Journal of System Simulation

Volume 28 | Issue 2 Article 4

8-17-2020

Modeling and Simulation of Executable Architecture Based on P-DEVS

Jianpeng Hu
1. College of Electronic and Electrical Engineering, Shanghai University of Engineering Science, Shanghai 201620, China;;2. Dept. of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China;

Linpeng Huang
2. Dept. of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China;

Follow this and additional works at: https://dc-china-simulation.researchcommons.org/journal

Part of the Artificial Intelligence and Robotics Commons, Computer Engineering Commons, Numerical Analysis and Scientific Computing Commons, Operations Research, Systems Engineering and Industrial Engineering Commons, and the Systems Science Commons

This Paper is brought to you for free and open access by Journal of System Simulation. It has been accepted for inclusion in Journal of System Simulation by an authorized editor of Journal of System Simulation.

Modeling and Simulation of Executable Architecture Based on P-DEVS

Abstract

Abstract: Executable architecture modeling plays a vital role in the design of complex systems and System of Systems (SoS), but there is still no common view and standard, unified execution mechanism and environment for an executable architecture. Three major elements of executable architecting were analyzed and summarized with a comparison among the options of these elements. *To realize automated transformation from graphical models to executable codes, a generic modeling approach to executable architecture of SoS based on Parallel Discrete Event System Specification (P-DEVS) was proposed*, and a case study of airport SoS shows its feasibility and effectiveness.

Keywords

DEVS, executable architecture, system of systems, modeling & simulation;

Recommended Citation

Hu Jianpeng, Huang Linpeng. Modeling and Simulation of Executable Architecture Based on P-DEVS[J]. Journal of System Simulation, 2016, 28(2): 283-291.

This paper is available in Journal of System Simulation: https://dc-china-simulation.researchcommons.org/journal/vol28/iss2/4

第 28 卷第 2 期
2016 年 2 月
系统仿真学报©
Journal of System Simulation
Vol. 28 No. 2
Feb., 2016

基于 P-DEVS 的可执行体系结构建模与仿真方法

胡建鹏 [1,2]，黄林鹏 [2]
（1.上海工程技术大学电子电气工程学院，上海 201620；2.上海交通大学计算机科学与工程系，上海 200240）

摘要：可执行体系结构建模与仿真验证在复杂大系统和体系的设计与构建中起着至关重要的作用，但目前相关的概念和方法还不是很成熟，缺乏统一的认识和标准，针对不同的可执行需求存在着很多不同的执行机制和执行环境。分析和总结了可执行体系结构建模的三大要素，并且比较了这些要素的可选项，*提出了一个基于 Parallel Discrete Event System Specification(P-DEVS)的通用可执行体系结构建模与仿真方法，实现了从图形化建模到仿真代码自动生成的全过程*，结合实例介绍了所提方法在机场综合信息系统中的应用来证明该方法的可行性和有效性。

关键词：离散事件系统规范；可执行体系结构；系统的系统；建模与仿真
中图分类号：TP391　　文献标识码：A　　文章编号：1004-731X (2016) 02-0283-09

Modeling and Simulation of Executable Architecture Based on P-DEVS

Hu Jianpeng[1,2], *Huang Linpeng*[2]
(1. College of Electronic and Electrical Engineering, Shanghai University of Engineering Science, Shanghai 201620, China;
2. Dept. of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China)

Abstract: Executable architecture modeling plays a vital role in the design of complex systems and System of Systems (SoS), but there is still no common view and standard, unified execution mechanism and environment for an executable architecture. Three major elements of executable architecting were analyzed and summarized with a comparison among the options of these elements. *To realize automated transformation from graphical models to executable codes, a generic modeling approach to executable architecture of SoS based on Parallel Discrete Event System Specification (P-DEVS) was proposed*, and a case study of airport SoS shows its feasibility and effectiveness.

Keywords: DEVS; executable architecture; system of systems; modeling & simulation

引言

可执行体系结构一直以来都是复杂系统工程的研究热点，尤其是在分布式网络信息技术快速发展的今天，普遍存在的集成化大系统作为一个由多个交互式组件系统组成的联合体，被称之为“系统的系统”(System of Systems，SoS)，或者“体系”，此类大系统的体系结构定义了系统各组成部分的结构和它们之间的关系，以及制约系统设计随时间演化的原则[1]，在实现系统使命目标和满足系统能力需求方面起着至关重要的作用。而构建可执行模型的目的是为了对体系结构设计进行验证和评估，判断体系结构产品所描述的体系结构方案是否满足系统的功能需求和非功能需求及其满足程度。在国防军事，航空航天以及交通运输等重大工程研究领域，“体系工程”研究者们将可执行体系结构的研究作为最重要的核心内容之一，提出了各种各样

收稿日期：2014-10-14　　修回日期：2014-12-14；
基金项目：国家自然科学基金项目(61232007, 91118004);
作者简介：胡建鹏(1980-)，男，湖北武汉，博士生，讲师，研究方向为软件工程、系统工程；黄林鹏(1964-)，男，教授，博导，研究方向为软件工程、分布式计算。

第 28 卷第 2 期 系统仿真学报 Vol. 28 No. 2
2016 年 2 月 Journal of System Simulation Feb., 2016

的解决方案，一部分直接利用某种集成建模仿真工具进行可执行体系结构设计，另一部分则利用模型转换方法使静态模型变得可执行。前者侧重于体系结构的分析和验证，而后者可以作为模型驱动开发过程中的一个关键环节，其体系结构产品还可以用于后续环节(如代码生成)，本文将着重讨论后者。由于许多相关的概念和方法还不是很成熟，缺乏统一的认识和标准，针对不同的可执行需求存在着很多不同的执行机制和执行环境。目前已有的各种解决方案从使用的执行环境来看可分为 3 类：(1) 可执行 UML 方法：文献[2]利用 xUML 和动作规约语言(Action Specification Language，ASL)对反导系统的体系结构进行了功能性验证，而文献[3]则利用 fUML(foundational UML)结合可执行活动模型代数验证了体系结构模型在逻辑上是合理的。此类方法通过虚拟机的方法实现 UML 模型的可执行，其行为描述同样依靠相应的动作语言。然而，图形化的建模符号并不适合于细节编程，此类方法过于依赖于编程语言，目前的研究也只限于功能性需求的验证。(2) 离散事件系统仿真方法：体系结构可执行的关键在于具有精确操作语义的仿真范式，广泛使用的 Petri 网和 DEVS(Discrete Event System Specification)就为此提供了形式化的仿真体系。目前国内外研究转化为各类 Petri 网的文献最多，包括对象 Petri 网(Object Petri Net，OPN)[4]，着色 Petri 网(Colored Petri Nets，CPN)[5]，时间约束 Petri 网(Timed CPN，TCPN)[6]以及层次化着色 Petri 网(Hierarchical CPN，HCPN)[7]。各种 Petri 网变体各有优势，但生成 Petri 网模型时人工干预的因素较多，可执行模型构建效率低、重用性差，不易实现静态模型到动态模型的自动转换。而 DEVS 则提供了模块化、层次化的建模方法论和统一的模型描述框架，是一种具有普适性的建模理论。并且 CPN 和时间自动机等一系列离散事件系统都可以用 DEVS 的理论进行描述。从 UML 到 DEVS 的转换也具有广泛研究[8]，而文献[9]提出的统一 DEVS 过程(DEVS Unified Process，DUNIP)为各种建模语言描述的体系结构模型自动转化为可执行的 DEVS 模型提供了技术路线。最近发布的 DEVS 建模语言 DEVSML 2.0[10]基于支持并行仿真的 Parallel DEV(P-DEVS)，去除了先前版本的诸多不足，但仍然受限于 Finite and Deterministic DEVS (FD-DEVS)，不能描述包括非确定性状态转移在内的复杂行为，对于多个外部事件同时发生的情况也未考虑。不论 Petri 网还是 DEVS 方法，均是利用基于严格形式化定义的仿真范式，也有很多相应的仿真环境支持，可灵活进行范式、建模语言和工具的扩展，应用十分广泛。(3) 使用商业仿真软件的方法：文献[11]建立从各体系结构产品元素到 ExtendSim 建模组件的映射关系，以此建立体系结构的可执行模型，此转换过程目前来看是手动进行的。而文献[12]建立了源模型的元模型到目标模型的元模型的映射关系，应用模型转换语言 ATL (ATLAS Transformation Language)及其转换引擎实现了将系统体系结构中的各种模型自动转换为 Simulink 仿真模型，不仅能够支持验证系统的逻辑和行为的正确性和一致性，还能够支持对系统的整体性能或效能进行预评估。不管使用哪一种商业仿真软件，其优势是仿真环境功能强大，稳定而成熟；但是缺点在于其内部执行机制包括数学基础并没有完全公开，很难进行相应扩展。

基于对以上 3 类方法的总结，本文还分析了可执行体系结构建模过程的三大要素，并对每一要素逐一进行可选项比较，提出一个基于 P-DEVS 的通用可执行体系结构建模方法，并利用 Eclipse 开源平台的工具集实现了从图形化建模到仿真代码自动生成的全过程。

1 可执行体系结构建模的三大要素

从上述 3 类方法的比较中不难得出，它们的本质区别在于各自所依赖的可执行语义，这是可执行体系结构的核心，可称之为建模与仿真范式(Modeling and Simulation Formalism[13])；除此之外，可执行体系结构建模过程中还必须包含体系结构可

执行元素和建模语言这两大要素。每一个要素都有很多不同的选择，一个好的可执行体系结构建模方法必须从中挑选出较为合理的选项并无缝的结合在一起，成为一个既能广泛适用又能灵活扩展的通用方法。

1.1 建模与仿真范式

上述第一类方法的执行语义依赖于可执行 UML 及其动作语言，建模过程过于依赖于编程。第 3 类方法的执行语义则在于商业仿真软件的定义，缺乏较好的灵活性和可扩展性。因此使用第 2 类方法中的离散事件系统仿真理论作为通用方法的基础较为合理。Petri 网和 DEVS 是其中 2 个使用最为频繁的建模与仿真范式，均具有严格的数学定义，相互之间也可以进行模型转换。其本质区别在于：Petri 网侧重进行系统业务流程仿真，其节点多表示为某道工序，可由活动图或者顺序图转换而来；DEVS 侧重进行系统各组件之间的交互仿真，其节点多表示为带状态的系统组件，可由组件图加上状态图转换而来。选择何种范式较为合适取决于具体验证需求，表 1 显示了 Petri 网及其变体和 DEVS 作为可执行体系结构建模与仿真范式的比较；除此之外，DEVS 源于系统理论，是对各个系统组件及其相互关系的抽象，从仿真界面上来讲更加直观；如果系统的业务流程过于复杂，用 Petri 网来表示则相当庞大且复杂。本文将介绍一种采用 DEVS 作为理论基础的通用方法。

表 1 Petri 网和 DEVS 的比较

比较项目	Petri 网及其变体	DEVS
模块化与复用	OPN 具备	具备
层次化	OPN 和 HCPN 具备	具备
并行性	具备	具备
时间约束	TCPN 具备	具备
仿真工具	CPN-tools，OPMSE 等	DEVS-suite (开源)等
体系结构模型转换	自动转换较难	可自动转换
可验证范围	功能、性能、服务质量[14]、效能等	功能性与非功能性需求(含各类质量属性[15])

1.2 可执行体系结构元素

并不是所有的体系结构元素都和可执行体系结构相关。一个体系结构框架(Architecture Framework)通常在不同的视角下定义了许多体系结构元素及其相互关系，构成了多视图的体系结构产品，而与可执行体系结构相关的元素只是其中的子集。目前已有很多成熟的体系结构框架，包括 Zachman 框架，The Open Group 体系结构框架以及美国国防部的 DoDAF，英国国防部的 MODAF 等。其中 DoDAF 已广泛应用于 SoS，因为这个框架特别适合应用在含有复杂集成和互操作的大系统中。在众多的相关文献中，DoDAF 几乎成为体系工程领域的一个标准。然而，DoDAF 只是呈现静态的体系结构模型，而且过于专注于什么需要被描述而不是怎么被描述，关于如何开发可执行体系结构也缺乏相应指导。更明确地讲，DoDAF 2.0 版本中一共有 8 个类别 52 个视图，但只有一部分视图是跟可执行体系结构直接相关的，我们必须从中选取相关者，并按照一定的开发过程获取这些元素，该过程也可以看作是起始环节和基础。通过对 Unified Profile for DoDAF and MODAF(UPDM)的元模型进行检查可获得可执行体系结构的相关候选元素，主要涉及 DoDAF 的数据视点、操作视点和系统视点。文献[8]给出了 UML 中不同视图中元素与 DEVS 元素的映射关系，实质上也给出了可转换为基于 DEVS 可执行体系结构的候选视图。文献[5]则以 CPN 为最终目标，给出了详细的使用 SysML 的一套标准建模过程。而 DUNIP 支持 DoDAF 产品到 DEVS 模型的转换，至于如何获取相应的 DoDAF 产品在此过程中并未提及。因此本文将在总结前人工作基础上给出一个以 DEVS 为最终目标，基于 DoDAF 的通用建模过程去获取可执行体系结构的相关元素。

1.3 建模语言

体系结构模型的描述都是通过建模语言实现的，目前多使用图形化或者符号化的标准记号。体

第 28 卷第 2 期　　系统仿真学报　　Vol. 28 No. 2
2016 年 2 月　　Journal of System Simulation　　Feb., 2016

系结构的建模语言可从 2 个方面进行讨论：(1) 用来描述静态体系结构模型的通用建模语言，诸如 UML，SysML 和 IDEF 结构化描述语言，甚至是用于过程建模的 BPMN(Business Process Modeling Notation)；(2) 具有动作语义的可执行体系结构描述语言，如 DEVSML，用于 Petri 网建模的 PNML (Petri Net Markup Language)以及文献[12]提到的 Simulink 建模语言。利用模型转换技术实现可执行体系结构建模的一般步骤都是使用如 ATL 这样的基于元模型的模型转换工具将前者描述的静态体系结构模型转换为后者描述的可执行模型，最终在相应的执行环境中进行仿真实验。因此选择合适的两方面的建模语言对于实现自动的、完整的、并保持语义一致性的模型转换是非常重要的。

在静态建模语言方面，从能否描述所有 DoDAF 中可执行体系结构元素候选项的角度来看，SysML 的满足程度几乎达到了 100%，UML 的满足程度则只有约 70%，其他 2 种都不及 50%，可见 SysML 作为前端语言来描述静态体系结构最为全面。而从可执行建模语言来看，建模与仿真范式的选择就直接决定了此种语言的使用。两者之间的转换若采用人工的方法效率较低，某些人为的差错还会影响语义的完整性和一致性，所以使用自动转换技术更为合理。鉴于本文选择 DEVS 作为理论基础，在此我们只讨论如何实现从静态体系结构模型到 DEVS 的转换。一般来说，要实现到 DEVS 的转换有 2 种途径：(1) 直接使用 M2T（Model to Text）转换技术将各种 UML 或 SysML 的视图直接映射到 DEVS 范式并生成可执行的代码；(2) 将相关视图产品先转换为某种 DEVS 建模语(如 DEVSML)所描述的模型，然后再自动转换为可执行代码。前者所转换的 DEVS 可执行代码只能在单个平台上执行，而且转换过程相对复杂。后者使用的 DEVSML 是平台独立的建模语言，可生成多平台可执行代码；另外在进行 DEVS 仿真之前需要从多视图产品中综合提取所需信息并去除重复信息的过程，两步走的方法显得更加合理和流畅。Mittal 提出的 DEVSML 2.0 及其配套方法[10]为领域建模和 DEVS 建模之间提供了一座桥梁，使得 DUNIP 成为一个很好的可执行体系结构建模框架，但是 DEVSML 如前所述作为关键环节仍然存在着一些缺点。为此，本文对 DEVSML 进行了一些扩展，实现了用 SysML 描述的体系结构模型与 DEVS 仿真之间的无缝连接。

2　扩展的 DEVS 建模语言

DEVSML 的形式化基础是 P-DEVS，其中模型分为原子模型和耦合模型：原子模型描述了离散事件系统的自治行为，包括系统状态转换、外部输入事件响应和系统输出等。原子模型可以通过连接形成耦合模型。耦合模型的成员既可以是原子模型，也可以是耦合模型。一个原子模型 M 可定义为：

$$M = < IP, OP, X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta >$$

式中：IP，OP 分别是输入输出端口，S 是系统状态集，X，Y 分别是输入输出事件集，$X=\{ (p,v) \mid p\in IP, v\in dom(p) \}$, $Y=\{(p,v) \mid p\in OP, v\in dom(p) \}$,其中 v 为输入输出的数据值，dom 为值域函数。ta 是时间推进函数，ta(s)表示在没有外部事件到达时系统状态保持为 s 的时间量。δ_{int} 是内部转移函数。如无外部事件到达，系统经过 ta(s)时间后，状态 s 将转移到 $\delta_{int}(s)$。δ_{ext} 是外部转移函数，若有外部事件 x 到达，且系统在状态 s 已停留时间为 e，则它立即转移到 $\delta_{ext}(s, e, x)$。δ_{con} 是冲突处理函数，决定了内部事件和外部事件同时发生时内部和外部转移函数的执行顺序。λ 是输出函数，输出事件在系统内部状态转移时产生，且状态转移前的状态 s 用于产生输出 $\lambda(s)$。一个耦合模型 N 可以定义为：

$$N = < IP, OP, X, Y, D, EIC, EOC, IC >$$

式中：IP，OP，X，Y 的含义与原子模型中相同。D 是组件集，是已定义的原子模型和耦合模型的集合。EIC, EOC, IC 分别代表外部输入连接，外部输出连接以及内部连接。由于 DEVSML 不支持对非

确定性状态转移的描述，其冲突函数还不能处理由多个外部事件同时到达引起的状态转移冲突，因此本文对其进行以下扩展：

(1) I/O 变量：组件之间的通信是通过消息传递的形式实现，一个消息实质上是一个打包的输入或者输出事件集。当一个消息包含多个外部事件同时到达时，可以先将各个输入数据存放在相应输入变量 v_x，同样将待输出的的数据存放在相应的输出变量 v_y 中：

$$v_x = \mathrm{Rev}(x), y = \mathrm{Sed}(v_y), v_y = DP(v_x)$$

式中：Rev 为输入接口函数，*Sed* 为输出接口函数，*DP* 为 v_x 到 v_y 数据处理函数，I/O 变量可以是简单数据类型，也可以是队列或者堆栈这样的数据容器，起到缓冲作用。这样做的目的一是不让同时到达的数据发生丢失，同时还可以将消息处理过程与状态转移分离，使得原子模型的行为建模更加清晰明了。

(2) 条件状态转移：首先看看确定性状态转移(Deterministic State Transition，DST)和非确定性状态转移(Nondeterministic State Transition，NST)的区别：

$$\mathrm{DST} = \delta: S \times X^b \rightarrow S$$

$$\mathrm{NST} = \delta: S \times X^b \rightarrow P(S)$$

式中：*P(S)*表示 *S* 的幂集。完全的 NST 不易控制，可使用条件状态转移(Conditional State Transition，CST)来近似实现：

$$\mathrm{CST} = \delta: S \times X^b \times C \rightarrow S$$

式中：*C* 为约束条件集，表示当某个条件满足时状态转移才发生，可设置多个条件实现同一个输入事件触发目标不同的状态转移。这样的条件一般是跟状态变量相关的，也可以是和 I/O 变量相关。I/O 变量作为缓冲区储存还未处理的数据值，其自身就可以作为约束条件实现没有额外外部事件发生的情况下触发状态转移，这样就解决了多个外部事件同时到达引起状态转移冲突的问题，保证每一个外部事件都能被处理。

(3) 增加了一些辅助手段来增强语言的描述能力，如用户自定义函数和代码模板功能，可利用嵌入式的编程语言来描述更为复杂的控制逻辑详细的动作行为。因为 DEVSML 本身是缺乏逻辑表达式和运算符号的，在实现转移条件判断和转移动作时需要一些简单的嵌入式代码支持。本文使用 Eclipse 平台上 Xtext 工具，基于扩展的巴科斯范式(Extended Backus-Naur Form，EBNF)来定义扩展的 DEVS 建模语言，将模型分为实体(Entity)、原子(Atomic)和耦合(Coupled)三类。其中实体和耦合模型的定义与文献[10]基本一致，原子模型作了较大改动，如图 1 所示。

```
Atomic : 'atomic'  name=ID
( 'extends' superType=[Atomic |QualifiedName])?
'{'  'vars''{'(variables += Variable)*'}' /*变量定义*/
     'interfaceIO'  '{' (msgs += Port)*'}'/*端口集*/
 'state-time-advance'  '{' (stas += STA)*'}' /*状态集*/
 'state-machine''{' 'start in ' init=InitState/*状态机*/
 (stm1=Deltext)? (stm2=Outfn)? (stm3=Deltint)?
 (stm4=Confluent)?'}' (func += UserFunction)* '}';
Port : type = ('input'| 'output')    /*端口定义*/
       ref=[Entity |QualifiedName] name=ID;
STA : name=ID timeAdv=TimeAdv;  /*状态定义*/
TimeAdv: tav=DOUBLE|inf='infinity'|tvar=[Variable];
InitState: state=[STA ] (code=Code)?; /*初始状态定义*/
CodeTemplate:'code-template' name=ID  /*代码模板*/
   '(' (codePara += Parameter)* ')'
   '{' (templateBody += TemplateBody)+ '}';
TemplateBody: code=Code | para=[Parameter];
UseTemplate: 'use'ct=[CodeTemplate]  /*使用代码模板*/
            '['(para+=[Variable|QualifiedName])*']';
UserFunction: code=Code ; /*用户自定义函数*/
Parameter: str=STRING;
Code:  '{' str=STRING  '}' ; /*代码块*/     /*外部转移函数*/
Deltext:'deltext' '{'(rm += ReceiveMessage)+
(st += StateTransition)*'}';                /*内部转移函数*/
Deltint:'deltint' '{'(st += StateTransition)+'}';
Outfn: 'Outfn' '{' (so += StateOutput)+ '}'; /*输出函数*/
Confluent: 'confluent'  con=( 'ignore-input'|/*冲突处理*/
'input-only'| 'input-first' | 'input-later' );
SendMessage:'send' '['out=[Port]']'sender=[Variable];
ReceiveMessage:    /*消息接收与发送，实现 Rev 和 Sed 函数*/
'receive' '['in=[Port]('.'subMsg = QualifiedName)?']'
receiver = [Variable];
StateOutput:'S:'state=[STA]'{'(sm += SendMessage)*'}';
StateTransition:'S:' state=[STA]
   '{'(ct += ConditionalTransition)+ '}';
ConditionalTransition: /*条件状态转移*/
( 'when' '(' condition=STRING ')' )?
  '{'(dp += DataProcess)* act = Action '}';
Action: con='continue'|sig = SetSigma|ts=Transition;
SetSigma:                         /*状态转移的各种动作和数据处理*/
'sigma' '('sdv= DOUBLE|inf='infinity'|v=[Variable]')';
Transition: 'goto' target=[STA] (sig=SetSigma)?;
DataProcess: code=Code | ut = UseTemplate;
```

图 1　原子模型的语法定义

3　基于 P-DEVS 的建模与仿真方法

对于一个大型的系统体系，传统的系统工程方法已经不再适用，尤其是当 SoS 功能庞大而复杂时，通常需要将其分解成小的若干子体系，由不同

的职能部门进行建设和管理。但是子体系之间又不是完全独立的，即当某个具体任务需要执行时，往往会将多个子体系中的系统又组成一个任务体系去完成相关使命。因此我们从两个层面去考虑 SoS 的需求：从战略层面来看，SoS 可按照职能划分为不同的子体系；同时从战术层面来看，我们也定义一些任务子体系。基于这个思想，我们将 SoS 的需求分析阶段也分为战略层和战术层。图 2 给出了本文所提方法的主要过程以及涉及到的模型图，相关的模型解释请看表 2。

战略层建模是面向目标和基于能力的需求分析，目的是识别体系的所有参与者及其能力需求。而战术层建模是面向场景的任务建模，目的是识别任务之间的关系以及不同场景中的系统交互。业务体系结构设计是基于以活动为中心的业务流程建模。系统体系结构则是经由业务层逻辑节点映射到物理节点以及进一步精化而得到。当获得完整的静态体系结构模型之后，即可进行模型转换得到平台独立的 DEVS 模型，最终则自动生成可执行代码在仿真环境中进行实验。可执行体系结构元素的获取不是一蹴而就的，而是一个逐步细化的过程。在 DoDAF 所包含的众多视点和模型图中，本文只关注其中的核心要素：能力视点(Capability Viewpoint, CV)，数据信息视点(Data and Information Viewpoint，DIV)，操作视点(Operational Viewpoint，OV)以及系统视点(Systems Viewpoint，SV)中的部分模型，而最终获得的 SV 产品才是 DEVS 仿真的前端模型。基于 DoDAF 的体系结构建模和 SysML 的应用研究已有很多，在此不再赘述。

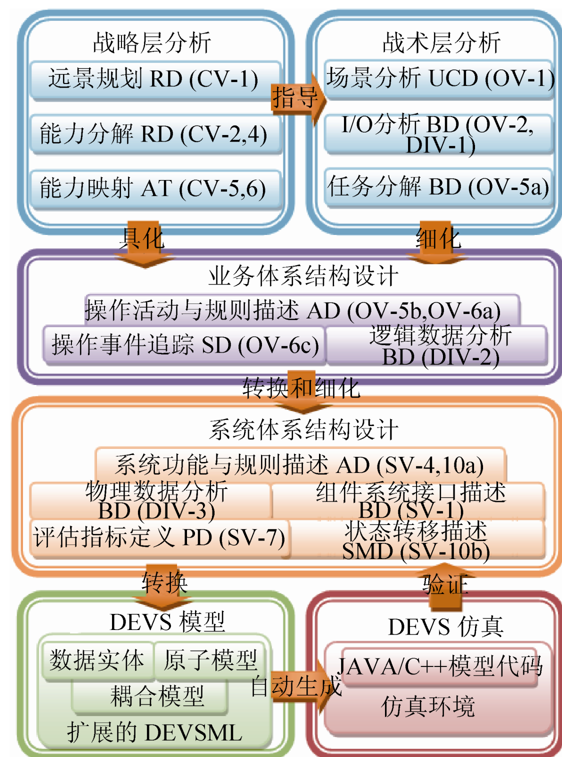

图 2　可执行体系结构建模过程

表 2　与可执行体系结构相关的 DoDAF 模型

DoDAF 视图	SysML 图	说明
CV-1：远景规划	Requirement Diagram (RD)	高层级的能力规划，战略设想
CV-2,4：能力分类与依赖	Requirement Diagram (RD)	能力的分类以及能力之间的隶属关系
CV-5,6：能力映射	Allocation Tables (AT) or link	能力与开发组织的映射关系，能力与活动对应关系
OV-1：操作概念	Use Case Diagram (UCD)	图形化的高层级操作概念
OV-2：操作资源流 DIV-1：概念化数据	Block Diagram (BD)	活动之间交换的资源流，高层数据概念及其关系
OV-5a：操作活动分解	Requirement Diagram (RD)	按层级结构组织的能力和活动
OV-5b,6a：操作活动及规则	Activity Diagram (AD)	能力和活动的背景以及输入输出，约束活动的规则
OV-6c：事件跟踪描述	Sequence Diagram (SD)	跟踪想定或事件顺序中的行为
DIV-2：逻辑数据	Block Diagram (BD)	数据需求和结构化业务过程的规则文档
SV-4,10a 系统功能与规则	Activity Diagram (AD)	系统的功能和数据流以及对系统的约束
DIV-3：物理数据	Block Diagram (BD)	逻辑数据模型实体的物理实现格式
SV-1 系统接口描述	Block Diagram (BD)	确认系统、系统组成要素和它们之间连接
SV-7 系统评估指标	Parametric Diagram (PD)	界定适当时间段对系统模型元素的度量和评估指标
SV-10b 系统状态转移描述	State Machine Diagram (SMD)	确认系统对事件的响应

利用模型转换实现可执行体系结构构建并不需要太多的静态视图作为源模型，图 2 中 DIV-3、SV-10b 和 SV-1 与 DEVSML 的实体、原子以及耦合模型相对应，主要映射关系如表 3 所示。如前所述，本文也使用两步走的方法将 SysML 描述的体系结构模型转换为可执行代码。因为扩展的 DEVSML 是文本型语言，首先将 SysML 的 XMI 文件作为输入源，利用 Eclipse 平台上模型转换工具 Xpand 将其转换为 DEVSML，然后再由 Xtext 自带的 Xtend 工具自动生成 java 代码。图 3 给出了在 Xpand 中定义的状态机图转换模型的部分代码。

4　案例分析

机场综合信息系统是一个典型的 SoS，包含有上百种组件系统，在系统设计初期对其进行体系结构建模非常重要，本文以其中一个局部的飞机降落后的停泊场景为例，验证本方法的可行性和有效性。图 4 为飞描述机停泊的业务流程的活动图，涉及的系统包括航班信息系统(FIS)、登机口操作系统(GOS)、泊位引导系统(DGS)、操作员面板(OP)和飞机。FIS 预先将航班信息发送给 GOS，GOS 分配任务给相应的 DGS 并要求操作人员做好准备，飞机落地后会发出泊位请求，降落的航班信息得到核实之后 DGS 与飞机之间会有若干交互，待飞机完成泊位后会向 FIS 进行结果反馈。

表 3　DEVS 模型与 SysML 图的映射关系

原子 DEVS	扩展的 DEVSML	SysML 状态机图
S	state-time-advance	State
ta		Constraint
δ_{int}	Deltint	Transition effect
δ_{ext}	Deltext	Transition effect
λ	Outfn	State exit
X/Y	receive/send	Transition Event/Action
耦合 DEVS	扩展的 DEVSML	SysML 内部块图
IP	interfaceIO input	Inbound flow ports
OP	interfaceIO outout	Outbound flow ports
D	Models	Blocks
EIC	couplings eic	External input ItemFlows
EOC	couplings eoc	External output ItemFlows
IC	couplings ic	Internal ItemFlows

```
...
«DEFINE Behavior FOR Model»  «EXPAND main FOREACH allOwnedElements().typeSelect(StateMachine)» «ENDDEFINE»
«DEFINE main FOR StateMachine» «FILE name.toString() + ".fns"-» «EXPAND reg FOREACH region» «ENDFILE» «END
«DEFINE reg FOR Region-» «REM»«EXPAND pseudo FOREACH subvertex.typeSelect(uml::Pseudostate)» «ENDREM»
state-time-advance{ «EXPAND subv_advance FOREACH subvertex.typeSelect(State)-» }
state-machine{
    start in «EXPAND start_in FOREACH transition.select(e|e.name == 'start in')»
    deltint{ «EXPAND state_in FOREACH subvertex.typeSelect(State)-» }
    deltext{ «EXPAND state_ext FOREACH subvertex.typeSelect(State)-» }
    outfun{ «EXPAND subv_out FOREACH subvertex.typeSelect(State)-»  }
«ENDDEFINE»
...
«DEFINE trans_in FOR Transition-»       S: «source.name»{
            «IF guard.specification != null-»   when("«guard.specification.stringValue()»")
            { «EXPAND effect FOR effect-»   goto «target.name»  }   «ENDIF»
            «EXPAND target FOREACH target.outgoing.typeSelect(Transition)»      }
«ENDDEFINE»
...
```

图 3　Xpand 中定义的状态机图转换模型

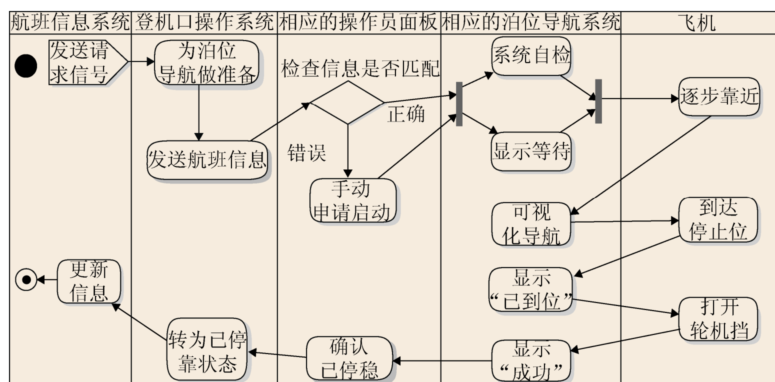

图 4　飞机停泊流程的活动图

Published by Journal of System Simulation, 2016

第 28 卷第 2 期 系统仿真学报 Vol. 28 No. 2
2016 年 2 月 Journal of System Simulation Feb., 2016

图 5 所示为核心部件 GOS 的状态机图，当有消息发送至 GOS 时，先进行条件判断然后根据需要向其它组件发出消息，图中嵌入使用了 DEVSML 的语句，这种 SysML 和 DEVSML 进行协同建模是实现全自动模型转换的前提，同时还使用了 java 语言中简单逻辑表达式和算术表达式，图 6 显示了 GOS 组件的 DEVSML 模型和 java 代码的对比。描述组件及其相互联系的 SV-1 用内部块图(Internal Block Diagram)表示，其外观与图 7 仿真结果界面十分相似。

经过模型转换过程最终生成的 java 模型代码在开源仿真软件 DEVS-suite 中执行，可以看到该业务流程执行和消息传送的动画效果，如图 7 所示，增加的传感器 transducer 是一个统计设备，用来记录飞机停泊所用时间，并计算整体效能；而 Aircraft 实质上是虚拟飞机的生成器，会按照不同间隔发出停泊请求。仿真界面的“SimView”功能窗口显示了系统组件之间的连接和信息交互，在仿真过程中动态显示每个组件的状态变化和组件之间消息的传递；而“TimeView”功能窗口可以用时间轴的形式显示了登机口操作系统不同端口发送或接收的消息以及状态转移轨迹，以上两个功能可以很好的验证系统功能性需求；而控制台窗口可以同步显示总的飞机停泊数量和平均消耗时间，以验证诸如性能等时间相关的非功能需求，验证何种非功能性需求取决于实验场景的设计。

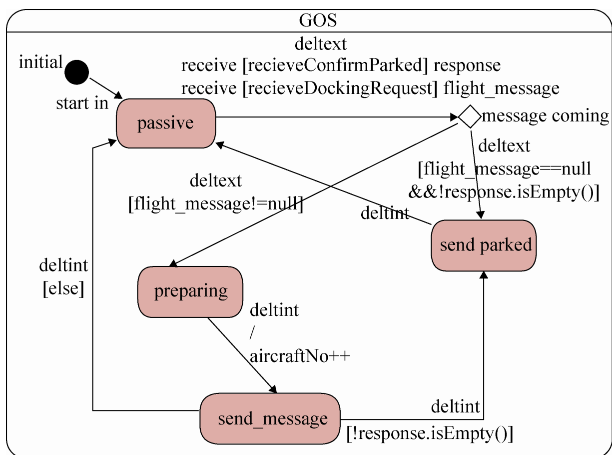

图 5 登机口操作系统的状态机图

```
deltext{
            receive [recieveConfirmParked] response
            receive [recieveDockingRequest] flight_message
            S: passive{
                when ("flight_message != null")
                {   goto preparing  }
                when("flight_message == null && !response.isEmpty()")
                {   goto send_parked }
            }
        }
Outfn{
    S:send_message{send [sendFlightInformation] flight_message
    S:send_parked{send [SendBlocks_onMessage] response }
}
                                                    GOS.fns
-----------------------------------------------------------
                                                    GOS.java
public void deltext(double e, message x)
{   Continue(e);
    for (int i=0; i< x.size();i++)
    {  if (messageOnPort(x,"recieveConfirmParked",i))
        {   entity val = x.getValOnPort("recieveConfirmParked",i);
            DockingDone job = (DockingDone)val;
            response.add(job);  }
        if (messageOnPort(x,"recieveDockingRequest",i))
        {   entity val = x.getValOnPort("recieveDockingRequest",i);
            DockingRequest job = (DockingRequest)val;
             flight_message=job; }
    }
    if(phaseIs("passive"))
    {   if (flight_message != null)
        {   holdIn("preparing",30.0);return;}
        if (flight_message == null && !response.isEmpty())
        {  holdIn("send_parked",0.0);return;}
    }
}
public message out()
{   message  m = new message();
    if(phaseIs("send_message"))
    {   m.add( makeContent("sendFlightInformation",flight_message));
        flight_message = new ent.flight();
        return m;
    }
    if(phaseIs("send_parked"))
    {  for (int i= 0; i< response.size();i++)
        {   entity  job = (entity)response.get(i);
            m.add( makeContent("SendBlocks-onMessage",job));}
        response = new Queue();
        return m;
    }
}
```

图 6 GOS 组件的 DEVSML 模型与 java 代码的比对

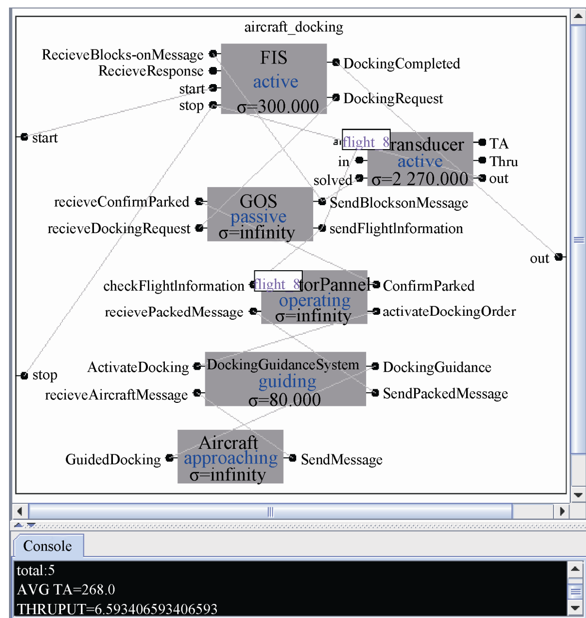

图 7 仿真界面与仿真结果

5 结论

本文提出的基于 P-DEVS 的通用可执行体系结构建模方法主要包含 3 方面的内容：一个扩展的

DEVSML 建模语言，一个通用的获取可执行体系结构元素的建模过程，以及利用 SysML 和扩展的 DEVSML 之间模型自动转换实现协同建模的方法。这一通用方法不仅具有严格的数学基础，可灵活应用于各种复杂系统的体系结构建模，与以往的方法相比还具有以下优点：(1) 扩展的 DEVSML 增强了对于复杂行为的描述能力；(2) 实现了从静态体系结构模型到可执行模型的自动转换；(3) 使建模人员无需深入 DEVS 理论和仿真软件的代码，较好地解决了以往 DEVS 建模困难的问题。

参考文献：

[1] U.S. Department of Defense. Department of Defense Architecture Framework. V2.0” 2009. [EB/OL]. [2014-12-11]. http://dodcio.defense.gov .

[2] 陈岩, 李志淮, 谭贤四, 等. 基于 xUML 的 DoDAF 可执行体系结构开发与验证 [J]. 系统仿真学报, 2014, 26(1): 152-158.

[3] 何红悦, 王智学, 王庆龙, 等. 基于 fUML 的 C4ISR 体系结构可执行建模及分析[J]. 系统工程与电子技术, 2014(9):1874-1881.

[4] 张晓雪, 罗爱民, 黄力, 等. 基于 DM2 的体系结构可执行模型构建方法 [J]. 国防科技大学学报, 2013(2): 27-33.

[5] Wang Renzhong, Cihan H Dagli. Executable system architecting using systems modeling language in conjunction with colored Petri nets in a model-driven systems development process [J]. Systems Engineering, (S1520-6858), 2011, 14(4): 383-409.

[6] Xuling Chang, Linpeng Huang, Jianpeng Hu, et al. Transformation from Activity Diagrams with Time Properties to Timed Coloured Petri Nets [C]// 2014 IEEE 38th Annual Computer Software and Applications Conference. USA: IEEE, 2014.

[7] 倪枫, 王明哲, 周丰, 等. 可执行体系结构的 HCPN 建模方法 [J]. 系统工程与电子技术, 2010, 32(5): 959-965.

[8] Risco-Martín José L, et al. eUDEVS: Executable UML with DEVS theory of modeling and simulation [J]. Simulation Transactions of the Society for Modeling & Simulation International (S0037-5497), 2009, 85(11/12): 750-777.

[9] Mittal Saurabh. Devs unified process for integrated development and testing of service oriented architectures [D]. University of Arizona Tucson, AZ, USA, Dissertations & Theses - Gradworks, 2007.

[10] Mittal S, S A Douglass. DEVSML 2.0: The Language and the Stack [C]// Symposium on Theory of Modeling and Simulation, Spring Simulation Multiconference, San Diego, CA, USA, Society for Computer Simulation International, 2012.

[11] 葛冰峰, 任长晟, 赵青松, 等. 可执行体系结构建模与分析 [J]. 系统工程理论与实践, 2011, 31(11): 2191-2201.

[12] 夏晓凯, 吴际, 徐络, 等. 模型驱动的系统体系结构仿真验证研究 [J]. 系统工程与电子技术, 2013, 35(11): 2424-2429.

[13] Shuman E A. Understanding executable architectures through an examination of language model elements [C]// Proceedings of the 2010 Summer Computer Simulation Conference. San Diego, CA, USA,Society for Computer Simulation International, 2010: 483-497.

[14] 于晓浩, 罗雪山, 陈洪辉, 等. 面向服务军事信息系统的可执行建模仿真方法 [J]. 系统仿真学报, 2010, 22(11): 2479-2484.

[15] Verónica Bogado, Silvio Gonnet, Horacio Leone. Modeling and simulation of software architecture in discrete event system specification for quality evaluation [J]. Simulation Transactions of the Society for Modeling & Simulation International (S0037-5497), 2014, 90: 290-319.