中图分类号：TP311　　论文编号：1028716　14-S085
学科分类号：081200

硕士学位论文

基于 SysML 的可执行模型验证工具的研究与实现

研究生姓名　　俞晓锋
学科、专业　　计算机科学与技术
研究方向　　数据库系统及应用
指导教师　　王 立 松　副教授

南京航空航天大学

研究生院 计算机科学与技术学院

二〇一四年六月

Nanjing University of Aeronautics and Astronautics
The Graduate School
College of Computer Science and Technology

Research and Implementation of a Verification Tool For Executable Model Based on SysML

A Thesis in

Academic Master, Computer Science and Technology

by

Yu XiaoFeng

Advised by

Associate Prof. Wang liSong

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Engineering

June, 2014

承诺书

本人声明所呈交的硕士学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得南京航空航天大学或其他教育机构的学位或证书而使用过的材料。

本人授权南京航空航天大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

(保密的学位论文在解密后适用本承诺书)

作者签名：_____

日　　期：_____

摘 要

系统建模语言(Systems Modeling Language，SysML)的提出使得模型驱动开发(Model Driven Development，MDD)可以应用于系统工程领域。由于模型驱动开发对自动代码生成的支持，使得目标系统的验证在整个项目开发周期中的位置可以推前到系统的设计阶段进行。在 MDD 中，模型可分为平台无关模型和平台相关模型，通过验证平台无关模型(Platform Independent Model，PIM)来预览系统的行为功能可以减少系统工程实施的风险。在系统工程领域中，针对平台无关模型进行验证具有重要的意义，合理的平台无关模型可以组织成为高内聚低耦合的可复用组件，提高系统工程的质量和效率，大大减少测试目标系统的时间和开销。系统建模语言是统一建模语言(Unified Modeling Language，UML)的扩展，将会广泛应用到模型驱动开发中，但 SysML 还在不断完善中，使用原生 SysML 进行建模不能满足平台无关模型验证的需求，主要原因为：一方面，SysML 本身不具备可执行性；另一方面，缺少针对 SysML PIM 进行设计、分析和验证的工具原型。论文针对目前存在的问题和需求开展了相关研究和实践，主要工作如下：

(1) 在 SysML 规范的基础上定义了可执行的平台无关模型。它有一个清晰的模型结构并加入了动作规约语言进行增强，可执行性是进行平台无关模型验证的前提和基础；

(2) 设计实现了支持基于 SysML 的可执行平台无关模型的工具原型。用它可以创建和验证可执行的平台无关模型；

(3) 在可执行模型和工具原型的基础上实现了平台无关模型的验证功能。通过一个具体的模型实例，采用论文中的原型系统进行实验，体现了验证平台无关模型的可行性。

关键词：MDD， SysML，平台无关模型，可执行，验证

ABSTRACT

Model Driven Development (MDD) can be applied to system engineering result from Systems Modeling Language (SysML) proposed. Because of the supporting of automatic code generation in Model Driven Development (MDD), the position of verification for target system in the entire development liftcycle can be set in the design stage of system. In the field of MDD, model can be divided into Platform Independent Model and Platform Specifical Model. The behavior of the system function can be previewed through the verification of Platform Independent Model (PIM) which leads to the risk-reduction of the project enforcing. In the field of system engineering, validation of platform independent model has the vital significance. Reasonable platform independent model can be organized into reusable components of high cohesion and low coupling, it improves the quality and efficiency of system engineering and greatly reduces time and cost of testing target system. SysML extends Unified Modeling Language (UML) and will be widely applied to MDD. However, SysML still keeps improving, in the case, it can't meet the requirement of validation of platform independent model through using original SysML to build platform independent model simply. It suffers from a few reasons: On the one hand, SysML itself lacks the enforceability, On the other hand, there is still not an efficient SysML-based tool prototype supporting the design, analysis and verify for platform independent model. Aiming at these existing problems and needs, the paper carried out relevant research and practice. The main work is as follows:

(1) It defines an executable platform independent model on the basis of SysML specification. The model has a distinct architecture supporting action specification language. Enforceablity is the premise and foundation of platform independent model validation;

(2) Design and realize the modeling tool prototype supporting executable SysML model. It can create and validate an executable platform independent model;

(3) Implement the function of model verification on the basis of executable model and modeling tool prototype. The prototype system of the paper performs experimentation of a concrete model instance. The experimental results show the feasibility of verification of platform indepent model.

Keywords: MDD, SysML, platform independent model, executable, verification

目 录

图表清单

缩略词

缩略词	英文全称
OMG	Object Management Group
UML	Unified Modeling Language
INCOSE	International Council on Systems Engineering
SysML	Systems Modeling Language
MDA	Model Driven Architecture
PIM	Platform Independent Model
PSM	Platform Specific Model
MDD	Model Driven Development
ASL	Action Specification Language
UML	Unified Modeling Language
MOF	Meta Object Facility
EMF	Eclipse Modeling Framework
GEF	Graphical Editor Framework
GMF	Graphical Modeling Framework
MVC	Model View Controller
TDM	Tool Definition Model
GDM	Graph Definition Model
MM	Mapping Model
SCXML	State Chart XML
XMI	XML-based Metadata Interchage
XML	eXtensible Markup Language
W3C	World Wide Web Consortium
DTD	Document Type Definition
RUP	Rational Unified Process

第一章　绪论

1.1 课题研究目的

随着信息化产业的快速发展，软件开发技术不断进步。同软件开发的实现技术一样，业务需求的变化同样很快。快速的技术进步和业务需求变化的剧烈融合导致了人们努力从实现业务需求的技术中分离出业务需求的模型。为了实现这些目标，国际对象管理组织(Object Management Group，OMG)提出了基于模型驱动的软件开发方法[1]，它提供了一种开放、通用的方法和一组标准建模语言来建立系统。

统一建模语言[2] (Unified Modeling Language，UML)已经成功地应用于软件开发领域。它在软件开发领域带来的好处促使人们希望打造出一种针对系统工程领域的建模语言。为了满足系统工程的这种需求，国际系统工程学会 (International Council on Systems Engineering，INCOSE)和 OMG 提出了系统建模语言[3] (Systems Modeling Language，SysML)。UML 提供了三种扩展结构：约束(Constraints)、构造型(Stereotypes)、标签值(Tag)，它们都可以在不改变 UML 元模型的前提下对 UML 进行扩展。SysML 是 UML 在系统工程领域的扩展。

模型驱动开发(Model Driven Development，MDD)是一种基于模型转换[4]的开发方法，整个系统起源于模型。在系统工程领域的 MDD 中，开发者在一个基于 SysML 的抽象层上生成适当的开发工件，这些工件以 SysML 模型的形式存在。在 MDD 中有两种 SysML 模型的基本类型：平台无关模型[5] (Platform Independent Model，PIM)和平台相关模型(Platform Specific Model，PSM)，建模人员构建的是平台无关模型，平台相关模型按照一定的映射规则[6][7]从平台无关模型转换而来，最终生成目标系统。

- 平台无关模型

PIM 获取和表达目标系统的所有业务需求，是系统功能、行为和结构的形式化表示。每个 PIM 对应一个主题事务域，并且不被其它的主题事务和具体的实现技术细节所“污染”，独立于任何实现平台。在 MDA 中，“平台”是指与软件基本功能无关的技术与工程细节。通过从平台和实现相关问题中抽象出来，PIM 将注意力放在业务问题上。这使得 PIM 更容易被客户理解，更容易在业务需求变更时进行维护，并且更容易应用到任何平台和实现技术上。“形式化”要求构建模型的建模语言具有良定义的语法语义，目前用于构建 PIM 的建模语言主要是 UML 和 SysML。由于 PIM 的目标是抽象出系统的业务逻辑，所以在使用 UML 和 SysML 对系统进行面向对象分析时，描述模型的静态结构和动态行为等建模工作都必须严格限制在业务逻辑的范畴之内。

- 平台相关模型

PSM 是一种已被具体实现技术细节所“污染”的主题事务模型。它包含了 PIM 表示的所有内容。PSM 主要有两种存在形式：可以是从 UML 或 SysML 元模型转换得到的原型代

码，也可以是某种具体的编程语言，比如 C/C++。由于 PSM 与特定的平台（比如 J2EE、.NET 等）关联，PSM 只对了解相关平台和开发环境的开发者有意义。平台相关模型通过映射规则从 PIM 转换而来，不需要单独开发，这是 MDA 的高级特性。常见的映射规则有 Marking、MetaModel Transform 以及 Model Transform。

PIM 和 PSM 在模型的合理性方面具有一致性。由于 MDD 中自动代码生成技术[8]的支持，目标系统的验证在整个系统开发周期中的位置可以推前到模型的设计阶段进行。在系统工程领域的模型驱动开发中，针对平台无关模型进行验证有以下几个优点：

(1) 组织大规模的高内聚低耦合的可复用组件

MDD 将分析模型看作理解所有主题事务的过程，理解的结果在可执行 SysML 模型中形式化地表示出来，引导开发者如何基于特定技术来实现目标系统。一个组件要成为可复用的基本原则是保证组件是需求合理的并且不能包含不同的主题事务以及具体实现技术。平台无关模型由 SysML 中的一个域来表示，每个域都维护着独立的主题事务信息，具有域内高内聚，域间低耦合的特点，平台无关性使得 PIM 与具体实现技术无关。通过平台无关模型的验证可以产生出高质量的分析模型，开发者将这些合理的分析模型组织成大规模的高内聚低耦合的可复用组件，提高系统工程的开发效率。

(2) 提高系统工程的质量

建模最终目的是为了产生实际可用的系统。人们希望系统表现出的行为能够符合预期要求。然而，随着应用系统功能越来越复杂，系统运行更加偏向自动化。复杂性导致人们很难全面地考虑到系统的潜在问题，自动化在缓解人工作业压力的同时也给系统带来了不稳定的因素。系统工程领域有别于软件开发，为系统建立 SysML 模型需要建模人员和领域专家共同合作完成，领域专家掌握着系统所有的主题事务，他的职责主要是保障 SysML 模型在业务表现方面的合理性。平台无关模型与实现技术无关，它有助于建模人员与领域专家之间关于需求和设计进行有效沟通。在系统设计阶段，建模人员可以在领域专家的参与下完成平台无关模型的验证，提高系统工程的质量。

(3) 减少测试目标系统的周期和开销

在 MDD 过程中，平台无关模型是源头，其它阶段的模型产物都是由 PIM 转换而来。通过平台无关模型预览系统的行为功能可以发现系统存在的潜在问题，合理的 PIM 在最后生成实际系统之后可以大大减少测试目标系统的周期以及开销。

Shlaer 和 Mellor 在他们的“递归”设计方法[9]中强调了建立平台无关模型的价值，它在最后会被自动地转换生成应用系统。建模人员和领域专家在设计完 PIM 后可以实施 MDA 的整个开发过程。然而 SysML 作为面向对象的标准符号系统并不足以建立精确的平台无关模型。SysML 为了保持描述的清晰易懂，在给出自身语义说明以及描述 PIM 时都采用半形式化方法，该方法降低了平台无关模型的精确性，不利于 PIM 的自动分析和验证。主要问题体现在以下两点：

(1) SysML 表示法没有统一上下文环境。SysML 跟 UML 一样规定了一个图形语言，使得系统可以通过一组不同类型的图式系统化地定义。但 SysML 在这些图形的使用方法上并不正规，如状态模型用来表示用例、子系统以及对象。这就要求用户必须首先确定所读图形的上下文才能理解它。

(2) SysML 包含许多语义较弱的元素，不足以进行精确建模。简单使用所有 SysML 的表示法很容易产生出令人难以理解的模型，这样不利于处理问题的复杂性。并且 SysML 本身没有提供一个对 SysML 模型进行推理的合理机制。

在系统工程领域，由于 SysML 本身的不精确性，使用原生 SysML 建立 PIM 不能满足验证 PIM 的需求。因此 MDD 通常只是将 PIM 理解为设计编写文档的工具，而 PSM 则被认为表示最后真实的系统，它将重心放在了 PSM 阶段。

本文在研究国外商用软件建模技术的基础上设计实现了面向 SysML PIM 的系统建模工具，并实现 SysML PIM 的验证功能。由于原生 SysML 构建的平台无关模型不具备可执行性，本文利用 SysML 的子集和动作规约语言[10](Action Specification Language，ASL)定义了具有精确语义的 xSysML(eXecutable SysML)，工具原型在 xSysML 基础上建立具备特定模型结构的可执行[11]PIM，可执行 PIM 的语义相比于 ICE-PIM 更侧重于执行，能够满足针对平台无关模型进行验证的需求。在模型验证方面，本文通过分析 Rhapsody[12]状态图执行实现机制[13][14][15]，开发可执行框架用于验证可执行 PIM。

1.2 国内外现状

在系统工程领域中，为了降低项目的实施风险，通常有一个模拟系统全部或者部分行为功能的仿真过程，它建立在系统的抽象模型的基础上。当前有多种描述系统模型的方案和架构，常见的有 CIM-OSA(计算机集成制造-开放体系结构)，它为制造企业的系统集成和企业建模提供了方法论，面向过程的 ARIS(集成信息系统)和面向需求的 PERA(企业参考体系结构)都由 CIM-OSA 框架衍生出来，然而它们主要针对业务分析层次建模，并未提供如何实施系统设计的方法。

SysML 的提出提供了另一种建立系统抽象模型的方法，国内学者在 SysML 提出之后提出了相关综述[16]。此后基于 SysML 的研究主要集中在军工方面，例如使用 SysML 的现有功能与 DODAF[17][18](美国国防体系结构框架)相结合。SysML 是面向对象的建模语言，基于它的表示法能够建立通用的可视的抽象模型，SysML-MDA 方法贯穿从系统设计到实施的各个阶段。

SysML-MDA 中的模型形式分为 PIM 和 PSM 两种，当前针对 PIM 的验证主要基于模型转换来间接实现，使用 OCL 构建 PIM 后变成 ICE-PIM(可配置可互操作 PIM)，然而 ICE-PIM 中的精确语义主要是用于转换为 PSM，由于 PSM 增加了软件基本功能相关的技术与工程细节，它本身已经具备可执行性，通过执行 PSM 来完成模型仿真可以使用 Rhapsody、Rational

Rose 和 Enterprise Architecture 等建模工具来实现，这种验证思想并不是直接作用于 PIM，具备了一定的平台相关特性。下面是这三大主流建模软件的相关特点。

- Rational Rose

Rational Rose 是 Rational 公司开发的一款面向 UML 的可视化建模工具。Rational Rose 用于可视化建模和公司级水平软件应用的组件构造，主要面向软件工程领域。

Rational Rose 能够通过回溯和更新模型的其余部分来保证原型代码的一致性，从而展现出被称为"来回旅程工程"的能力，Rational Rose 是可扩展的，可以使用可下载附加项和第三方应用软件，它支持 COM/DCOM (ActiveX)，JavaBeans 和 Corba 组件标准。

- Rhapsody

Rhapsody 是行业领先的基于 UML 2.0 及 OMG SysML 的可视化编程开发环境。通过特有的将 UML 各类视图映射为具体目标机程序语言的技术，Rhapsody 提供了完整的复杂实时嵌入式应用软件开发环境。开发者可以创建跟踪链接、自动生成跟踪文档，并支持从多个源导入开发需求，具有对软件完整生命周期的跟踪能力。在开发需求捕获方面，Rhapsody 提供直观的需求捕获视图和多种需求定义与描述方式，同时能够对需求的覆盖和需求的变更情况进行分析，并通过执行和动画演示的方式对模型的原型代码进行合理性验证，最终生成可执行的完整应用程序。Rhapsody 的功能特点主要有以下三点：

(1) 实现了模型的可执行

Rhapsody 通过内置的 OXF[19]（对象可执行框架）实现模型动态行为的执行。通过状态图，顺序图、活动图的执行反映系统的动态行为。这样系统工程师，软件工程师和测试工程师可以对目标原型进行验证，通过模型的执行保证设计的高质量。

(2) 完整的、可定制代码自动生成

代码生成框架的最显著的特点是能够做到模型-代码的双向关联。代码和模型作为同一设计的不同展示而共同存在，在模型发生改变时，对应的代码文件得到更新，代码发生改变时，对应的模型信息也同时发生改变，这种设计保证了模型和代码的一致性。Rhapsody 提供了 150 多个属性用来定制代码的生成，其中包括对可执行程序的定制，指定特定的文件，指定需要进行编译的具体元素，指定代码生成风格，选择实时时钟和模拟时钟等。Rhapsody 还可以保存常用的配置，用于简化根据应用程序的不同需求来生成原型代码的过程。用户能够自定义 Rhapsody 的代码生成框架以满足特定的需求，这种灵活的代码生成机制能够在缩短开发周期的同时，提高代码质量。

(3) 目标原型的调试和验证

Rhapsody 的实时运行框架在生成的原型代码中添加了调试以及平台相关的结构信息，可以用于监视和控制生成的原型代码在任何特定平台的运行。开发人员只需在代码生成过程中配置工程属性，选择恰当的调试方式，就能够使用 Rhapsody 的代码调试功能。代码的调试模式有“动画”和“跟踪”两种模式。在“动画”模式中，状态机当前所处的状态会高亮

显示，在“跟踪”模式中，Rhapsody 最强大的功能是可以同时从多种角度对调试的程序段的动态信息进行展示。

- Enterprise Architecture

EA 是一个成本低廉、功能丰富的建模工具。它支持模型驱动的软件开发全生命周期，主要功能包括：UML 建模、BPMN 建模、模型仿真、应用程序执行与调试、双向代码工程、需求管理、项目管理、版本管理、测试点管理等，并且能够生成 PDF 格式、 RTF 格式和 HTML 格式的文档报告。EA 能够通过模型生成原型代码并且通过执行代码进行仿真验证，在整个开发过程中可以建立需求追踪关系，提高项目开发的进度。

这三款建模工具都是以目标原型为向导的软件系统，它们虽然支持基于 SysML 的 MDD，但它们将重点放在如何生成基于模型转换的目标原型上，在 PIM 映射到 PSM 之前并没有一个专门验证 PIM 的过程，对目标原型的补救主要通过重新生成和验证 PSM 来完成。

1.3 本人主要研究工作

针对没有专门支持平台无关模型验证的 SysML 建模工具的现状，本文在研究 ICE-PIM 的基础上采用 xSysML 构建可执行 PIM，相比与 ICE-PIM，可执行 PIM 中的精确语义侧重于执行，并给出直接作用于 PIM 的验证工具。可执行 PIM 和 ICE-PIM 使得 MDA 过程更加健全完善。本文主要内容从 PIM 的可执行性分析、工具原型和平台无关模型验证三个方面展开，具体包括以下几点：

(1) 研究 SysML 符号系统，挑选一个 SysML 符号系统的完备子集，并在这个子集中添加动作规约语言进行增强，此时的 SysML 变为 xSysML，在 xSysML 的基础上研究模型的可执行性。这步工作主要是为了得到可执行 PIM，它是进行平台无关模型验证的基础和前提。

(2) 深入分析与研究现有 MDD 工具的建模机制。在 GMF 插件的基础上开发了满足课题需求的图形化建模编辑器，用它可以建立可执行 PIM。GMF 插件生成的 XMI 文档将状态模型信息按照状态结点和迁移分开来存放，这种存放方式影响了可执行框架执行状态机的效率。由于模型文件是建模框架与可执行框架之间交互的纽带，本文提出了模型文件的序列化操作，改变了状态图信息的组织方式，提高执行效率。这一步工作主要是实现建模框架。

(3) 设计实现支持平台无关模型验证的可执行框架，主要由 ASL 解析器模块、消息驱动框架、状态解析引擎以及实例生成器组成，它是工具原型进行平台无关模型验证的基础。PIM 的验证建立在对象状态机的执行基础上。状态图位于模型的动态行为层，它可以描述系统生命期内的行为过程。在执行对象状态机的过程中产生的信号事件队列以及标记值的变化反映平台无关模型的业务特征。

(4) 定义平台无关模型验证的合理性标准并给出了验证的机制和过程。利用工具原型建立列车运行系统的平台无关模型，主要从它的需求分析、可执行 PIM 的创建和验证三个方面对模型进行研究。

1.4 本文主要内容和组织结构

本文首先研究了与课题相关的技术，主要是如何从 SysML 符号集合中定义出可执行 PIM，它是进行平台无关模型验证的基础。对现有 MDD 工具进行详细地研究，从建模框架、可执行框架和自定义功能插件三大部分设计完成支持可执行 PIM 验证的工具原型，并在这基础上实现可执行 PIM 的验证功能，最后通过模型案例对工具原型的验证功能进行测试。

论文结构组织如下：

第一章 绪论：介绍了课题的研究背景及目标、国内外现状、作者的主要工作内容以及本论文主要内容与组织结构。

第二章 可执行 PIM 相关技术研究：介绍了 SysML，xSysML，动作规约语言，可执行 PIM 以及基于可执行 PIM 的 MDD 过程。

第三章 可执行 PIM 工具原型设计与实现：介绍了工具原型的功能需求，并在功能分析的基础上介绍了建模框架、可执行框架以及自定义功能插件的实现。

第四章 可执行 PIM 工具原型的应用：介绍了平台无关模型验证的标准、模型验证机制并通过实例测试工具原型验证 PIM 的可行性。

第五章 总结与展望：总结本文主要研究内容并对论文的下一步研究工作进行展望。

第二章 可执行 PIM 相关技术研究

使用原生 SysML 进行建模不能得到可执行 PIM，它通常只是半形式化展示模型信息，人们不能进一步处理模型。本章主要目的是利用 SysML 符号集的完备子集和动作规约语言来定义 xSysML，并在 xSysML 基础上定义了特定模型结构的可执行 PIM，使构建的平台无关模型具备可执行性。模型具备可执行性是进行 PIM 验证的基础和前提。

2.1 SysML 的提出

系统工程发展过程中一直缺乏一种规范的“标准”建模语言。这严重限制了系统工程师和其它学科人员之间关于系统需求和设计的有效协作与交流，影响了系统工程的质量和效率。为了满足系统工程领域的建模需求，国际系统工程学会和对象管理组织在 UML2.0 的基础上进行了相应的重用和扩展，提出一种新的建模语言 SysML(Systems Modeling Language)作为系统工程的标准建模语言。2005 年 1 月 10 日，OMG 公布了第三次修改后的 SysML0.9 版。SysML0.9 版是 SysML 发展过程中的里程碑，它确定了 SysML 核心的图形展示。2012 年 6 月 2 日，OMG 公布了 SysML 的最新版本 SysML1.3。

2.2 SysML 语义

SysML 为系统的结构模型、行为模型和需求模型定义了语义。SysML 语义是对现实世界进行抽象和描述的符号系统，这些符号为开发者进行系统建模提供了标准。SysML 语义基于 SysML 元模型，通过元模型可以说明 SysML 建模概念的语义。

2.2.1 元模型理论

元模型由描述模型、构建模型并为模型的实例化和自定义机制提供必要支持的元信息组成，它属于定义模型的模型。OMG 指定的 SysML 模型包括四层元模型结构：实例、模型、元模型、元元模型。在 MOF[20][21] (Meta Object Facility)规范中，元元模型位于 M3 层，元模型位于 M2 层，模型位于 M1 层，实例位于 M0 层。

M3 层具有最高抽象层，为定义 M2 层元素和各种机制提供最基本的概念，它通过元素(Element)进行自描述，元素是 SysML 最底层的模型元素。M3 层包括元类、元操作、值规范等。

M2 层是元元模型的实例，它提供了表达系统的各种包、模型元素的定义类型、标记值和约束等。M2 层包括状态 state、块结构 block、泛化 generalization 和关联 Association 等。

M1 层的每个概念都是 M2 层概念的实例，这一层用来定义特定领域描述语言的模型，例如定义一个列车运行系统块结构图中的 train block，它根据给定个体来表达 M0 层的沟通语言。

M0 层是 M1 层的实例，用来实现复杂系统的功能和性能，例如一个列车运行系统的块结构图。

2.2.2 SysML 表示法

SysML 的视图表示定义了 SysML 语义的表示法，为建模工具使用图形符号和文本语法进行系统建模提供了可视化的标准，SysML 视图在语义上属于 SysML 元模型。SysML 表示法定义了九种视图来表示模型的各个部分，按组织方式分为：行为视图、结构视图和需求视图。行为视图包括状态图、时序图、用例图和活动图，结构视图包括块结构图、包图、参数图和装配图，需求视图包括需求图。SysML 复用了 UML 的状态图，时序图，用例图和包图，在 UML 基础上扩展了活动图、块结构图和装配图，增加了面向系统工程领域的需求图和参数图。SysML 组织形式[22]如图 2.1 所示。

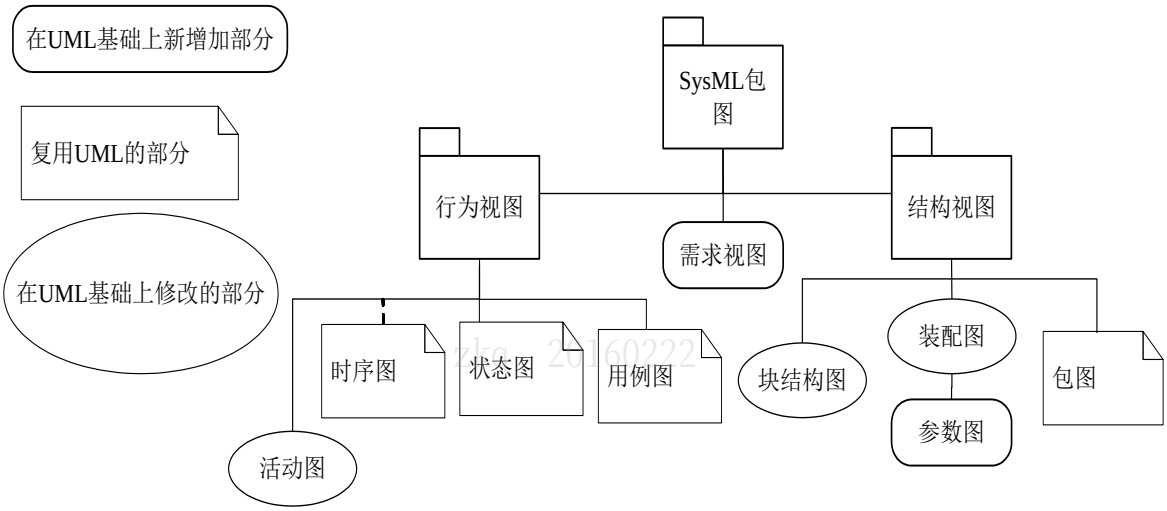

图 2.1 SysML 组织形式

- 状态图元模型分析

状态图由信号事件、迁移、动作和状态结点组成。状态结点表示对象所在的情形，它定义了信号事件、迁移和动作的上下文环境，信号事件和迁移是一种因果关系，信号事件导致了迁移的发生，动作指对象处于该状态时所发生的执行序列。状态图是状态机的图形表示，状态机总是与某个对象相关联，它的基本结构展示了对象生命期的状态集合和转换过程。SysML 状态图复用了 UML 状态图，本文针对平台无关模型的验证主要通过状态图来完成。

- 时序图元模型分析

时序图按照时间顺序描述对象的行为过程，它主要包括对象生命线以及生命线之间的消息传递。消息传递展示了对象的交互过程，这种过程可以表示为对象在给定条件下所发生的事务序列。时序图强调了对象交互的时间顺序，因此它通常被用来图形化地表示对象在某个场景下的用例描述。SysML 时序图复用了 UML 时序图。

- 用例图元模型分析

用例图由用例和参与者组成，它用来表示系统可以向参与者提供哪些用例，从参与者角度分析，一个用例表示系统能够提供的服务。用例之间存在包含、拓展和泛化关系，对于参与者而言，由于它们没有任何属性和操作，参与者之间只有一种泛化关系。用例图在建模过程中提供的有效信息太少，它的用途是有限的。用例图作为用例和系统边界的可视化索引并不被看作开发过程中的重要可交付工件。用例图在建模过程中的真正价值是存在于对它的用例描述中。SysML 用例图复用了 UML 用例图。

- 包图元模型分析

包图用来组织模型元素，每个包代表模型的一个独立部件。包图可以由任何一种 SysML 视图组成，通常是用例图或者块结构图。本文用包图来组织块结构图，表示平台无关模型的一个主题事务。SysML 包图复用了 UML 包图。

- 活动图元模型分析

活动图通常用来描述业务流程，它主要由活动结点和迁移组成。活动结点与状态结点类似，每个活动结点表示业务角色的一次处理过程。当完成一次活动后，业务流并不等待事件触发而是直接跳转到下一个活动结点。在表达业务角色行为过程方面，状态图可以很好地替代活动图，因此本文并没有将它引入行为模型中。SysML 活动图扩展了 UML 活动图的输入和数据流部分。

- 块结构图元模型分析

块结构图作为模型的静态视图，抽象地表示了系统的体系结构。块结构图扩展了 UML 类图，它包括系统属性、操作以及约束条件。块结构图中除了类图已有的对外服务端口外还有流端口，它可以通过特定数据结构的属性来表示，流端口主要处理连续的数据信息，比如电信号。服务端口和流端口是块与块之间通信的接口，它们以系统属性和操作的方式存放在块里。为了表示系统中各种实体单位以及实体测量值，SysML 定义了值类型(ValueType)向块中的属性和流提供单位元素。流端口和约束条件是 UML 类图没有的，它主要为了满足系统领域中的实际需要而添加到 SysML 中，本文用块结构图表示模型的静态结构。

- 装配图元模型分析

SysML 装配图扩展了 UML 组合结构图，它展示了块结构图的白箱视图，提供了部件、端口和连接器。部件描述块内部的实体，端口是部件间通信的交互点，与块结构图一样分为服务端口和流端口，端口之间通过连接器相连。装配图适用于描述复杂的块结构图。

- 需求图元模型分析

需求图是针对系统工程领域而提出的新视图，描述了复杂系统必须具备的行为能力，它是需求分析模型的重要组成部分。每个需求由需求结点、属性结点和测试用例结点构成，在 SysML 表示法中，需求结点被描述为一个类，它有两个属性：需求描述和需求标识，需求描述主要说明了系统的业务用例，需求标识用来区分不同类型的需求。需求与需求之间存在包含关系<<include>>、导出关系<<derive>>、满足关系<<satisfy>>和验证关系<<verify>>[23]。

以需求 A、需求 B 和测试用例 C 为例，A<<include>>B 表示需求 B 是需求 A 的组成部分，A<<derive>>B 表示要实现需求 A 必须先实现需求 B，满足关系<<satisfy>>表示某个需求所依赖的功能模块，C<<verify>>A 表示测试用例 C 能够验证需求 A。

- 参数图元模型分析

SysML 参数图和需求图一样是新提出的模型视图，表示模型属性之间的约束关系。参数图说明某个属性的变化如何影响到其它属性，它通常与块结构图配合使用。从可执行角度分析，使用动作语言可以更精确地描述属性之间的约束关系，完成参数图的任务。

2.3 可执行 PIM

SysML 采用可视化元素构建 PIM 时，为了保持描述的清晰易懂，在给出自身语义说明的同时采用了半形式化的描述方法，这种半形式化降低了 SysML 模型的精确性，给模型的进一步处理带来了困难。所以，当要设计一个精确的 SysML 模型时，原生 SysML 的不足也就显露出来，因此有必要为 SysML 语义进行进一步形式化以增强 SysML 模型的精确性。下面使用 SysML 子集和动作规约来定义 xSysML。

2.3.1 SysML 子集

模型驱动开发方法弥补了传统以代码为中心的软件开发方法的不足，但分析阶段产生的模型往往由于过多地使用 SysML 符号而变得难以理解，这样在描述系统的问题复杂性时缺乏确切的上下文环境。SysML 模型不仅应该被建模人员理解，也应该被领域专家理解，这对一个成功的分析过程非常重要。因此，本文规定了一个简单、一致的 SysML 符号的完备子集，这些符号的选择基于现实模型的静态结构以及执行语义。并不是所有的 SysML 元素都具备模型的执行语义，装配图不具有运行时行为，用例图是辅助分析的符号集合也没有特定的执行语义，因此这个子集中并没有包括这些 SysML 符号。包图、块结构图和需求图能够代表 SysML 模型的静态结构，状态图和时序图用来表示模型的执行语义，本文选取的完备子集定义为{包图、块结构图、状态图、需求图、时序图}。

2.3.2 ASL

传统编程语言用来操纵所有和实现相关的事物，它并不能操纵一个对象模型的元素，因此把它应用到 PIM 中并不合适。Wilkie 等人提出了平台无关的动作规约语言 ASL，它没有对中间件、实现语言或者软件设计策略等作出任何假定，它们能够改变模型的状态，体现业务变化。ASL 的关键特征主要有对象句柄、对象操作、关联操作、调用操作和信号发送。

- 对象句柄

对象句柄用来引用块结构图对象。它是信号发送、调用操作、关联操纵以及对象操作的前提。完成这些操作之前都需要指明一个对象，这个对象就是以对象句柄的形式存在。

下面通过一个 UML 类图实例来说明对象句柄的使用，图 2.2 表示一个学生类和教师类以及它们之间的关联关系，关联名为 R1。

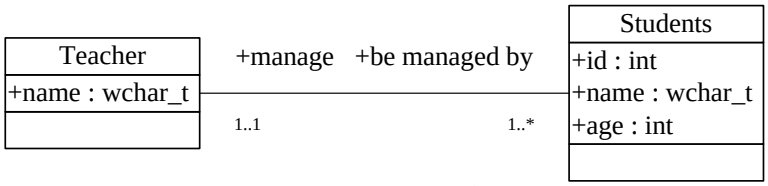

图 2.2 UML 类图实例

引用学号为 1 号的学生对象表示为：selectedStudent = find–only Students where id = 1。“selectedStudent”的句柄指向 id = 1 的 Students 对象。关键字“find-only”表示被测试的条件仅能被一个对象满足，ASL 中用“this”指代本对象。对象集合用来存放多个满足要求的对象，ASL 描述为：{selectedStudents} = find Students where id < 20。关键字“find”返回所有符合条件的对象，返回的对象句柄集合用{selectedStudents}表示，大括号表示该变量是一个集合。

- 对象操作

对象操作处理对象的创建、销毁以及对象属性的读写，给图 2.2 中的学生类创建一个新的学生对象表示为：newStudent = create Students with id = 2 & name = “abc”。创建成功时返回该对象的句柄“newStudent”。 “create”、 “with”和“&”是关键字，其中“&”表示并且，创建对象时不必规定所有的属性值，属性值可以是本地变量。

通过“newStudent”可以读出新创建的学生对象的属性值，结果作为变量返回：selectedId = newStudent.id。也可以写入对象的属性值：newStudent.id = newId。通过“delete”删除对象句柄：delete newStudent。也可以删除对象句柄的集合，表示为：delete {selectedStudents}。

- 关联操作

关联类似链表中的指针域，通过链表指针可以创建结点、删除结点以及查找对应的数据。在 UML 类图的上下文中，关联表示各个块对象之间的相互关系，它是对象之间调用操作和发送信号的基础。 ASL 通过关联操纵来创建关联、删除关联以及漫游关联，漫游类似于链表中的查找。关键字“link”， “unlink”和“->”分别表示创建关联，删除关联以及漫游关联。

参照图 2.2，从 Students 类对象“newStudent”漫游到 Teacher 类对象可以表示为：theTeacher = newStudent -> R1。漫游可以双向进行，根据关联的多重性，可以从 Teacher 类漫游到 Students 类的对象集合上，表示为：{students} = theTeacher -> R1。在关联能被漫游之前必须先创建对象之间的关联，表示为：link theTeacher R1 newStudent，“link”操作通常在“create”操作之后立即进行。“unlink”用来删除对象之间的关联，表示为：unlink theTeacher R1 newStudent。

- 调用操作

操作可以有任意多个输入和输出参数，这些参数由“[]”指示，同时在调用一个操作的时候要指明该操作所属的类名以及该操作所作用的对象句柄。参照图 2.2，要更新“newStudent”

的学号，可以表示为：[newId] = Students:updateId [] on newStudent。[newId]是输出参数集合，[]定义了输入参数集合，这里为空。

- 信号发送

信号发送同调用操作一样，是对象之间的交互手段，它通过异步方式进行交互。异步的特性决定了信号发送没有返回值，信号总是传递给特定的对象。例如向电梯对象发送一个上升的信号，可以表示为：generate LiftHardware : Up() to thelift。“generate”是信号发送的关键字，“LiftHardware”是块名，是信号的发生源，“()”放置信号参数，关键字“to”指明接收信号的对象句柄。

2.3.3 xSysML 定义

SysML 的完备子集解决了 SysML 模型缺乏统一上下文环境的问题，规范了 SysML 视图的使用方法。然而该子集中的语义不足以在平台无关的层面上实现可执行的目的，它还需要精确语义的平台无关动作规约语言的支撑，ASL 是一种支持 UML 和 SysML 的动作语言，遵从 OMG 的动作语义，与 SysML 元素兼容。因此，本文将 ASL 引入到 SysML 子集中用于增强 SysML 语义，此时的 SysML 就变成了 xSysML，定义为{{包图、块结构图、状态图、需求图、时序图}，ASL}。在 xSysML 中，ASL 发挥了重要的作用，它提供了操作 SysML 元素的方法，并能用于详述模型执行的初始化条件以及测试用例。

2.3.4 模型结构

组织和集成 xSysML 的方式必须遵循严格的规则，这些规则能够保持整个模型规约的清晰性。建模人员不会以他们自己的个人风格建模，也不会去构建任意层次深度的系统模型。本文定义的 PIM 是基于信息、行为和操作的抽象层，而不是建立在不断增加细节的层次结构上。在后一种分层方式中，每一层的信息都在其下一层中重复地表达，它消除了用大而复杂的符号系统刻画系统模型时产生的各种特征。

系统的每个主题事务都由一个平台无关模型来表示。根据信息、行为和操作的分层规则，平台无关模型由三个层次构成：第一层是基于信息的静态结构层，它是 PIM 的基石，描述了系统内各个功能模块以及相互之间的关联关系。第二层是基于行为的动态行为层，它描述了静态结构层对象对于异步信号激励和同步调用的行为特征。第三层是基于操作的动作规约层，动态行为层中对象的行为特征在这里被详细定义。

本文定义的可执行 PIM 用包图表示，它描述了系统所包含的主题事务。静态结构层用块结构图表示，块结构图由类图扩展而来，它维护着模型中用来体现业务知识的数据信息。块结构图描述了各个块所具有的属性、操作集合以及相互之间的关联关系。在动态行为层中，行为模型用于详述块对象对于异步信号激励和同步调用操作的响应，每个块都关联着对应的行为模型，行为模型用 SysML 状态图表示。由于状态图的表示方式有很多，为了不造成混淆，本文使用的 SysML 状态图遵循 Moore 状态机[24]规范，即块对象的状态相关行为只在状

态结点的入口处定义，状态图模拟了块对象的生命期。动态行为层描述了块对象在受到信号事件激励时所表现出来的行为特征，行为特征是本文验证平台无关模型是否表现出应有业务需求的重要依据。在动作规约层中，描述行为特征的动作在两个地方定义：在状态中，动作定义为对信号的响应；在操作中，动作体现了同步调用的实现过程。

虽然原先的 SysML 语义规范是必需的，但使用现有状态图语义构建的动作规约层不具备精确的动作执行语义。由于 OMG 已经扩充了 SysML 标准，支持元数据交换，因此它可以与平台无关语言混合使用。针对这一情况，本文在动作规约层中引入 ASL，ASL 可以无歧义地描述平台无关模型的行为动作，具备操作平台无关模型的能力。它和原先的状态图语义共同支撑起平台无关模型中的动态行为层和动作规约层。

基于信息、行为和操作的分层规则使得整个平台无关模型具有清晰、简单的模型结构。ASL 的引入使得 PIM 中的动态行为层和动作规约层具备精确的动作执行语义。本文定义这样的模型是基于 xSysML 的可执行 PIM，如图 2.3 所示。

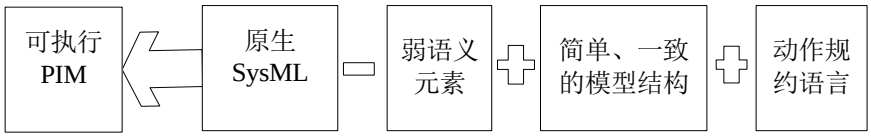

图 2.3 基于 xSysML 的可执行 PIM

2.3.5 MDD 的完善

MDD 过程经历 PIM 的设计，PIM 到 PSM 的映射以及目标原型生成三个阶段[25]。传统 MDD 过程通常将系统的验证放到平台相关阶段进行，这样的开发模式对于系统工程领域而言会付出很大的代价。基于 xSysML 的 MDD 方法在模型设计阶段能产生出可执行 PIM，可执行 PIM 在 MDD 中主要体现出以下几个特点：

(1) 更受关注的可交付工件

可执行 PIM 为每个域提供了一个完整的行为规约。通过仿真运行来验证这些规约可以证明它们是否符合用例的行为需求，问题被更早地发现并解决。因此可执行 PIM 相比不可执行的模型具有更好的质量，也更有用。开发者可以对它们进行定量的评估，创建出高质量的可交付 PIM 工件。

(2) 更完善的构建过程

由于可执行 PIM 建立在 xSysML 的形式化表示法的基础之上，它在映射为 PSM 之前能够被验证。验证 PIM 的测试用例由需求分析模型给出，系统每个需求都需要使用测试用例验证。需求分析模型中的用例用时序图表示，它与状态图之间存在相互关系[26][27]。时序图用在需求分析阶段，是人们期望系统所具有的行为过程。状态图描述系统在运行期间实时表现的行为过程。它们都能展示系统行为过程，在反映系统信息方面存在相交性。当 PIM 对所有给定的测试用例都能表现出预期要求时，PIM 的设计工作才算完成。在原先的 MDD 过程中，PIM 的不可执行性导致了 PIM 到 PSM 映射的可靠性远低于 PSM 到目标原型转换的

可靠性。可执行 PIM 弥补了 PIM 到 PSM 的映射具有低可靠性的缺陷，完善了 MDD 的构建过程。目标原型的缺陷可以更直接地反映到 PIM 中，因此开发者并不需要花很多精力去维护目标原型，减少了测试目标原型的周期和开销。

综上所述，基于可执行 PIM 的 MDD 所体现的特点主要包括：1.建立精确、可执行的 PIM；2.PIM 被转换为目标原型之前就被验证；3.更完善的自动构建过程。基于可执行 PIM 的 MDD 过程如图 2.4 所示，其中圆角矩形代表 MDD 的完善部分。

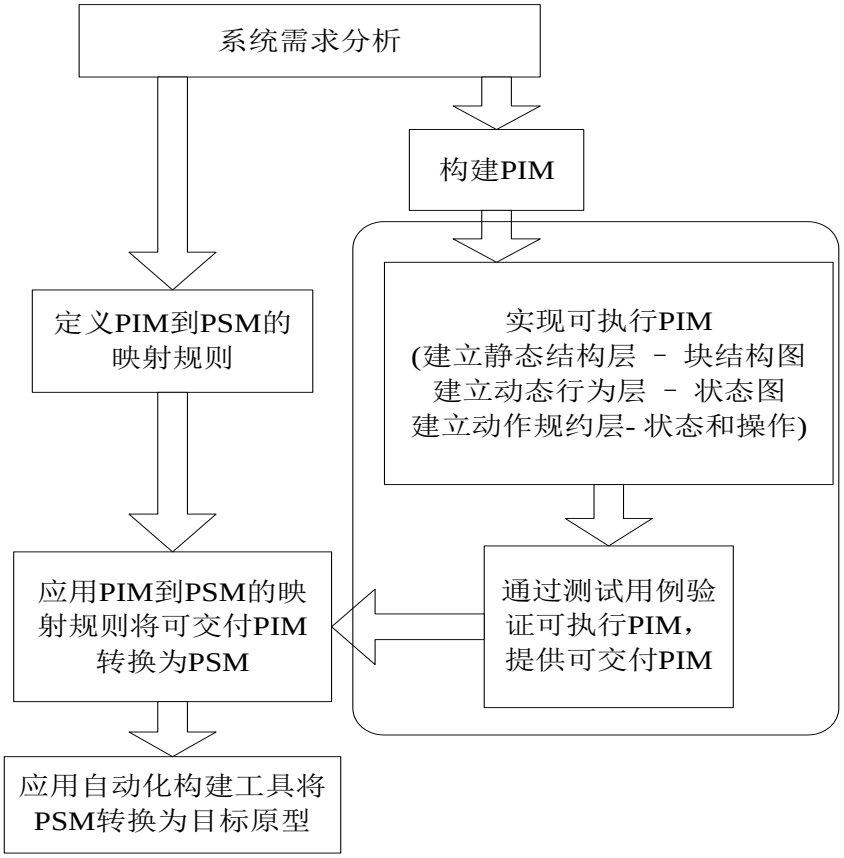

图 2.4 基于可执行 PIM 的 MDD 过程

2.4 本章小结

本章介绍了可执行 PIM 的相关技术。由于原生 SysML 构建的 PIM 不具备可执行性，本章在 SysML 子集和 ASL 基础上定义了 xSysML，并在 xSysML 的基础上定义了可执行 PIM，并介绍了基于可执行 PIM 的模型驱动开发特点。本章内容给出了工具原型关于模型方面的需求。

第三章 可执行 PIM 工具原型设计与实现

SysML 是近年来才提出的用于系统工程领域的建模语言，国内还没有支持 SysML 的建模工具，系统建模缺乏有效工具支持。国外出现了支持 SysML 规范的建模工具，例如 papyrus、Rhapsody、ROSE 等。papyrus 只是简单地遵循 SysML 标准，对 SysML 模型之间的关系以及行为模型的研究不够深入，在使用工具进行 SysML 建模时无法对 PIM 的正确性进行验证，缺乏 PIM 的检测机制。Rhapsody、ROSE 等建模工具支持整个 MDD 过程，对目标原型的合理性验证主要通过平台相关模型约束来完成，并不重点面向平台无关模型，没有进行 SysML PIM 的验证工作，因此它们不属于专门针对 SysML PIM 进行设计、分析和验证的工具。

第二章介绍了基于 xSysML 的可执行 PIM，可执行性满足了本章设计实现的面向 SysML PIM 的工具原型关于模型方面的需求。工具原型支持可执行 PIM 的创建和验证，它的目的是产生可交付 PIM。可交付 PIM 代表 RUP[28]初始阶段和营造阶段的工作量，并且使得构建阶段和移交阶段的工作量大大减少。工具原型弥补了其它建模工具在可交付 PIM 方面的欠缺，通过与 Rhapsody 等工具的协同使用可以更高质量地完成基于 SysML 的 MDD 过程。

3.1 工具的功能需求

通过对一些现有建模工具的分析与研究[29][30][31][32][33]，本文在结合课题目的的基础上设计实现了一个支持可执行 PIM 的工具原型。工具原型主要满足以下功能需求：

- 与 UML 类图编辑器、ROSE 等图形化建模工具一样需要提供一个编辑器和一个工具箱，通过工具箱能够创建 SysML 模型的静态结构视图、动态行为视图和需求图。它们以图形的方式展示在模型编辑区，用户可以通过拖动或者单击的方式对所选的模型视图进行编辑操作，比如查看或修改某个节点或连接线的属性。
- 支持静态结构视图和动态行为视图在模型上下文中的一致性约束。这两种视图代表了 xSysML 模型的静态结构层和动态行为层，一致性确保了动态行为层中的状态图能够模拟块结构图对象的生命期。
- 支持在 PIM 中添加 ASL。它提供操作 PIM 元素的方法，并能用于详述模型执行的初始化条件以及测试用例。ASL 的精确语义支撑起可执行 PIM 的动态行为层和动作规约层。
- 支持平台无关模型的验证功能。该功能主要基于 Rhapsody 的状态图执行机制，通过可执行框架执行对象状态机来实现，并在执行完 PIM 后生成验证信息反馈给用户。
- 提供模型资源管理器，以工程树的形式管理模型元素，支持模型的统一排版和布局。
- 包含其它有利于用户体验的自定义功能插件，主要有控制台插件，工具栏和菜单插件，模型管理器插件和属性页插件。

3.2 工具的功能分析

3.2.1 功能结构

工具原型的功能需求主要分为可执行 PIM 的创建和验证两部分。

可执行 PIM 的创建部分包括需求分析视图管理、静态结构视图管理、动态行为视图管理。需求分析视图管理负责创建和维护包图、需求图和时序图，它们代表 PIM 的业务需求。静态结构视图管理负责创建和维护可执行 PIM 的静态结构层，主要由块结构图组成。动态行为视图管理负责创建和维护可执行 PIM 的动态行为层，主要由状态图组成。

可执行 PIM 的验证部分包括模型数据管理和验证环境管理。模型数据管理负责平台无关信息的处理。验证环境管理负责执行动态行为层中的对象状态机。为了辅助可执行 PIM 的创建和验证，工具原型中还增加了自定义功能管理，它主要用来生成用户界面和辅助性功能的集成。

3.2.2 插件机制

Eclipse[34]是一个运行在 java 虚拟机环境下的应用程序框架，插件机制是 Eclipse 最有价值的地方，它附带了一个标准的插件集，通过插件集可以在原生框架上生成功能丰富的应用系统。Eclipse 支持插件工程，插件工程可以产生功能性插件，它按照功能部件的方式组装到 Eclipse 中，支持即插即用。Eclipse 插件信息以插件文件清单(plugin.xml)的方式呈现给用户，它描述插件集成到框架所需要的信息。插件文件清单中最重要的部分是扩展点，它描述了插件扩展、实现了怎样的功能以及在 Eclipse 工作台中如何显示。Eclipse 已经定义了很多扩展点，例如要添加新的属性页，可以通过添加 Eclipse 提供的属性页扩展点。

3.2.3 开发平台

工具原型拟采用开源 Eclipse Workbench 平台作为基础开发平台，其优点主要是：开放的体系结构和丰富的模型插件支持，一方面可以确保平台的稳定性，提高开发进度；另一个方面保持平台的可扩展性。在 Eclipse 平台的基础上实现建模框架、可执行框架以及自定义功能插件，其中建模框架完成可执行 PIM 的创建，可执行框架完成可执行 PIM 的验证，自定义功能插件完成自定义功能。

3.2.4 建模框架

建模框架是建立在 Eclipse 上的两大扩展功能部件，分为建模部件和模型管理模块。

建模部件能够满足用户建立模型的需求，它基于 GMF[35] (Graphical Modeling Framework)插件。GMF 融合了 EMF[36] (Eclipse Modeling Framework)在模型规范性方面以及 GEF[37](Graphical Editor Framework)在图形化编辑操作方面的优点，主要体现在以下几个方面：

(1) GMF 采用 MVC(Model View Controller)架构[38]，可以方便地将模型文件和图形化编辑操作关联在一起。

(2) GMF 支持 MOF 元模型理论，并提供了根据元模型来生成图形化建模编辑器的解决方案。该图形化建模编辑基于 MVC 架构。

(3) GMF 具有自定义功能扩展点，通过定制可以实现添加 ASL 的导入功能，ASL 能与元数据按照视图逻辑组织在一起，使得建立的 PIM 具备可执行性。

(4) GMF 的模型文件以元数据方式持久化，支持层次结构，确保了可执行 PIM 静态结构视图和动态行为视图在模型上下文中的一致性约束。

由于状态图信息的组织问题，建模部件生成的状态图文件不能够直接被可执行框架的消息解析引擎处理，为了提高消息解析引擎访问动态行为层的效率，需要由模型管理模块对 xSysML 状态图进行序列化处理。经模型管理模块处理之后，模型文件可供可执行框架访问。

3.2.5 可执行框架

本文针对可执行 PIM 的验证主要基于 Rhapsody 的状态图执行实现机制。Rhapsody 软件已经能够实现动态行为模型的执行，其以状态机为核心执行模型，采用 Bruce 提出的状态图模型进行复杂系统建模过程中的动态行为验证。通过对 Rhapsody 的状态图执行实现机制的分析，设计开发了支持可执行 PIM 验证的可执行框架(eXecutable Framework，XF)。不同于 Rhapsody 之处是它在 PIM 阶段而不是目标代码生成阶段完成模型的测试。

可执行框架由消息解析引擎、实例生成器、消息驱动框架和 ASL 解析器组成。消息解析引擎负责将模型文件中的状态图部分提取到克里普克结构中。实例生成器负责将模型文件中块结构图信息提取到对象实例表。克里普克结构和对象实例表共同构成了 PIM 的执行现场。xSysML 状态图语义表明了 XF 中的执行进程是受事件激励的，它通过执行动态行为层中的对象状态机来改变 PIM 的状态，并将模型的标记记录作为验证的反馈信息呈现给用户，这个功能主要由消息驱动框架来完成。xSysML 状态图遵循动作规约层中的行为规则，由于动作规约层是由 ASL 和状态图语义共同组成，ASL 作为平台无关描述语言并不能够直接被执行进程处理，因此在执行对象状态机的过程中，ASL 需要在 ASL 解析器模块的帮助下转换为执行进程能够识别的行为规则。它们相互配合来完成 PIM 的验证。

3.3 工具的总体架构

工具原型以 Eclipse 为基础，通过 Eclipse 的插件扩展点实现了建模框架、可执行框架以及自定义功能插件。建模框架和可执行框架之间通过模型文件进行联系。用户通过基于 GMF 的图形化建模编辑器创建 PIM 并生成模型文件，该模型文件经过模型管理模块的处理后可供可执行框架访问。可执行框架中的消息解析引擎和实例生成器将该模型文件转为可供消息驱动框架访问的执行现场。消息驱动框架在 ASL 解析器模块的配合下执行模型文件中的状态图部分，并且更新模型的执行现场。图 3.1 展示了工具原型的总体架构。

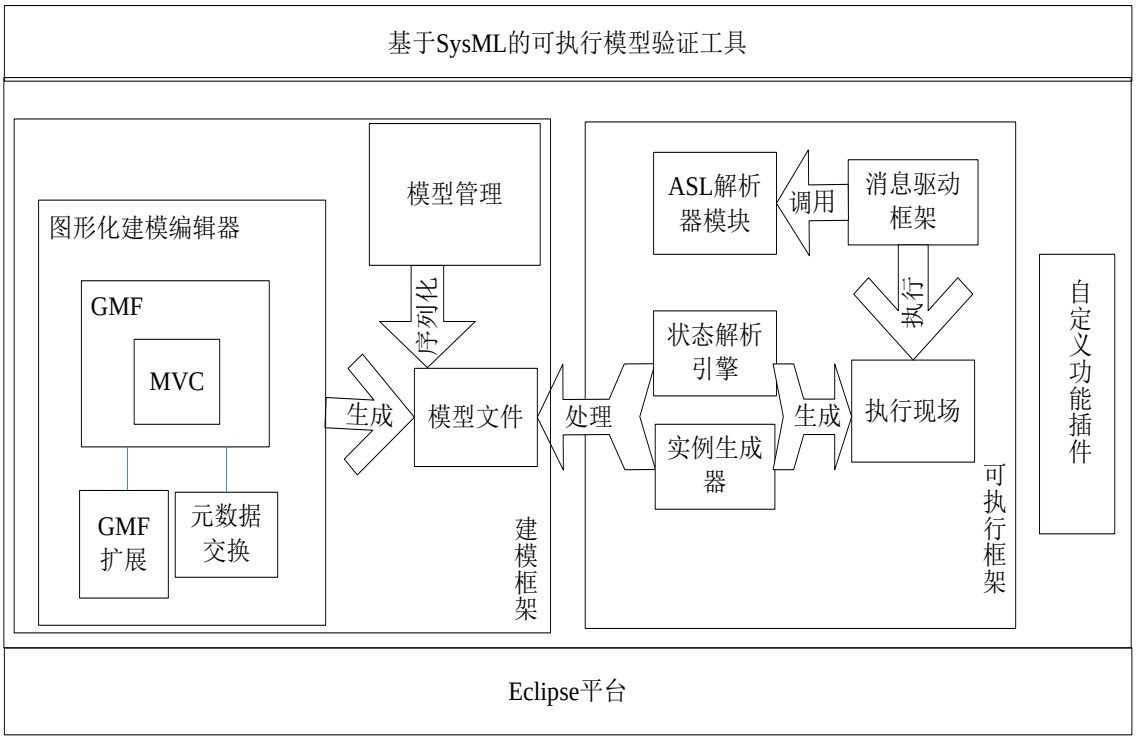

图 3.1 工具原型的总体架构

3.4 建模框架实现

3.4.1 GMF MVC 架构

GMF 内置了用于模型图形化展示的 MVC 框架，用户对模型的编辑操作通过这个框架来实现。MVC 由模型、控制器和视图组成。控制器是整个框架的核心部分，是模型和视图之间联系的桥梁。MVC 的结构如图 3.2 所示，图中的 EditPart 代表控制器，Editor Graphicalviewer 对应视图，箭头序号表示 EditPart 工厂根据模型产生对应的 EditPart 和 GraphicalViewer，并将 GraphicalViewer 与 EditPart 关联在一起。

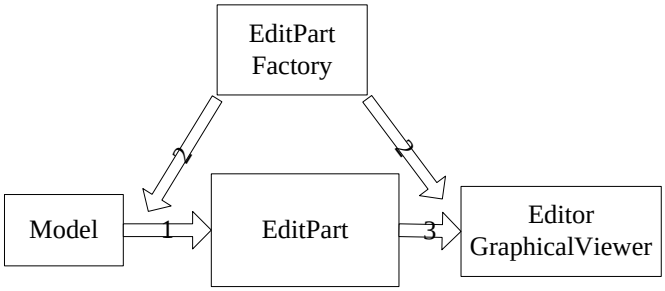

图 3.2 MVC 架构

控制器(Controller)负责更新视图(View)并把用户界面事件转换为请求(Requests)，在 GMF 中，Controller 由一个或者多个 EditPart 对象组成，每个 EditPart 对象关联一个模型(Model)。每个 EditPart 类由若干个编辑策略(EditPolicy)对象组成，EditPart 收到用户请求后会将实际操作权移交给所属的 EditPolicy 。通过 EditPolicy 的定制可以实现自定义操作功能。

EditPart 分为 graphical 和 tree 两种。graphical EditPart 使用 Draw2D 图形作为 View，Draw2D 用于展示编辑区视图，它内置了三种常见的图形集：1.形状(shape)，如正方形、多边形、梯形等；2.控件(widget)，如标签、按钮、滚动条等；3.层(layer)，提供图形缩放、滚动等功能。tree EditPart 使用图形编程框架里面的 treeitem 作为 View，属于大纲展示视图。这两种 EditPart 都继承自 AbstractEditPart 类。

在 GMF 中，当用户编辑模型时，GraphicalViewer 接受操作，同时发送事件到工具栏视图中的 ActiveTool，ActiveTool 将用户操作转换为 Request 分配给 EditPart 中的 EditPolicy，由 EditPolicy 创建适当的 Command 来修改模型。Command 会保留在 EditDomain 的命令栈 CommandStack 里用于实现撤销和重做功能，一旦模型发生改变，其对应的 EditPart 侦听到模型的改变在 GraphicalViewer 上作出反馈，MVC 的工作流程如图 3.3 所示。

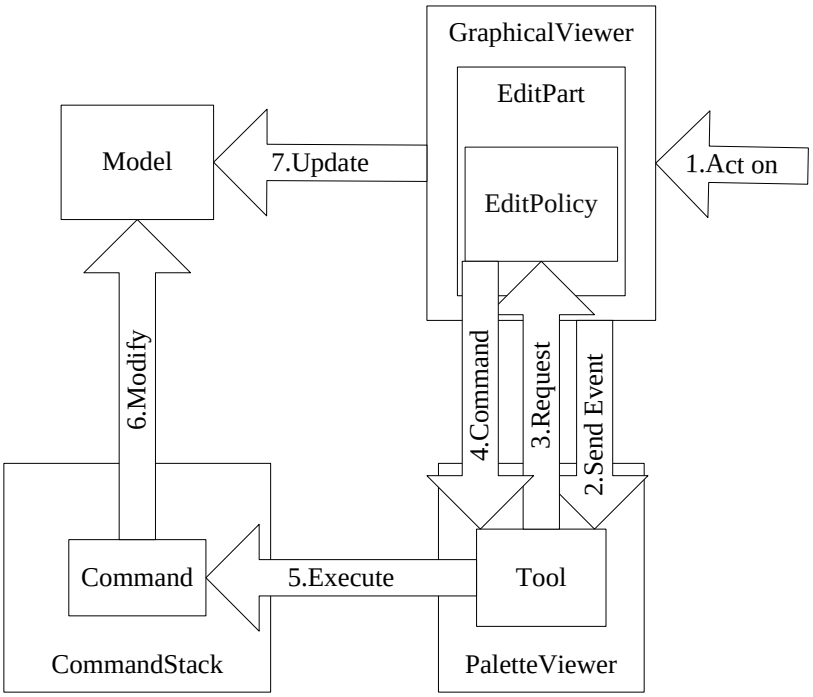

图 3.3 MVC 工作流程

3.4.2 图形化建模编辑器

GMF 能够定义数据模型，包括模型元素的属性、模型元素之间的关联关系、模型元素支持的操作以及模型元素和关系的简单约束。这个数据模型通过一套 Ecore 元模型定义，Ecore 元模型用 MOF 中的一个子集来表示，它在 MOF 规范中处于 M3 层。用 Ecore 元模型可以定义 ecore 模型(位于 M2 层)，ecore 模型在 GMF 中是一个.ecore 文件，它是 M1 层模型的构建规范。GMF 提供了类图编辑器来创建 ecore 模型。

xSysML 包括块结构图、状态图、时序图、需求图和包图的元模型。在构建这些视图的 M1 层模型之前需要分别对它们创建一个 ecore 模型，ecore 模型符合 UML 类图规范，下面建立这五种 M2 层模型的 ecore 模型。

- 状态图元模型

状态图包含的 M2 层模型信息中有初始状态、结束状态、选择状态、简单状态、复合状态、迁移、域、Fork 状态以及 Join 状态。其中初始状态、结束状态、历史状态、Fork 状态和 Join 状态由状态类型进行区分，复合状态由简单状态结点构成。转换(transition)用于描述两个状态之间跳转的方式和条件，驱动着状态图的状态改变。转换发生的条件和执行操作使用标签 “trigger[guard]/activity”表示。其含义是：trigger 是触发转换的事件，guard 是转换发生的条件，只有条件满足事件才执行，activity 是在转换发生后执行的动作。图 3.4 定义了状态图的元模型。

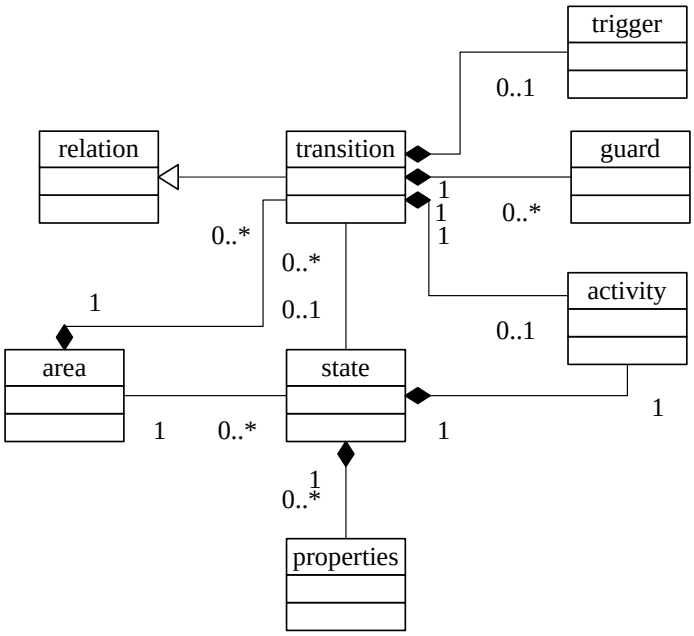

图 3.4 状态图元模型

- 包图元模型

包图用来组织平台无关模型的主题事务信息，它由包(package)以及包之间的依赖(dependency)关系组成。每个 package 中都有一个 statement 属性，它保存着每个包的描述信息。图 3.5 定义了包图的元模型。

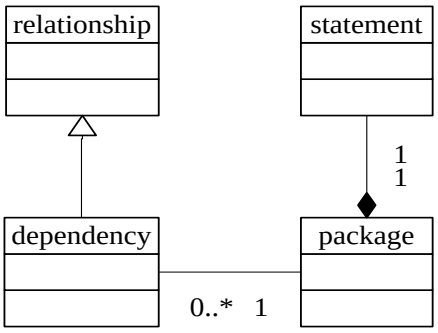

图 3.5 包图元模型

- 块结构图元模型

块结构图中的 M2 层模型信息包括块、值类型、关联和泛化，块(block)是块结构图的最基本元素扩展于 UML 的类元素，它是包括了系统的属性(properties)、操作(operations)和系

统模块约束条件(constraints)。为了表示系统中各种实体的单位以及实体测量值，SysML 中定义了单位元素用于表示物理学科中定义的一些标准单位比如米等，它们由值类型元素(valuetype)指明数据类型，例如用 EInt 来表示长度单位。图 3.6 定义了块结构图元模型。

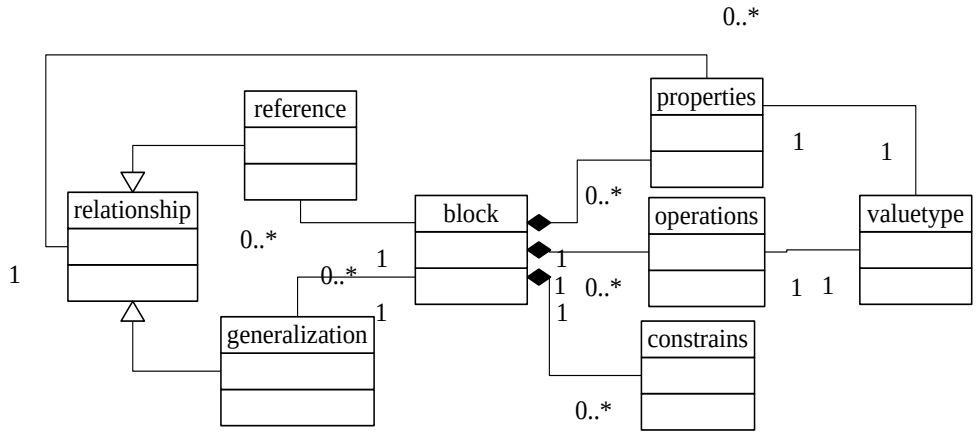

图 3.6 块结构图元模型

- 时序图元模型

时序图所包含的 M2 层模型信息有生命线、可执行区间、消息传递以及生命线的创建和销毁，其中生命线由执行部分组成，生命线之间通过同步调用和异步信号进行通信，生命线可以由另一个生命线创建和销毁，图 3.7 定义了时序图的元模型。

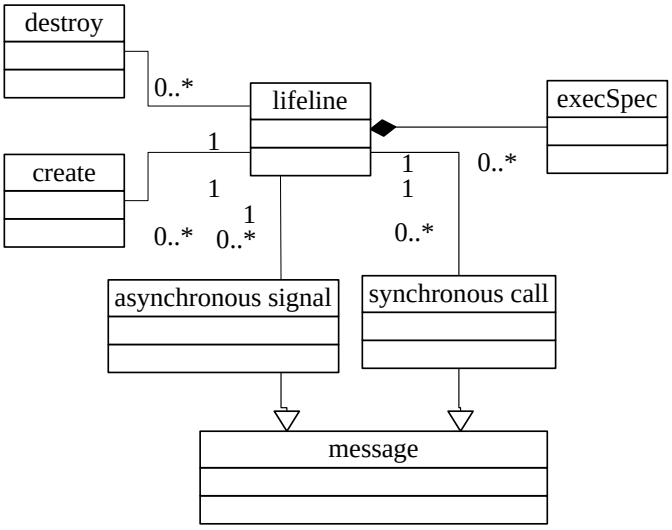

图 3.7 时序图元模型

- 需求图元模型

需求图所包含的 M2 层模型信息包括需求、用例、包含关系、满足关系、导出关系和验证关系，需求图中的需求元素(requirement)是 UML 类的构造型，包含两个属性(properties)：text 和 id，前者是描述需求的文本信息，后者是需求的唯一标示符。需求之间有包含(include)、满足(satisfy)、验证(verify)和导出(derive)关系，需求通过测试用例(testcase)来验证。图 3.8 定义了需求图元模型。

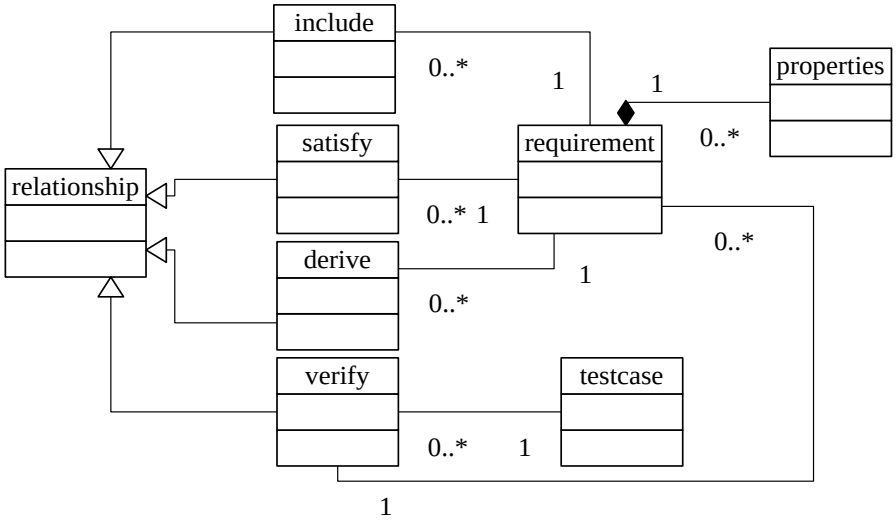

图 3.8 需求图元模型

由于 GMF 是 MDA 在 Eclipse 上的一个实现，通过 ecore 模型可以生成在 Eclipse 上执行的模型代码。GMF 可以通过 ecore 模型产生出面向 M1 层模型的图形化建模编辑器。

原生图形化建模编辑器包含一个工具箱，工具箱存放着创建模型所需的模型元素，拖放模型元素可以在编辑区里显示出对应的图形。模型、图形元素以及工具元素之间存在着紧密的联系。基于 GMF 的编辑器开发方法一方面通过 ecore 模型来生成工具定义文件以及图形定义文件，设置它们的映射关系可以得到一个映射模型，这个模型关联了工具元素和图形元素。另外一方面，GMF 通过 ecore 模型生成了 genmodel 模型。最终的编辑器代码由映射模型和 genmodel 模型共同生成，基于 GMF 的图形化建模编辑器开发方法如图 3.9 所示。

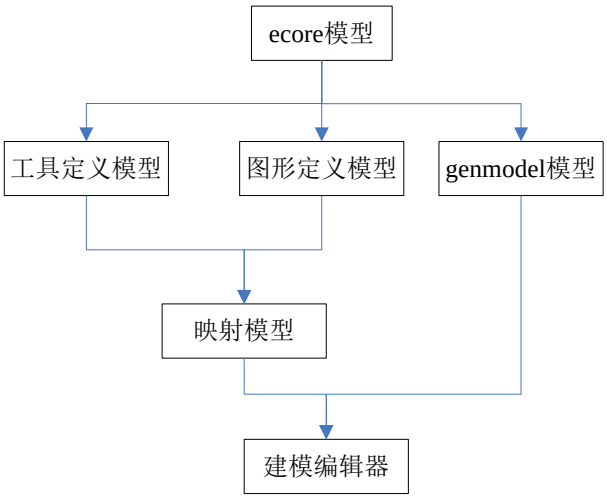

图 3.9 基于 GMF 的编辑器开发方法

本文需要为 xSysML 的五种模型视图建立图形化的建模编辑器，并根据功能需求对建模编辑器进行定制。下面以块结构图为例说明构建图形化建模编辑器的解决方案：

(1) 构建块结构图元模型，这是一个 ecore 模型，它通过 GMF 提供的类图编辑器创建。它定义了块结构图所包含的模型元素、属性和连接关系。

(2) 根据 ecore 模型生成 genmodel 模型。genmodel 模型用于产生模型代码。

(3) 创建和配置工具定义模型(Tool Definition Model，TDM)。ecore 模型通过 GMF 转换为 TDM，TDM 定义了工具箱中的工具元素，主要是结点和连接线。

(4) 创建和配置图形定义模型(Graph Definition Model，GDM)。ecore 模型通过 GMF 转换为 GDM，GDM 定义了 ecore 模型的图形元素，主要有背景画布，图形描述符，结点，连接线和标签五种元素。图形描述符定义了结点和连接线的形状，标签用于显示结点和连接线上的信息。结点，连接线和标签是模型、工具元素和图形元素映射时的图形接口。

(5) 创建映射模型(Mapping Model，MM)。MM 由 TDM 和 GDM 组合而成，它定义了图形元素和工具元素的映射关系。配置默认的 MM，调整每个元素的属性以及它们的组合关系得到符合要求的映射模型。

(6) 生成图形化建模编辑器。GMF 将 genmodel 模型和 MM 转换为 gmfgen 模型，gmfgen 模型是图形化建模编辑器的平台相关模型，最终生成编辑器代码。代码分模型、控制器和视图三个部分，采用了典型的 MVC 架构。

(7) 为了满足功能需求，对(6)中的编辑器进行定制，包括功能和界面两部分，主要实现 ASL 导入功能。

(8) 利用 Eclipse 的功能部件工程将块结构图插件打包成功能部件，通过 Eclipse 中的软件安装向导将功能部件安装到 Eclipse 中。

3.4.3 ASL 导入功能

ASL 在模型中的本质是一个字符串变量，这个变量以成员属性的方式保存在模型元素中。MVC 的模型类中存在一些变量的 get 和 set 操作，ASL 的导入相当于实现 set 操作，而 ASL 的调用则对应 get 操作。伪代码如下：

```
public class block extends Model{
    private String ASL;
    …
    public String getASL(){
        return ASL;
    }
    public void setASL(String ASL){
        this.ASL = ASL;
    }
}
```

从代码可以看出 ASL 的导入功能可以通过改变模型类的成员属性的方式来实现。为了方便用户添加，本文实现了右键菜单的自定义命令，并关联在块结构 block 和状态结点 state 元素上。用户只有在这两个元素上右键点击，弹出菜单里才有这个自定义命令。以 block 为例，实现 ASL 导入的步骤如下所示：

(1) 配置 popupMenus 扩展点，使右键菜单中出现“Set”菜单项并关联着“SetASL”命令。

```
<extension
        point="org.eclipse.ui.popupMenus">
  <objectContribution
      adaptable="false"
      id="com.sysmltool.custom.objectContribution.blockEditPart"
      objectClass="com.sysmltool.diagram.edit.parts.blockEditPart">
    <menu id="BMASet"
        label="&amp;Set"
        path="additions">
    <separator name="group1"/>
    </menu>
    <action
        class="com.sysmltool.diagram.popup. SetASLAction"
        enablesFor="1"
        id="com.sysmltool.diagram.popup. SetASLAction"
        label="&amp;SetASL"
        menubarPath="BMASet/group1"/>
  </objectContribution>
</extension>
```

(2) 实现上一步定义的 Action 类 SetASLAction。它继承自 GMF 的 AbstractActionDelegate 类，并实现 IObjectActionDelegate 接口。SetASLAction 的关键成员函数 doRun()的实现步骤为：

(a) 打开 ASL 编辑对话框。对话框下设有写入按钮。写入按钮将 ASL 关联到全局变量 str。

(b) 取得当前选中模型元素的 editpart。

(c) 创建一个 SetRequest 实例，它包含了改变属性操作的所有信息，包括业务模型对象、属性的名字和新属性值。由于 GMF 里 editpart 的 getModel()方法不是直接返回业务模型元素，而是 View 对象，因此需要再次调用 View#getElement()才能得到业务模型里的元素。利用 ViewUtil#resolveSemanticElement()方法可以直接得到 Activity 业务模型对象。另外，GMF 使用 EMF 的 Transaction 项目来操作模型，所以 editpart.getEditingDomain()方法得到的是一个 TransactionalEditingDomain 类型。

(c) 使用 request 作为构造参数创建一个 SetValueCommand()，这是一个 GMF 命令（实现 org.eclipse.gmf.runtime.common.core.command.ICommand），用来改变属性值。Command 通过 CommandStack 来执行，这样才能达到 undo/redo 的目的，但 editpart.getDiagramEditDomain().getDiagramCommandStack()得到的 CommandStack 只能执行 GEF 命令（org.eclipse.gef.commands.Command），因此需要用 ICommandProxy()对 Command 进行包装。具体实现代码如下所示：

```
public class SetASLAction extends AbstractActionDelegate
    implements IObjectActionDelegate {
      protected void doRun(IProgressMonitor progressMonitor) {
          Runtime rn = Runtime.getRuntime();
          Process p = rn.exec("ASLActivity.exe");
          // 获取选择模型对象的 editpart
          IStructuredSelection structuredSelection = getStructuredSelection();
          Object selection = structuredSelection.getFirstElement();
          IGraphicalEditPart editpart = (IGraphicalEditPart) selection;
          // 创建 SetRequest 实例，用于修改 ASL 属性值
          SetRequest request = new SetRequest(
              editpart.getEditingDomain(),
              ViewUtil.resolveSemanticElement((View) editpart.getModel()),
              BmaPackage.Literals.ASL__NAME,//属性名称
              str);//属性值  String str = “ASL Code”
          SetValueCommand command = new SetValueCommand(request);
          //执行赋值命令
          editpart.getDiagramEditDomain().getDiagramCommandStack().execute(new
      ICommandProxy(command));
      }
  }
```

3.4.4 元数据交换

GMF 在持久化一个业务模型实例的时候，会得到两个文件，一个是业务模型持久化信息，另一个是记号模型持久化信息。记号模型维护着模型图形方面的信息，依赖于业务模型。业务模型文件包含了整个 PIM 的业务逻辑，是建模框架和可执行框架交互的纽带。业务模型以 XMI[39](XML-based Metadata Interchage)文档的形式保存在存储介质中，XMI 是对象管理组织提出的为用户提供元数据交换的标准方法，XMI 的目的在于标准化了使用统一建模语言以及其它不同语言和开发工具的开发人员之间的数据交互方式，这是 GMF SysML 支

持平台无关动作语言的理论基础。它通过标准的 XML(eXtensible Markup Language)文档格式和 XML Sechema 或 XML DTD(Document Type Definition)为 xSysML 模型定义了一种基于 XML 的数据交换格式。任何能够用 MOF 表示的元数据（比如模型和元模型）都可以使用 XMI 规范进行数据转换。同时支持完整的模型或是一个模型的片断到 XML 的转换。

3.4.5 模型文件管理

将主题事务保存在同一个 XMI 文档中可以保证模型层次描述的一致性。本文采用域-块结构图-状态图-动作规约的层次组织模型，块结构图是域的孩子节点信息，状态图是块结构图的孩子节点信息，基于这种父子关系，状态图能访问到块结构图中的所有信息，块结构图能访问到域中的所有信息。这种分层的模型结构定义了行为模型的上下文环境，保证了模型验证的一致性。GMF 产生的状态图 XMI 信息分为迁移和状态结点两个部分存储，在这种存储方式下，它将迁移信息存放到了迁移结点节点中，但状态结点节点并没有给出状态结点的完备信息。每处理一个状态结点时都需要遍历整个状态图 XMI 文档才能获得它的完备信息，这对模型执行效率造成很大的影响。为了解决这个问题，模型管理模块在可执行框架处理行为模型之前会将动态行为层中的状态图序列化为 SCXML[40](State Chart XML)。

State Chart XML 是 W3C[41](World Wide Web Consortium)近年来提出的一种基于 Harel 状态表[42]的通用的可执行状态机表示法，它是针对状态图的一种扩展标记语言。SCXML 以状态结点为单位组织状态图信息，因此每次处理一个状态结点时只需深度遍历完本节点就能够获得它的完备信息。下面通过一个实例来说明 SCXML 的表示方法，它给出了状态图中某个状态结点所包含的信息。

```
<State type="SimpleState">
    <StateName>MaintainingAccelerationCurve</StateName>
    <Tansition Num="2">
        <Destination Name="AccelerationCurveComplete" Event="endDistanceReached"
        Cond="1"/>
        <Destination Name="MaitainingAccelerationCurve" Event="TimeToAdjustSpeed"
        Cond="1"/>
    </Tansition>
    <DynamicArea Num = "1">
        <assign location = "Speed" onEntry = "ACCom" Activity = “ASL 描述文本”/>
    </DynamicArea>
</State>
```

从 SCXML 中可以得到状态结点的类型是简单状态，迁移信息和状态信息都存放在它的孩子节点里。它有两个迁移分别通往相应的后继状态结点，状态入口动作用 ASL 描述。当进入该状态时，首先执行入口动作，endDistanceReached 信号和 TimeToAdjustSpeed 信号有

可能是由入口动作产生也有可能是外部激励。收到 endDistanceReached 信号时状态迁移到 AccelerationCurveComplete 状态，收到 TimeToAdjustSpeed 信号时状态迁移到原状态。

序列化[43]是指将对象的状态信息转换为可以存储或传输的形式的过程，可以由二进制序列化和 XML 序列化两种技术来实现。建模框架与可执行框架之间在模型文件格式上的差异决定了模型管理模块需要对模型文件进行序列化操作。由于 EMF 生成的状态图 XMI 文档也采用了扩展标记语言，因此本文采用 XML 序列化的方式。如图 3.10 所示。

图 3.10 模型序列化操作

模型管理模块以树的方式组织状态信息，运用深度优先遍历算法将 XMI 文档中的每个状态结点的信息和迁移关系都提取出来，然后按照 SCXML 标准重新生成一个 XML 文档。在状态图 XMI 中<transition/>代表迁移节点，<subvertex/>代表状态结点节点，<region/>代表状态图的根节点，<ownedRule/>代表状态结点中的变量信息节点，<entry/>代表状态结点的入口动作节点，<trigger/>代表迁移的事件信息节点，序列化操作的主要算法步骤如下所示：

(1) 在状态图 XMI 文档中主要处理<transition/>和<subvertex/>这两种标签，它们都是<region/>标签的孩子节点，<transition/>和<subvertex/>互为兄弟节点。初始时，指针定位在<region/>节点。

(2) 定位到<region/>节点的第一个孩子节点。

(3) 判断该节点的类型

if 该节点是<transition/>节点

跳到(4)；

else if 该节点是<subvertex/>节点

{

提取出状态类型信息、状态 id 以及状态结点名称，并定位到<subvertex/>节点的孩子节点；

if 孩子节点为空

跳到(4)；

else if 孩子节点是<ownedRule/>节点

提取出状态结点中的变量信息；

if 孩子结点是<entry/>节点

提取出状态结点中的入口动作信息；

跳到(4)；

}

(4) 定位到<transition/>或者<subvertex/>的下一个兄弟节点。如果兄弟节点不为空，跳到(3)。否则，将指针重新定位到<region/>节点的第一个孩子节点并跳到(5)。

(5) 判断该节点的类型

if 该节点是<transition/>节点

{

提取出迁移 id、源状态结点名称和目标状态结点名称，并定位到<transition/>节点的孩子节点；

if 孩子节点为空

将提取出的源状态结点名称与(3)中的状态结点名称做比对；

if 结点名称相同

将<transition/>节点信息存放到源状态结点里并跳到(6)；

else if 孩子节点是<trigger/>节点

提取出事件信息，然后将提取出的源状态结点名称与(3)中的状态结点名称做比对；

if 结点名称相同

将<transition/>节点信息存放到源状态结点里并跳到(6)；

}

else

跳到(6)；

(6) 定位到<transition/>或者<subvertex/>的下一个兄弟节点。如果兄弟节点不为空，跳到(5)。否则，跳到(7)。

(7) 此时状态图 XMI 文档中的每个<subvertex/>节点都包含了状态结点的所有信息。将每个<subvertex/>节点依次按照 SCXML 的编排格式输出到一个新的 XML 文档中。

(8) 序列化操作结束。

3.5 可执行框架实现

3.5.1 状态解析引擎

可执行框架访问动态行为层需要状态解析引擎的支持。状态解析引擎用于提取动态行为层中经过序列化之后的状态图信息，状态图文件经过解析之后得到一组状态相关数据，这些数据代表了动态行为层中的行为规则，它们保存在能被消息驱动框架访问到的克里普克(Kripke)结构中。对象状态机的执行是在此 Kripke 结构表示的有限状态空间内完成。

状态解析引擎为消息驱动框架创建了状态图的执行现场，它是一个有限状态空间，为了保障对象状态机执行的安全性，需要对有限状态空间进行合理性检查。状态解析引擎负责从SCXML 中生成状态迁移图(State Transition Diagram)并检查其合理性。图 3.11 说明了状态解析引擎的用例。

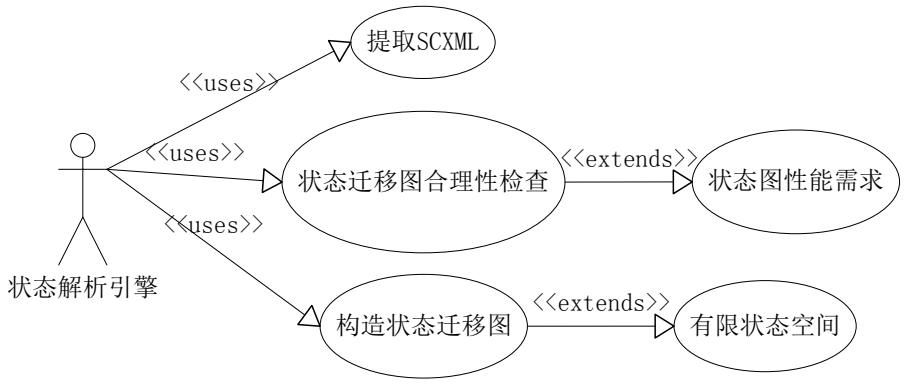

图 3.11 状态解析引擎用例

- 状态图性能需求

状态图的性能需求[44]指对象状态机要求在有限的时间内运行完成。完整的生命期包括对象的创建、对象的运行过程以及对象的销毁，从状态图体系结构的角度分析，必须存在一条由初始结点到达终止结点的路径，其中初始结点代表对象创建，终止结点代表对象销毁，路径代表对象运行过程，并且这条路径能够到达状态图的任意结点。

- 有限状态空间

SCXML 含有 xSysML 状态图体系结构的所有信息，使用 Kripke 结构[45]可以表示对象状态空间，通过状态解析引擎将 SCXML 信息提取到 Kripke 结构中。根据 Kripke 结构可以方便地构造出 SysML 状态图的对象状态迁移图。本文在遵从 OMG SysML 规范的前提下重新定义了一些状态图语义的形式化表示方式，包括顺序和层次自动机的定义、单元的定义以及层次自动机操作语义的定义，并将状态图转化为层次自动机(Hierarchical Automaton，HA)，HA 由顺序自动机(Sequential Automaton，SA)组成。

定义 1（顺序自动机） $SA = (\Psi_A, S_A^0, \lambda_A, \delta_A)$，$\Psi_A$ 是有穷状态集合，S_A^0 表示初始状态，λ_A 表示迁移标记，$\delta_A = \Psi_A \times \lambda_A \times \Psi_A$ 表示迁移关系。

定义 2（层次自动机） *HA = (Fi, Ev)*，其中 *Fi* 代表一个或多个有穷顺序自动机，$\forall SA1, SA2 \in Fi$，$\Psi_{SA1} \cap \Psi_{SA2} = \Phi$，*Ev* 代表事件集合。

HA 中的全局状态由单元表示，一个单元既可以代表一个基本状态又可以代表一个复合状态，每个复合状态由某个简单顺序自动机的基本状态组成。

定义 3（单元） *HA* 的一个单元是一个集合 $Cell \subseteq \cup_{SA \subset F_{\psi SA}}$ 使得：

(1) $\exists s \in \psi_{SA}$ 满足 $s \in Cell$；

(2) 若父状态位于一个单元内，则它的子状态都位于该单元内。$\forall s, SA$，其中 *SA* 是以 *s* 为父状态的简单顺序自动机并且 $s \in Cell$，则 $\exists s' \in SA, s' \in Cell$。

$Cell_{HA}$ 表示 *HA* 中所有的单元。

在 xSysML 状态图中，每个状态由单元和当前环境组成，环境包括事件队列和状态变量，对于事件队列环境 *Ev*，ΘEv 表示事件队列的所有可能情况，对于状态变量环境 *Va*，ΘVa 表示变量所有可能取值。下面用标记转换系统（LTS）定义 *HA* 的操作语义。

定义 4（HA 的操作语义）*HA* 的操作语义是一个 LTS：$OP = (S, S_0, \rightarrow)$，其中 $S = Cell_{HA} \times \Theta Ev \times \Theta Va$ 是状态集合，S_0 是初始状态集合，$\rightarrow = S \times S$ 是状态迁移集合。

Kripke 结构是一个四元组，$K = (Cell_{HA}, \rightarrow, L, Cell_0)$，其中 $Cell_0 \subseteq Cell_{HA}$ 是初始单元的合集；$(Cell_1, Cell_2) \in \rightarrow$ 表示存在从 $Cell_1$ 到 $Cell_2$ 的迁移，假设 $\rightarrow$ 中的迁移互不相同；$L: Cell_{HA} \rightarrow 2^{AP}$ 是每个单元的原子命题。状态解析引擎采用邻接表表示 Kripke 结构，图 3.12 给出了用邻接表表示 Kripke 结构的一个实例。

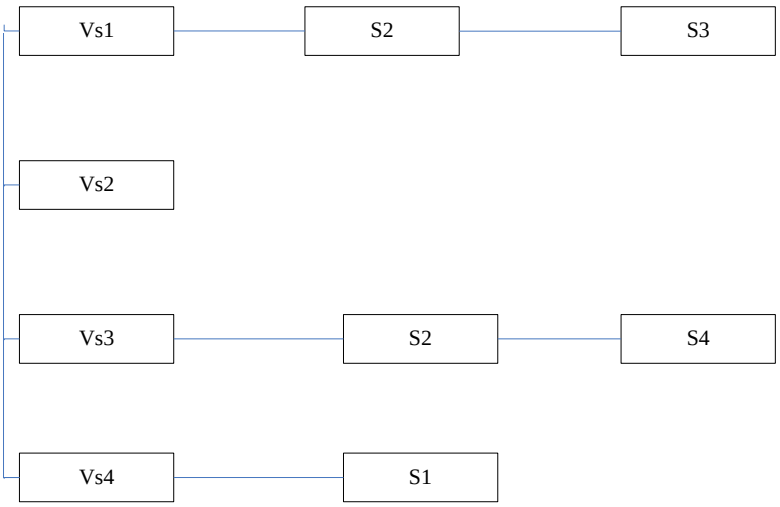

图 3.12 邻接表表示 Kripke 结构的实例

图中 *Vs1, Vs2, Vs3, Vs4* $\in Cell_{HA}$，所有邻接于 *Vsi* 的顶点 *Sj* 链成一条单链表，其中 *Vsi* ，*Sj* $\in Cell_{HA}$，$\exists i \in \{1..num\}$，$\sum_{j \in \{1..num\}} (Vsi, Sj) \in \rightarrow$，此例中 *num* 取 4。

根据定义 4，给出 $s \in S$ 的 SCXML 表示（其中 $e_1..e_n \in Ev$，$a_1..a_m \in Va$）。

```
<State type="undef">
    <StateName>Vsi</StateName>
    <Tansition Num="n">
      <Destination Name= Sj1  Event= e1  Cond="true" />
      …
      <Destination Name= Sjn Event= en  Cond="true" />
    </Tansition>
    <DynamicArea Num="m">
        <assign location=" a1 " onEntry="Entry" Activity="ASL description"/>
        …
        <assign location=" am " onEntry="Entry" Activity="ASL description"/>
```

</DynamicArea>

</State>

它表明该状态的类型未定义，状态结点名为 Vsi，它有 n 个迁移，受到事件 e_1 激励时迁移到 Sj_1，受到事件 e_n 激励时迁移到 Sj_n。它有 m 个变量，入口名为 Entry，动作由 ASL 描述。从 SCXML 的描述中可知 xSysML 状态图的操作语义，结合 Kripke 结构，便能构造出对象状态迁移图。

- 合理性检查标准

xSysML 状态图是否满足性能需求指块对象能否运行完整个生命期，这就要保证对象能够安全地创建，安全地销毁并且在生命期内不存在死锁。根据这个要求，本文定义了对象状态迁移图的三大安全性条件[46]并给出了相关的推论和证明。

条件 1 $\forall s \in \cup_{SA \subset F_{\psi SA}}$，$(i \xrightarrow{*} s) \Rightarrow (s \xrightarrow{*} o)$，其中 i、o 分别代表初始状态和结束状态。

条件 2 $\exists t \in \rightarrow$，$s \xrightarrow{t} s'$，其中 s' 为除初始状态和终止状态外的任意状态。该条件表明对象在生命期内不存在死锁情况。

推论 1 满足条件 1 和条件 2 的状态图是符合状态可达的，即状态图中的任意一个状态都是可达的。

证明： 如果 $\exists s' \in \cup_{SA \subset F_{\psi SA}}$ 是不可达的，即 $s \xrightarrow{t} s'$ 不成立，这与条件 2 违背。同理，$(i \xrightarrow{*} s')$不成立，这与条件 1 违背。所以 $\forall s' \in \cup_{SA \subset F_{\psi SA}}$ 是可达的。

条件 3 $tsi = (s1, s2 \ldots sj) \in \{ts1..tsn\}$是对象的其中一条并发迁移序列，则 $ts1 \cap .. \cap tsi \cap .. \cap tsn \notin \{o\}$, 其中 o 为终止状态。该条件表明对象收到销毁信号时，不可能存在该对象产生的某个或多个子对象还未销毁的情况，否则子对象会一直存在于系统中占用资源。该条件保证了对象安全地销毁。

- 提取 SCXML 并构造状态迁移图

状态解析引擎负责将 SCXML 中的状态结点信息提取到邻接表表示的 Kripke 结构中，生成对象状态迁移图。Kripke 结构由状态结点组成，为了能够存放状态信息，本文定义了状态结点 StateNode 的数据结构。它是一个结构体，各字段名以及含义如表 3.1 所示。

表 3.1 状态结点的各字段名及含义

字段名（数据类型）	含义	字段名（数据类型）	含义
StateName(string)	状态名称	StateType(int)	状态类型
NestNum(int)	嵌套层数	SubState_Num(int)	子状态个数
TransitionStruct (TRANSITION_STRUCT *)	迁移链表首地址	Nest(StateNode *)	嵌套状态的内部首地址
FuncName(string)	入口函数名	OnEntry(string)	入口动作描述
IsActiveTrack(bool)	活动标志	HasParent(bool)	后继标志

其中 TRANSITION_STRUCT 也是一个结构体，里面存放了目标状态名以及迁移条件信息。由于状态结点中包含了迁移链表的首地址，对象状态迁移图其实是一个状态结点的数组。对象状态迁移图的生成算法如下：

(1) 提取出状态结点的个数。

(2) 动态生成状态结点数组。

(3) 在 SCXML 中定位到第一个状态结点的根节点，按照深度优先遍历该节点的所有孩子节点的信息，根据状态类型的差异分别进行如下操作：

(a) 如果状态类型为初始状态，则在状态结点结构体中保存状态名信息，状态类型信息，入口函数名，入口动作描述，后继迁移数目并根据迁移字段信息生成迁移链表。

(b) 如果状态类型为终止状态，则在状态结点结构体中保存状态名信息，状态类型信息，入口函数名以及入口动作描述。

(c) 如果状态类型为选择状态，则在状态结点结构体中保存状态名信息，状态类型信息，入口动作描述信息并根据迁移字段信息生成迁移链表。

(d) 如果状态类型为 FORK 或 JOIN 状态，则在状态结点结构体中保存状态名信息，状态类型信息，并根据迁移字段信息生成迁移链表。

(e) 如果状态类型为简单状态或者历史状态，则在状态结点结构体中保存状态名信息，状态类型信息，后继迁移数目，根据迁移字段信息生成的迁移链表，入口函数名以及入口动作描述。

(f) 如果状态类型为复合状态，则在状态结点结构体中保存状态名信息，状态类型信息，嵌套数目，根据嵌套字段信息生成的嵌套链表，后继迁移数目，根据迁移字段信息生成的迁移链表，入口函数名以及入口动作描述。

(4) 在 SCXML 中定位到上一步中的根节点的兄弟节点，如果兄弟结点为空则跳到(6)，否则按照深度优先遍历该节点的所有孩子节点信息，根据(3)中的规则将孩子节点信息保存到数组中的下一个结构体中。

(5) 重复进行(4)。

(6) 对象状态迁移图生成完成。

- 状态迁移图合理性检查

状态解析引擎在完成对象状态迁移图的创建之后，对状态图进行性能需求方面的验证。在给出验证算法前，先定义两种异常的状态结点，分别为“奇迹”状态和“黑洞”状态。

定义 5（“奇迹”状态) “奇迹”状态是一种只有迁移出，没有迁移入的非初始状态。对象处于这种状态时说明它没有被正确地创建。

定义 6（“黑洞”状态）“黑洞”状态是一种只有迁移入，没有迁移出的非终止状态。对象处于这种状态说明它已经陷入死锁。

由推论 1 可知 SysML 状态图不能存在“奇迹”和“黑洞”状态，下面给出性能需求的验证算法：

(1) 遍历对象状态迁移图，判断状态结点的单链表是否为空，如果为空并且该状态结点不是终止状态，则存在“黑洞”状态，标记异常，否则符合安全性的条件 1。

(2) 遍历对象状态迁移图，如果状态结点的单链表非空，则遍历该单链表，并给单链表中的每个状态结点标记上后继标志。

(3) 在(2)的基础上重新遍历对象状态迁移图，判断状态结点中的状态是否有后继标志，如果没有并且该状态不是初始状态，则存在“奇迹”状态，标记异常，否则符合安全性的条件 2。

(4) 如果终止状态的前继状态只有 1 个，则符合安全性的条件 3，直接跳到(8)，否则进行(5)(6)(7)。

(5) 遍历对象状态迁移图，判断状态结点的类型，如果为 FORK 伪状态，则给该单链表中的每个状态结点标记上活动标志，如果为 JOIN 伪状态，则清除顶点域中状态的活动标记。

(6) 如果状态结点具有活动标志，则给该单链表中的每个状态结点标记上活动标志。

(7) 在(5)(6)的基础上重新遍历对象状态迁移图，判断状态结点的类型，如果为终止状态并且具有活动标记，说明不符合安全性的条件 3，标记异常，否则符合安全性的条件 3。

如果条件 1、2、3 都满足，则该对象状态迁移图是合理的。

3.5.2 实例生成器

可执行框架访问静态结构层需要对象实例生成器的支持。对象实例生成器创建块的运行时现场，该现场在工具原型中是以对象实例表的形式存在，实例生成器声明了具有十行大小的实例表类型。实例表中的行类型是个结构体，它的成员变量是一个结构体数组，数组下标指引成员变量，结构体中的成员对应成员变量的名称、类型和值。实例表 TABLE 声明过程如下所示：

```
typedef struct Attribute
{
    char Property_name[20];
    char Property_type[20];
    char Property_value[20];//可以存放变量的引用地址
} ATTRIBUTE;
typedef struct Row
{
    ATTRIBUTE AttrIndex[100];
} ROW;
typedef struct Table
```

```
{
    ROW RowIndex[10];
} TABLE;
```

在执行对象状态机之前，实例生成器为静态结构层中的每个块结构创建一个 TABLE 类型变量，并初始化为空，当 ASL 解析器处理某个块的 create 语句时，实例生成器会将对象的初始化信息赋给 TABLE 变量，每个对象对应一行，行由 RowIndex 数组下标指示。

实例生成器提取静态结构层信息的规则基于以形式化表示为中心的方法[47]，规则对应一个静态结构层的元模型。以形式化表示为中心的方法根据元模型的组织关系编写从模型元素到目标原型的直接转换处理过程，这个过程表示为以 ASL 详述的针对静态结构层元模型的处理模型。静态结构层元模型的组织关系如图 3.13 所示。

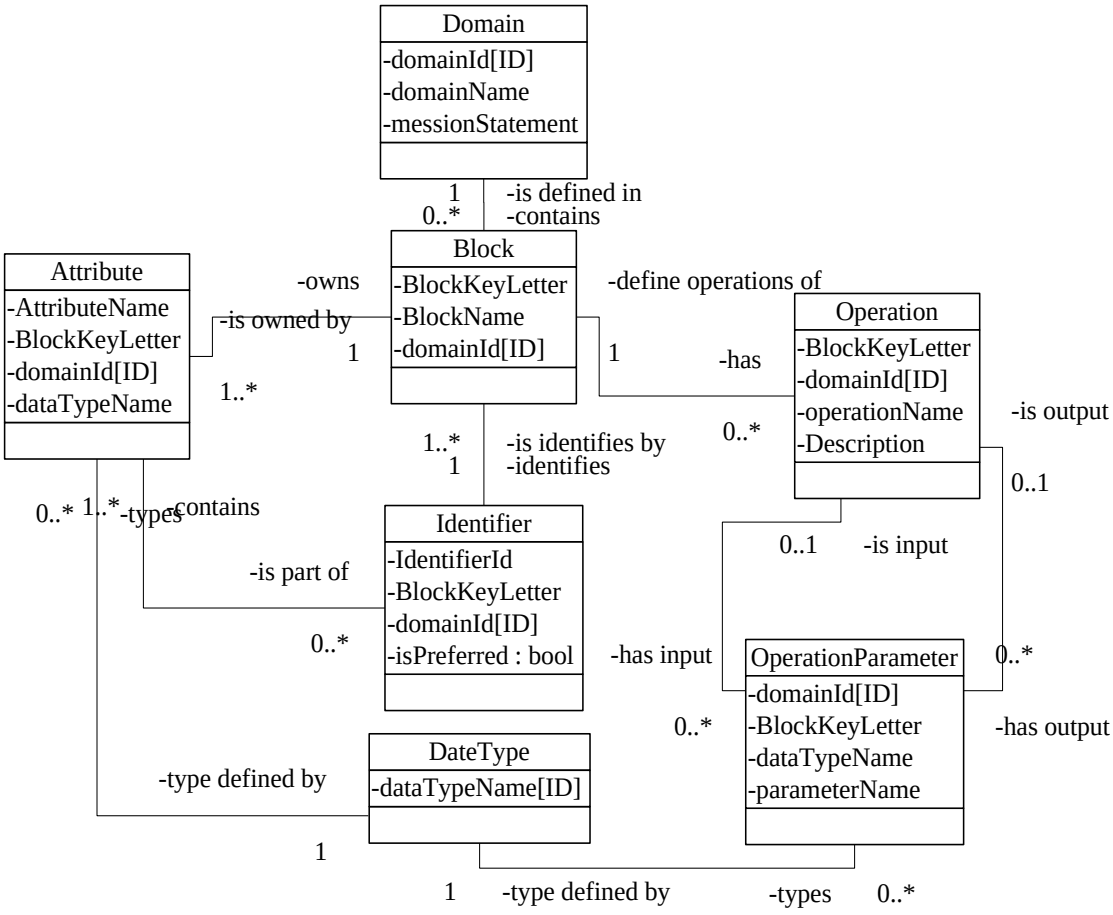

图 3.13 静态结构层元模型

从图 3.13 可以看出静态结构层元素的组织关系，针对一个实际的平台无关模型，根据组织关系可以实例化每个块结构的对象，ASL 表示的针对静态结构层元素的提取规则如下：

```
allDomains = find-all Domains
for aDomain in {allDomains} do
    #遍历域中的每个块结构图
```

```
{BlocksForDomain} = aDomain -> Block
for aBlock in {BlocksForDomains} do
    #遍历块结构图中的信息
    {AttributesForBlock} = aBlock -> Attribute
    for aAttribute in {AttributesForBlock} do
        #提取属性信息 构造 TABLE 变量
    end for
    {OperationsForBlock} = aBlock -> Operation
    for aOperation in {OperationsForBlock} do
        #提取操作信息
    end for
 end for
end for
```

对象实例表存储了 PIM 的业务信息。在执行对象状态机的过程中会改变表中的属性，有些属性的变化并不被建模人员和领域专家所关心，通常他们会选择一些具有某种业务事实的属性作为监视对象。为了解决这个问题，本文扩展 ATTRIBUTE 结构如下：

```
typedef struct Attribute
{
    bool SetAsTag;
    char Tag[20];//存放 Property_name 额外说明信息(Tag Scheme)地址，Tag Scheme 在
                 //块结构中保存在 constraints 里。若 SetAsTag = false，则忽略该项。
    char Property_name[20];
    char Property_type[20];
    char Property_value[20];//可以存放变量的引用地址
} ATTRIBUTE;
```

标记是由建模人员和领域专家使用的提供有关模型元素的额外信息的方法。这些额外信息可以被模型转换系统使用。这些信息是基于分析域的视角看到的事实，而不是有关系统结构的信息，即标记是面向业务的，而不是面向设计。它在模型文件中通过标签属性添加到块结构的成员属性元素<ownedAttribute/>中，它用来指示成员属性是否具备业务事实。在执行状态机的过程中，这些标记的成员属性值会发生变化，代表业务事实的变化。文本定义的 Tag Scheme 包含下列元素：

- 标记名；
- Property_value 的允许范围(通常用于描述模型执行过程中某个属性的变化范围，如果超过这个范围，表示模型的执行与实际结果不符，存在异常)；

- 业务需求视点的描述信息；
- 使用这个标记的成员属性；

块结构对象的创建和销毁对应实例表中行的插入和删除。表项代表块结构中的属性，比如列车块结构的速度属性。每个块都能且只能关联一张对象实例表，它们通过表类型的变量名进行区分。对象实例生成器确保对象的生成规则，主要体现为：

1. 对象实例表中的每个单元格（ATTRIBUTE 结构体）只能关联一个成员属性。
2. 行和列的次序由 RowIndex 和 AttrIndex 的下标指定。
3. 对象实例表中的每行通过一个或多个属性惟一标识。

在验证 PIM 时，可以将其中的第 N 列 AttrIndex[N]的 SetAsTag 设置为 true，这样在反馈验证信息时只将做为标记的成员属性值呈现给用户。

3.5.3 ASL 解析器原型

本文设计实现的 ASL 解析器原型基于以 ASL 元模型形式化表示为中心的文本替换规则，这些规则能将 ASL 的平台无关操作转换到平台相关实现上。ASL 本身也是一个模型，任何一条完整的 ASL 语句都是该模型的一个实例。当 ASL 解析器扫描一条 ASL 语句时，它开始处理 ASL 模型的元模型。ASL 元模型提供了微观的处理视图，每条 ASL 语句结构由 ASL 元模型中的元类(metaclass)的实例来表示。ASL 元模型中包含有元类 specified process，而这个元类又派生出诸如 create、generate、delete、if、switch 等关键字类。每个关键字类中维护着参数属性、指引属性以及文本替换规则。参数属性指明了该关键字所支持的参数，指引属性指明了关键字之间的关联关系，文本替换规则是一张由参数属性以及指引属性决定的文本映射表，它完成两部操作：首先将关键字替换为平台相关实现语言中的关键字；随后将参数属性和指引属性按照映射表中的格式与替换后的关键字组织在一起。五个主要的 ASL 关键特征的映射格式如下所示。

对象句柄：Param1 = find Param2 where Param3 --> Param2.Param1(Param3);

对象操作：Param1 = create Param2 with Param3 --> Param1 = new Param2 (Param3);
delete Param1 --> delete Param1;

关联操作：Param1 = Param2 -> Param3 --> Param1 = Param2.Param3;
link Param1 Param2 Param3 --> Param1.Param2 = Param3;
unlink Param1 Param2 Param3 --> Param1.Param2 = null;

调用操作：[Param1] = Param2:Param3 [Param4] on Param5 -->
Param1 = Param2.Param5.Param3 (Param4);

信号发送：generate Param1:Param2 to Param3 --> Param1.SendEvent(Param2, Param3);

ASL 元类实例中的参数属性和指引属性为解析器中的词法分析和语法分析提供了依据。词法分析器通过元类实例名来判断关键字的合法性，并根据实例中的参数属性来判断关键字所附带的参数格式。语法分析器根据实例中的指引属性来判断关键字与关键字之间的相互关

系，若元类 Key1 中有 Key2 的指引，说明关键字 Key2 需要同 Key1 一起使用。ASL 解析器在词法分析和语法分析的基础上将 ASL 语句替换为平台相关语句，这个功能由替换域(Replace Domain)来完成。在以形式化表示为中心的 ASL 解析器中，替换域是协调解析的一个域，它还依赖于格式化服务域和错误处理域，格式化服务域用来处理冗余的 ASL 语句，还对替换后的文本代码的布局和外观进行改进，错误处理域完成 ASL 语句发生语法词法错误后的后续处理。ASL 语句经过替换域、格式化服务域和错误处理域的协调处理后由 ASL 解析器的输出域输出平台相关实现语句，例如 js、lua 等解释执行的脚本。图 3.14 给出了本文设计的 ASL 解析器模块的包图。

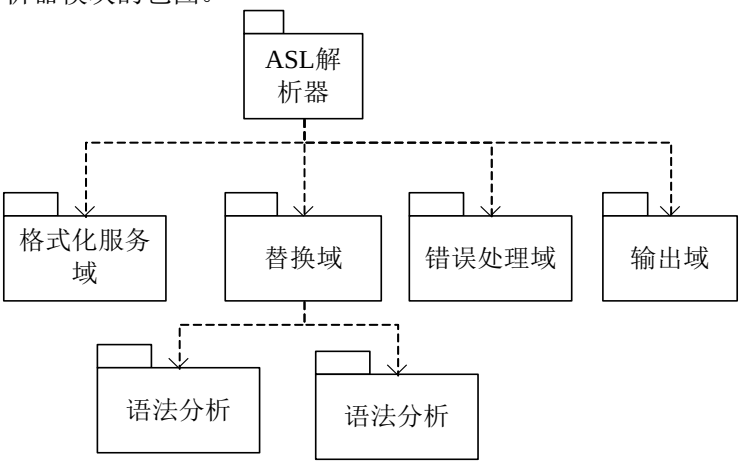

图 3.14 ASL 解析器模块的包图

下面以 ASL 语句 InputString =“newStudent = create Students with id = 2 & name = “abc””作为 ASL 模型的实例来说明 ASL 解析器原型的文本替换过程 ASLProc。specified process 中的元类 create 和 with 具有如图 3.15 所示关系。

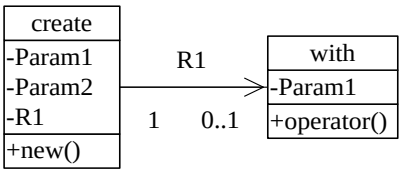

图 3.15 关键字类关系

文本替换过程 OutputString = ASLProc(InputString)的实现步骤如下：

(1) 替换域中的词法分析器遍历 InputString 得到关键字类 create 和 with 的实例。

(2) 分别使用 newStudent 和 Students 构造 create 的参数属性 Param1 和 Param2。

(3) 语法分析器通过 create 的指引属性 R1 得到下一个关键字 with。

(4) 使用 id = 2 & name = “abc”构造 with 的参数属性 Param1。

(5) create 实例中 new()完成 create 关键字的替换规则：Param1 = create Param2 变成 Param1 = new Param2。

(6) with 实例中 operator()完成 with 关键字的替换规则：with Param1 变成 (Param1)。

(7) 替换域将 create 和 with 实例整合为 Param1 = new Param2(Param1)之后由对应的参数属性进行替换，转换为 newStudent = new Students(id = 2 & name = “abc”)。

(8) 经过格式化服务域和输出域处理后输出 OutputString = “newStudent = new Students(2, “abc”);”。

ASL 解析器处理多条 ASL 语句{InputString1..InputStringN}的解析规则如下：

```
判断 InputString1 中是否有关键字 create;
if (InputString1 中存在 create)
{
    调用实例生成器;
    Execute(ASLProc(InputString1));//Execute 是脚本执行引擎的调用函数，工具原型使
                                    //用 Mozilla 的 SpiderMonkey 引擎
}
else
{
    Execute(ASLProc(InputString1));
}
…
判断 InputStringN 中是否有关键字 create;
if (InputStringN 中存在 create)
{
    调用实例生成器;
    Execute(ASLProc(InputStringN));
}
else
{
    Execute(ASLProc(InputStringN));
}
```

第二章介绍了 ASL 在可执行 PIM 中的作用。在可执行框架中，执行进程需要完成：1.通过测试用例来初始化静态结构层的《terminator》对象。2.执行动态行为层的过程中，对象状态机在信号事件的激励下完成状态迁移。其中信号事件和状态入口动作都由 ASL 进行描述。3.处理块结构中的操作。这些都需要可执行框架加载 ASL 解析器来实现，图 3.16 说明了 ASL 解析器模块的用例。

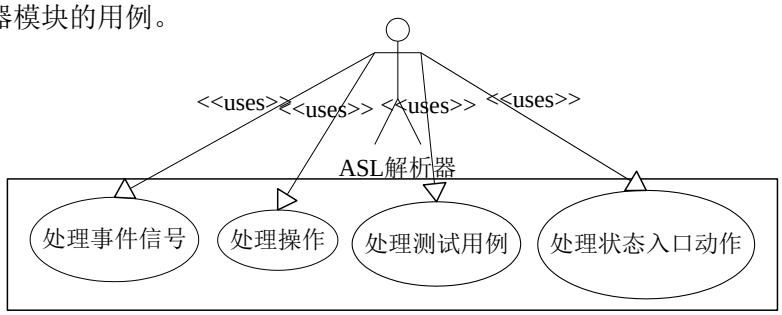

图 3.16 ASL 解析器用例

3.5.4 对象状态机的执行

依据 Rhapsody OXF 的执行原理，本文将对象状态机的执行交由消息驱动框架处理。消息驱动框架在信号事件的激励下对 PIM 的动态行为层进行调度，具体过程是用户注入信号

事件到消息驱动框架，消息驱动框架进行信号事件的分发和管理，在适当的时候分发信号事件给状态机对象，进行信号事件的处理，达到可执行的目的。

整个过程由执行进程实现，消息驱动框架中的执行进程包括主进程、状态机进程、消息发送进程和时钟进程。其中主进程通过 OS 原语创建，它是其它三个进程的父进程。主进程一旦启动之后会立即创建消息发送进程和时钟进程，它不停地轮询事件队列，如果发现事件队列中有未决信号事件则创建状态机进程。信号事件的发送和接收分别委派给消息发送进程和状态机进程处理。时钟进程处理信号事件中的时间事件，并监视状态机进程的状态。图 3.17 和图 3.18 分别说明了四进程的生命期特征和相互关系。

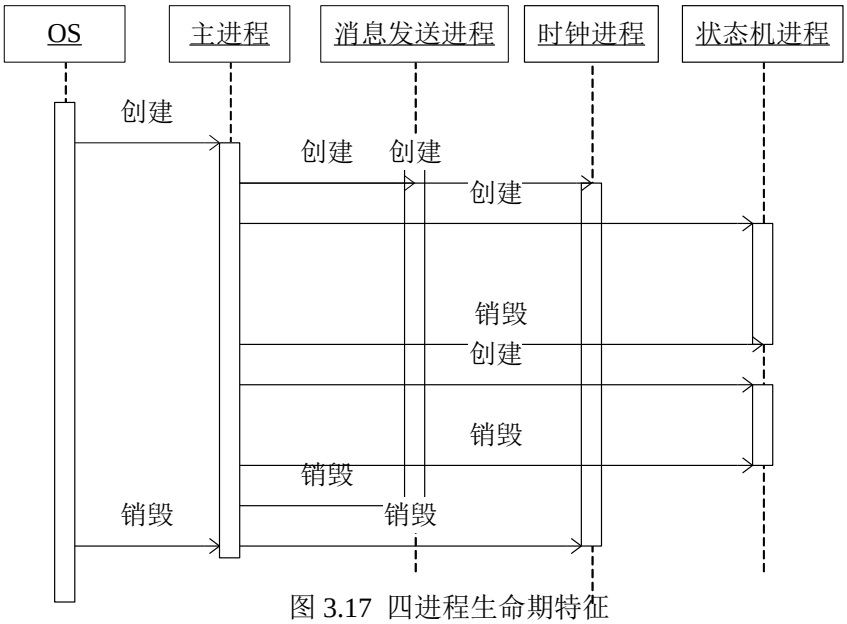

图 3.17 四进程生命期特征

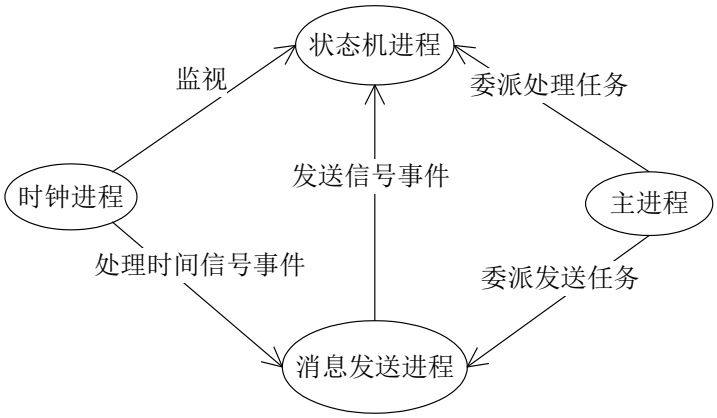

图 3.18 四进程相互关系

消息驱动框架通过消息发送进程来模拟外部信号事件激励，信号事件产生后进入主进程维护的事件队列。主进程一旦检测到事件队列为非空，则创建状态机进程，状态机进程负责执行对象状态迁移图。在执行状态迁移图的过程中，时钟进程会记录状态机对象在每个状态下的执行时间，如果发现超时情况则直接通知主进程销毁状态机进程并抛出异常说明。当状态机进程执行完对象状态迁移图之后，状态图没有到达终止结点，状态机进程在退出之前会保存对象状态迁移图的历史状态结点信息，作为下一次执行的初始状态结点，否则输出在执

行过程中产生的信号事件序列以及记录的标记信息并重置历史状态结点为初始结点。四进程各司其职，相互合作，共同完成对象状态机的执行任务。消息驱动流程如图 3.19 所示。

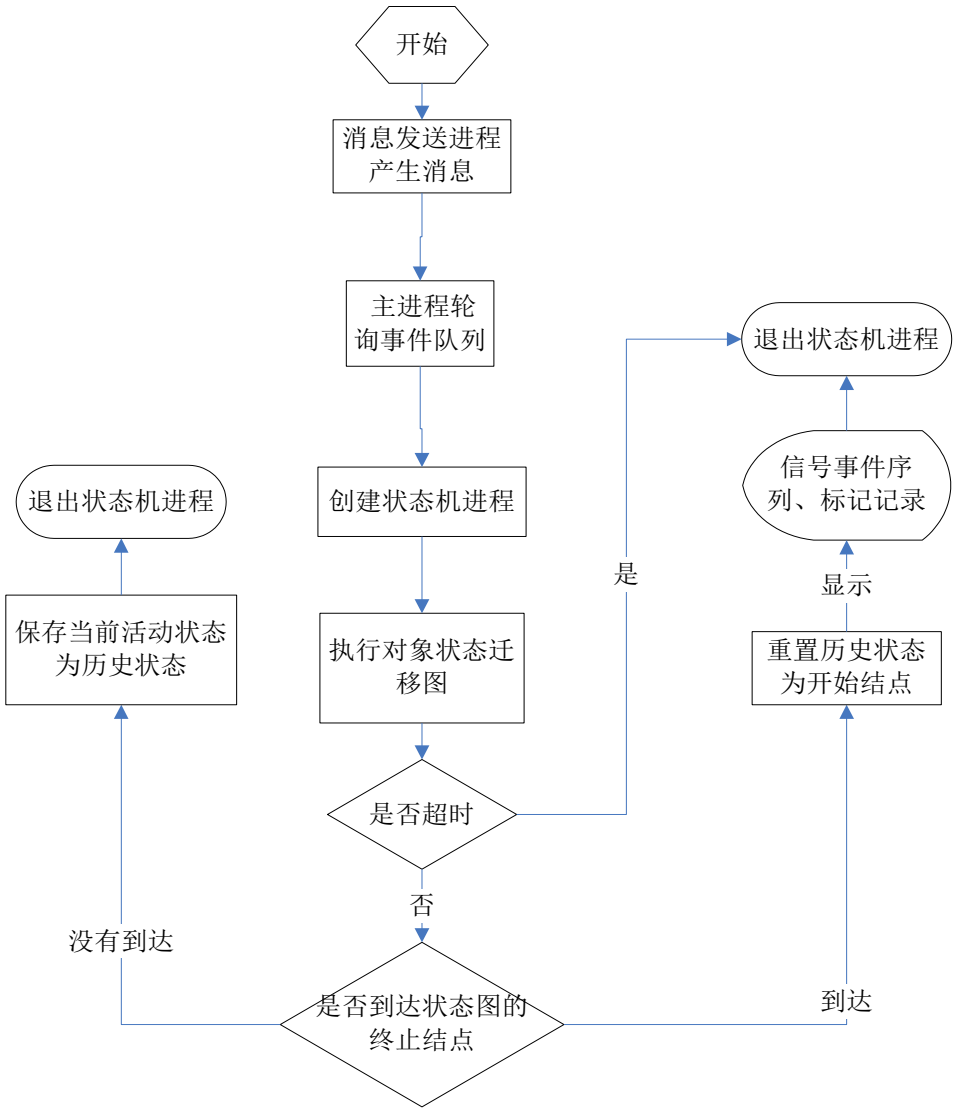

图 3.19 消息驱动流程图

- 消息发送进程

消息发送进程给状态机进程递送事件，它由一个事件类构成，主要维护 EVENT 数据结构。事件 EVENT 数据结构各个字段信息如表 3.2 所示。

表 3.2 EVENT 数据结构

字段名	字段类型	描述信息
event_type	unsigned int	事件类型信息，分未定义类型、信号类型、时间类型、条件类型。
event_id	unsigned int	事件 ID，用来区分各个事件。
isSyn	Bool	判断事件是否为同步调用。
event_msg	String	事件内容，包括事件名、参数等。

事件类有两个派生类分别为事件收集类和事件发送类。消息的收集和发送由这两个派生类完成，事件类结构信息如图 3.20 所示。

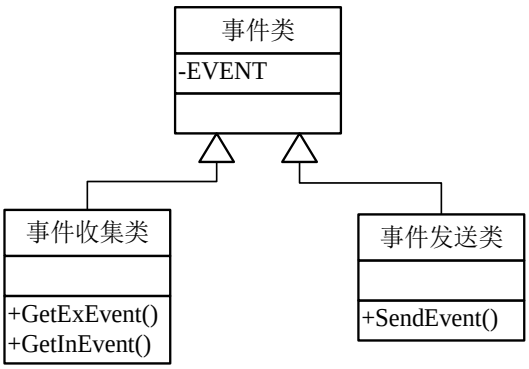

图 3.20 事件类结构信息

GetExEvent 函数获取外部激励消息，主要有时间事件、信号事件和条件事件，GetInEvent 函数获取状态机的自导信号，它由状态机进程产生。获取到的事件按 FIFO 关系存放到主进程的事件队列中，如果事件队列已满，则挂起等待。这两个函数模拟验证的触发条件。

SendEvent 函数在发送事件队列里的事件到状态机进程之前会判断事件队列的标识值，如果为真，表示队列非空，此时从事件队列中获取事件并发送，否则挂起等待。图 3.21 描述了消息发送进程工作流程图。

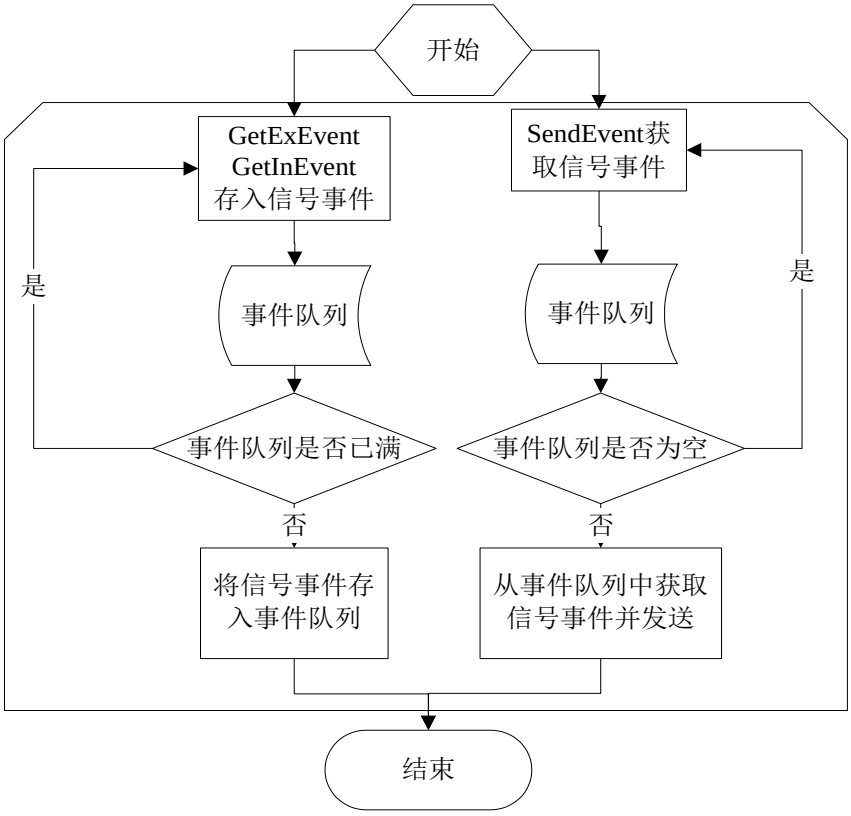

图 3.21 消息发送进程工作流程图

- 时钟进程

时钟进程主要完成以下功能。

1．负责时间事件的分发，通常用来发送采样信号。

2．超时事件的处理，将超时的时间事件添加到事件队列中。

3．防止状态机进程中的死循环。

时钟进程维护一个时钟管理器，它接收应用程序产生的时间事件，在时间事件过期时添加到事件队列中。它还监视对象在某个状态下的执行时间，一旦超时，则直接剥夺执行权力，抛出异常。图 3.22 描述了时钟进程的工作流程图。

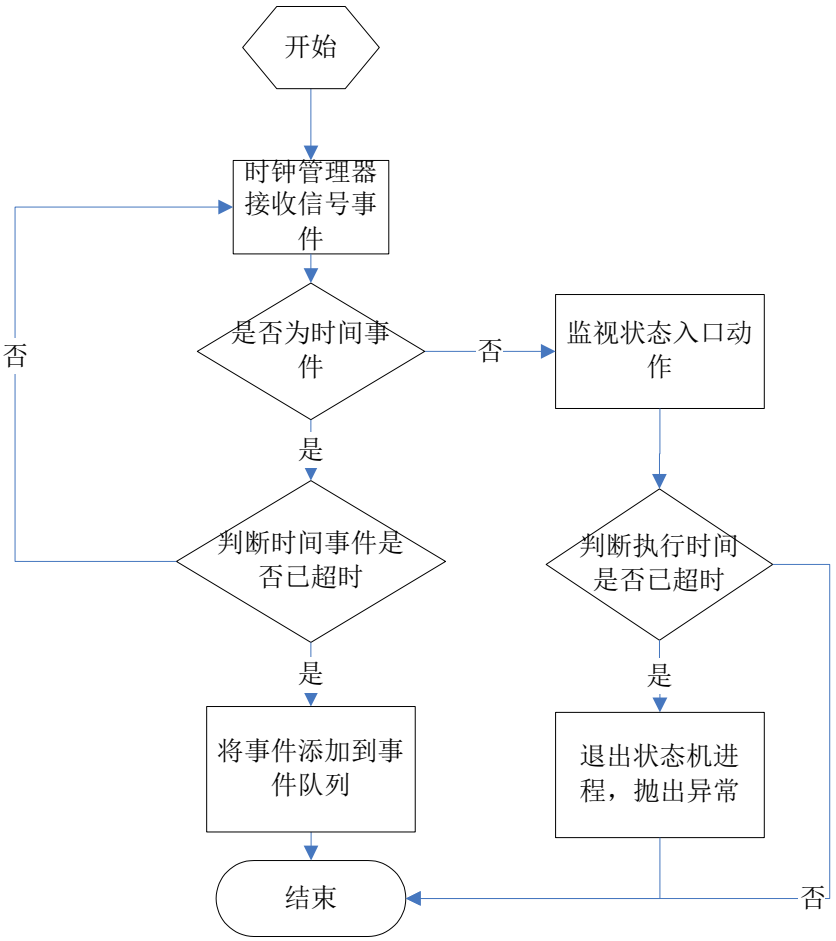

图 3.22 时钟进程工作流程图

- 状态机进程

状态图中每一个状态结点都有对应的入口动作，它描述了对象在该状态下的执行序列。状态机进程根据信号事件实现状态结点的跳转并执行对应的入口动作，在执行过程中产生信号事件序列并更新对象实例表中的属性值。下面通过状态机进程的执行函数清单说明它的工作过程。

```
void MainExecute(event) {
    While(1){//进入循环状态
        signal(event);//状态机接收到信号事件
        /*如果当前能响应该信号事件的状态结点为活动状态*/
        if(IsActive(current_node, event)) {
```

```
            /*根据当前状态结点对信号事件做出响应，该响应在状态的入口动作中完成，
            然后转移到下一个状态结点并将该结点设置为活动结点，函数返回一个内部
            信号即自导信号，如果没有则返回 NULL */
                EVENT Res = StateChartExecute(current_node, next_node, ASLParse/*调用
        ASL 解析器*/);
                /*如果没有产生自导信号，则将转移后的结点作为活动状态结点，并退出
                状态机进程执行函数*/
                if(Res == NULL) {
                    ActiveNode = next_node; break;
                } else {
                    /*如果产生了自导信号，则将下一个状态作为活动状态，同时判断该
                    活动状态是否是结束状态，如果是则状态图执行完成，活动状态重
                    置为初始状态并退出状态机进程执行函数，否则继续循环*/
                    current_node = next_node;
                    ActiveNode = next_node;
                    if(IsFinish(next_node)) {
                        Reset(ActiveNode);
                        return; /*退出 while 循环*/
                    }
                    event = Res;
            /*如果信号事件作用的状态不是活动状态则忽略该信号，退出状态机进程执行函
        数*/
            } else {
                return; /*退出 while 循环*/
            }
        }
    }
```

其中 StateChartExecute 函数定义了状态结点根据信号事件产生的跳转规则，结合跳转规则给出状态机的工作过程描述。

(1) 定位到状态图的历史状态结点，标记状态结点为活动状态。

(2) 如果历史状态为初始状态，则自动迁移到它的后继状态结点，并标记后继状态结点为活动状态。

(3) 如果活动状态能对当前信号事件做出响应，并且活动状态为：

(a) 简单状态结点，则根据结点的入口动作决定下一个后继状态结点。

(b) 复合状态结点，则从状态的嵌套链表中找到初始子状态，并自动迁移到初始子状态的后继子状态结点。

(c) 终止状态结点，则跳转到(6)，并将活动状态重置为初始状态结点。

(d) 子状态的终止状态结点，则根据父状态的入口动作决定下一个后继状态结点。

(e) 条件状态结点，则根据结点入口动作中的守卫条件决定下一个后继状态结点。

(f) FORK 状态结点，自动迁移到后继状态结点。

(g) JOIN 状态结点，自动迁移到后继状态结点。

后继状态结点的入口动作有可能产生自导信号，标记后继状态或者子状态为活动状态。

(4) 重复执行(3)直到活动状态不能对信号事件做出响应或者信号事件为空。

(5) 将活动状态保存为历史状态，跳到(7)。

(6) 输出执行过程中保存的信号事件序列。

(7) 退出状态机进程执行函数。

状态机进程工作流程图如图 3.23 所示。

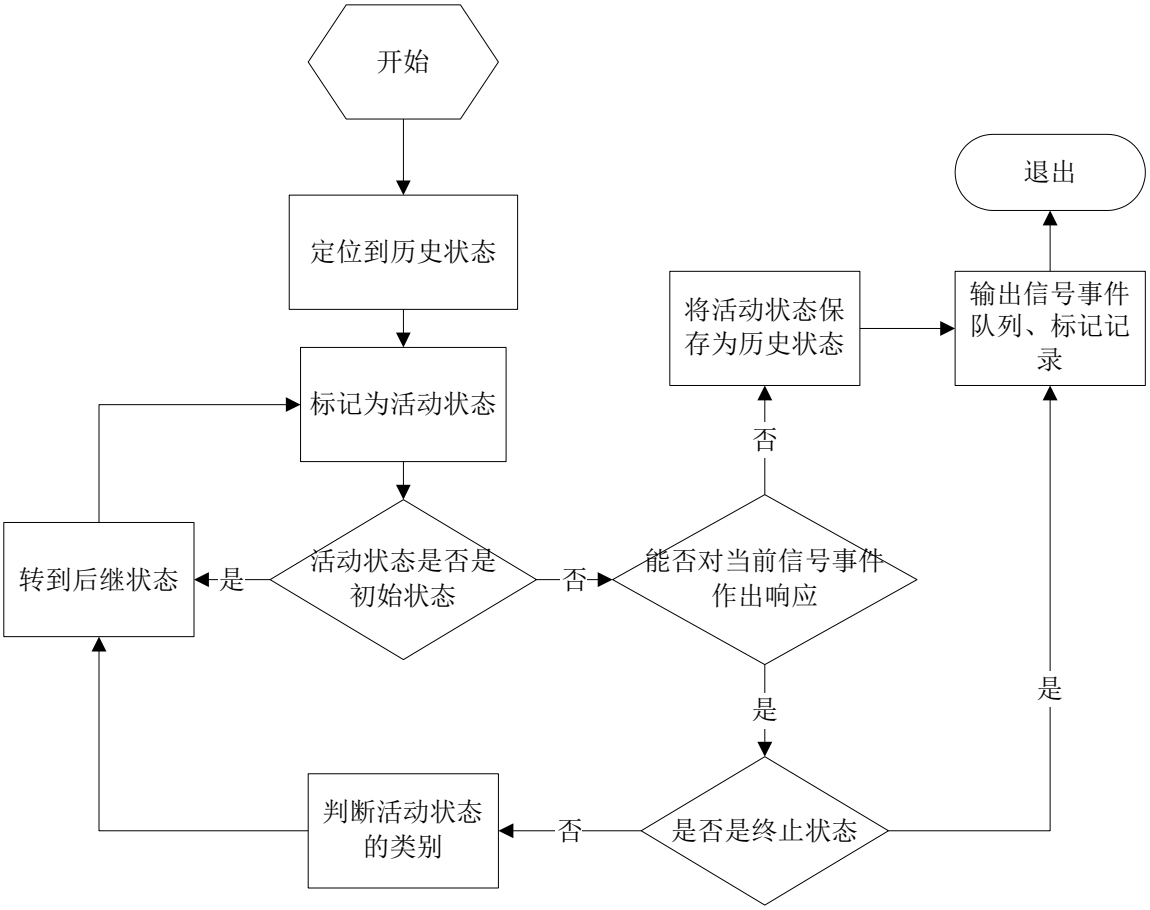

图 3.23 状态机进程工作流程图

- 主进程

主进程主要完成以下功能。

1．调用实例生成器初始化执行现场，检索对象实例表。

2．负责创建和销毁状态机进程，消息发送进程以及时钟进程。

3．维护待发送的 EVENT 事件队列，通过定期轮询事件队列更新队列标识值。

图 3.24 描述了主进程的工作流程图。

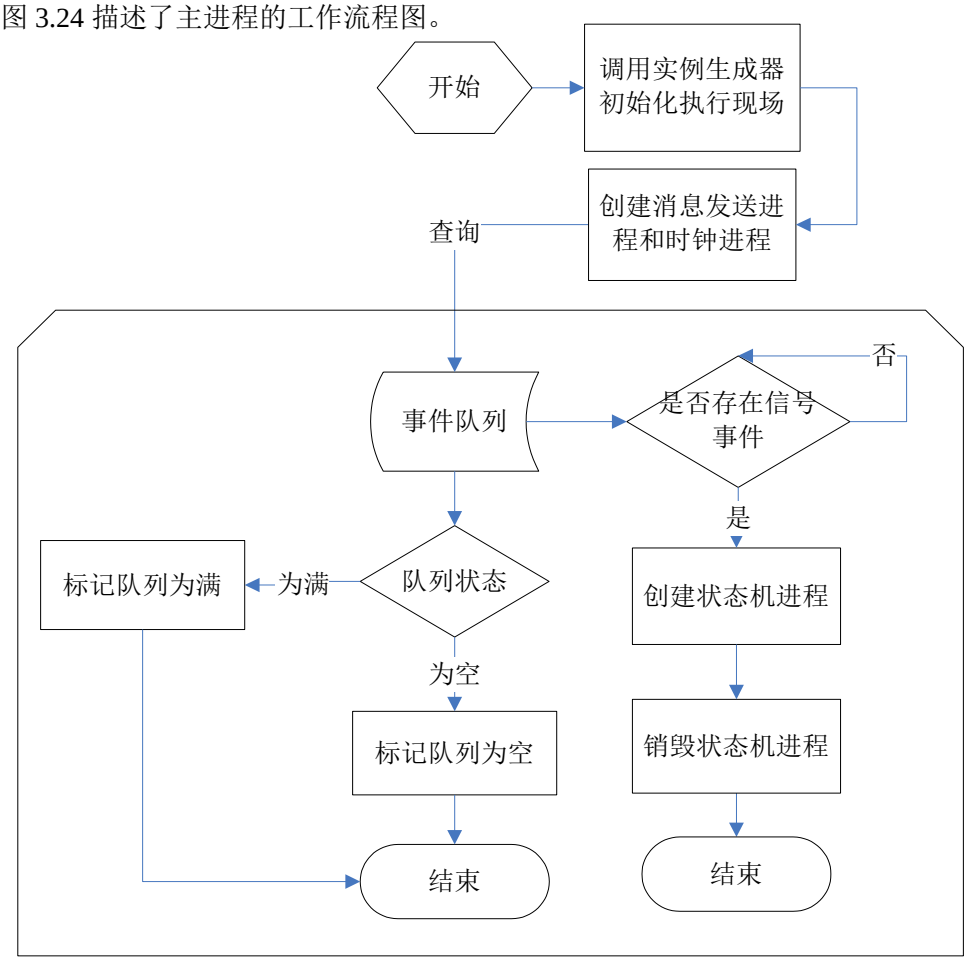

图 3.24 主进程工作流程图

3.6 自定义功能插件

为了更好地提升用户体验，除了完成建模和可执行两大框架外还设计增加了一些其它的功能，这些功能并不是工具必须具备的，它们主要是为了辅助建模框架和可执行框架。它们被组织成自定义功能插件。表 3.3 总结了几个主要的自定义功能插件以及它们的作用。

表 3.3 自定义功能插件清单

插件名称	功能描述
控制台插件	从 Eclipse 的控制台中输出模型验证的相关信息。
工具栏和菜单插件	在原先 Eclipse 的工具栏和菜单上添加一些便利性按钮，如为模型元素添加标记按钮。
模型管理器插件	将模型的层次结构信息以工程树的形式显示在视图中。
属性页插件	在原先的属性页基础上增加属性项，如状态结点的入口动作输入框。

下面结合控制台插件简要说明自定义功能插件的开发过程：

(1) 通过 Eclipse 工程设置向导新建一个“Plug–in Project”，工程名为“ExecuteMemo”，选择默认模板后生成一个工程文件目录。编辑 src 文件夹中 ExecuteMemo.handlers 包下的 java 文件。

(2) 由于验证信息是在可执行框架执行完状态图之后产生，控制台其实是加载了可执行框架生成的 Memo.log 文件(里面存放验证信息)。定义 ReadLogFile 函数，通过 ConsolePlugin 得到 ConsoleManager 对象，并把新建立的 console 对象添加进去。最后新建一个 MessageConsoleStream 对象，该对象用于将模型验证信息显示到 console 对象的视图中。

(3) 通过 Eclipse 工程设置向导新建一个“Feature Project”，在 feature.xml 文件的插件片段向导页中添加(2)中生成的插件，生成功能部件。

(4) 通过 Eclipse 工程设置向导新建一个“Update Site Project”，在“Site Map”页添加(3)中生成的功能部件，并在“Archives”页中指明更新站点信息。

(5) 重启 Eclipse，并在软件安装向导找到(4)中的更新站点，向 Eclipse 添加功能部件。再次重启 Eclipse，此时的 Eclipse 已经拥有控制台插件所具备的功能。

其它的插件也都按照类似的步骤来完成。

3.7 本章小结

本章介绍了工具原型的设计与实现。工具原型包括建模框架、可执行框架以及自定义功能插件。建模框架使用 GMF 做为基础插件，根据可执行 PIM 的要求对建模功能进行改进，并在这基础上对模型文件进行转换。可执行框架主要由状态解析引擎、实例生成器、ASL 解析器和消息驱动框架组成，它们主要完成执行模型文件的功能。工具原型是进行平台无关模型验证的基础设施。

第四章 可执行 PIM 工具原型的应用

第二章研究了可执行 PIM 建模理论和方法，解决了 SysML 不可执行的问题，第三章设计实现了针对可执行 PIM 进行分析验证的工具原型。本章主要在前两章的基础上实现可执行 PIM 的验证功能，完成可交付 PIM 工件，弥补 Rhapsody 等其它建模工具在 SysML PIM 验证方面的欠缺，可以更高质量地完成 MDD 过程。

4.1 模型验证分析

SysML 实现了系统工程领域中业务概念的抽象，通过 SysML 构建的 PIM 属于 OO 分析中的领域模型(Domain Model)。领域模型是描述业务用例实现的对象模型，针对业务角色和业务实体之间应该如何联系和协作来执行业务用例进行抽象。领域模型从业务角色内部的观点定义了业务用例。该模型为产生标称模型(Nominal Model)中预期效果确定了业务角色和业务实体之间应该具有的静态和动态关系，这些关系限定了领域模型在当前迭代开发的用例场景的范围，通过迭代不断演进。

标称模型作为领域模型的基准，是一种确定性模型，它规定了领域模型的业务准则。基于 SysML 的可执行 PIM 根据需求来描述系统，是需求分析模型(Requirements Model)的重要产出。在系统工程建模领域，需求分析模型是 PIM 的标称模型，它引导了 PIM 中的业务用例。在执行 PIM 的过程中，PIM 表现出的业务用例是不确定的，它与标称模型之间会有偏差。该偏差反应了 PIM 验证的合理性度量。基于需求分析模型的验证标准将重心放在 PIM 的演练上，通过执行特定场景的测试用例来体现 PIM 的业务特征，当所有测试用例都满足需求，PIM 的验证才算完成。图 4.1 描述了系统工程领域建模的验证标准。

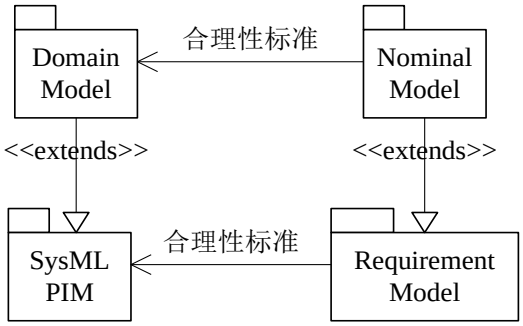

图 4.1 PIM 验证标准

4.1.1 需求分析模型

需求分析模型由需求图和时序图组成。从需求图元模型的组成元素可知它由需求结点、属性结点和测试用例结点组成，需求结点定义 PIM 的业务用例，属性结点描述 PIM 受业务用例驱动的业务事实。

业务用例从黑盒、外部的角度描述了业务实体的运行过程。用它可以推断出业务角色对请求或激励做出何种响应。全体用例集合能够完全地描述出业务角色和业务实体之间的交互过程，这些行为反应了绝大多数的业务需求。每个用例表示业务实体所需的一种特定类型的行为，它可以驱动 PIM 的行为模型，并指导该 PIM 的模拟运行和测试。业务角色与业务实体之间的每次交互都会改变业务实体中的业务事实，它通过需求分析模型中的属性结点呈现出来。在 SysML 构建的领域模型中，业务事实通过块结构图成员属性的方式形式化。需求分析模型中的每个用例都对应一个业务事实的快照，当执行 PIM 时，块结构图成员属性的变化可以反映当前业务实体的用例情况。图 4.2 描述了业务用例和业务事实的关系。

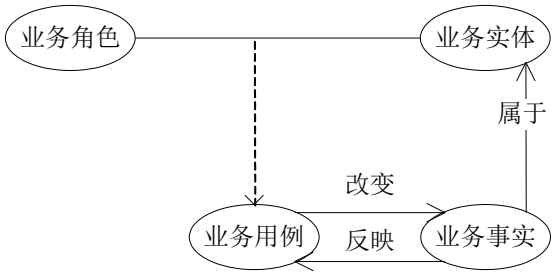

图 4.2 业务用例和业务事实的关系

用例通常用业务角色和用例实体之间交互的事务序列来描述，它由建模人员和领域专家共同决定，通常以用例描述的方式文档化。用例描述定义了业务实体需要实现什么功能，而不是怎样实现，体现了 PIM 的业务需求。用例在一个特定的场景中定义，业务实体在不同的场景中展现不同的行为模式。例如一个自动饮料机系统，参与者投币并购得饮料是其中的一种场景。事务的不同序列对应不同的场景，这些场景中有一个主场景，它表示了系统的常规行为。除了主场景外，还有多个次场景，通常用来表示系统的异常情况。表 4.1 给出在主场景下的自动饮料机系统的用例描述。

表 4.1 主场景下的自动饮料机系统的用例描述

系统需求	自动售货
参与者	顾客
场景名称	成功购买饮料
用例描述	1.参与者向系统投入硬币 2.系统显示已投硬币数目 3.系统显示当前硬币数目下可以购得的饮料 4.参与者选择其中一种饮料的按钮 5.系统递出饮料 6.系统判断是否需要找零 7.参与者拿走饮料 8.系统重新待机

从表 4.1 的用例描述中可以看出业务角色与业务实体之间的交互过程具有时序性，为了增加需求分析模型的直观性，本文用 SysML 时序图描述一个用例。表 4.1 中的用例描述可以用如图 4.3 所示的 SysML 时序图表示。

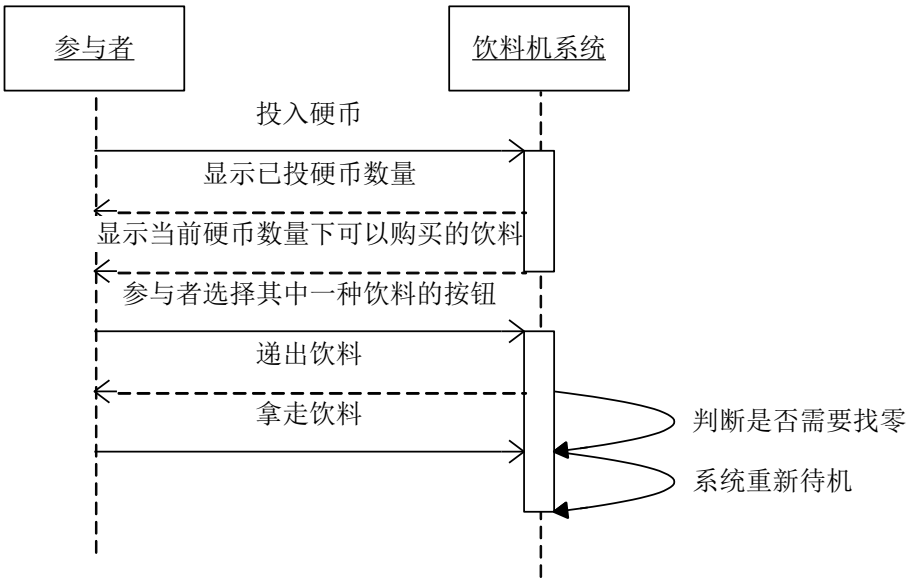

图 4.3 主场景用例的 SysML 时序图

4.1.2 可执行 PIM 的验证标准

动态行为层中的状态图描述 PIM 的业务用例，在 Moore 机原型中，对象在某个状态下的执行序列由状态入口动作定义。通过不同迁移和状态组合可以产生出不同的执行序列。状态图语义表明迁移是由事件的激励而引起，当对象接收到外部激励信号时，表现的行为由对象所在状态的入口动作决定。此时对象可能会产生自导信号并在自导信号的作用下迁移到新的状态，也有可能停留在原状态等待新的外部激励，对象在这两种情况下产生收发信号事件的序列。这些信号事件序列粗粒度地描述了 PIM 中业务用例的过程。

静态结构层中的块结构图描述系统主题事务的业务实体，它拥有与业务事实相关的一些特性，这些特性被表示为块的成员属性。块结构图是声明性规约，代表了用于支持业务计算所需要的数据。将具有业务事实的数据作为标记(例如在电梯系统中，数据 0 表示关门，数据 1 表示开门)，当执行对象状态机时，业务用例在 PIM 中映射为标记的变化(例如电梯门对象在接收到开门信号后执行开门动作，在虚拟的仿真环境中[48]，这个开门动作是通过将标记值由 0 改变为 1 来完成)。通过跟踪记录标记变化轨迹可以模拟出实际系统业务事实的变化。状态入口动作使用动作规约层中的 ASL 来描述，根据第二章中 ASL 的关键特性可知，状态入口动作能够操作对象的成员属性，这是标记发生变化的充分条件。处于某个状态下的对象在受到信号事件激励后都会产生一个对应的标记快照，整个状态迁移过程产生出一个与信号事件序列对应的标记变化轨迹，它细粒度地表现了 PIM 中业务事实的变化。

由于对象状态机的执行是信号事件驱动的，每个激励的信号事件都对应一个标记快照，因此，本文从两个方面评判可执行 PIM 的验证标准：

- 从信号事件角度分析 PIM 是否符合需求分析模型中的业务用例；
- 从标记跟踪角度分析 PIM 是否符合需求分析模型中的业务事实。

4.2 模型验证机制

基于 xSysML 的可执行 PIM 从层次上分为静态结构层、动态行为层和动作规约层。位于静态结构层块结构图中的每个块都与其他一定数量的块进行协作以在整体系统中发挥它的作用，协作的方式主要是通过块对象之间的同步调用操作和异步信号发送来实现。由于动态行为层中的状态图模型可以同时具有同步调用操作和异步信号发送的功能，本文为块结构图中的每个块结构都建立状态图。状态图的集合为整个 PIM 确定了一个协调的策略集合，本文称之为块结构协作模型(Block Cooperative Model)。BCM 表达了 PIM 在对象级别上的交互过程，这些交互过程最终会在模拟 PIM 运行的时候作为预期结果使用。

块结构对象的生命期从可执行框架执行状态图的那一刻开始直到执行完毕，执行中的状态图称为对象状态机，每个块结构可以有多个对象状态机，它们共享同一个状态图副本。客户对模型的激励映射为信号事件对块结构对象状态机激励，在执行状态机的过程中，xSysML 语义允许引用模型元素、操纵和关联对象以及发送同步和异步消息，块结构图中所有块的状态以及块之间的关联关系都可以通过状态机的执行反映出来。BCM 的基本策略包括：

- 确定块结构图中的《terminator》，它是模型验证时需要初始化的块结构对象，用来接收客户对模型的激励。
- 自上而下地创建动态块结构对象。
- 自底向上地删除动态块结构对象。
- 委派消息(同步调用和异步信号)自上而下传送。在任何时刻，如果一个块结构对象需要其它对象帮助它来完成一个任务，则它通过向下属传递消息的方式委派任务。
- 反馈消息(同步调用和异步信号)自底向上传送。当一个块结构对象完成上属对象的任务后，它向上属发送完成的反馈消息。

BCM 使得每个块结构对象处于相互协作的模式中，当执行《terminator》状态机时可以在整个 PIM 的抽象层次上描述行为，实现模型验证。

建模框架构造完成可执行的平台无关模型后，消息驱动框架中的主进程调用实例生成器和 ASL 解析器模块，为模型的执行做初始化准备。主要分两步完成：首先，通过实例生成器为 PIM 模型中的每个块生成一张空的对象实例表。其次，ASL 解析器模块处理 PIM 模型验证的初始化 ASL 代码，这些代码通常是一条或者多条 create 语句，用于创建《terminator》块的对象，实例生成器将这些对象的初始信息按行填入各个块对应的对象实例表中。

初始化结束后，主进程会检索并判断静态结构层中每个块的对象实例表，如果表非空，则进行下面处理调用状态解析引擎将对应块的状态图信息提取到 Kripke 结构中，每个块状态图对应一个 Kripke 结构。此时块对象就能够接收来自客户的信号事件激励，例如输入用于执行平台无关模型的测试用例。主进程在轮询事件队列的过程中一旦发现队列中存在信号

事件，则为目标对象实例创建状态机进程。由于对象实例可能不止一个，因此可能会同时存在多个状态机进程的情况。

在执行对象状态机的过程中，信号事件驱动状态入口动作，它主要完成了两个任务：1.改变对象的属性，工具原型会跟踪记录那些被选为标记的块对象属性的变化过程。2.将块对象的当前状态设置为历史状态，或者产生新的信号事件进行下一步迁移。一旦到达了对象的终止状态结点或者此时没有信号事件激励时，主进程就会销毁该对象的状态机。如果到达了对象的终止状态结点，模型管理模块还将清空对象实例表中对应的表项。通常一个测试用例带动了一次 PIM 的验证过程。图 4.4 描述了单个块的 PIM 的验证过程。

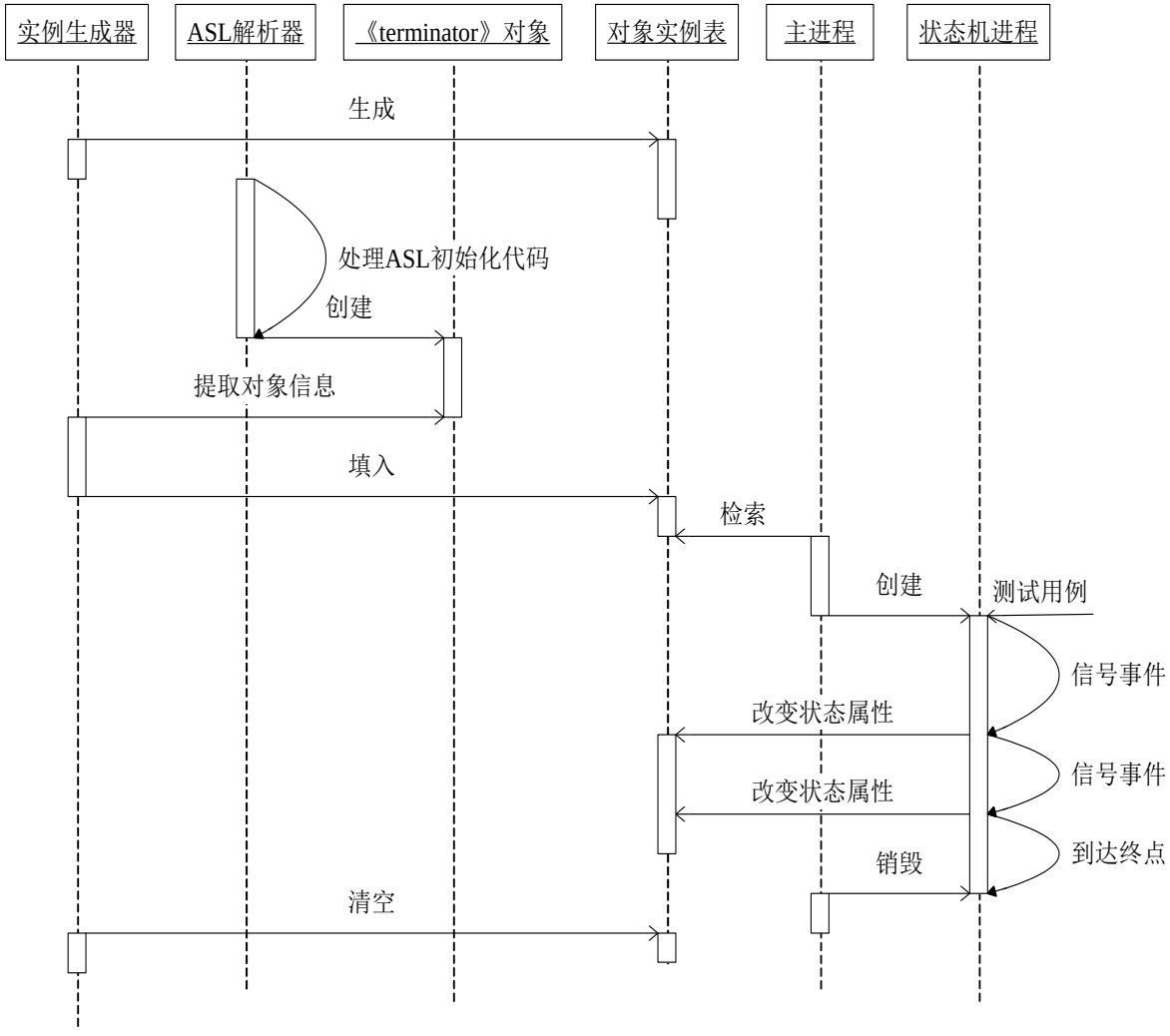

图 4.4 PIM 验证过程

4.3 模型演练

基于 xSysML 的 PIM 具有精确的执行语义，可执行性使得 PIM 从原来的设计文档演变为面向对象的解决方案，工具原型通过演练可执行 PIM 可以模拟系统工程领域模型业务事实的变化情况。这些变化情况为领域专家评估系统提供了很大的参考价值，例如通过模型演练来配置合理的参数使得模型行为能够符合预期，这种预先性降低了系统工程的实施风险。

4.3.1 案例需求分析

列车运行系统控制列车的行驶，主要由牵引设备和旅行日志两部分组成。旅行日志记录各个车站间的期望行驶曲线，它维护了列车行程与期望速度之间的关系。牵引设备控制列车实际运行速度，驾驶员根据期望行驶曲线控制牵引设备来调节速度。

牵引设备通过牵引级参数的设置来调节速度，按照期望行驶曲线为牵引设备设置牵引级是列车运行系统自动驾驶模式的主要部分。为了获取某型号机车自驾模式牵引参数，领域专家需要借助可执行 PIM 来模拟出某型号机车按照期望行驶曲线的牵引级序列。通过工具原型绘制出列车运行系统自驾模式需求图如图 4.5 所示。

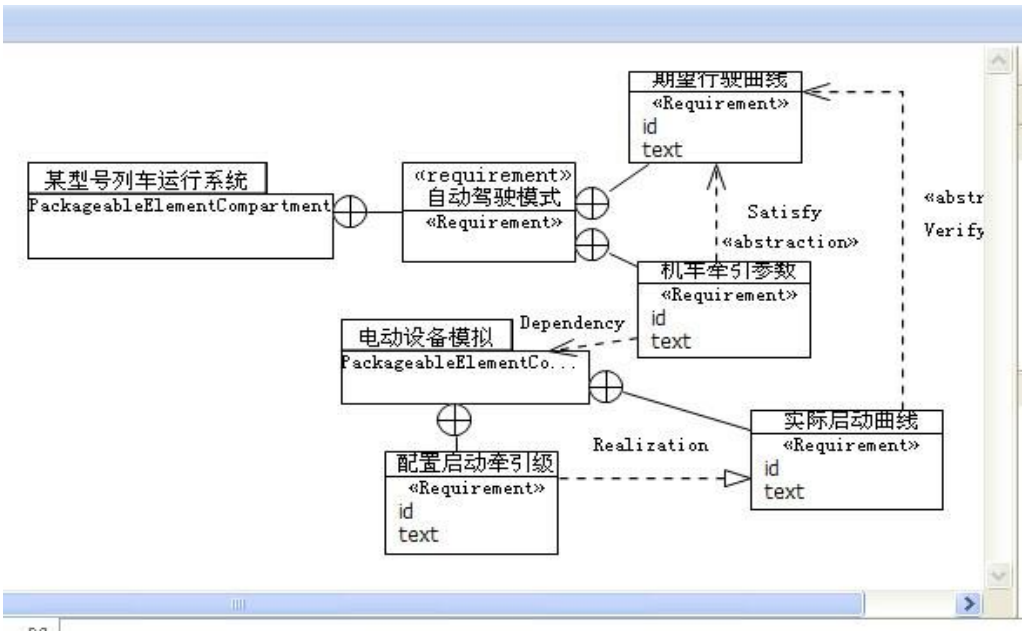

图 4.5 列车运行系统自驾模式需求图

电动设备模拟的核心部分是机车的速度特性和牵引力特性。机车速度特性描述机车运行速度 v 与牵引电动机电枢电流 Ia 之间的关系曲线。工具原型的验证环境中没有考虑机车动轮摩擦系数，在一定的电网电压下，某型号机车的速度特性按下列函数控制。

$$Ia = 100Gears \quad \ldots\ldots(1)$$

$$Ia = 400Gears - 75v \quad \ldots\ldots(2)$$

$$Ia = 820 \quad \ldots\ldots(3)$$

式中 Gears(0 ~ 13)代表牵引级，Ia 取三式中的最小值，如机车在 5 级位启动时，Ia 由 500A 启动，随着速度增加，(2)式中的 Ia 随之变化，当(1)和(2)式中的 Ia 相同时，列车进入准恒速区。准恒速区的列车开始变加速运行，当(2)式 Ia 减为 0 后，列车达到准恒速终值。

牵引力特性描述机车轮周牵引力与电枢电流之间的关系曲线。在一定牵引级下，电机电枢电流 Ia 按本级位启动电流恒流运行，此时机车牵引力恒定。当机车速度到达准恒速点后，机车功率达到级位最大值，机车牵引力随着电流线性下降直到零，由于忽略了摩擦系数，电枢电流与列车加速度之间存在着线性关系。

期望行驶曲线的业务用例为控制列车按照期望速度在站点之间运行，根据业务用例将列车的行驶距离和期望速度列为业务事实。通常列车的旅程由若干路段(相邻站台之间的区间)组成，而路段又是由多个加速曲线组成。业务用例将路段的期望行驶曲线理想化为三段加速曲线：(1) 启动时的匀加速过程，(2) 匀速运行过程，(3) 即将到站的匀减速过程。图 4.6 通过运行剖面图描述了列车管理域的业务用例。

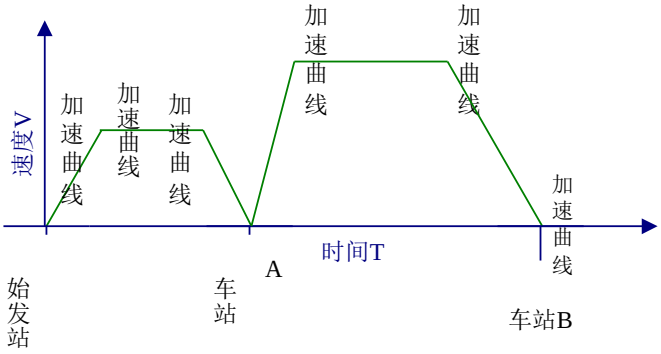

图 4.6 期望行驶曲线

按照期望行驶曲线和机车设备参数，可以在车站之间配置牵引级序列，当前，领域专家首先需要模拟出启动牵引级使得列车运行符合期望行驶曲线的第一段和第二段加速曲线。

4.3.2 构建可执行 PIM

- 静态结构层

静态结构层由六个业务角色组成，分别为列车 Train、牵引设备 Motor、控制台 Console、旅程 Journey、路段 Hop 和加速曲线 AccelerationCurve。这六个业务角色之间的主要关系如图 4.7 所示。

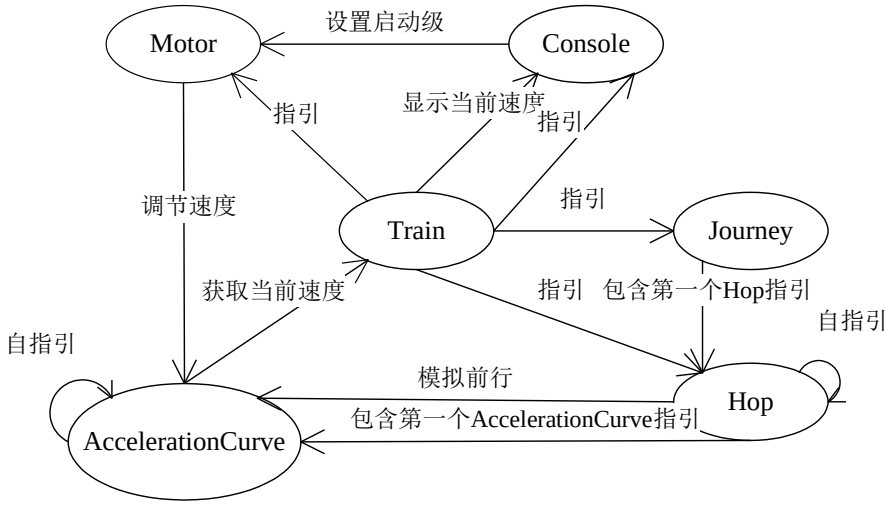

图 4.7 业务角色关系图

业务角色在工具原型中由块结构表示，块结构不同于 UML 类，它提供了 constraints 部分用来说明实际系统中的限制条件。本案例在 Train 的 constraints 中添加了针对期望行驶曲线的限速和限时条件：限速条件定义为[v-4，v+4]，其中 v 表示期望恒速；限时条件定义为

[t-4，t+4]，其中 t 表示到达期望恒速的时刻。如果列车到达准恒速的时刻 t’在[t-4，t+4]，并且恒速值 v’在[v-4，v+4]，则当前启动曲线是符合期望要求的。通过工具原型绘制块结构图如图 4.8 所示。

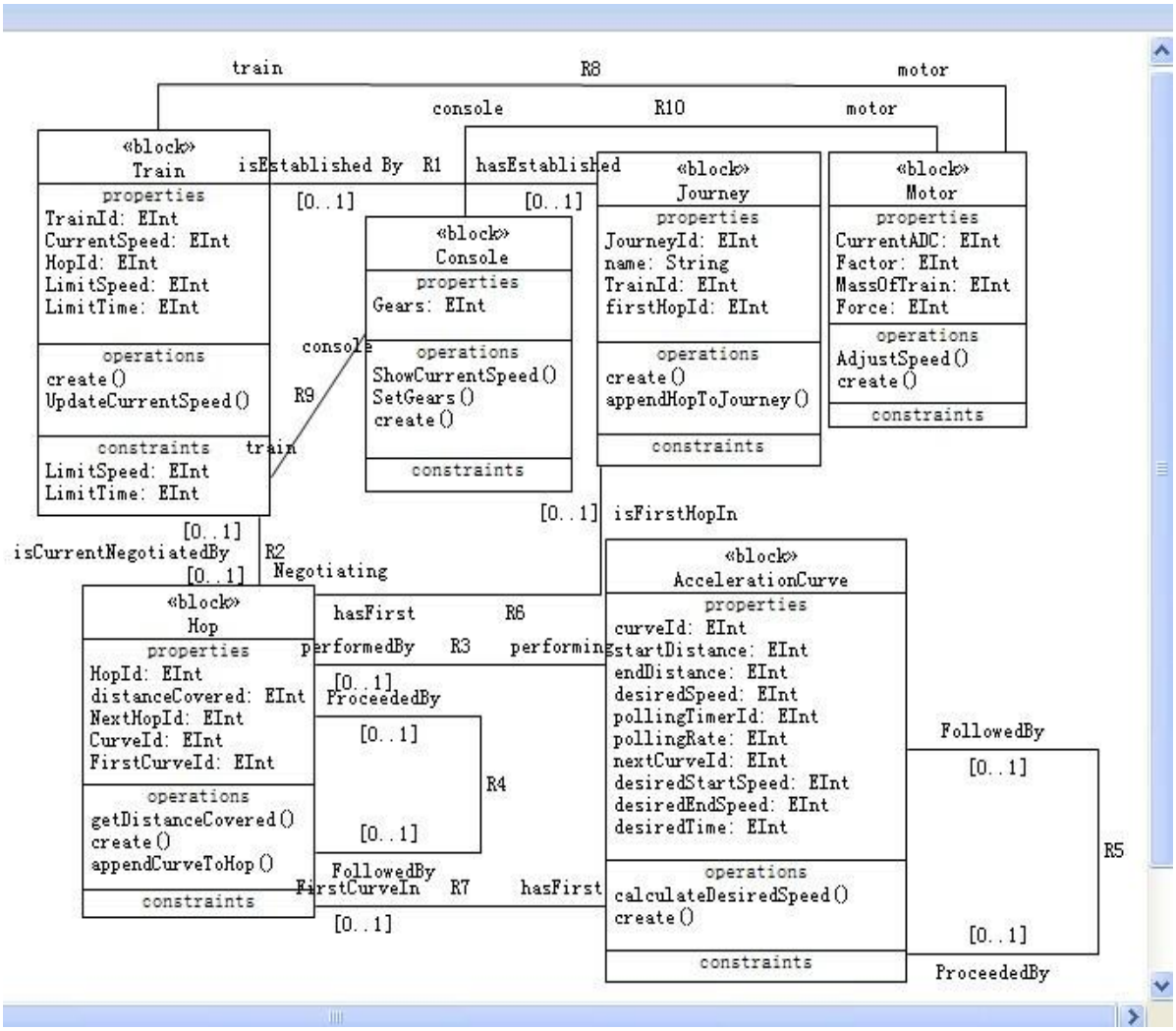

图 4.8 块结构图

关联在块结构图元数据中表示为指引标签，类似 R1 这样的名称代表标签名，关联两端有角色短语标签，例如 R1 两端的“isEstablished By”和“hasEstablished”，角色短语的作用主要是辅助建模人员和领域专家更好地理解模型，例如在自反关联中区分前趋和后继结点。

- BCM

六个业务角色之间进行相互协作以在整体系统中发挥各自的作用，协作的方式主要通过块结构对象之间的同步调用操作和异步信号发送来实现。BCM 展示了对象级的交互，它指明每个对象动态行为之间的相互关系。每个业务角色对应一个有限状态空间，其中 train 状态图包含了 hop、journey、motor 和 console 的子系统，hop 状态图中包含 accelerationCurve 的子系统。在状态图语义中，子系统是宿主的一个复合状态，因此该 PIM BCM 可以抽象为

一个更大的分层状态机，向最顶层状态机发送激励信号可以到达 BCM 每个业务角色的有限状态空间，这种分层的状态机保持了整个块结构图行为的一致规约性。宿主发送给子系统的信号称之为层间信号，层间信号实现了对象级的交互。该 PIM BCM 主要部分如图 4.9 所示。

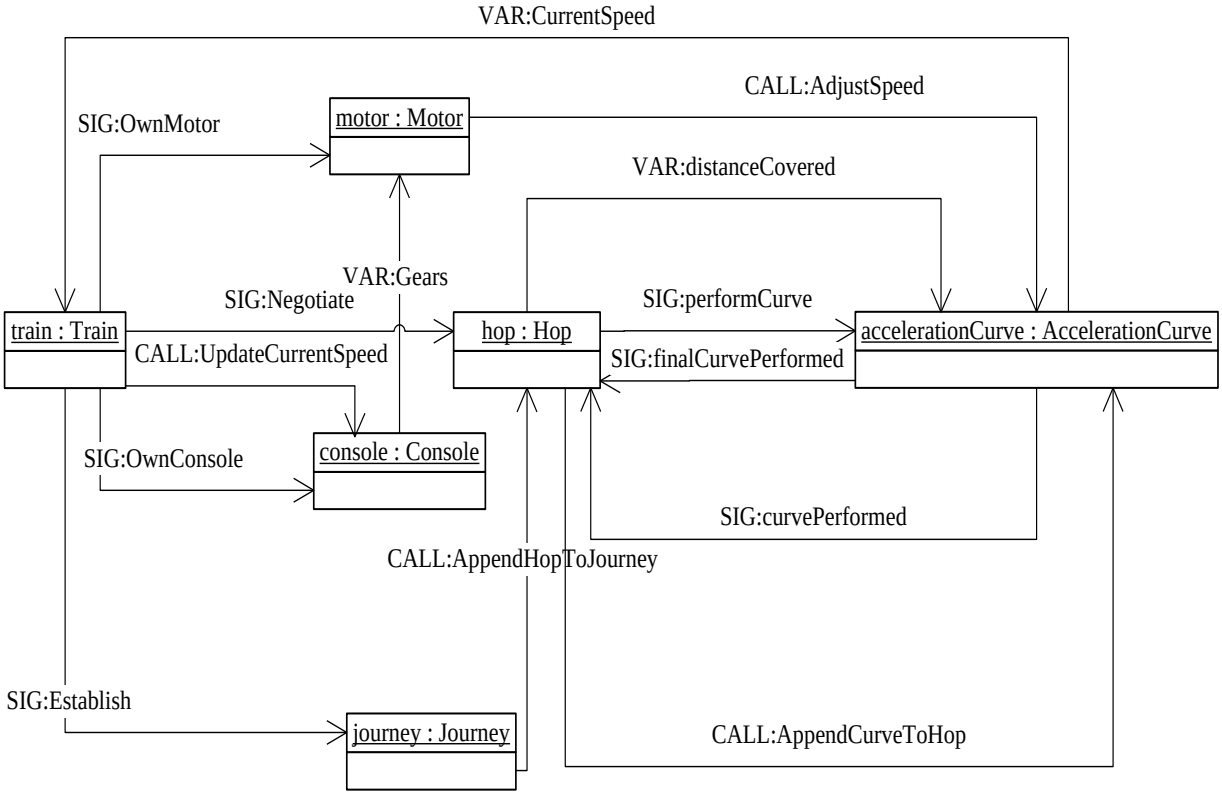

图 4.9 PIM BCM 图

通过 BCM 可以确定

- 块结构图中的《terminator》为 train，它是模型验证时需要初始化的块结构对象，用来接收客户对模型的激励。
- train 向 motor、console、journey、hop 发送层间信号，hop 向 accelerationCurve 发送层间信号。
- 速度的调节通过 accelerationCurve 调用 motor 的 AdjustSpeed 操作来完成，并将 CurrentSpeed 反馈到 train。因此，寻找启动牵引级的工作主要借助 AccelerationCurve 的有限状态空间。
- accelerationCurve 向 hop 反馈 finalCurvePerformed 和 curvePerformed 信号。

- 动态行为层

AccelerationCurve 模拟了一段距离的抽象，AccelerationCurve 对象在创建之后模拟了列车的一段行驶过程，因此 AccelerationCurve 对象在这段距离上必须执行一个持续的操作，这个操作通过轮询信号实现。在轮询信号的作用下，AccelerationCurve 对象完成期望速度和实际行驶速度的计算。轮询信号在 AccelerationCurve 对象到达 AccelerationCurve 终点后退出

并产生终点抵达信号，此时 AccelerationCurve 对象判断当前 AccelerationCurve 是否为 Hop 的第一段并搜索后继 AccelerationCurve，如果后继 AccelerationCurve 不存在，则产生 Hop 完成信号，否则产生 AccelerationCurve 完成信号。使用工具原型绘制的 AccelerationCurve 对象的行为过程如图 4.10 所示。

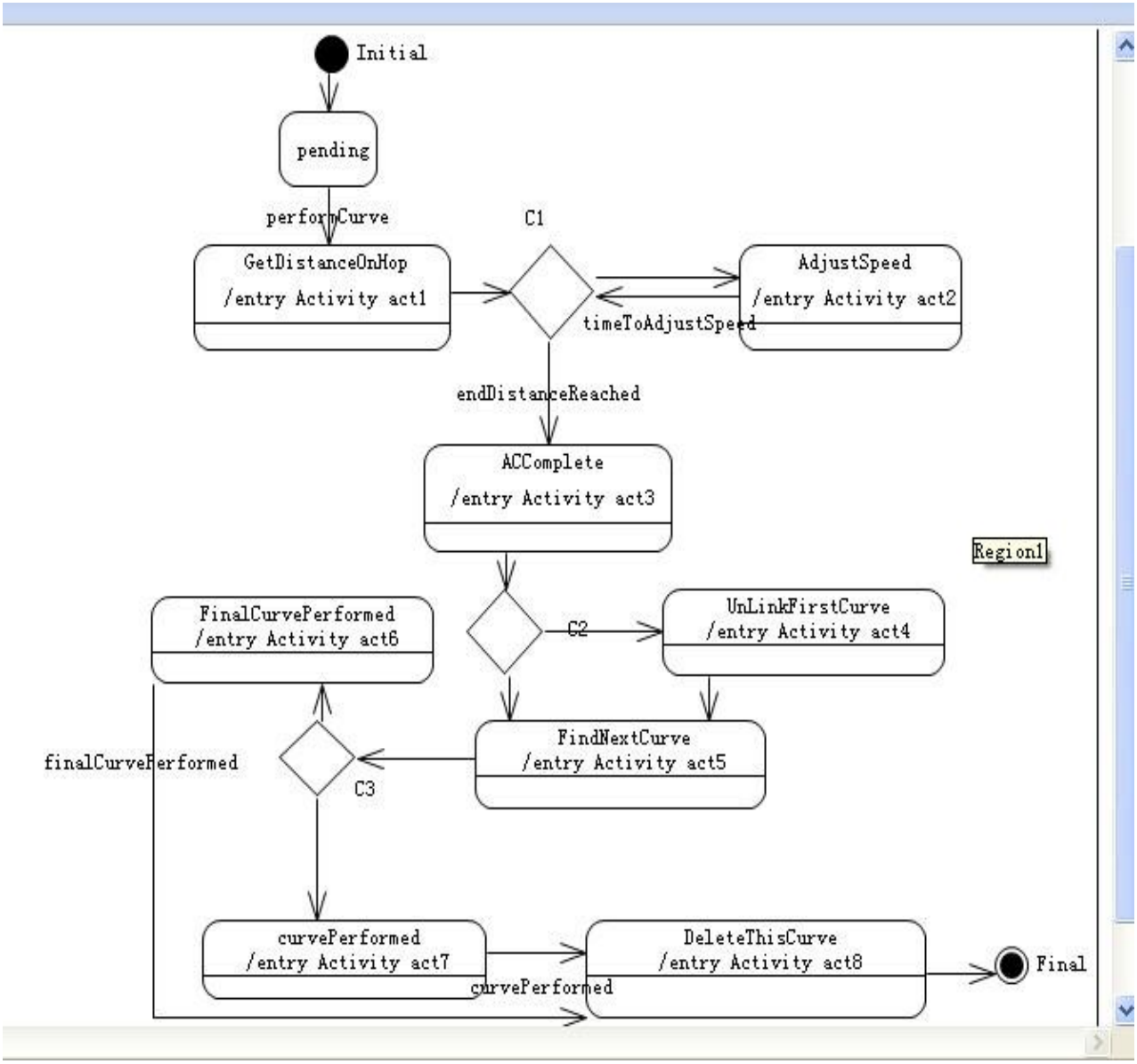

图 4.10 AccelerationCurve 对象行为过程

- 动作规约

entry Activity 在 Moore 状态图中用来定义状态动作，在状态图语义中表示状态属性的类型。ASL 导入对话框建立 entry Activity 类型的变量并通过 setASL 按钮实现变量和 ASL 的关联。使用工具原型为 AccelerationCurve 状态图添加动作规约，右键单击 GetDistanceOnHop 状态，在 ASL 的导入向导中输入入口动作变量名 act1，并在编辑框内输入动作语义，整个入口动作获得 distanceCovered 的值，如图 4.11 所示。

图 4.11 GetDistanceOnHop 状态动作描述

accelerationCurve 其余动作规约如下。

C1: actualDistanceCovered < this.endDistance

act2: train = this -> R3 -> R2

[DesiredSpeed] = AccelerationCurve:calculateDesiredSpeed[distanceCorved] on this

motor = train -> R8

[] = AdjustSpeed[Gears] on motor

Set_Timer(pollingTimerId, pollingRate, timeToAdjustSpeed)

generate timeToAdjustSpeed to this

act3: generate endDistanceReached to this

hop = this -> R3

unlink this R3 hop

C2: theHopContainingFirstCurve = this -> R7

theHopContainingFirstCurve == UNDEFINED

act4: unlink this R7 theHopContainingFirstCurve

act5: nextCurve = this -> R5

C3: nextCurve == UNDEFINED

act6: generate finalCurvePerformed to hop

act7: unlink this R5 nextCurve

link nextCurve R3 hop

generate curvePerformed to hop

act8: delete this

右键点击各个 block 的 operations 部分可以向操作导入 ASL，其中 getDistanceCovered、calculateDesiredSpeed 和 AdjustSpeed 是三个关键操作，它们的主要实现过程如下。

#getDistanceCovered 计算当前轮询时刻 pollingRate 下的距离#

accelerationCurve = this -> R3

Acceleration = (accelerationCurve.desiredEndSpeed * accelerationCurve.desiredEndSpeed - accelerationCurve.desiredStartSpeed*accelerationCurve.desiredStartSpeed)/2*(accelerationCurve.endDistance - accelerationCurve.startDistance)

distanceCovered = accelerationCurve.desiredStartSpeed * pollingRate + 0.5 * Acceleration * pollingRate*pollingRate

#calculateDesiredSpeed 计算 desiredSpeed 过程#

lengthOfCurve, SpeedDiff, SpeedGrad, CurrentAlongCurve, deltaSpeed : EInt

lengthOfCurve = this -> endDistance – this -> startDistance

SpeedDiff = this -> desiredEndSpeed – this -> desiredStartSpeed

SpeedGrad = SpeedDiff / lengthOfCurve

CurrentAlongCurve = distanceCorved – this -> startDistance

deltaSpeed = CurrentAlongCurve * SpeedGrad

desiredSpeed = deltaSpeed + this -> desiredStartSpeed

#AdjustSpeed 函数在准恒速与恒速之间计算 CurrentSpeed 的操作过程#

const1, k : EInt

console = this -> R10

train = this -> R8

const1 = CurrentADC / Force #const1 是牵引特性曲线的斜率#

k = const1 * MassOfTrain #MassOfTrain 是列车质量#

accelerationCurve = this -> R8 -> R2 -> R3

train.CurrentSpeed = (400 / 75) * (Console.gears) – Factor * exp (-75 / k * (accelerationCurve.pollingRate))

#向 train block 的 constraints 部分添加约束条件#

LimitSpeed >= DesiredSpeed – 4

LimitSpeed <= DesiredSpeed + 4

LimitTime >= 2 * LengthOfCurve / SpeedDiff – 4

LimitTime <= 2 * LengthOfCurve / SpeedDiff + 4

4.3.3 PIM 验证

初始化为测试用例定义了一个执行的场景，实例中表现为“一列进行某次旅程的列车正运行在某个路段的某个加速曲线上”。消息驱动框架主进程调用实例生成器创建 8 个 TABLE 类型变量 journey、train、hop、A1、A2、A3、console、motor，并在 ASL 解析器的配合下完成变量初始化，具体步骤如下所示：

(1) #初始化变量 journey，其中标识号为 1，名称为 JTest#

JourneyId = 1 name = ‘JTest’ TrainId = 10 firstHopId = 2

[journey] = Journey:create [journeyId, name, TrainId, firstHopId]

(2) #初始化变量 train，进行“JTest”旅程#

TrainId = 10 CurrentSpeed = 0 hopId = 2 LimitSpeed = 0 LimitTime = 0

[train] = Train:create[TrainId, CurrentSpeed, hopId, LimitSpeed, LimitTime]

(3) #初始化变量 hop#

hopId = 2 distanceCorved = 0 CurveId = 5 firstCurveId = 5 NextCurveId = 6

[hop] = Hop:create[hopId, distanceCorved, CurveId, firstCurveId, NextCurveId]

(4) #初始化 hop 中的三个加速曲线对象变量，轮询率设置为 1 秒#

#1#

curveId=5 startDistance=0 endDistance=50 desiredStartSpeed=0 desiredEndSpeed=10 desiredSpeed=0 pollingRate=1 pollingTimerId=0 nextCurveId=6 desiredTime=0

[A1] = AccelerationCurve:create[curveId, startDistance, endDistance, desiredStartSpeed, desiredEndSpeed,desiredSpeed, pollingRate, pollingTimerId, nextCurveId, desiredTime]

#2#

curveId=5 startDistance=50 endDistance=100 desiredStartSpeed=10 desiredEndSpeed=10 desiredSpeed=0 pollingRate=1 pollingTimerId=0 nextCurveId=6 desiredTime=0

[A2] = AccelerationCurve:create[curveId, startDistance, endDistance, desiredStartSpeed, desiredEndSpeed,desiredSpeed, pollingRate, pollingTimerId, nextCurveId, desiredTime]

#3#

curveId=5 startDistance=100 endDistance=150 desiredStartSpeed=10 desiredEndSpeed=0 desiredSpeed=0 pollingRate=1 pollingTimerId=0 nextCurveId=6 desiredTime=0

[A3] = AccelerationCurve:create[curveId, startDistance, endDistance, desiredStartSpeed, desiredEndSpeed,desiredSpeed, pollingRate, pollingTimerId, nextCurveId, desiredTime]

(5) #初始化变量 motor#

CurrentADC=0 Factor=1 MassOfTrain=1000 Force=0

[motor] = Motor:create[CurrentADC, Factor, MassOfTrain, Force]

(6) #初始化变量 console#

Gears=0

[console] = Console:create[Gears]

消息驱动框架主进程判断 AccelerationCurve 对象实例表为非空状态后调用状态解析引擎创建 AccelerationCurve 状态图的有限状态空间并检查合理性，该有限状态空间结构如图 4.12 所示。

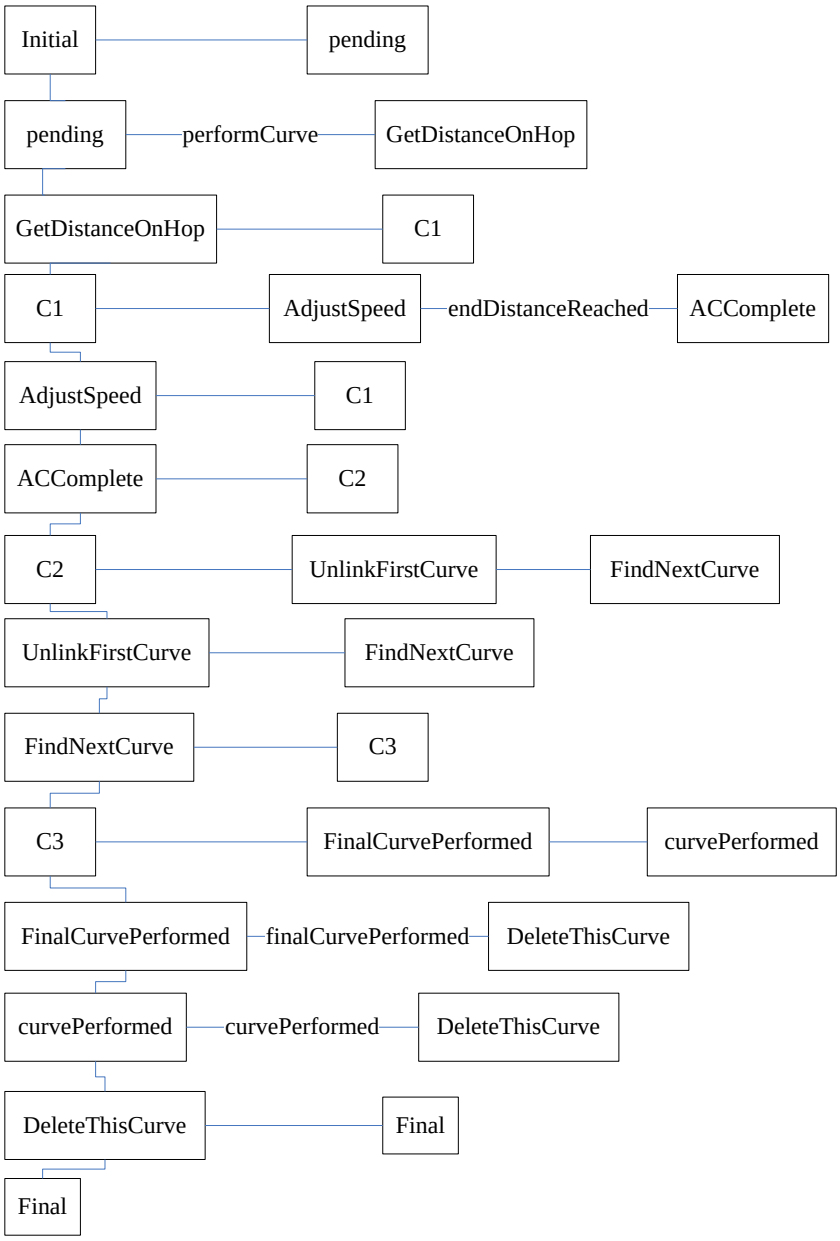

图 4.12 AccelerationCurve 有限状态空间

领域专家需要获取列车在不同 Gears 下启动时的 CurrentSpeed 的变化过程，并将其与期望启动曲线进行比较。匹配期望启动牵引级需要监视的业务事实为实际行驶速度，实际行驶距离以及期望行驶速度。将 train 的 CurrentSpeed，hop 的 distanceCovered 以及 A1、A2、A3 的 desiredSpeed 属性成员通过工具原型设置为标记，并设置跟踪时间间隔为 0 到 14 秒，轮询间隔为 1 秒。

BCM 确定了 PIM 的《terminator》对象为 train，通过《terminator》对象向子系统发送层间信号：首先向 train 发送层间信号 OwnConsole 进入 console 状态空间完成 Gears 的设置，然后向 train 发送层间信号 OwnMotor 进入 motor 状态空间启动轮询信号，其次向 train 发送层间信号 Negotiate 进入 hop 状态空间，最后向 hop 发送层间信号 performCurve 持续更新 CurrentSpeed、distanceCovered 和 desiredSpeed 的值。在工具原型中输入测试用例如下。

train = find Train where TrainId = 10

generate OwnConsole(Gears) to train#工具原型模拟 Gears = 1~ 4 级的启动曲线#

generate OwnMotor to train

generate Negotiate to train

generate performCurve to hop

消息驱动框架主进程判断事件队列中存在 performCurve 层间信号后创建 AccelerationCurve 状态机进程。由于 AccelerationCurve 状态迁移图已经满足安全性条件，工具原型在此基础上执行状态机进程，反馈的 desiredSpeed 的业务用例如图 4.13 所示。

图 4.13 AccelerationCurve 业务用例

Gears = 1 时，工具原型反馈的 AccelerationCurve 业务事实如图 4.14 所示。

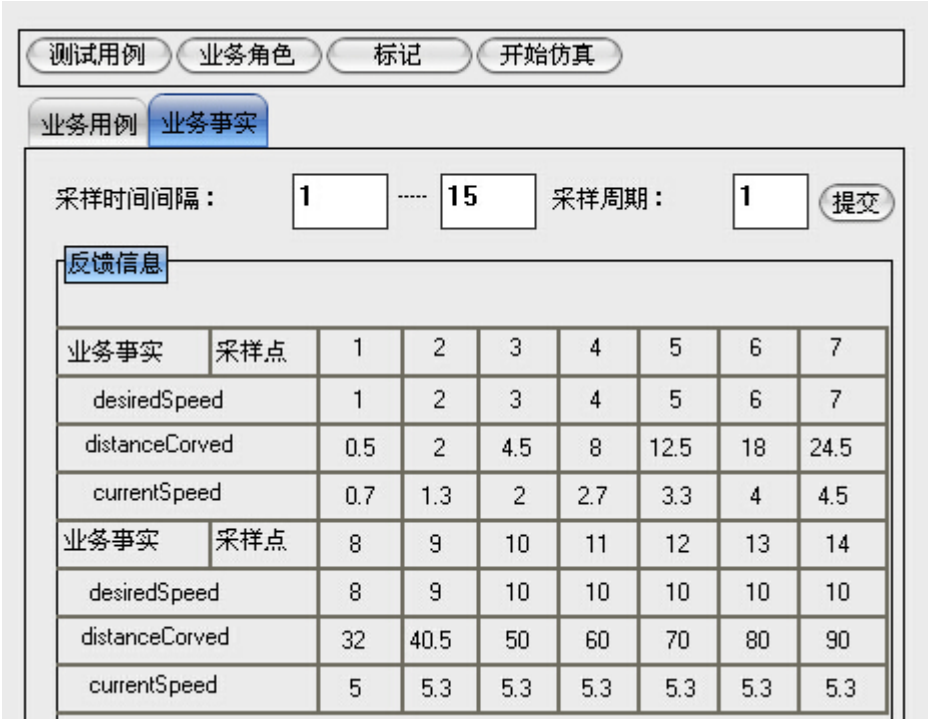

图 4.14 N1 级 AccelerationCurve 业务事实

Gears = 3 时，工具原型反馈的 AccelerationCurve 业务事实如图 4.15 所示。

业务事实	采样点	1	2	3	4	5	6	7
desiredSpeed		1	2	3	4	5	6	7
distanceCorved		0.5	2	4.5	8	12.5	18	24.5
currentSpeed		1.2	2.5	3.7	5	6.2	7.4	8.7
业务事实	采样点	8	9	10	11	12	13	14
desiredSpeed		8	9	10	10	10	10	10
distanceCorved		32	40.5	50	60	70	80	90
currentSpeed		9.7	11	12	13.4	14.6	15.5	16

图 4.15 N3 级 AccelerationCurve 业务事实

工具原型模拟 Gears 从 1 级到 4 级的启动曲线如图 4.16 所示。

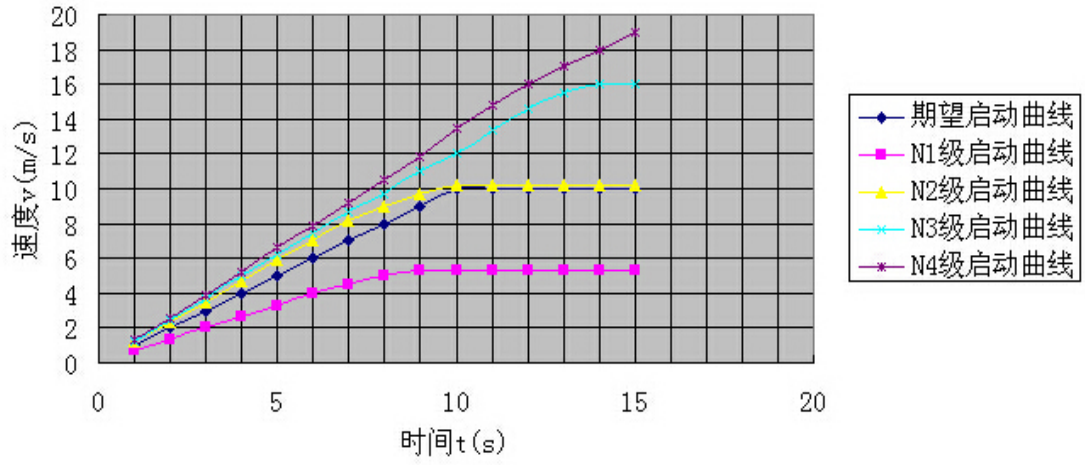

图 4.16 启动曲线比较

验证结果反映了工具原型操纵平台无关模型元素的可行性，从模拟数据中可以看到列车在四个启动级下的启动过程，其中 N2 级启动曲线的业务事实符合期望启动曲线的限制条件，它到达准恒速的时刻在 6 秒到 14 秒之间，并且恒速值在 6 米每秒到 14 米每秒之间。通过分析业务用例和标记数据可以细粒度地判断平台无关模型业务事实的变化情况，领域专家在配置该期望启动曲线的启动牵引级时选择 N2 级启动。

通过本文设计实现的基于 SysML 的可执行模型验证工具，建模人员和领域专家能够使用 xSysML 建立可执行平台无关模型并通过执行来实现平台无关模型的验证，完成 RUP 的初始阶段和营造阶段。

4.4 本章小结

本章阐述了工具原型针对 SysML PIM 进行验证的功能。首先通过模型验证分析得出了验证的合理性标准。然后说明了模型验证机制，接着结合可执行框架介绍了平台无关模型验证的过程，并通过模型实例测试了工具原型在创建和验证平台无关模型方面的可行性。

第五章 总结与展望

5.1 总结

MDA 的提出解决了传统精细手工开发过程中存在的弊端，SysML 的引入使得 MDA 开发方法能够应用于系统工程领域。在系统工程设计阶段提出一个精确无歧义的可执行 SysML 模型可以很好地将面向实现的解决方案与业务需求分离开来，有利于技术人员与领域专家之间的分工合作。可执行性使得模型的验证提前到模型驱动开发过程的设计阶段进行。合理的平台无关模型可以组织成大规模的高内聚低耦合的可复用组件，减少测试目标系统的时间和开销，保障了系统的质量和安全性。本文对 SysML 进行可执行扩展，在工具原型的基础上研究了平台无关模型的验证，主要工作总结如下：

(1) 研究 SysML 符号系统，挑选一个 SysML 符号系统的完备子集，并在这个子集中添加动作规约语言进行增强，此时的 SysML 变为 xSysML，在 xSysML 的基础上研究模型的可执行性。最后说明了基于可执行 SysML 的模型驱动开发过程。

(2) 在 xSysML 基础上研究了工具原型的设计与实现。采用 Eclipse 作为开发平台，通过分析现有建模工具的实现技术设计实现了建模框架、可执行框架以及自定义功能插件。

(3) 建模框架主要由 GMF 和模型管理模块组成，使用 GMF 作为基础插件来开发图形化的建模编辑器，根据模型需求对编辑器进行了二次开发。由于编辑器保存的模型文件将动作规约层中的状态图信息按照状态结点信息和迁移信息分开来存放，这种存放方式影响了可执行框架执行状态图的效率。模型管理模块对动态规约层中的模型文件进行序列化操作，改变了状态图信息的组织方式，提高执行效率。该框架满足了工具原型的建模需求。

(4) 可执行框架主要由 ASL 解析器模块、消息驱动框架、状态解析引擎以及实例生成器组成，它是工具原型进行平台无关模型验证的基础。PIM 的验证建立在对象状态机的执行基础上。状态图位于模型的动态行为层，它可以描述系统生命期内的行为过程。在执行对象状态机的过程中产生的信号事件队列以及标记值的变化反映平台无关模型的业务特征。

(5) 利用工具原型创建列车运行系统平台无关模型，主要从它的需求分析、可执行 PIM 的创建和验证三个方面对模型进行研究。通过执行 AccelerationCurve 行为模型对工具原型的可行性进行测试。

5.2 展望

由于目前国内对 SysML 的研究还很不充分，也没有相关的建模工具，加之 SysML 在 MDA 领域研究内容很丰富，本文并没有挖掘出 SysML 的所有功能。鉴于水平以及时间所限，本文有待于对 SysML 和模型验证工具作进一步的深入学习和研究，主要有以下几个方面：

(1) 本文定义的 xSysML 中只引用了包图、块结构图、状态图，时序图和需求图，并没有考

虑到其它的视图。下一步工作是在静态结构视图中加入参数图和块内部定义图进行补充，在行为视图中加入活动图，并研究它们之间的相互关系。

(2) 在模型验证的功能需求合理性标准中只考虑到了体系结构的安全性内容，这是不完全的[49][50][51]，没有时序逻辑方面[52][53][54]的约束，验证平台无关模型只是很粗略地通过事件序列和标记值进行判断。下一步工作是细化模型验证的标准以及实现过程。

(3) 在模型验证的展示方面做得还不够人性化，下一步工作主要应用反序列化技术将模型文件的执行过程以图形化的方式展示出来[55]。

(4) 进一步加强工具原型的功能，当前版本还不能支持某些 SysML 模型元素和 ASL 操作需求，比如流元素以及诸如 union-of 或者 intersection-of 等集合论操作。

参考文献

[1] OMG. Model Driven Architecture. http://www.omg.org/, 2000.

[2] OMG. Unified Modeling Language. http://www.uml.org/, 2001.

[3] OMG. Systems Modeling Language specification. http://www.omg.org/spec/SysML/1.3/, 2012.

[4] 王学斌，吴泉源，史殿学. 模型驱动架构中的模型转换方法. 计算机工程与科学，2006，11：133~135.

[5] 董珊珊. 基于 MDA 的 Web-MIS 平台研究与实现[硕士学位论文]. 南京:南京航空航天大学, 2010.

[6] Brown, A.W.. MDA Redux: Practical Realization of Model Driven Architecture. Seventh International Conference on Composition-Based Software Systems, 2008, 174~183.

[7] 王赟华, 陈蔚薇. 模型驱动开发方法的应用研究. 计算机工程, 2006, 13:63~65.

[8] 凌华德. 基于 MDA 的代码自动生成技术的研究与实现[硕士学位论文]. 上海:华东师范大学，2006.

[9] Leon Starr，Yourdan Press. How to Build Shlaer-Mellor Object Model. London：Prentice Hall PTR，1996.

[10] Chris Raistrick ., et al. Model Driven Architecture with Executable UML. 北京：机械工业出.版社, 2006.

[11] 姜军，罗雪山，罗爱民等. 可执行体系结构研究. 国防科技大学学报，2008，03：77~80.

[12] I-Logix Inc. Code Generation Guide. Rhapsody in C. 美国, 2003.

[13] Douglass, Bruce Powel. UML Statecharts. i-Logix, European, 2004, 1~23.

[14] A. David, M. O. Moller, W. Yi. Formal verification of UML statecharts with real-time extensions. 2002, 218~232.

[15] Schattkowsky, T., Muller, W. . Transformation of UML state machines for direct execution. IEEE, 2005, 117~124.

[16] 蒋彩云，王维平，李群等. SysML:一种新的系统建模语言. 系统仿真学报，2006，18(6)：1483~1487.

[17] DoD Architecture Framework Working Group. DoD Architecture Framework Version 1.0 Volume I: Definitions and Guidelines. U.S.: Department of Defense, 2003.

[18] 吴娟. 基于 SysML 的 DoDAF 产品设计研究[硕士学位论文]. 武汉:华中科技大学,2006.

[19] IBM Corporation, C++ Framework Execution Reference Manual. USA, 2008.

[20] OMG. Meta-Object Facility(MOF). http://www.omg.org/mof.

[21] 李晓春，刘淑芬，于卓尔等．一种基于 MOF 的两级建模工具的设计与实现．计算机应用与软件，2008，03:6~8.

[22] OMG. OMG Systems Modeling Language. http://www.omgsysml.org，2013.

[23] 赵立军. 基于 SysML 的需求分析研究. 计算机技术与发展, 2011, 12：139~141.

[24] 阎 石．数字电子技术基础．北京：高等教育出版社，1998.

[25] 吕小强.基于 SysML 模型驱动的软件开发应用与研究[硕士学位论文]．大连：大连海事大学，2008.

[26] 史耀馨,崔萌,李宣东等．基于 MDA 的 UML 模型转换技术——从顺序图到状态图．计算机工程与应用，2004，13：40~45.

[27] 王洪媛. UML 行为模型之间模型转换的研究[博士学位论文]．长春:吉林大学，2007.

[28] Kruchten,P.K,P. Rational 统一过程实践者指南．北京：中国电力出版社，2004.

[29] 庞世春．GMT:一种面向方面的领域建模工具的研究与实现[硕士学位论文]．长春:吉林大学，2005.

[30] 麻志毅,刘辉,何啸等．一个支持模型驱动开发的元建模平台的研制．电子学报,2008，04:731~736.

[31] 陈洪辉,苏伟,柳海峰. SysML 及其在 C~4ISR 系统建模中的应用研究．计算机仿真，2007，11:60~64.

[32] 蒋浴芹．M-Design-多域产品系统建模平台的研究与实现[硕士学位论文]．杭州:浙江大学，2011.

[33] 吴娟,王明哲,方华京．基于 SysML 的系统体系结构产品设计．系统工程与电子技术，2006，04:594~598.

[34] The Eclipse Foundation:http://www.eclipse.org.

[35] Eclipse Graphical Modeling Framework，Eclipse Foundation. http://www.eclipse.org/gmf/.

[36] Dave Steinberg，Fank Budinsky，Marcelo Paternostro，et al：Eclipse Modeling Framework 2.0 中文版(战晓苏译)．北京:清华大学出版社，2010:30~100.

[37] Eclipse Graphical Editing Framework，Eclipse Foundation. http://www.eclipse.org/gef/.

[38] 任中方，张华，闫明松等. MVC 模式研究的综述．计算机应用研究，2004，10：1~8.

[39] OMG. XML Metadata Interchange (XMI) Specification. Version 2.0. 2003.

[40] Jim Barnett，Genesys. State Chart XML (SCXML)：State Machine Notation for Control Abstraction. http://www.w3.org/TR/scxml/. 2007.

[41] W3C. World Wide Web Consortium. http://www.w3c.org/，1994.

[42] 唐云吉．面向 Web 应用的测试用例生成技术研究[硕士学位论文]．上海：上海大学，2009.

[43] 郭荷清，王增勋. XML 数据绑定及对象序列化的应用研究．计算机应用与软件，2006，

05：66~68.

[44] 朱蕾蕾. 基于 UML 状态图的软件性能测试研究[硕士学位论文]. 长春：长春理工大学. 2012.

[45] 林杰，徐建坤. 基于 Kripke 结构的程序正确性证明. 计算机应用，2011，05,：1425~1428.

[46] 陆公正，张广泉. 基于 UML 状态图的工作流建模与验证[硕士学位论文]. 苏州：苏州大学，2006.

[47] R.J.Bril. A Model-oriented Method for Algebraic Specifications Using COLD-l as Notation, Berlin：Springer-Verlag, 1991.

[48] 陈利，张庆明，王晓英. 仿真模型验证方法与程序设计. 北京科技大学学报，2002.

[49] G Kwon. Rewrite Rules and Operational Semantics for Model Checking UML Statecharts. Berlin：Springer-Verlag，2000.

[50] S Gnesi, F Mazzanti. A Model Checking Verification Environment for UML Statecharts. Udine: XLIII Annual Italian Conference AICA，2005.

[51] R Eshuis. Symbolic model checking of UML activity diagrams. ACM Transactions on Software Engineering and Methodology，2006，1(15)：1~38.

[52] 李广元，唐稚松. 基于线性时序逻辑的实时系统模型检测. 软件学报，2002，13(2)：193-202.

[53] 王小兵，段振华. 面向对象的时序逻辑语言. 计算机工程与应用，2009, 38（1）：98-101.

[54] 唐达，徐超. 工作流建模中时态逻辑的研究与应用，2010, 10（4）：389-392.

[55] Sun Yong，Gong Guanghong，Han Liang，et al. Research on Framework and Tool of MDA Oriented Simulation System. 2010.

致 谢

时间过的很快，美好的研究生学习生活阶段即将进入尾声，在论文即将完成之即，我要向指导和帮助过我的导师、同学还有亲友表示由衷的感谢。

首先，我要特别感谢我的导师王立松副教授。他严谨的治学态度、亦师亦友的谆谆教诲给了我很大的帮助，无论在工作还是为人处事上都是我今后学习的楷模。在论文方面，王老师在选题内容上给了我关键的指导。起初，我并没有理解课题的研究目的，走了不少弯路，正是王老师耐心的教导使得我回到了研究的正路上来，同时，王老师还提供给我一些具有指导性意义的资料，这些资料让我对课题的研究更加的顺利。在遇到技术瓶颈时，王老师总能够提出一个解决方案，指引我攻克一个又一个的难题。特此向王老师表示最衷心的感谢。

感谢一起努力的同窗好友：陈凯、章倩、王尚北、钱正麟，他们在学习和生活上给予我很大的帮助，同时感谢我的室友杨金龙，他在我论文撰写过程中提出很多宝贵的意见。感谢我的朋友俞顺正，他在我做课题的过程中给出了很多的参考意见。

尤其感谢我的父母，没有他们的支持，我走不到现在这一步。他们含辛茹苦地供我上学，为我付出太多太多，滴水之恩当涌泉相报，我会用一辈子的时间去用心地报答他们。

最后衷心感谢各位专家、教授在百忙之中抽时间对论文进行评审并提出宝贵意见。

在学期间的研究成果及发表的学术论文

攻读硕士学位期间发表（录用）论文情况

1. 俞晓锋，王立松. SysML 状态图合理性验证研究与实现. 电子科技，2014：第五期.