Journal of System Simulation

Volume 33 | Issue 9 Article 13

9-17-2021

Exploring Formal Model Transformation Techniques within Model Driven Engineering

Zhu Zhi
College of System Engineering, National University of Defense Technology, Changsha 410073, China;

Lei Sen
College of System Engineering, National University of Defense Technology, Changsha 410073, China;

Yonglin Lei
College of System Engineering, National University of Defense Technology, Changsha 410073, China;

Exploring Formal Model Transformation Techniques within Model Driven Engineering

Abstract

Abstract: With the increasing complexity of simulation system and the wide use of simulation models, the higher requirements of the efficiency and quality for simulation models are needed. Currently, model-driven engineering is mostly applied in many simulation software tools, which cannot really carry out the formal analysis at the model level. *Based on model driven engineering, the domain specific language with metamodeling and engineering model continuity is designed. Taking a group fire control channel system as the example, the domain specific language is designed and the conceptual models are transformed into other precise semantics to carry out the final executable simulations*. The research results can effectively guide the model transformation, which can improve the efficiency and quality of simulation models.

Keywords

model driven engineering, metamodeling, model transformation, language engineering

第 33 卷第 9 期 系统仿真学报© Vol. 33 No. 9
2021 年 9 月 Journal of System Simulation Sep. 2021

基于模型驱动工程的形式化模型转换技术

朱智，雷森，雷永林
(国防科技大学 系统工程学院，湖南 长沙 410073)

摘要：随着仿真系统的复杂性及其仿真模型在各个领域的广泛应用，用户对仿真模型的开发效率和质量提出了更高的要求。针对当前模型驱动工程主要应用于有关仿真软件工具的工程化实现，尚未达到模型层面而未能进行形式化分析，*基于模型驱动工程，从元建模、模型转换两个方面设计了领域特定语言及形式化的模型转换体系，以火控通道控制系统为例，将概念模型自动化转换为具有精确语义形式体系表示的可执行仿真模型，直到最终代码框架的生成。*研究成果具有通用性，能为各类模型转换工作提供有效指导，对提升仿真模型的开发效率和质量有重要意义。

关键词：模型驱动工程；元建模；模型转换；语言工程
中图分类号：TP391 文献标志码：A 文章编号：1004-731X (2021) 09-2119-09
DOI: 10.16182/j.issn1004731x.joss.20-0374

Exploring Formal Model Transformation Techniques within Model Driven Engineering

Zhu Zhi, Lei Sen, Lei Yonglin
(College of System Engineering, National University of Defense Technology, Changsha 410073, China)

Abstract: With the increasing complexity of simulation system and the wide use of simulation models, the higher requirements of the efficiency and quality for simulation models are needed. Currently, model-driven engineering is mostly applied in many simulation software tools, which cannot really carry out the formal analysis at the model level. *Based on model driven engineering, the domain specific language with metamodeling and engineering model continuity is designed. Taking a group fire control channel system as the example, the domain specific language is designed and the conceptual models are transformed into other precise semantics to carry out the final executable simulations.* The research results can effectively guide the model transformation, which can improve the efficiency and quality of simulation models.

Keywords: model driven engineering; metamodeling; model transformation; language engineering

引言

传统的系统仿真模型基于通用建模语言进行描述，缺乏精确的、非歧义的语义表示，或者通过通用性编程语言编写的仿真器在执行该模型时语义才得以表达，而没有在模型层面得到很好的定义，亟需针对特定的领域定义一整套较为通用的形式化模型转换体系。该模型转换体系能将各种建模语言描述的概念模型装换为具有精确语义形式体系表示的仿真模型，最后生成可执行仿真模型，乃至最终通用编程语言表示的代码，并集成到特定仿真系统或平台，从真正意义上工程化实现仿真模型的组合重用。在这个过程中主要涉及到 3 个建模仿真领域广泛研究的问题，模型组合、模型异构和模型连续性。

模型组合着重于仿真建模中各模型之间在语法和语义上的匹配与关联，相对于模型异构和模型

收稿日期：2020-06-17 修回日期：2020-08-17
基金项目：国家自然科学基金(61273198)
第一作者：朱智(1989-)，男，博士，讲师，研究方向为仿真建模。E-mail：zhuzhi@nudt.edu.cn

连续性，它强调模型驱动的系统开发中诸多模型之间的集成，形成有效、有意义的仿真应用[1]。模型异构是探讨在语言定制的过程中不同建模语言在语法上的不兼容，通常引起模型异构的因素来源于不同的系统组成部分，以及仿真模型的不同抽象层次、不同的视角、不同的开发阶段都需要不同的建模方法或建模规范。模型连续性主要是研究模型转换中转换前后源模型与目标模型的语法、语义一致性，涉及到目标模型的语法正确性、模型转换的完整性和语义正确性，区别但又与模型组合、模型异构紧密关联。

本文就模型连续性探讨设计形式化的模型转换体系，基于该模型转换体系从概念建模、模型定义和模型实现 3 个阶段定义仿真模型之间的转换规则，实现概念模型到可执行模型，乃至最终代码的自动化或半自动化实现过程。并以火控通道控制系统(Group Fire Control Channel System，GFCCS)[2]为例，实现其到并行 DEVS(Parallel-Discrete Event System Specification，P-DEVS)、JAVA，再到代码生成(Code Generation)的过程。最后，根据模型转换正确性、完整性和终止性 3 个基本评价标准，分析评估了该模型转换过程。

1 模型转换

1.1 模型转换基本概念

形式化的模型转换要求参与转换的模型是由良好的建模语言定义的，且转换的规则也是由良好的模型转换语言定义，这样才能保证模型转换是在良好的模型转换模板下进行。在模型转换过程中，为了保持模型的连续性，目标模型应尽可能包含源模型的信息以及保留初始的建模关系[3]。图 1 是典型的模型转换基本模式。

根据目标模型的具体表现形式，模型转换有 2 种类型：模型到模型(Model to Model，M2M)和模型到文本(Model to Text，M2T)。在 M2T 转换中，当文本是源代码时，也称为代码生成。一个模型驱动开发(Model-Driven Development，MDD)过程通常包括一系列的 M2M 转换和最终的一个 M2T 转换。

源模型与目标模型的元模型是同一个元模型，称该模型转换是内生的，反之称该模型转换是外生的模型转换[4]。源模型和目标模型在同一个抽象层次上，则称该模型转换为横向的模型转换；反之，该模型转换称为纵向的模型转换。同一个源模型可能会有不同类型的模型转换，但它们都尽可能的保留源模型信息，以最大限度地重用已有的模型信息。

为了自动生成 MDD 中特定系统在不同抽象层次上的表达，以及重用已有的模型信息，模型转换语言的定义在模型转换中发挥了重要的作用。模型转换语言用于编写形式化的转换规则，给定源语言和目标语言的语法以及由源语言定义的源模型，模型转换工具的语言解析器通过解析这些转换规则，并解释源模型，而模型转换引擎在该源模型上运用转换规则，根据解释器的结果生成目标模型。如果模型转换中的模型表现形式是图，那么此时的模型转换称为图形转换，模型转换工具称为图形转换工具[5]。典型的 M2M 转换语言如声明式和命令式的模型转换语言 QVT(Query/View/Transformation)[6]，ATL(ATLAS Transformation Language)[7]。

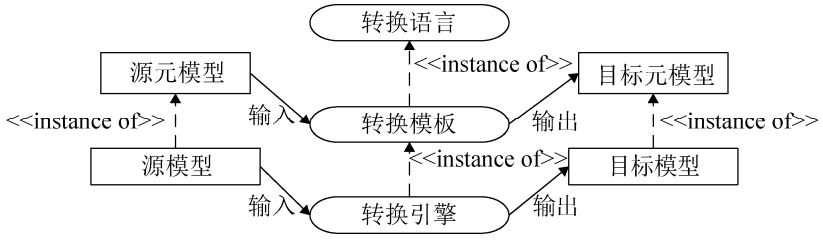

图 1 模型转换基本模式
Fig. 1 Basic mode of model transformation

DOI: 10.16182/j.issn1004731x.joss.20-0374

1.2 模型转换的评价标准

同一个源模型可以有多种形式的模型转换，不同的模型转换显然具有不一样的评价标准，如何选择合适的模型转换是 MDD 中至关重要的环节。寻求统一的向导式规则以供选择最佳的模型转换方式显然是不现实的，但可以借鉴软件工程中一些基本评价标准：

正确性：模型转换的正确性包括语法和语义两方面。当且仅当目标模型符合目标元模型规范，那么该模型转换是语法正确的；当且仅当模型转换保留了源模型的语义，那么这个模型转换是语义正确的；

完整性：对于源模型的每一个元素，如果在目标模型上都能找到与之对应的模型元素，那么这个模型转换具有完整性；

终止性：模型转换总能停止并输出一个结果；

唯一性：模型转换对特定的源模型都能生成唯一的目标模型；

可读性：模型转换规则是可读的、可理解的；

效率：模型转换包括多少个转换步骤，运用了多少个转换规则；

可维护性：模型转换在更改、扩展和重用方面的能力；

可拓展性：模型转换扩展到处理大范围模型的能力，且不以牺牲模型转换的性能为代价；

精确性：模型转换管理不完整的源模型、处理模型转换中可能发生的错误的能力；

鲁棒性：模型转换处理突发错误、管理无效源模型的能力；

有效性：能对模型转换进行系统测试和校验，以确定其是否具有正确的行为；

一致性：模型转换能检测并解决内部的冲突和不一致；

可追溯性：在模型转换过程中的各个阶段，源模型和目标模型对应元素之间存在记录链接；

可逆性：模型转换从 s 转换为 t，如果也能从 t 转换到 s，那么该模型转换是可逆的，这个属性多用于取消模型转换。

正确性、完整性和终止性是评判某个模型转换是否成功的基本标准。首先，一个模型转换应当能确保语法、语义上的正确性；其次，为了尽可能保留和重用源模型的信息，模型转换应当具备完整性，模型转换的完整性通常通过检测它是否覆盖到源元模型和目标元模型的所有元素；最后，模型转换语言的解析器一般能提供模型转换的终止性，故很多时候只关注模型的正确性和完整性。

另外，MDD 涉及以下几个方面的特性：

抽象性：提高模型的抽象层次，以使其更接近问题域，不用考虑具体的技术实现细节；

元建模：基于元模型形式化定义建模语言；

模型转换：定义模型映射机制，实现模型之间的转换，包括代码生成；

自动化：利用计算机技术以自动化地生成模型、建模工具、源代码以及各类文件；

通用性：不依赖与特定的具体技术、工具或平台。

1.3 仿真模型开发过程

软件开发过程中各个阶段所使用的语言不尽相同。根据抽象层次和主要目标，可以将软件开发过程大致分为需求、分析、设计、编码和测试 5 个阶段[8]。在建模与仿真领域存在很多仿真模型开发过程[9]，这些过程中的各个阶段明显区分，以提高仿真模型的开发效率。在不同的应用中，各个阶段以及各个阶段的输入输出、参与人员等虽然有所不同，但都着重从 3 个阶段解读仿真模型开发过程，即概念建模、模型定义和模型实现，如图 2 所示。

1.3.1 概念建模

在需求分析阶段，用户给出仿真应用系统的目标、需求，以及设定仿真应用系统的边界，确定影响仿真应用系统性能的关键因素。需求分析一般可以通过专门的需求定义语言来描述[10]，并在用户与概念建模人员开始交互时，能有一个初步的解决方案，这个阶段输出项目开发计划。

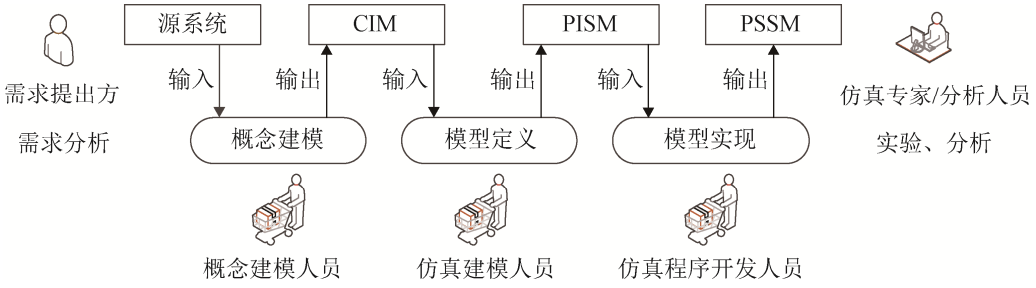

图 2 仿真模型开发过程

Fig. 2 Process of simulation model development1.

当用户给出了需求，启动仿真模型的开发，下一步就是概念建模人员根据仿真应用系统的概览定义一个高层的抽象模型，这类模型与现实层面的数据结构和操作无关，即计算无关模型(Computation Independent Model，CIM)。概念模型是不可执行的系统高层抽象，即根据所采用的概念建模语言的语义，没有相应的算法解释其模型的行为[11]。

1.3.2 模型定义

在用户和概念建模人员对所建立的 CIM 达成一致意见后，仿真建模人员将会根据特定的系统定义语言，如 DEVS、Petri 网、偏微分方程等，把 CIM 转换为形式化的模型。这类模型是对系统的规约，包括系统的体系结构、数据结构以及操作的抽象设计，如果存在一个能够执行该模型的抽象机，那么该模型的结构和行为应该符合这项规约。由于这类模型的设计并不考虑具体的实现技术细节，只涉及到系统的抽象计算，也叫平台无关仿真模型(Platform Independent Simulation Model，PISM)，对应 MDA 中的 PIM。PISM 是 CIM 中有关的过程和活动的数学表示，还包括执行这些过程和活动的数据资源。根据其数学描述，PISM 能被人工地执行，然而要能在特定的仿真平台上运行，还需要具体的技术来实现系统的体系结构、数据结构、操作和配置，形成对于系统的另一项规约。

1.3.3 模型实现

当概念建模人员和仿真建模人员关于 PISM 达成一致意见后，仿真编程人员将会基于特定的平台编写平台相关仿真模型(Platform Specific Simulation Model，PSSM)，对应 MDA 中的 PSM。一般地，PSSM 能自动生成最终的可编译或可执行的仿真模型(Simulation Model，SM)，注意 PSSM 与 SM 是在同一个抽象层次上不同侧面上的表达。PSSM 用比特定编程语言高一个抽象层次的建模语言定义，SM 通常是用编程语言生成或编写的可执行代码。在模型实现阶段，通过 SM 的运行、测试，以验证仿真模型是否正确、是否准确地表达源系统的行为。

当 SM 验证正确后，仿真专家就会设计仿真实验在仿真器上执行 SM，以校验仿真模型的行为是否与最初仿真模型的开发目标一致，最终得到仿真实验结果。仿真实验的设计包括想定的编辑、实验数据的准备、实验因子和实验响应的选取、脚本的定制等内容，这些内容形成一个实验配置文件，也叫仿真实验模型(Simulation Experiment Model，SEM)。

2 形式化的模型转换体系

MDD 过程包括一个初始模型、多个中间模型和最终源代码，主要是通过具有连续性的模型转换从某一个初始模型获得多个中间模型，最后生成源代码。

【定义 1】MDD 过程：

记多元组，$mdd=\{n, MML, ML, MO, SL, pl, MTP, STP, MT, sc, TO\}$为一个 MDD 过程，其中：$n$ 为中间模型的个数；$MML=\{ll_0, ll_1, \cdots, ll_{n-1}\}$ 为元建

DOI: 10.16182/j.issn1004731x.joss.20-0374

模语言的有序集合； $ML=\{l_0(mm_0), l_1(mm_1),\cdots, l_{n-1}(mm_{n-1})\}|\text{conform to } mm_i, ll_i\ (0\leqslant i\leqslant n)$为建模语言的有序集合； $MO=\{m_0,m_1,\cdots,m_{n-1}\}|\text{conform to } m_i, l_i\ (0\leqslant i\leqslant n)$为模型的有序集合，其中， m_0 为初始模型， m_{n-1} 为最终模型； *SL* 为附加语言集，包括至少一门模型转换语言，以编写转换规则； *pl* 为所生成代码的编程语言； $MTP=\{p_0,p_1,\cdots,p_{n-1}\}$ 为模型转换模式集，其中， $p_i\ (0\leqslant i\leqslant n)$是模型转换模式， p_{n-1} 是代码生成模式，且 $p_{n-1}=\{l_{n-1}(mm_{n-1}), pl, r\}\in MTP$ ，即至少包含一个代码生成模式； *STP* 为附加的模型转换模式集； $MT=\{\text{transform}(x,p)=y\,|\,x\in M\wedge p\in MTP\}$ 为模型转换集，其中， $(\text{transform}(m_{n-1},p_{n-1})=sc)\in MT$ ，即至少包含一个代码生成； *sc* 为最终代码； *TO* 为工具集。

【定义 2】MDA 过程：

记多元组， *mda*={*n*, *MML*, *ML*, *MO*, *SL*, *pl*, *MTP*, *STP*, *MT*, *SM*, *TO*}为一个 MDA 过程，其中： $n=3(\text{CIM},\text{PIM},\text{PSM})$ ； $MML=\{ll_0,ll_1,ll_2\}$ 为元建模语言有序集； $ML=\{l_0(mm_{\text{CIM}}),l_1(mm_{\text{PIM}}),l_2(mm_{\text{PSM}})\}$ ，有 $\text{confrom to}\ (mm_{\text{CIM}},ll_0)$ ， $\text{confrom to}\ (mm_{\text{PIM}},ll_1)$ ， $\text{confrom to}(mm_{\text{PSM}},ll_2)$ ； *MO*={CIM, PIM, PSM}，CIM 为初始模型， PSM 是最终模型，且 $\text{instance of}\ (\text{CIM})=mm_{\text{CIM}}$ ， $\text{instance of}\ (\text{PIM})=mm_{\text{PIM}}$ ， $\text{instance of}\ (\text{PSM})=mm_{\text{PSM}}$ ； *SL* 为模型转换语言集； *pl* 为具体的编程语言，MDA 过程最后的模型表现形式； $MTP=\{p_{\text{CIM}},p_{\text{PIM}},p_{\text{PSM}}\}$ ， $p_{\text{CIM}}=\{l_0(mm_{\text{CIM}}), l_1(mm_{\text{PIM}}),r_0\}$ ， $p_{\text{PIM}}=\{l_1(mm_{\text{PIM}}),l_2(mm_{\text{PSM}}),r_1\}$ ， $p_{\text{PSM}}=\{l_2(mm_{\text{PSM}}),pl,r_2\}$ ； *STP* 为附加的模型转换模式； $MT=\{\ \text{transform to}\ (CIM,\ p_{\text{CIM}})=\text{PIM}$ ， $\text{transform to}(\text{PIM},p_{\text{PIM}})=\text{PSM}$ ； $\text{transform to}(\text{CIM},p_{\text{PSM}})=\text{SM}\ \}$ ；SM 为最终可执行仿真模型； *TO* 为工具集。

上述形式化模型转换理论体系实质上是 MDD 在仿真建模过程中的应用，因此根据 MDD 的 5 个基本要求，对该理论体系进行总结和评价：

抽象性：MDA 过程包含了 4 类模型，即 CIM，PIM，PSM，SM。这些模型各自处在不同的抽象层次上，例如，CIM 没有执行语义，最靠近问题域；PSM 有执行语义，但没有实现细节；PSM 两者兼有。因此，上述 MDA 过程的形式化定义满足抽象性的要求；

元建模：MDA 过程基于元建模定义建模语言，至少存在 3 种元模型，即 CIM 元模型、PIM 元模型和 PSM 元模型，附加一些描述模型转换规则的元模型。因此，该 MDA 过程满足元建模的需求；

转换：MDA 过程定义了 3 种模型转换，即 CIM 到 PIM，PIM 到 PSM，PSM 到 SM 的转换，每一个转换都需要通过模型转换语言编写转换规则。因此，该 MDA 过程满足转换的需求；

自动化：MDA 过程的形式化定义中，最后一个元素 *TO* 即是描述支持该过程的工具集，通过这些工具以及现成的元建模环境可以自动化地生成模型、建模工具和源代码等，也能通过模型转换工具自动生成模型。因此，该 MDA 过程满足自动化的需求；

通用性：该 MDA 过程基于 MDA 提供了仿真模型开发的通用方法，独立于具体形式体系和应用平台，能比较好地适应特定的平台和技术进行定制。因此，该 MDA 过程满足通用性的需求。

3 火控通道控制系统仿真实现

3.1 火控通道控制系统仿真建模

指定 GFCCS DSL 为概念建模语言以定义概念模型，P-DEVS 为建模形式体系以定义 PIM，JAVA 作为底层的编程语言以定义 PSM。因此，GFCCS 的仿真建模过程包括以下几类元模型和模型转换：

(1) CIM 元模型是 GFCCS 元模型；

(2) PIM 元模型是 DEVS 元模型；

(3) PSM 元模型是 JAVA 元模型；

(4) CIM-PIM 转换是 GFCCS-DEVS 转换；

(5) PIM-PSM 转换是 DEVS-JAVA 转换；

(6) PSM-SM 转换是 JAVA-java 代码转换。

【定义 3】GFCCS 仿真建模过程：

记多元组，$gfccs=\{n, MML, ML, MO, SL, pl, MTP, STP, MT, SM, TO\}$ 为一个 GFCCS 仿真建模过程，其中：$n=3$(CIM, PIM, PSM)；$MML=\{Ecore, Ecore, Ecore\}$ 是元建模语言的有序集；$ML=\{l_0(mm_{\text{GFCCS}}), l_1(mm_{\text{DEVS}}), l_2(mm_{\text{JAVA}})\}$，confrom to $(mm_{\text{GFCCS}}, Ecore)$；confrom to$(mm_{\text{DEVS}}, Ecore)$；confrom to $(mm_{\text{JAVA}}, Ecore)$；$MO=\{\text{CIM, PIM, PSM}\}$，其中 CIM 为初始模型，PSM 是最终模型，有 instance of $(\text{CIM})=mm_{\text{GFCCS}}$，instance of $(\text{PIM})=mm_{\text{DEVS}}$，instance of $(\text{PSM})=mm_{\text{JAVA}}$；$SL=\{ATL, Acceleo\}$ 为模型转换语言集；$pl=$JAVA 为最后的编程语言；$MTP=\{p_{\text{CIM}}, p_{\text{PIM}}, p_{\text{PSM}}\}$，有 $p_{\text{CIM}}=\{l_0(mm_{\text{GFCCS}}),\ l_1(mm_{\text{DEVS}}), gfccs2devs.atl\}$；$p_{\text{PIM}}=\{l_1(mm_{\text{DEVS}}),\ l_2(mm_{\text{JAVA}}), devs2\,java.atl\}$；$p_{\text{PSM}}=\{l_2(mm_{\text{JAVA}}),$ JAVA, $java2code.mtl\}$；$SIP=\varnothing$ 为附加的模型转换模式；$MT=\{$transform to $(\text{CIM}, p_{\text{CIM}})=\text{PIM}$，transform to $(\text{PIM}, p_{\text{PIM}})=\text{PSM}$，transform to$(\text{CIM}, p_{\text{PSM}})=SM\}$；$SM$ 为最终可执行仿真模型；$TO=\{ATL, Acceleo, EMF, GMF, Eclipse\_IDE\}$。

3.2 具体实现过程

基于 MDA 的 GFCCS 仿真建模过程分为 3 个具体过程：GFCCS 到 P-DEVS(CIM-PIM)[12]，P-DEVS 到 JAVA(PIM-PSM)，JAVA 模型到源代码(PSM-SM)，如图 3 所示。GFCCS 到 P-DEVS 转换的模型元素匹配关系如表 1 所示，P-DEVS 到 JAVA 转换如表 2 所示。

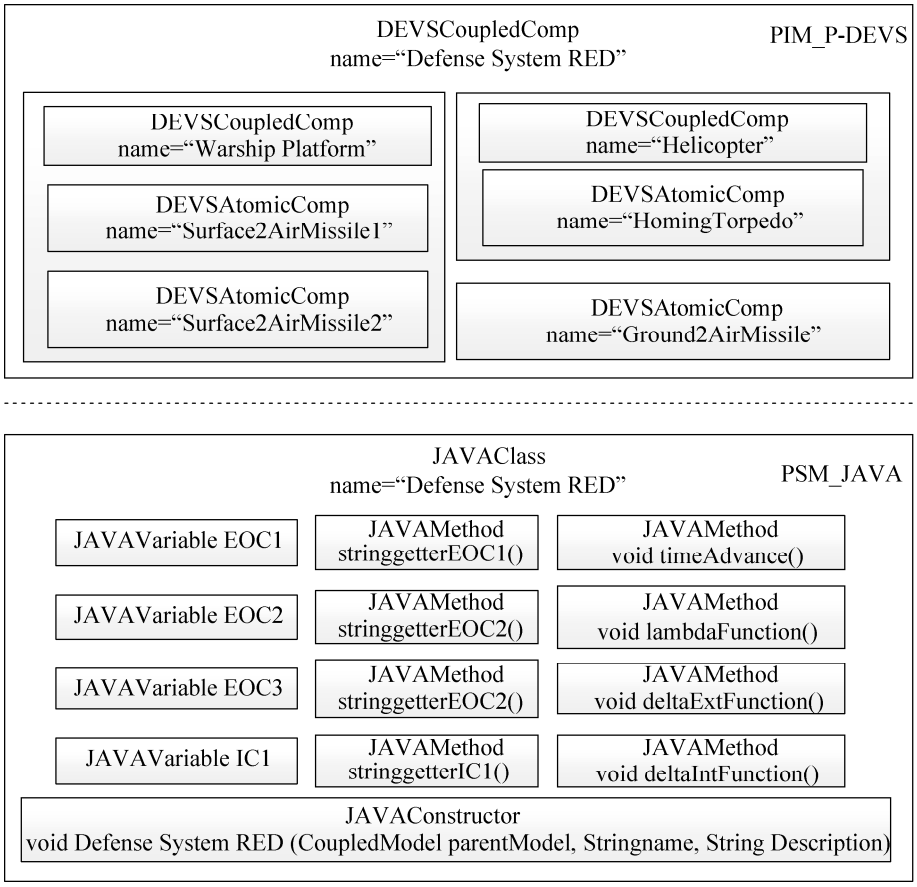

图 3 基于 MDA 的 GFCCS 仿真建模过程示例

Fig. 3 A sample of GFCCS to P-DEVS to JAVA transformations based on MDA

DOI: 10.16182/j.issn1004731x.joss.20-0374

表 1 GFCCS 到 P-DEVS 转换匹配关系
Tab. 1 Matching rules of GFCCS to P-DEVS transformation

GFCCS 元模型	P-DEVS 元模型
GroupFireControlSystem	DEVSModel
Group	DEVSCoupledComp
GroupNode	DEVSCoupledComp
Channel	DEVSAtomicComp
Weapon	DEVSAtomicComp
Target	DEVSAtomicComp
COPShare	DEVSOutToIn_ICConnection
disjointWith	DEVSOutToIn_ICConnection
affliated	DEVSOutToIn_ICConnection
fireControl	DEVSOutToOut_ICConnection + Source.out: DEVSOutputPort + Target.in: DEVSOutputPort + SourceParents.EOC Ports: DEVSOutputPort
mutualExclusive	DEVSOutToIn_ICConnection
taretList	DEVSOutToIn_ICConnection
assigned	DEVSOutToOut_ICConnection + Source.out: DEVSOutputPort + Target.in: DEVSOutputPort + SourceParents.EOC Ports: DEVSOutputPort
weaponList	DEVSInToIn_ICConnection + Source.out: DEVSInputPort + Target.in: DEVSInputPort + SourceParents.EIC Ports: DEVSInputPort

表 2 P-DEVS 到 JAVA 转换匹配关系
Tab. 2 Matching rules of P-DEVS to JAVA transformation

P-DEVS 元模型	JAVA 元模型
DEVSModel	JAVAPackage + javaClasses: JAVAClass + javaConstructors: JAVAConstructor + javaExpressions: JAVAExpression
DEVSCoupledComp	JAVAClass +JAVAVariable + javaConstructors: JAVAConstructor + javaExpressions: JAVAExpression
DEVSAtomicComp	JAVAClass +JAVAVariable + javaConstructors: JAVAConstructor + javaExpressions: JAVAExpressi
DEVSInputPort	JAVAVariable
DEVSOutputPort	JAVAVariable
StateVariable	JAVAVariable
DEVSOutToIn_ICConnection	JAVAExpression
DEVSInToIn_EICConnection	JAVAExpression
DEVSOutToOut_EOCConnection	JAVAExpression
Expression	JAVAExpression
DeltaIntFunction	JAVAMethod
DeltaExtFunction	JAVAMethod

4 GFCCS 实现的 MDD 评估

根据前面列出的模型转换评价标准，结合 GFCCS 到 P-DEVS、到 JAVA、再到源代码的转换过程，首先能轻易看出该实现过程满足了终止性、唯一性和可读性。然而，上述示例毕竟只是个例，因此不完全满足对有效性、可维护性、可拓展性以及重用性的要求，这类标准一般需要更多的案例进行论证评估。再者，由于生成的模型编辑器通常会自动检测模型的正确性，所以精确性和一致性可以得到充分保证。而鲁棒性在模型转换语言编译器中得到保证。最后，考虑到模型转换能否成功有效地进行，聚焦于分析模型转换的正确性和完整性，着重从 3 个方面进行分析。

(1) 目标模型的语法正确性

前面提到，如果目标模型符合目标元模型，那么称该模型转换在语法上是正确的。然而，在编写模型转换规则时所使用的语言和编译器并不能确保目标模型的正确性，所以必须人为地确保目标模型的正确性。例如，在编写模型转换规则时可以为某一变量赋多种类型的值，但目标元模型并不总是能支持所有的种类。所以，确保目标模型的正确性通常是通过仔细检查转换规则，或者通过增强规则编写工具的功能。而在本例中，可以通过在 Eclipse 集成开发环境中注册元模型，然后验证得到的目标

模型是否符合注册的元模型，从而得出该模型转换在语法上是否正确。

(2) 模型转换的完整性

完整性一般通过测量源元模型和目标元模型的模型元素用于模型转换的覆盖率，前者指模型转换中用到的模型元素在源元模型中所有模型元素的比例，而后者则指模型转换中用到的模型元素与目标元模型中的所有模型元素的比例。源元模型覆盖率体现模型转换中应用到的源元模型中的模型元素量，而目标元模型覆盖率则体现所生成的目标模型的准确性。本章的示例中，各个 ATL 编写的转换引擎完全覆盖了源元模型和目标元模型的所有元素，而对于代码生成，只提供了源元模型覆盖率，而没有体现目标元模型覆盖率，这是因为仅用到了 JAVA 编程语言的一小部分元素。因此，该模型转换提供了完整性以保留和重用源模型中的结构和信息。在转换中，在源模型和目标模型中也确保了标签的相似性，假设在现有信息基础上的增加或扩展不会引起信息的丢失，例如名为“Defense System RED”的模型组件在代码生成中转换为了“Defense_System_RED”，本文认为该转换能保留信息，因为在该转换中想要获得原来的名字也是简单可行的。

(3) 模型转换的语义正确性

假设模型 a 转换为模型 b，那么模型 b 需要保留模型 a 的语义，即模型转换的语义正确性，因为只有连续的模型转换才是有用和有意义的。语义正确性可以通过执行语义映射并对比映射结果来核查，通常有不同的方法来检查语义正确性，如追踪等值、双向仿真和行为等值。

在追踪等值法中，模型的行为被看成是一系列踪迹，模型的踪迹指一个可能的执行片段，它通常是标签的有序列表，表示在一个执行片段中按时间序列发生的事件。通常认为，当且仅当 2 个模型产生相等的踪迹集合时，这 2 个模型被认为是踪迹等值的，即将 $Tr(m)$ 为模型 m 的踪迹集合，其元素称为 m 的一个踪迹，m_1 和 m_2 踪迹等值，当且仅当 $Tr(m_1) \cong Tr(m_2)$ [13]。另外，如果 $Tr(m_1) \subseteq Tr(m_2)$，有 $Tr(m_1) \cong Tr(m_2)$。本章示例中的 GFCCS 到 P-DEVS 的转换，前者的每一个部分都映射到了一个或多个后者的元素，所以其语义是正确的。而 P-DEVS 到 JAVA 的转换是根据 P-DEVS 的操作语义执行的，P-DEVS 模型的语义能够在 JAVA 模型中得到完全保留。最后，代码生成是一个一一映射的过程，显然其也是语义正确的。综上所述，本章示例中所有的模型转换是具备完整性和正确性的，能充分确保模型的连续性。

5 结论

模型转换是提高仿真模型开发效率和质量的重要途经，良好的模型转换需要保持转换前源模型的结构、信息在转换后的目标模型中得到重用，以减少仿真模型重复开发带来的不必要开销。本文在介绍模型转换概念及其基本模式的基础上，基于 MDA 总结了通用的仿真模型开发过程，并建立了形式化的模型转换理论体系，以支撑良构的模型转换过程。此外，基于上述定义实现了 GFCCS 概念模型到最终代码的半自动化转换过程，分析结果表明该模型转换在很大程度上能支持模型连续性，但也需要更多的应用和实验来验证其可维护性、可拓展性以及有效性。

参考文献：

[1] 张霖, 周龙飞. 制造中的建模仿真技术[J]. 系统仿真学报, 2018, 30(6): 1997-2012.
Zhang Lin, Zhou Longfei. Modeling & Simulation Technology in Manufacturing [J]. Journal of System Simulation, 2018, 30(6): 1997-2012.

[2] 雷永林, 李群, 杨峰, 等. 武器装备效能仿真的可组合建模框架研究[J]. 系统工程理论与实践, 2013, 33(11): 2954-2966.
Lei Yonglin, Li Qun, Yang Feng, et al. A Composable Modeling Framework for Weapon Systems Effectiveness Simulation [J]. Systems Engineering-Theory & Practice, 2013, 33(11): 2954-2966.

[3] Ehrig H, Ermel C. Semantical Correctness and Completeness of Model Transformations Using Graph and Rule Transformation[C]// 4th International Conference on Graph Transformations. Leicester, UK:

https://dc-china-simulation.researchcommons.org/journal/vol33/iss9/13
DOI: 10.16182/j.issn1004731x.joss.20-0374

Springer, 2008: 194-210.

[4] Agarwal S, Dixit S, Aggarwal A. Model to Model Transformation for Declarative Models[J]. International Journal of Computer Science and Engineering (S2347-2693), 2018, 6(11): 164-170.

[5] Ege F, Tichy M. A Proposal of Features to Support Analysis and Debugging of Declarative Model Transformations with Graphical Syntax by Embedded Visualizations [C]// ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion. Munich, Germany: IEEE, 2019: 326-330.

[6] Mukhtar M A O, Abdullah A, Downe A G. Preliminary Overview about Relations QVT: Query/View/ Transformation Model Transformation Language [C]// 2011 National Postgraduate Conference. Kuala Lumpur, Malaysia: IEEE, 2011: 1-5.

[7] Jouault F, Allilaire F, Bezivin J, et al. ATL: A Model Transformation Tool[J]. Science of Computer Programming (S0167-6423), 2008, 72(1/2): 31-39.

[8] Ghezzi C, Jazayeri M, Mandrioli D. Fundamentals of Software Engineering [M]. 2nd ed. NY: Prentice Hall, 2002: 58-60.

[9] Balci O. A Life Cycle for Modeling and Simulation [J]. Simulation (S2310-4791), 2012, 88(7): 870-883.

[10] 李伯虎，柴旭东，张霖，等. 面向新型人工智能系统的建模与仿真技术初步研究[J]. 系统仿真学报，2018, 30(2): 349-362.
Li Bohu, Cai Xudong, Zhang Lin, et al. Preliminary Study of Modeling and Simulation Technology Oriented to Neo-type Artificial Intelligent Systems [J]. Journal of System Simulation, 2018, 30(2): 349-362.

[11] Olive A. Conceptual modeling of information systems [M]. Berlin: Springer-Verlag, 2007: 98-101.

[12] Zhi Z, Lei Y L, Alshareef A, et al. Domain Specific MetaModeling for Deep Semantic Composability [J]. IEEE Access (S2169-3536), 2018, 6(1): 18276-18289.

[13] Nain S, Vardi M Y. Trace Semantics is Fully Abstract [C]//24th Annual IEEE Symposium on Logic in Computer Science. Los Angeles, CA: IEEE, 2009: 59-68.