Q_0(\text{out}), B_0\}$，其中 $Q(\text{in}), Q(\text{out})$ 分别是输入输出的流量。

δ_{ext}：外部转移函数，$Q \times X \to S_1$。在加工设备中，由于流量或者生产方案的切换，加工设备首先会进入一个暂态 S_1，在经过一段时间后才会处于最终的新的稳态 S_2，而外部转移函数则是从旧的状态变量到暂态的转变。

δ_{int}：内部转移函数，$\delta_{\text{int}} : S_1 \to S_2$。加工设备从暂态到稳态的过程。

λ：输出函数，$S \to Y$，输出物料输出改变时新的输出流量。

在 protégé 中，可以将上述模型描述为 DEVSMO 的实例，在图 9 中，左边是装置 Eq02001 模型的各种元素，右上可以看到 Eq02001 所拥有的 DEVS 模型元素。这些元素在 3.2 节中都已经较为详细的描述过。图 9 右下是模型中的一个 action 和该 action 执行的条件，在 action 的 comment 中存放有 action 相应的动作。这个 action 是输出函数中的一个行为。在输出函数中还有 action_7~action_14，分别代表着两种不同的生产方案下，4 个物料流出端口的流量计算公式。该 action 需要满足一定的条件才能执行，即 outPutSet_Cond_1，当装置生产方案 Schedule_value 为 1，也就是按照 1 号方案生产时，和端口 port_out_2 绑定的 out_value_2 按照 action_8 来执行状态转移的过程。

根据这样的方式，可以建立文献[24]中的全厂物流模型。并全部通过 DEVSMO_Mapper 映射成可执行模型。其中映射生成的装置模型 Eq02001

代码见图 10。

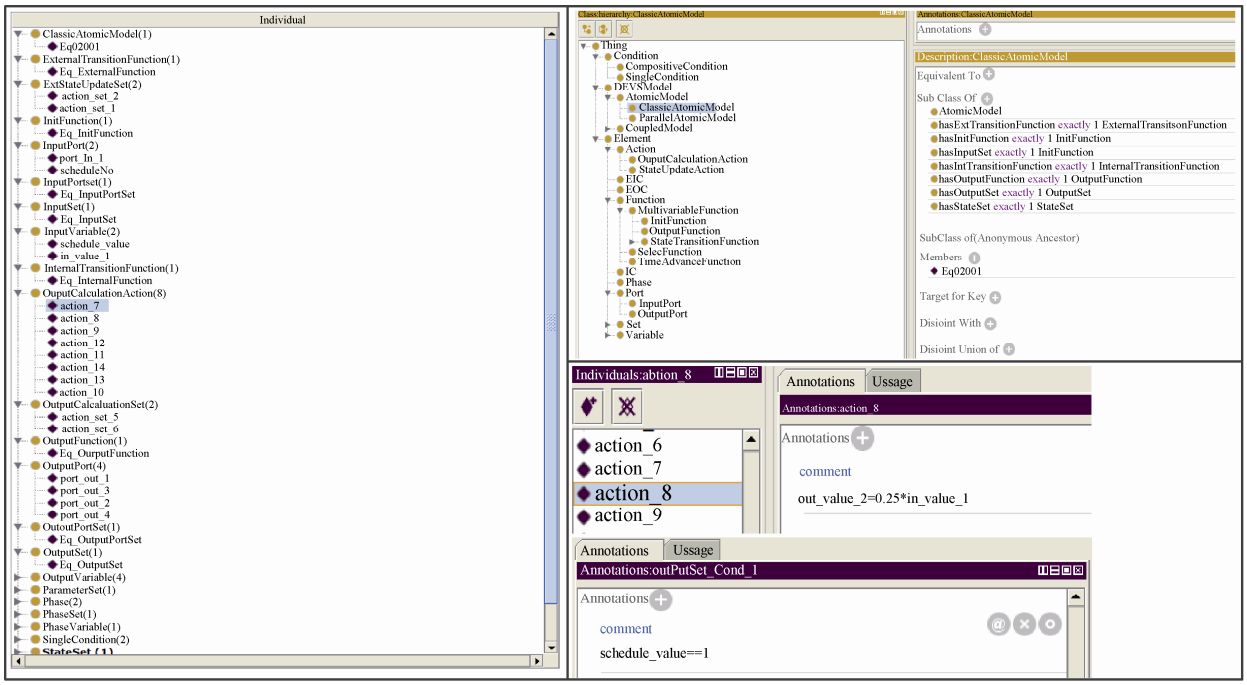

图 9 装置模型在 DEVSMO 中的描述

```
/*** the constructor ***/
Eq02001::Eq02001(conststring &name) :
            Atomic(name)
          , port_In_1(this->addInputPort("port_In_1"))
          , scheduleNo(this->addInputPort("scheduleNo"))
          , port_out_1(this->addOutputPort("port_out_1"))
          , port_out_2(this->addOutputPort("port_out_2"))
          , port_out_3(this->addOutputPort("port_out_3"))
          , port_out_4(this->addOutputPort("port_out_4"))
{

}

/*** externalFunction ***/
Model &Eq02001::externalFunction(const ExternalMessage &msg)
{
    if(msg.port() == port_In_1) {
       in_value_1 = msg.value();
       sigma = Time(0, 0, 0, 0);
       p_phase = active;
       holdIn(p_phase, sigma);
       return*this;
       }
    if(msg.port() == scheduleNo) {
          schedule_value = msg.value();
          sigma = Time(0, 0, 0, 0);
          p_phase = active;
          holdIn(p_phase, sigma);
          return*this;
       }
       return*this;
}
```

```
/*** internalFunction ***/
Model &Eq02001::internalFunction(constInternalMessage &msg) {
     p_phase = passive;
     sigma = Time::Inf;
     return*this;
}

/*** outPutFunction ****/
Model &Eq02001::outputFunction(constInternalMessage &msg) {
     if(schedule_value == 1) {
        out_value_1 = 0.25 * in_value_1;
        this->sendOutput(msg.time(), port_out_1, out_value_1);
        out_value_2 = 0.25 * in_value_1;
        this->sendOutput(msg.time(), port_out_2, out_value_2);
        out_valuc_3 = 0.25 * in_valuc_1;
        this->sendOutput(msg.time(), port_out_3, out_value_3);
        out_value_4 = 0.25 * in_value_1;
        this->sendOutput(msg.time(), port_out_4, out_value_4);
        return*this;

     }
     if(schedule_value == 2) {
           // ...
        return*this;
     }
     return*this;
}
```

图 10 生产装置 Eq02001 模型代码片段

4.2 可执行模型验证

为了验证模型的正确性，本节利用 CD++仿真环境来验证软件产生的可执行模型。在 CD++环境中，我们只使用上面软件生成的可执行模型，人工设置文献[24]中的模型输入数据，然后通过对比本

文的模型仿真结果和文献中仿真结果，来确定模型的正确性。

通过 CD++环境仿真，可得一系列结果，如罐的罐存变化，装置的出料流量等。下面选取了一些有代表性的仿真输出来对比。表 3 是 Eq02002 装置 7 号侧线以及 8 号侧线出料的对比结果。在仿真过程中，该装置进料流量有部分波动，并且在 2008-01-02 T 07:00 点配置了一个方案切换的事件，导致 7 号侧线在方案切换后有物料流出，而 8 号侧线物料流量则降为 0。从表 3 中可以看出，本文仿真结果同文献[24]仿真结果基本一致。同时也可以证明，通过本文提出的方法建立的 DEVS 模型，然后通过 DEVSMO_Mapper 软件映射而成的可执行模型文件是正确的。

表 3 仿真模型运行结果

时间	装置	7 号侧线		8 号侧线	
		文献中结果	本文结果	文献结果	本文结果
2008-01-01 T 8:00	eq02002	0	0	75.92	75.92
2008-01-01 T 9:00	eq02002	0	0	83.2	83.2
2008-01-01 T 10:00	eq02002	0	0	75.4	75.4
2008-01-01 T 12:00	eq02002	0	0	78	78
2008-01-01 T 15:00	eq02002	0	0	91	89.622
2008-01-01 T 16:00	eq02002	0	0	85.8	85.8
2008-01-01 T 18:00	eq02002	0	0	76.7	76.7
2008-01-01 T 22:00	eq02002	0	0	87.36	87.36
2008-01-02 T 2:00	eq02002	0	0	85.28	85.28
2008-01-02 T 5:00	eq02002	0	0	79.56	79.56
2008-01-02 T 7:00	eq02002	0	0	72.28	72.28
2008-01-02 T 8:00	eq02002	31.6	31.6	0	0
2008-01-02 T 10:00	eq02002	33.94	33.94	0	0
2008-01-02 T 12:00	eq02002	35.12	35.12	0	0
2008-01-02 T 13:00	eq02002	34.06	34.06	0	0
2008-01-02 T 15:00	eq02002	31.2	31.2	0	0
2008-01-02 T 18:00	eq02002	31.6	31.6	0	0

5 结论

本文针对仿真领域模型重用率低的问题，利用本体驱动建模的思想，将DEVS仿真模型利用DEVSMO建模本体表达出来，利用本体这种能够被计算机所处理的特性，在一定程度上解决了DEVS模型和仿真环境结合紧密的问题。本文还通过软件DEVSMO_Mapper将OWL形式的DEVS仿真模型映射成CD++环境中的可执行模型并用案例验证，为本体表达的DEVS模型的应用做了示范，同时也证明了利用DEVSMO表达的DEVS模型并没有丢失原有的信息，保证了DEVS仿真的正确性。

参考文献:

[1] Zeigler B P, Praehofer H, Kim T G. Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems [M]. USA: Academic Press, 2000.

[2] Martín J L, Mittal S, López-Peña M A, *et al*. A W3C XML schema for DEVS scenarios [C]// Proceedings of the 2007 Spring Simulation Multiconference-Volume 2. Society for Computer Simulation International, 2007: 279-286.

[3] Meseth N, Kirchhof P, Witte T. XML-based DEVS

modeling and interpretation [C]// Proceedings of the 2009 Spring Simulation Multiconference. Society for Computer Simulation International, 2009: 152.

[4] Mittal S, Risco-Martín J L, Zeigler B P. DEVSML: automating DEVS execution over SOA towards transparent simulators [C]// Proceedings of the 2007 Spring Simulation Multiconference-Volume 2. Society for Computer Simulation International, 2007: 287-295.

[5] Bellipady K R. Ontology driven simulation using ontology mediation techniques [D]. USA: University of Georgia, 2009.

[6] Guizzardi G, Wagner G. Towards an ontological foundation of discrete event simulation [Z]. New York, USA: IEEE, 2010: 652-664.

[7] Silver G A, Miller J A, Hybinette M, *et al*. DeMO: An Ontology for Discrete-event Modeling and Simulation [J]. Simulation (S0037-5497), 2011, 87(9): 747-773.

[8] Lacy L W. Interchanging Discrete event simulation Process Interaction Models using the Web Ontology Language-OWL [D]. Orlando, Florida, USA: University of Central Florida, 2006.

[9] A F, J R, R F. The ontolingua server: A tool for collaborative ontology construction [J]. International Journal of Human-Computer Studies (S1071-5819), 1997, 46(6): 707-727.

[10] Macgregor R M. Inside the LOOM description classifier [J]. ACM Sigart Bulletin (S0163-5719), 1991, 2(3): 88-92.

[11] Chaudhri V K, Farquhar A, Fikes R, *et al*. OKBC: A Programmatic Foundation for Knowledge Base Interoperability [C]// AAAI/IAAI. USA: AIAA, 1998: 600-607.

[12] Lenat D B, Guha R V, Pittman K, *et al*. Cyc: toward programs with common sense [J]. Communications of the ACM (S0001-0782), 1990, 33(8): 30-49.

[13] Klyne G, Carroll J J. Resource description framework (RDF): Concepts and abstract syntax [J]. W3C Recommendation, 2004.

[14] Welty C, Mcguinness D L. OWL web ontology language guide, W3C Recommendation, W3C (February 2004) (2009-11-12) [2014-06-11]. [J/OL]. http://www. w3. org/TR/2004/REC-owl-guide-20040210. 2004.

[15] Hailemariam L, Venkatasubramanian V. Purdue ontology for pharmaceutical engineering: part I. Conceptual framework [J]. Journal of Pharmaceutical Innovation (S1872-5120), 2010, 5(3): 88-99.

[16] Hailemariam L, Venkatasubramanian V. Purdue ontology for pharmaceutical engineering: Part II. Applications [J]. Journal of Pharmaceutical Innovation (S1872-5120), 2010, 5(4): 139-146.

[17] Suresh P, Hsu S, Akkisetty P, *et al*. OntoMODEL: Ontological mathematical modeling knowledge management in pharmaceutical product development, 1: Conceptual framework [J]. Industrial and Engineering Chemistry Research (S0888-5885), 2010, 49(17): 7758-7767.

[18] Suresh P, Hsu S, Reklaitis G V, *et al*. OntoMODEL: Ontological mathematical modeling knowledge management in pharmaceutical product development, 2: Applications [J]. Industrial and Engineering Chemistry Research (S0888-5885), 2010, 49(17): 7768-7781.

[19] 张童, 刘云生, 查亚兵. 基于本体的仿真服务定制与组合 [J]. 国防科技大学学报, 2007, 29(4): 105-109.

[20] 彭春光, 彭勇, 赵鑫业, 等. 本体在建模与仿真中的应用 [J]. 系统仿真学报, 2009 (21): 6739-6742.

[21] Quilitz B, Leser U. Querying distributed RDF data sources with SPARQL [M]. Germany: Springer Berlin Heidelberg, 2008, MLA.

[22] Morbach J, Wiesner A, Marquardt W. OntoCAPE-A (re)usable ontology for computer-aided process engineering [J]. Computers & Chemical Engineering (S0098-1354), 2009, 33(10): 1546-1556.

[23] Morbach J, Yang A, Marquardt W. OntoCAPE—A large-scale ontology for chemical process engineering [J]. Engineering Applications of Artificial Intelligence (S0952-1976), 2007, 20(2): 147-161.

[24] 郑丽钰. 基于 DEVS 的石化多层次物流模型建模和仿真研究 [D]. 杭州: 浙江大学, 2008.