

# 面向复杂装备系统设计的 MBSE 模型传递技术研究

陈锐

2024 年 6 月

面向复杂装备系统设计的MBSE模型传递技术研究

陈锐

北京理工大学

中图分类号：TP18

UDC 分类号：621.3

## 面向复杂装备系统设计的 MBSE 模型传递技术研究

作者姓名	陈锐
学院名称	机械与车辆学院
指导教师	王国新教授
答辩委员会主席	张发平教授
申请学位	工学硕士
学科/类别	机械工程
学位授予单位	北京理工大学
论文答辩日期	2024 年 6 月

**Technical Research on Model-Based Systems  
Engineering (MBSE) Model Transfer for Complex  
System Equipment Design**

Candidate Name:	<u>Rui Chen</u>
School or Department:	<u>School of Mechanical Engineering</u>
Faculty Mentor:	<u>Prof. Wang Guoxin</u>
Chair, Thesis Committee:	<u>Prof. Zhang Faping</u>
Degree Applied:	<u>Master of Science in Engineering</u>
Major:	<u>Mechanical Engineering</u>
Degree by:	<u>Beijing Institute of Technology</u>
The Date of Defence:	<u>June, 2024</u>

## 研究成果声明

本人郑重声明：所提交的学位论文是我本人在指导教师的指导下独立完成的研究成果。文中所撰写内容符合以下学术规范（请勾选）：

☒ 论文综述遵循“适当引用”的规范，全部引用的内容不超过 50%。

☒ 论文中的研究数据及结果不存在篡改、剽窃、抄袭、伪造等学术不端行为，并愿意承担因学术不端行为所带来的一切后果和法律责任。

☒ 文中依法引用他人的成果，均已做出明确标注或得到许可。

☒ 论文内容未包含法律意义上已属于他人的任何形式的研究成果，也不包含本人已用于其他学位申请的论文或成果。

☒ 与本人一同工作的合作者对此研究工作所做的任何贡献均已在学位论文中作了明确的说明并表示了谢意。

特此声明。

签 名： 陈锐

日 期：2024 年 6 月

## 关于学位论文使用权的说明

本人完全了解北京理工大学有关保管、使用学位论文的规定，其中包括：

①学校有权保管、并向有关部门送交学位论文的原件与复印件；

②学校可以采用影印、缩印或其它复制手段复制并保存学位论文；

③学校可允许学位论文被查阅或借阅；

④学校可以学术交流为目的，复制赠送和交换学位论文；

⑤学校可以公布学位论文的全部或部分内容（保密学位论文在解密后遵守此规定）。

签 名：陈锐

日期：2024 年 6 月

导师签名：王立新

日期：2024 年 6 月

## 摘 要

在采用基于模型的系统工程（Model-based System Engineering, MBSE）方法进行复杂装备系统开发的背景下，设计信息以模型为载体在架构设计与仿真验证等环节中闭环传递。模型传递保证了模型在设计上下游阶段之间信息的一致性，模型传递的效率直接决定架构设计与仿真验证两个环节的协同效率。然而，面向 MBSE 架构设计-仿真验证一体化过程中的模型传递，仍存在以下问题：（1）缺乏对于模型传递及其特征进行完整描述的指导性框架，导致设计人员无法完整识别复杂模型关联关系进而降低设计效率。（2）由于架构与仿真模型语法语义异构，导致工具级的模型集成与传递困难，设计信息一致性难以得到保证。（3）现有模型传递过程中需要对架构与仿真工具中模型进行转换、参数配置等大量手动操作，无形中提高了设计过程的错误率。针对上述问题，面向架构设计-仿真验证一体化过程，开展基于服务的 MBSE 模型传递技术研究，具体包括以下研究点：

**（1）MBSE 模型传递框架构建。**首先，面向架构设计-仿真验证一体化过程，给出模型传递的概念定义；其次，对模型传递的内涵与关键特征进行分析，包括互操作性与自动化能力两个维度的关键特征；最后，提出模型传递的技术框架，根据互操作性与自动化能力两个维度的关键特征进行技术选型，给出实施模型传递的指导性技术流程。

**（2）基于服务的架构-仿真模型一体化表达方法。**根据模型传递的互操作性关键特征，研究基于服务的架构-仿真模型一体化表达方法。首先，介绍开放生命周期协作服务规范（Open Services for Lifecycle Collaboration, OSLC）中的元数据模型，为后续统一表达架构模型与仿真模型提供集成基础；其次，分析架构模型与仿真模型的模型组成结构、参数和操作接口等关键特征，提出架构模型、仿真模型与 OSLC 元数据模型之间的映射方法，实现模型信息及操作在概念层的服务化表达；最后，根据模型的映射方法，开发工具适配器，实现设计信息在架构工具与仿真工具之间端到端的传递与集成。

**（3）面向模型持续集成的自动化 workflow 设计方法。**根据模型传递的自动化能力关键特征，研究面向模型持续集成的自动化 workflow 设计方法。面向架构设计-仿真验证一体化过程，分析 workflow 包含的模型传递活动，明确活动模型传递的输入与输出，并

建立起 workflow 活动与基于 OSLC 的模型操作服务之间的关联关系；然后，介绍基于流水线脚本的工作流表达形式，建立起流水线脚本元素与 workflow 活动之间的映射关系；其次，构建 workflow 活动定义与执行的流水线语义模板，为后续基于特定场景进行 workflow 设计提供可复用的框架。基于 OSLC 服务自动执行 workflow，用于验证 MBSE 架构模型与仿真模型传递的自动化特征。

**（4）面向航空发动机系统的 MBSE 模型传递案例验证。**该案例以航空发动机系统为对象，构建航空发动机系统的架构模型与仿真模型，根据模型传递框架，识别模型传递过程中架构模型与仿真模型传递的关键要素，构建模型持续集成 workflow，并关联到航空发动机系统的架构模型与仿真模型服务，打通架构工具与仿真工具之间的闭环数据传递链路，进而验证所提出方法的有效性。

**关键词：**模型传递；基于模型的系统工程；开放生命周期服务；持续集成



## Abstract

In the context of complex equipment System development using Model-based System Engineering (MBSE) method, the design information is transferred in a closed loop through architecture design and simulation verification. Model transfer ensures the consistency of information between the upstream and downstream stages of model design, and the efficiency of model transfer directly determines the collaborative efficiency of architecture design and simulation verification. However, in the integrated process of architecture design-simulation verification for MBSE model, there are still the following problems: (1) The lack of a guiding framework for the complete description of model transfer and its features leads to the inability of designers to fully identify complex model correlation and thus reduce design efficiency. (2) Due to the heterogeneity of syntax and semantics of architecture and simulation model, it is difficult to integrate and transfer the model at the tool level, and the consistency of design information is difficult to be guaranteed. (3) In the process of transfer of existing models, a large number of manual operations such as model conversion and parameter configuration in architecture and simulation tools are required, which invisibly increases the error rate in the design process. In view of the above problems, service-oriented MBSE model delivery technology research is carried out for the integrated process of architecture design-simulation verification, including the following research points:

(1) Construction of MBSE model transfer framework. Firstly, the concept definition of model transfer is given for the integration process of architecture design-simulation verification. Secondly, the connotation and key features of model transmission are analyzed, including the key features of tool integration and automation capability. Finally, the technical framework of model transfer is proposed, the technology selection is carried out according to the key characteristics of tool integration and automation capability, and the guiding technical process of implementing model transfer is given.

(2) Service-based architecture-simulation model integration expression method. A

According to the key features of tool integration delivered by the model, the service-based architecture-simulation model integration expression method is studied. Firstly, the Open Services for Lifecycle Collaboration (OSLC) metadata model is introduced, which provides integration basis for subsequent unified expression of architecture model and simulation model. Secondly, the key characteristics of architecture model and simulation model, such as model composition structure, parameters and operation interface, are analyzed, and a mapping method between architecture model, simulation model and OSLC metadata model is proposed to realize the service expression of model information and operation in the concept layer. Finally, according to the mapping method of the model, a tool adapter is developed to realize the end-to-end transfer and integration of design information between the architecture tool and the simulation tool.

(3) Model-oriented continuous integration automation workflow design method. According to the key features of automation capability delivered by model, the design method of automated workflow oriented model continuous integration is studied. Firstly, the definition specification of workflow and its component elements are introduced to provide a theoretical basis for the formalization of workflow. Secondly, for the integration process of architecture design-simulation verification, the relationship between workflow activities and OSLC-based model operation services is established, and workflow activities are executed by service technology, which provides a unified operation basis for automation. Then, the mapping relationship between specification elements and workflow activities is established, and the semantic template of workflow activity definition and execution is constructed, which provides a reusable framework for workflow design based on specific scenarios.

(4) MBSE model transfer case verification for aero-engine system. This case takes the aero-engine system as the object, builds the architecture model and simulation model of the aero-engine system, identifies the key elements of the architecture model and simulation model transfer in the process of model transfer according to the model transfer framework, builds the continuous integration workflow of the mod

el, and relates it to the architecture model and simulation model service of the aero-engine system. The closed-loop data transfer link between the architecture tool and the simulation tool is opened to verify the effectiveness of the proposed method.

**Keywords:** model transfer; model-based systems engineering; Open Services for Life cycle Collaboration; continuous integration

## 目 录

摘 要 .....	I
第 1 章 绪论 .....	1
1.1 研究背景 .....	1
1.2 国内外研究现状 .....	3
1.1.1 数据互操作研究现状 .....	3
1.1.2 MBSE 工具链应用现状 .....	5
1.1.3 模型持续集成的研究现状 .....	6
1.3 研究意义 .....	9
1.4 研究内容与组织架构 .....	10
1.5 本章小结 .....	11
第 2 章 MBSE 模型传递理论与研究框架 .....	13
2.1 引言 .....	13
2.2 MBSE 模型传递概念及内涵 .....	13
2.2.1 MBSE 模型传递的定义与内涵 .....	13
2.2.2 MBSE 模型传递的场景与范围 .....	15
2.3 MBSE 模型传递的关键特征 .....	17
2.3.1 MBSE 模型传递的互操作性特征 .....	18
2.3.2 MBSE 模型传递的自动化能力特征 .....	18
2.4 MBSE 模型传递的技术框架 .....	19
2.5 本章小结 .....	21
第 3 章 基于服务的架构-仿真模型一体化表达方法 .....	22
3.1 引言 .....	22
3.2 开放生命周期协作服务元数据模型 .....	22
3.2.1 开放生命周期协作服务规范 .....	22
3.2.2 开放生命周期协作服务元数据模型 .....	24
3.3 架构模型与 OSLC 元数据模型的映射方法 .....	25
3.3.1 基于多架构建模语言的架构模型构建方法 .....	25
3.3.2 GOPPRR-E 元模型与 OSLC 元数据模型映射方法 .....	28

3.4 仿真模型与 OSLC 元数据模型的映射方法 .....	33
3.4.1 基于多架构建模语言的架构模型构建方法 .....	33
3.4.2 Modelica 元模型与 OSLC 元数据模型映射方法 .....	37
3.5 基于服务的工具适配器构建方法 .....	40
3.6 本章小结 .....	42
第 4 章 面向模型持续集成的自动化 workflow 设计方法 .....	44
4.1 引言 .....	44
4.2 架构与仿真过程中的 workflow 活动分析 .....	45
4.2.1 模型及模型元素演化活动分析 .....	45
4.2.2 模型检查、转换与仿真活动分析 .....	46
4.3 workflow 活动与流水线脚本元素映射方法 .....	48
4.3.1 流水线脚本定义及组成元素分析 .....	49
4.3.2 workflow 活动与流水线元素映射方法 .....	50
4.4 workflow 的语义模板设计 .....	52
4.4.1 模型及模型元素演化活动的流水线语义模板定义 .....	52
4.4.2 模型分析、转换或仿真活动的脚本定义 .....	55
4.5 本章小结 .....	58
第 5 章 面向航空发动机系统的 MBSE 模型传递案例验证 .....	59
5.1 引言 .....	59
5.2 航空发动机系统设计背景 .....	59
5.3 面向航空发动机系统架构设计与仿真验证过程的模型传递 .....	60
5.3.1 航空发动机系统架构建模与仿真建模 .....	60
5.3.2 基于服务的航空发动机系统架构-仿真模型一体化表达 .....	65
5.3.3 航空发动机系统架构与仿真持续集成 workflow 设计 .....	76
5.4 案例总结与分析 .....	85
5.5 本章小结 .....	85
第 6 章 总结与展望 .....	87
1 全文总结 .....	87
2 工作展望 .....	88

参考文献.....	90
攻读学位期间发表论文与研究成果清单.....	97
致谢.....	98

## 第 1 章 绪论

### 1.1 研究背景

长期以来，复杂装备系统设计研制遵循传统的系统工程研发模式，图 1.1 左侧为典型的系统工程 V 型模型，左侧为系统设计阶段，包括系统需求分析、功能分析、架构设计、子系统设计等环节<sup>[1]</sup>。传统的研发模式中，不同部门之间通过文档的方式传递信息，具有传递效率低<sup>[2]</sup>、描述一致性差<sup>[3]</sup>、不支持早期的仿真验证<sup>[4]</sup>等问题，使得系统设计效率大大降低。

2007 年，INCOSE 在《系统工程 2020 年愿景》<sup>[5]</sup>中，提出了基于模型的系统工程(Model-based System Engineering, MBSE), MBSE 采用模型将系统的各个阶段以及开发活动进行形式化表达，以支持系统从概念设计阶段开始一直持续到开发阶段和后续生命期阶段的需求、设计、分析、验证和确认活动，如图 1.1 右侧所示。当采用 MBSE 方法进行系统开发，例如在左侧采用 SysML<sup>[6]</sup>建模语言构建系统架构模型进行系统设计，在右侧构建例如 Modelica<sup>[7]</sup>、Simulink 等模型对组件与子系统进行仿真与测试，从而在进行实物集成之前及时发现设计方案存在的问题，大幅提高设计研发的效率。

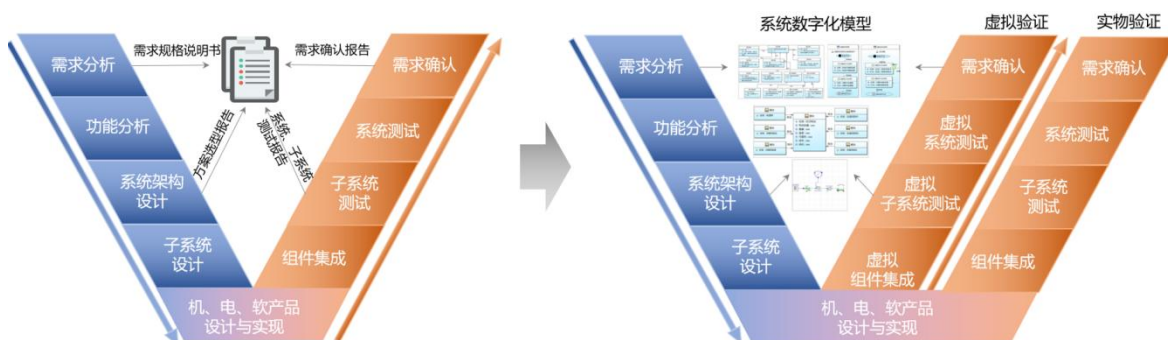


图 1.1 基于文档的系统工程到基于模型的系统工程

随着模型化技术的进步，复杂装备系统设计全生命周期中涉及到多个设计阶段<sup>[8]</sup>，不同设计阶段需要使用不同种建模工具，以模型核心跨越不同设计阶段与不同种建模工具将设计数据、信息及知识传递给下游是一个至关重要的过程，该过程可以被称为模型传递。模型传递涉及到模型在不同阶段、不同种工具之间的集成，确保了开发过

程中的信息一致性，直接影响到系统设计与验证&确认环节的协同效率<sup>[9]</sup>。例如在进行系统开发时，系统设计阶段需要使用 SysML 建模语言、MagicDraw 架构建模工具建立架构模型，在仿真验证阶段需要使用 Modelica 建模语言与 OpenModelica 建模工具建立物理模型，这些工具与其它建模、分析工具组合在一起形成系统开发的工具链<sup>[10]</sup>。设计数据、信息或者知识则以模型为载体在不同种建模与分析工具之间，即在工具链之中进行动态的传递。由于架构模型与仿真模型的语法语义异构、以及工具链之中不同种建模与分析工具底层结构异构，导致异构模型中所包含的数据、信息或者知识传递与流转困难。如果不打通架构模型与其它物理模型之间的关联，则无法将系统架构模型中包含的逻辑信息完整传递到物理设计等层面，也就无法高效进行包括多学科设计优化、详细设计或仿真验证等方面的活动，如图 1.2 所示。



图 1.2 MBSE 研制过程“数据孤岛”

在使用标准化的建模语言以及建模工具实现基于模型的开发过程，能够显著提高系统开发的一致性、准确性和效率<sup>[11]</sup>。然而，随着需求的频繁变化和系统交付日期的逐渐缩短，对系统的设计与反馈效率提出了更高的要求。目前基于模型的设计过程中，需要对架构与仿真工具中模型进行转换、分析、检查、参数配置等大量手动操作，手动操作一方面容易出现操作失误导致设计信息从系统架构设计传递到仿真验证阶段的一致性与准确性降低；另一方面难以及时响应系统架构设计-仿真验证“边设计边仿



真”的动态闭环反馈过程，模型在架构设计与仿真验证工具间流转的自动化水平低，严重降低了装备开发与迭代的效率。

因此，在当前多学科交叉的复杂系统设计背景下，目前面向复杂装备系统设计的 MBSE 模型传递尚具有一定的局限性。其一，缺乏对于模型传递及其特征进行完整描述的指导性框架，导致设计人员无法完整识别复杂模型关联关系进而降低设计效率。其二，由于架构与仿真模型语法语义异构，导致工具级的模型集成与传递困难，设计信息一致性难以得到保证。其三，现有模型传递过程中需要对架构与仿真工具中模型进行转换、参数配置等大量手动操作，导致架构与仿真动态、闭环一体化的验证过程效率降低。

## 1.2 国内外研究现状

### 1.1.1 数据互操作研究现状

在复杂装备系统全生命周期过程中，设计数据、信息与知识以模型为载体在不同阶段、不同种建模、仿真或分析工具之间进行动态交换。在使用 MBSE 方法进行复杂系统设计时，不同种建模与分析工具将产生大量的多源异构模型，不同格式的模型需要在工具链中进行数据交换与传递，数据互操作<sup>[12]</sup>是实现上述模型数据传递的一个重要手段。数据互操作是指不同工具之间的模型交换信息与使用已交换信息的能力，常见的数据互操作方式包括：点到点传递、数据中间件以及标准的集成规范。

#### （1）点到点传递

点到点传递是指对已有工具对外提供的接口进行扩展开发，从而实现工具间的数据及信息交换。Chen<sup>[13]</sup>等通过开发架构工具插件关联由 OSLC 适配器转化的仿真模型服务，实现架构模型与仿真模型的交换与传递。Badache<sup>[14]</sup>基于 Cappella 建模工具开发了一种由 Cappella 到 SysML 标准建模语言的工具转换器，从而将逻辑架构建模阶段的 Cappella 组件、功能以及相关的模型概念转化为 SysML 模型。该方法的优点是可以根据具体的需求定制模型传递与交换的方式和内容，实现高效的数据传输。缺点是需要对每个建模或分析工具进行单独的开发和维护，增加了开发成本和复杂度。因此，点到点的交换方式适用于模型种类较少、格式较为统一、交换需求较少的场景。

## （2）数据中间件

数据中间件是指一种在计算机间进行数据传输与交换的数据格式，常被用于 MBSE 设计过程中跨工具的模型信息交换。Huang 基于可扩展的 XML 元数据交换 (XMI) 实现 SysML 模型与其它领域工具之间的模型信息交换<sup>[15]</sup>。本体作为一种新的数据中间件，支持将特定领域的概念及其在不同语言之间的关系进行形式化表达。Chen 等人<sup>[16]</sup>定义了系统设计模型与本体模型之间的转换规则，并在系统设计阶段利用推理验证形式化的行为要求。Lu 等人<sup>[17]</sup>提出了一种基于图、对象、点、属性、角色和关系的本体扩展 (GOPRR-E)，支持形式化 MBSE 建模过程元模型。此外，Lu 开发了知识图谱模型，以支持全生命周期中模型的统一表达，并基于 GOPRR-E 本体支持模型及模型要素之间的数据集成。该方法的优点是可实现模型的标准化和规范化，降低了模型交换的复杂度和成本，提高了模型的一致性和可重用性。缺点是：需要对不同的建模及分析工具进行适配和转换，可能导致模型信息的损失或冗余。因此，数据中间件方式适用于模型种类较多、格式较为异构、交换需求较为静态的场景。

## （3）标准的集成规范

在基于模型的设计过程种，存在多种用于模型及数据信息交换的集成规范。如工程与建筑领域的 STEP 标准、质量管理与度量领域的 QIF 标准、系统仿真领域的 FMI 标准、航空航天等领域的 AP233 应用协议等。标准集成规范的目的是提供一种中立的机制和实现方式，独立于任何特定系统，并能完整地描述产品数据信息。Kwon 等人<sup>[18]</sup>提出了一种方法，将 STEP 表示的设计数据和 QIF 表示的检验数据融合到基于本体和知识图谱构建的数字线程中。Zadeh 等人<sup>[19]</sup>探讨了 STEP 和 OSLC 之间的协同作用，以整合生产工程中的产品制造数据和异构 IT 工具。Wu 等人<sup>[20]</sup>通过分析仿真机制和 PST 仿真系统逻辑结构，基于 FMI 标准将机械系统、控制系统、动力学等模型集成至联合仿真模型中。该方法的优点是可实现跨领域和跨平台的模型交互，提高了模型信息的通用性和互操作性。缺点是不同领域的标准之间存在冲突或不兼容。因此，标准的集成规范适用于特定域模型、格式较为异构、模型数据交换需求较为动态的场景。

### 1.1.2 MBSE 工具链应用现状

在 1.2.1 节提出, 在采用 MBSE 方法进行复杂复杂系统设计时, 设计数据、信息与知识以模型为载体在不同阶段、不同种建模、仿真或分析工具之间进行动态交换的过程, 而由多个建模或分析工具组成的 MBSE 工具链是发生上述模型传递过程的载体。因此对现有的 MBSE 工具链的应用现状进行调研, 从而归纳出 MBSE 工具链在形成模型设计信息传递链路上的不足。

首先根据 MBSE 工具链应用的业务场景, 定位 MBSE 工具链的应用功能。MBSE 工具链中工具的功能包括系统建模、验证和确认、仿真、几何建模、HIL 仿真、数据建模和管理、本体、知识管理和信息管理等, 工具链的功能是上述两个或多个工具功能的集成。考虑到系统全生命周期的复杂性, 开发人员难以开发出满足系统整体阶段需求的工具链, 来自不同领域的利益相关方通常根据自己关注的功能点开发工具链, 例如系统工程师关注建立系统的需求、架构模型从而进行系统设计, 仿真工程师关注建立多种系统仿真模型进行联合仿真。因此, 通过工具链应用的业务场景进行分类, 分析工具链的开发与使用情况。

在实现系统设计的业务场景中: Riccobene 等人<sup>[21]</sup>在模型驱动的嵌入式系统设计流程中, 集成了 SysML 和 SystemC-UML 配置文件来实现硬件软件协同设计。Carpanzano 等人<sup>[22]</sup>开发了一个基于模型和数据驱动的系统工程工具链, 实现两种不同的工具 (Capella 和 Valispace) 的协同, 保证系统设计与需求之间的一致性, 以及其更新和可追溯性的自动化, 用以支持小型卫星推进系统的设计。

在实现仿真的业务场景中: Lu 等人<sup>[23]</sup>提出了一种面向服务的工具链, 重点关注特定域的视图、仿真自动化和流程管理, 以支持航空发动机基于模型的系统工程。Vepsäläinen 等人<sup>[24]</sup>开发了一条工具链集成了数学仿真工具和系统建模工具, 以支持控制系统的自动验证。

在联合仿真场景的业务场景中: Lu 等人<sup>[25]</sup>提出了一个 MBSE 工具链用于支持信息物理系统的开发, 特别是联合仿真过程中的自动参数选择。此外, Lu 在<sup>[26]</sup>中使用本体提出了一个 MBSE 工具链用于支持使用基于模型的方法集成复杂系统开发和自动验证, 支持协同仿真。

在信息管理的业务场景中：Shani 和 Franke 等人提出了一种用于转换产品相关数据的统一本体方法，并将不同数据源的本体聚合成一个整体的产品服务系统<sup>[27]</sup>。Damir 等人<sup>[28]</sup>开发了一个基于模型的工具链，通过将可变性管理工具中的可变性信息添加到 SysML 模型中，支持汽车领域软件密集型系统设计的复杂性与可变性。Pavalkis<sup>[29]</sup>等人通过开发了一个工具链，将 MBSE 过程中需求、架构、仿真模型等的与 PLM 平台中的数据例如产线、物料清单等进行集成。Mousavi 等人<sup>[30]</sup>使用基于模型的系统工程方法和工具对半导体制造领域中所涉及到的供应链系统进行中断分析。

在国内，随着系统工程理念的推广和复杂产品研制需求的提升，MBSE 工具链的研究逐渐受到重视，研究主要集中于构建工具链支持系统设计。中国的研究机构和企业开始着手开发适应国内需求的 MBSE 工具链。徐州等人<sup>[31]</sup>根据基于 MBSE 的民机设计流程，从需求、功能、逻辑、物理四个层级介绍了常用的建模仿真软件，并根据软件特性提出了 MBSE 的工具链框架，用于支持民机系统的整体设计。朱睿等人<sup>[32]</sup>结合国内现有航空产品研制生命周期模型，提出了一种适配航空产品研制各生命周期阶段的多“V”迭代研发模式，构建从飞机级到系统级再到组件级需求、设计、仿真、验证的一体化工具链。

尽管国内外在 MBSE 工具链的研究上取得了一定的进展，但仍面临着诸多挑战。首先，大多数工具链开发用于系统的整体设计与仿真，不同建模与仿真或分析工具之间的接口标准不统一，多数不具有模型传递的能力或仍停留在单向的点到点传递过程中，导致模型传递过程的效率降低。其次，MBSE 工具链中模型传递的自动化水平不高，模型的转换与数据的传递大量依赖人工操作，增加了设计周期和成本。

### 1.1.3 模型持续集成的研究现状

通过上述对 MBSE 工具链的应用调研，发现 MBSE 工具之间存在数据、信息和知识交换或模型转换，但少有全部支持或部分支持数据、信息和知识交换以及模型转换的自动化能力，利用软件工程中持续集成的思想与方法可以用于解决模型传递自动化能力不足的问题。下面对持续集成的应用领域与技术实现展开调研。

在软件工程中，持续集成是指通过对频繁提交的代码进行自动化的构建、测试与部署，可以尽早发现和解决代码集成问题<sup>[33]</sup>。在持续集成的应用实践中，开发人员根据具体的业务场景，定义一套用于软件测试的工作流程，通过选定测试工具搭配成一

套测试工具链用于支持和实现该流程，目的是实现软件的自动化构建、测试与部署。例如使用 Git 与 GitLab 工具检测软件代码的版本提交，然后使用 Maven 工具对提交的软件进行自动化编码，然后使用 UXnit 工具实施白盒与黑盒两类测试排查代码运行逻辑错误，然后使用 SonarQube 对软件进行静态代码测试，最后使用命令行工具将测试成功的软件自动化部署到目标平台，工具链通过持续集成工具 Jenkins 自动执行上述各工具，从而实现自动化测试、构建与部署。

持续集成的方法在众多领域的软件系统开发上得到广泛应用：

在汽车嵌入式软件领域，李强等人<sup>[34]</sup>通过搭建持续集成工具链并设计开发流程，用于提高智能化汽车软件开发的敏捷性与效率。王帅等人<sup>[35]</sup>分析了汽车嵌入式软件的开发过程，引入持续集成测试方法，基于 Jenkins 搭建软件持续集成测试平台，支持集成调用编译器、模型在环（MIL）测试、硬件在环（HIL）测试等工具链，从而实现汽车嵌入式软件集成、测试和缺陷通知完全自动化。

在铁路运输领域，单立彬等人<sup>[36]</sup>构建了基于 Jenkins 的高铁列控仿真自动测试系统，用于解决现有实验室列控系统集成测试系统操作繁琐、软件架构较为分散、测试效率低下等问题。

在互联网技术领域，杨文远等人<sup>[37]</sup>给出了面向移动平台软件开发的持续集成系统设计方案，并提出了基于 Jenkins 的持续集成系统实现方法，实现了更加高效、更加优质的移动平台软件开发项目。

在电力工业领域，郭永和等人<sup>[38]</sup>提出了一种面向电力信息系统的分布式自动化测试持续集成平台，该平台能够将信息系统检修前运行测试和日常用户体验测试中重复性强的部分工作从手工化转变为快速自动化，从而提高测试效率。

随着系统的综合化和复杂度逐渐提高，基于模型的系统工程逐渐在型号设计中得到应用，持续集成也同样应用于复杂装备系统的软件开发中。尹伟等人<sup>[39]</sup>构建基于模型的软件开发方法，搭建了基于数据链接的集成开发环境和软件持续集成与验证环境，提升了航电系统软件研制效率和质量，该方法在 C919 民用飞机的航电系统软件研制过程中进行了应用。李昌等人<sup>[40]</sup>从软件测试角度作为切入点，研究航空机载嵌入式软件的结合方式，提出了一种航空机载软件测试工具链设计方案，完成了工具链技术架构的搭建，完成了持续集成环境与静态分析、代码审查、单元测试工具环境的集成开

发,并在某飞机的机电系统软件测试中进行应用,提高了测试效率。张健等人<sup>[41]</sup>自主研发了面向工业仿真软件的自动化持续集成平台,帮助仿真软件交互、封装与效率、功能扩展性及高性能集群环境部署等。赵辉等人<sup>[42]</sup>面向航天软件系统研发的 Sunwise AEM 一体化研发管理平台为载体,以 Bugzilla 缺陷管理系统和 Jenkins 持续集成工具的集成为例,设计了一种数据集成系统,实现了不同工具间的数据集成。

然而上述过程针对于复杂装备系统的软件开发过程,持续集成应用于二进制的软件代码,代码根据设定好的工作流在持续集成工具链中进行自动的构建、测试与部署。在文献<sup>[39]</sup>航天领域航电系统的软件系统开发中,前期采用基于模型的开发方法对软件系统做整体设计,后期采用持续集成的方法将软件代码进行自动化地构建、测试与部署。研究人员开始思考是否可以将持续集成的思想进一步拓展至复杂系统的整体设计中,将对代码的构建、测试等活动替换为系统的建模、分析与仿真等活动,强调在对系统建模过程中,能够根据反馈快速修改模型,并对模型进行自动化的生成、检查与仿真等。

Combemale 等人<sup>[43]</sup>在 2019 年首次提出将持续集成从纯粹的软件开发扩展到面向基于模型的开发过程中的愿景,为 MBSE 更广泛地采用持续集成方法奠定了基础。0 总结了近年来在制造、铁路、物流等多领域系统面向 MBSE 模型持续集成实践。

表 1.1 面向 MBSE 模型持续集成实践

年份	领域	对象	建模语言	持续集成	工具
2017 <sup>[44]</sup>	航空	立方体 卫星	SysML, CONOP S	原型(半自动), 依赖 矩阵, 仿真	MagicDraw
2017 <sup>[45]</sup>	物流	海上起 重机	SysML UML	重用, 优化	Architect
2017 <sup>[46]</sup>	通讯	雷达	UML, SysML, d ataflow, CONOP S	可追溯性、一致性检 查、系统风险分析、模 型发布	DOORS, DS M, Radiant w eb UI
2017 <sup>[47]</sup>	航空	小型卫 星系统	SysML	视点、规则	Sparx EA

表 1.1 面向 MBSE 模型持续集成实践（续表）

年份	领域	对象	建模语言	持续集成	工具
2019 <sup>[48]</sup>	铁路	商业系统	EA/BPMN	模型质量文档生成	CARLA, VIR ES
2019 <sup>[49]</sup>	汽车	自动驾驶汽车	动态模型	测试生成	——
2020 <sup>[50]</sup>	制造	CPS, 机器人	SysML, 3D CAD	代码生成脚本, 测试用例生成器	Capella, Gitlab CI, jupyter
2021 <sup>[51]</sup>	铁路	运输系统	SysML	请求机器人, 文件生成, 仿真	frama-C, OpenSCAP
2021 <sup>[52]</sup>	信息安全	防火墙	数据/威胁/基础设施/认证模型	可追溯性	Uppaal
2021 <sup>[53]</sup>	制造&物流	控制系统	有限状态机	针对攻击、代码检查的自动化测试生成	Sparx EA

基于上述表格，可以总结出将持续集成的应用于基于模型的开发过程中，用以支持 MBSE 模型之间信息的传递，存在以下几个问题：

（1）持续集成过程集成的不同的建模语言与领域工具，由于模型彼此之间格式不兼容、工具之间的接口标准不统一，该过程仅支持少部分开发活动的集成，导致持续集成自动化水平受限，设计信息无法以模型为载体跨多个设计阶段继续传递。

（2）缺少一种支持持续集成业务流程形式化定义的方法。大多数持续集成的过程只支持通过脚本定义工作流，不支持对工作流的进行显式地流程建模。通过脚本定义工作流存在学习成本高，脚本编写困难等问题，增加了实现系统开发自动化的成本。

### 1.3 研究意义

面向复杂装备系统的架构设计-仿真验证一体化过程中的 MBSE 模型传递，仍存在多源异构模型数据交换困难、数据信息交换与模型传递自动化水平低等问题。针对上述问题，开展面向 MBSE 模型传递技术研究。其研究意义在于：

(1) 构建面向 MBSE 模型架构设计-仿真验证一体化过程的模型传递框架。首先, 面向架构设计-仿真验证一体化过程, 给出模型传递的概念定义; 其次, 对模型传递的内涵与关键特征进行分析, 包括互操作性与自动化能力两个维度的关键特征; 最后, 提出模型传递的技术框架。该框架根据互操作性与自动化能力两个维度的关键特征进行技术选型, 给出了实施模型传递的指导性技术流程。

(2) 解决了模型传递过程中异构模型集成与传递自动化水平低的问题。首先, 本文提出了一种基于服务的架构-仿真模型一体化表达方法, 基于数据集成规范将异构的架构模型与仿真模型进行统一描述, 解决了模型跨工具统一集成的问题。其次, 本文提出了一种模型持续集成 workflow 设计方法, 解决了基于模型的设计流程中手动操作导致设计过程自动化水平低、错误率高的问题。该方法能有效实现系统设计过程中数据的集成与融合, 提高系统设计与仿真的效率, 推动数字工程的落地与实施。

## 1.4 研究内容与组织架构

本文拟对面向复杂系统设计的 MBSE 模型传递方法进行研究, 组织框架安排如下。

第一章: 绪论。从 MBSE 领域中存在模型传递的现象出发, 阐述国内外基于数据互操作实现模型传递、基于工具链实现模型传递以及基于持续集成实现模型传递模型的研究现状并分析目前存在的问题, 然后引出本文的研究意义, 其次给出研究内容和论文结构。

第二章: 面向 MBSE 模型的模型传递理论与研究框架。首先对面向 MBSE 模型的模型传递的概念进行定义, 然后分析向 MBSE 模型的模型传递的关键特征, 包括互操作性特征与自动化能力特征。最后, 提出面向 MBSE 模型的模型传递研究框架及技术流程, 为后续研究作整体指导。

第三章: 基于服务的架构-仿真模型一体化表达方法。首先, 介绍开放生命周期服务规范以及元数据模型。其次, 介绍多架构模型 KARMA 与开放生命周期元数据模型之间的映射方法; 同理, 介绍多领域仿真模型 Modelica 与开放生命周期元数据模型之间的映射方法。根据开放生命周期服务规范, 实现架构模型与仿真模型的统一描述, 实现模型传递的互操作性特征。

第四章: 面向模型持续集成的自动化 workflow 设计方法。首先, 介绍 workflow 定义规



范及其组成元素的表达形式，为后续形式化定义 workflow 提供理论基础。其次，面向架构设计-仿真验证一体化过程，分析 workflow 活动，建立起活动与模型操作服务之间的关联关系，基于服务统一执行 workflow 活动，为自动化提供统一的操作基础；然后，建立起规范组成元素与 workflow 活动之间的映射关系，构建 workflow 活动定义与执行的语义模板，为后续基于特定场景的进行 workflow 设计提供提可复用的框架，实现模型传递的自动化特征。

第五章：面向航空发动机系统设计的模型传递案例验证。以航发核心机模型为例，基于 KARMA 语言以及 RFLP 建模方法论构建航发核心机的架构模型，基于 Modelica 语言构建航发核心机的仿真模型，根据提出的模型传递框架，构建模型持续集成 workflow，并关联到航发核心机的架构模型与仿真模型服务，打通架构工具与仿真工具之间的闭环数据传递链路，进而验证所提出方法的有效性。

最后，在总结和展望部分，对本文的研究工作和创新点进行总结，并分析研究所存在的不足和未来的工作展望。

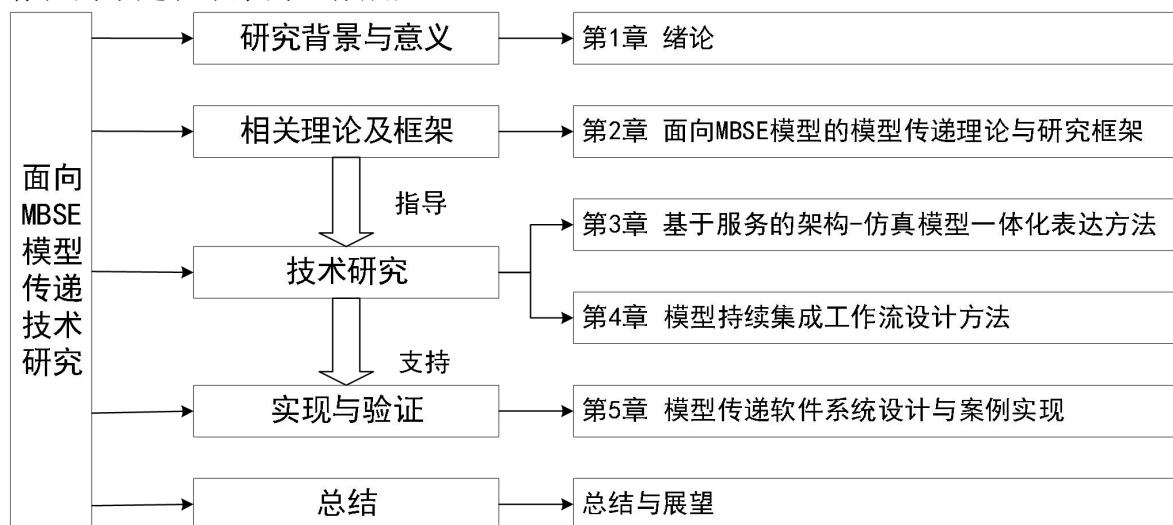


图 1.3 论文组织架构

## 1.5 本章小结

本章从基于模型的复杂系统设计出发，阐述了国内外在使用 MBSE 方法进行复杂系统设计的过程中，调研了（1）设计信息以模型为载体在不同阶段、工具之间传递的方式：数据互操作方式；（2）模型信息传递的载体：MBSE 工具链；（3）模型信息

传递自动化水平：包括软件工程与基于模型的系统工程领域中应用持续集成的研究现状。分析并总结了现有的研究方法，在设计信息以模型为载体在不同阶段、工具之间传递存在的不足，从而确定本文的研究内容和组织结构。

## 第 2 章 MBSE 模型传递理论与研究框架

### 2.1 引言

随着基于模型的系统工程（Model-Based Systems Engineering, MBSE）的不断发展和应用，模型在系统设计和验证过程中扮演着越来越重要的角色。MBSE 的核心在于利用模型来描述、分析和验证系统的行为和性能，从而提高系统开发的效率和质量。在 MBSE 过程中，设计信息以模型为载体在系统设计的不同阶段、不同工具之间进行传递，该过程是保证模型信息一致性的关键，同时该过程的传递效率也直接影响到不同设计环节之间的协同效率。针对设计信息以模型为载体的传递过程存在异构模型信息传递难、跨工具模型信息传递自动化水平低的问题，开展 MBSE 模型传递的理论研究，并提出面向复杂系统设计的 MBSE 模型传递研究框架，为后续的模型传递技术研究提供指导。

### 2.2 MBSE 模型传递概念及内涵

模型传递在 MBSE 领域扮演着至关重要的角色，它涉及了在系统开发过程中不同阶段、工具和平台之间进行模型数据的交换与传递，这一过程的核心目的在于确保模型信息的一致性和完整性，从而支持系统设计的协同性和迭代效率。本节旨在探讨 MBSE 模型传递概念及内涵，并结合 MBSE 工具链与工作流的概念，深入探讨 MBSE 工具链对模型传递的支持。

#### 2.2.1 MBSE 模型传递的定义与内涵

根据 INCOSE 在《系统工程 2020 年愿景》<sup>[5]</sup>中对基于模型的系统工程(Model-based System Engineering, MBSE)的定义，MBSE 是指对建模（活动）的形式化应用（formalized application of modeling），以便支持系统需求、设计、分析、验证和确认等活动，这些活动从概念设计阶段开始，持续贯穿到设计开发以及后来的所有寿命周期阶段。

系统是 MBSE 应用的对象，它是由相互作用的部分组成的整体，各个部分之间相互关联与作用涌现出一定的功能<sup>[54]</sup>。系统具有整体性、相关性、动态性等六大特性，

具有结构特征与行为特征<sup>[55]</sup>。系统间信息的流动和转换是形成系统功能和性能的关键。系统生命周期中包含着许多信息的传递和转换过程，例如在系统需求分析阶段捕获的需求信息需要传递到下游的功能分析与架构设计阶段。

MBSE 的核心是模型，模型是现实物理世界的抽象，承载着现实物理世界的信息，用于模拟或表达一个真实世界的过程或结构，具有客观性。建模的目的首先针对基于文本的描述方式，模型可以保证系统需求、功能、结构、行为、参数等表达的无歧义性与无模糊性；其次，模型具有模块化的特点，提高了系统生命周期中信息的流传、传递以及再利用的效率；再次，模型具有可执行可验证的特点，使用模型在早期进行对系统需求和设计进行验证，可以避免将设计错误带到系统实施阶段。

MBSE 实施包括不同的阶段，不同的阶段将会使用不同的领域工具，例如需求工具 Rhapsody 捕获系统需求信息，使用架构建模工具 MagicDraw 描述系统的功能、结构、行为与参数，使用领域仿真建模 OpenModelica 对系统行为与活动进行虚拟仿真。将由需求管理工具、架构建模工具、仿真建模工具以及其它分析工具共同组成的工具集，称为 **MBSE 工具链**<sup>[10]</sup>。在 MBSE 工具链中即不同的分析、建模与仿真工具之间，存在模型的传递与交换，例如需求工具 Rhapsody 捕获的系统需求条目需要向下游系统设计模型传递，系统设计模型中的结构、行为与参数信息需要传递到下游仿真模型的结构、行为与参数的构建。在上述 MBSE 模型传递与交换的过程中存在一系列活动和步骤，定义了如何创建、分析、验证和维护 MBSE 模型，这些活动和步骤的集合被称为 **MBSE 工作流**<sup>[10]</sup>。MBSE 工具链为执行 MBSE 工作流提供了所需的工具、技术与方法。

基于上述理论，结合在 MBSE 实施阶段存在模型传递的现象，本文首次提出将 **MBSE 模型传递**定义为：在系统工程过程中，为了实现系统设计、分析、验证和确认等活动，数据、知识、信息以模型为载体，在不同开发阶段、不同工具或环境之间进行有序的、动态的传递过程。这个过程涉及到模型的创建、更新、转换和验证，以确保系统需求、设计、分析和验证等活动之间的信息一致性和连续性。

模型传递在 MBSE 工作流中扮演着至关重要的角色，模型传递是发生在 MBSE 工作流中的动态过程，它是具有流动方向与流动逻辑。从小的颗粒度来看，模型传递是指发生在某个活动节点的，是指设计信息由一个模型经过某活动或操作传递到另一

个模型。从大的颗粒度来看，多个模型传递的节点，组成了一条模型传递的动态链路，这条动态执行的链路能够将模型有向地传递到很远的下游。MBSE 模型传递与 MBSE 工作流之间的关系可以从以下几个方面来理解：

（1）动态与静态的关系：MBSE 工作流是一个静态的概念，它定义了 MBSE 实施过程中应该遵循的步骤和活动。而模型传递是一个动态过程，它涉及到在这些工作流步骤中设计信息的实际流动和转换。

（2）信息流动：模型传递关注的是设计信息如何在 MBSE 工作流的不同阶段和活动中流动。这包括从一个模型到另一个模型的信息传递，以及从一个工作流节点到下一个节点的信息传递。

（3）活动节点的连接：在 MBSE 工作流中，每个活动或节点都可能涉及到模型的创建、更新或验证。模型传递确保了这些活动之间的连贯性，保证了设计信息在工作流中的每个阶段都能被正确理解和使用。

（4）方向性：模型传递具有明确的方向性，它指示了设计信息在 MBSE 工作流中的流动方向。这有助于确保设计决策和变更能够被有效地沟通和实施。

（5）链路形成：多个模型传递的节点可以组成一条动态链路，这条链路展示了设计信息在 MBSE 工作流中的完整路径。这种链路有助于分析和优化设计过程，确保设计信息的一致性和完整性。

（6）执行过程：从宏观角度看，模型传递可以视为整个 MBSE 工作流的动态执行过程。它不仅仅是单个活动之间的信息传递，而是整个系统工程活动序列中的信息流动。

（7）反馈和迭代：模型传递还涉及到反馈机制，允许在 MBSE 工作流中的不同阶段之间进行迭代和反馈。这有助于持续改进设计，并确保最终系统满足所有要求。

### 2.2.2 MBSE 模型传递的场景与范围

根据 MBSE 模型传递的定义，进一步分析 MBSE 模型传递的场景与范围。

#### （1）MBSE 模型传递的场景

首先根据 MBSE 实施过程中存在的系统设计、分析、验证和确认等活动，对 MBSE 模型传递的场景进行分析。它包括需求捕获与分析、系统架构设计、行为建模与仿真、模型转换与集成、验证与确认、迭代与优化、跨学科协作、项目管理与追踪、

维护与支持、信息的可追溯性这几个方面进行分析。其中，需求捕获与分析是指使用需求工具（如 Rhapsody）捕获利益相关者的需求，并分析这些需求的可行性和完整性，它涉及需求的收集、分类、优先级排序和详细描述。系统架构设计是指使用架构建模工具（如 MagicDraw）创建系统的高层和详细设计，包括功能、逻辑和物理架构，设计到系统组件的识别、接口定义、数据流分析和架构优化等。行为建模与仿真是指使用仿真工具（如 OpenModelica）用于模拟系统的行为和性能，验证设计是否满足需求，涉及建立系统的动态模型，进行仿真实验，并分析结果。模型转换与集成是指在 MBSE 工具链中，不同工具之间需要进行模型的转换和集成，以支持跨工具的一致性和协作，包括模型的导入、导出、格式转换和数据映射。验证与确认是指使用模型进行系统的需求验证、设计验证和最终的系统验证与确认，确保系统设计满足所有需求，并通过测试和审查来确认系统的正确性。迭代与优化是指在 MBSE 的迭代过程中，模型会根据反馈进行更新和优化，涉及模型的迭代改进、性能调优和风险缓解。跨学科协作是指不同领域的专家需要共享和使用模型，以确保系统设计的全面性和综合性，包括跨学科团队之间的沟通、协调和信息共享。项目管理与追踪是指项目管理工具用于追踪模型的状态、进度和变更，涉及模型版本控制、进度监控和风险管理。维护与支持是指在系统部署后，模型用于支持系统的维护和升级，包括系统维护文档的生成、故障诊断和性能改进。信息的可追溯性是指确保从原始需求到最终设计的每一步都有记录，以支持审计和合规性检查，涉及需求追溯、设计决策记录和变更管理。

MBSE 模型传递包含模型转换与集成。其中，模型转换是指将一个模型转换成另一个模型的过程，这种转换基于一系列预定义的映射规则。这些规则定义了源模型（输入）和目标模型（输出）之间的对应关系，确保了模型的语义一致性和结构的一致性。模型转换通常涉及到不同抽象层次或不同表示形式的模型之间的转换，例如，从概念模型转换到设计模型，或者从系统模型转换到仿真模型。模型集成侧重于将来自不同来源或领域的多个模型或模型组件合并到一个统一的、协调的系统中。这个过程可能包括解决模型之间的接口和交互问题，确保各个模型组件能够无缝协作，形成一个完整的系统模型。模型集成的目标是创建一个全面、一致的系统表示，它可以用于进一步的分析、验证和决策支持。

## (2) MBSE 模型传递的范围

然后, 基于 MBSE 模型传递的概念, 将模型传递的范围根据以下几个维度来界定:

- 开发阶段: 模型传递可以发生在系统开发的任何一个阶段, 从概念设计、详细设计、实现、测试到维护和退役等各个阶段。每个阶段可能需要特定类型的模型传递, 以满足该阶段的需求和目标。
- 工具和平台: 模型传递可能涉及多种不同的工程工具和平台, 如需求管理工具、建模工具、仿真工具、版本控制系统等。传递过程需要考虑这些工具之间的兼容性和接口标准。
- 模型类型: 传递的模型可以是需求模型、系统架构模型、物理模型、数据模型等。不同类型的模型可能需要不同的传递机制和技术。
- 组织和团队: 模型传递可能在不同的组织单元、项目团队或合作伙伴之间进行。这要求模型传递能够跨越组织边界, 同时保持信息的安全性和保密性。
- 技术和标准: 模型传递的实施需要依赖于一系列技术和标准, 如模型交换格式、中间件、APIs 等。这些技术和标准的选择和应用范围将直接影响模型传递的效率和质量。

## 2.3 MBSE 模型传递的关键特征

在基于模型的系统工程 (MBSE) 领域中, 模型传递存在于多种系统开发场景, 它面临的主要挑战是如何适应不同工具的特定要求在系统开发的不同阶段和工具之间传递信息, 并保持数据的一致性和准确性。然而, 在使用 MBSE 方法进行复杂系统设计的过程中, 存在模型数据格式不兼容、工具接口不统一等问题, 这些问题可能会对传递过程的有效性构成威胁。为了应对这些挑战, 分析并提取模型传递的特征就显得尤为重要。这些特征不仅指导了传递过程的设计和实施, 而且对于提高传递的可靠性和效率具有决定性的影响。

模型传递通常涉及一系列工具, 如需求管理工具、建模工具、仿真工具和测试工具等, 它们之间存在频繁的模型数据传递与交换。因此, 互操作性是模型传递的基础。这包括将不同工具的模型数据进行连接、转换和传递, 以实现信息的一致性和完整性。

此外, 模型传递还与 MBSE 工作流中的自动化密切相关。通过自动化的模型传递,

可以实现工作流中各个环节的无缝对接，减少人为错误，提高开发效率。自动化的模型传递还可以支持实时的数据更新和反馈，使得系统工程师能够快速响应设计变更和问题，从而实现更加灵活和动态的系统开发过程。

因此，通过上述分析，提取出了模型传递的两个特征，分别为是互操作性和自动化能力，具体介绍如下。

### 2.3.1 MBSE 模型传递的互操作性特征

模型传递涉及不同种建模与分析工具之间的数据传递。互操作性使得这些工具可以作为一个统一的工具链协同工作，模型数据可以在这些工具之间流畅传递，支持从概念设计到最终验证的整个开发流程。

互操作性（Interoperability）是指不同系统、组织、软件应用程序或工具之间的能力，以使用彼此的数据、资源或功能<sup>[56]</sup>。在 MBSE 中，互操作性是实现模型传递的前提，它允许不同来源的模型能够在多个工具和平台之间无缝交互和集成。这一概念起源于军事领域，用于描述不同国家和军种之间的装备和系统如何协同工作。随着技术的发展，互操作性已成为商业、工业和信息技术等多个领域的关键需求。

实现互操作性的关键技术方法包括：

- 标准化的数据模型和格式：如 SysML、XML<sup>[57]</sup>、JSON<sup>[58]</sup>等，它们提供了一种通用的语言，使得不同的工具可以理解和处理模型数据。
- 中间件技术：如 OSLC（Open Services for Lifecycle Collaboration）<sup>[59]</sup>，提供了一种机制，允许不同的应用程序通过共享数据和服务进行通信。
- APIs（应用程序编程接口）：定义了不同软件组件之间如何相互交互的规则，使得模型传递可以通过编程方式自动化。
- 模型转换和映射工具：用于将一种模型格式或语言转换为另一种，同时保持模型的完整性和准确性。

### 2.3.2 MBSE 模型传递的自动化能力特征

由于手动模型传递依赖于用户手动执行一系列操作来移动和转换模型数据。这通常涉及到导出模型、转换格式、导入到目标系统以及可能的手动数据映射和调整。手动传递的缺点包括耗时、容易出错、难以扩展和重复性工作量大。相比之下，自动模



型传递通过软件工具和脚本来实现，无需或很少需要人为干预。自动化可以显著减少错误，提高传递速度和一致性，使得模型更新更加及时和频繁。因此，模型传递需要实现自动化的数据传输和转换。自动化能力包括自动检测模型变更、自动触发数据传递、自动执行模型检查和仿真等。通过自动化，可以提高模型传递的效率，减少人工干预。

自动化传递的工具和技术包括各种模型转换工具、脚本语言、中间件、APIs 和集成开发环境（IDE）。例如，模型转换工具可以自动将一种模型格式转换为另一种，脚本语言（如 Python）可以编写自动化脚本来处理模型数据，中间件（如 OSLC）提供了标准化的服务来支持不同工具间的通信，APIs 允许不同系统之间进行编程交互，而 IDE 则提供了自动化构建、测试和部署模型的环境。

自动化模型传递对提高工程效率有着显著影响。首先，它可以减少重复性工作，让工程师专注于更高价值的任务。其次，自动化减少了人为错误，提高了模型传递的准确性和可靠性。此外，自动化加快了模型更新和迭代的速度，支持快速响应设计变更和市场需求。最后，自动化使得模型传递过程更加透明和可追溯，有助于提高团队协作和项目管理的效率。

尽管自动化带来了许多好处，但在自动化模型传递过程中仍然需要进行监控和控制。监控涉及跟踪传递过程的状态、性能和结果，以确保模型的正确传递和集成。控制则包括对自动化过程的管理，如调整传递参数、处理异常情况和实施安全措施。

## 2.4 MBSE 模型传递的技术框架

在上文基于 MBSE 模型传递的概念、内涵、范围以及所包括的两个关键特征进行分析后，提出本文面向复杂装备系架构设计与仿真验证一体化过程的 MBSE 模型传递技术研究框架，如图 2.1 所示，在该框架中涉及到两个关键技术：①基于服务的架构-仿真模型一体化表达方法。②面向模型持续集成的自动化 workflow 设计方法。

MBSE 模型传递技术框架与这两个关键方法之间的关系如下：

根据模型传递的互操作性关键特征，研究基于服务的架构-仿真模型一体化表达方法（即关键技术①）。首先，介绍开放生命周期协作服务规范（Open Services for Lifecycle Collaboration, OSLC）中的元数据模型，为后续统一表达架构模型与仿真模型

提供集成基础；其次，基于开放生命周期协作服务（Open Services for Lifecycle Collaboration, OSLC）规范，分析架构模型与仿真模型的模型组成结构、参数和操作接口等关键特征，提出架构模型、仿真模型与 OSLC 元数据模型之间的映射方法，实现模型信息及模型操作（例如模型的增删改查操作）在概念层的服务化表达；然后，根据模型的映射方法，开发工具适配器，实现设计信息在架构工具与仿真工具之间端到端的传递与集成，用于验证 MBSE 架构模型与仿真模型传递的互操作性特征。

根据模型传递的自动化能力关键特征，研究面向模型持续集成的自动化 workflow 设计方法。首先，面向架构设计-仿真验证一体化过程，分析 workflow 包含的模型传递活动，明确活动模型传递的输入与输出；然后，介绍基于流水线脚本的工作流表达形式，建立起流水线脚本元素与 workflow 活动之间的映射关系；其次，构建 workflow 对应的流水线脚本语义模板，构建 workflow 活动定义与执行的语义模板，并建立起 workflow 活动与关键技术①中实现的基于 OSLC 的模型操作服务之间的关联关系，基于 OSLC 服务自动执行 workflow，用于验证 MBSE 架构模型与仿真模型传递的自动化特征。

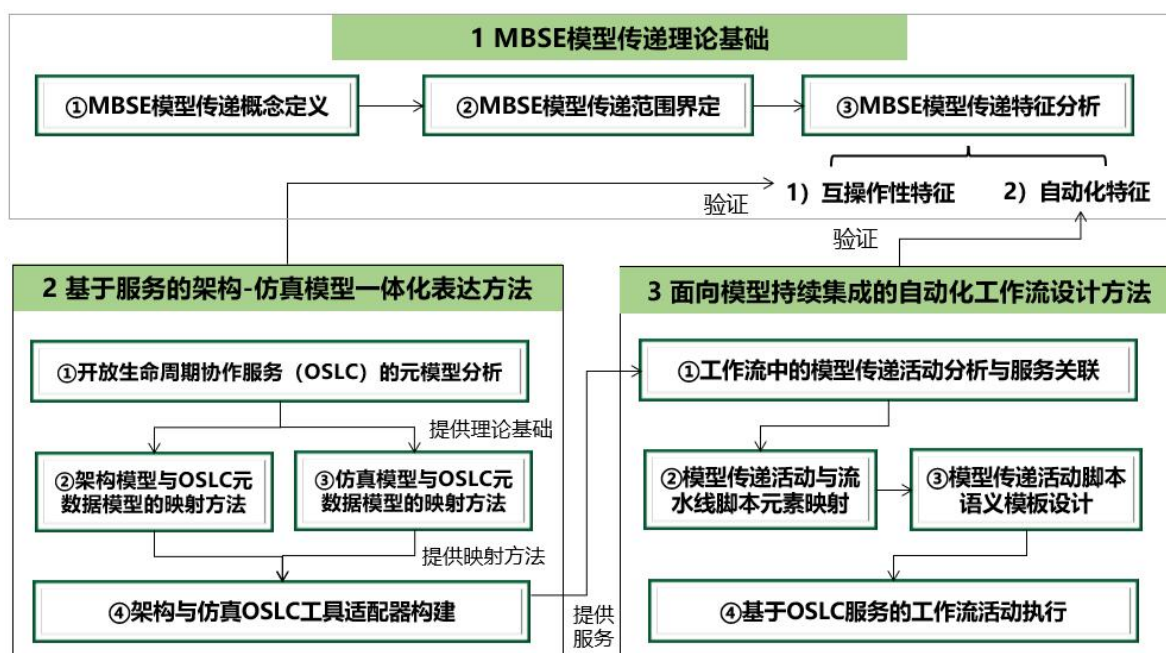


图 2.1 MBSE 模型传递技术研究框架

## 2.5 本章小结

本章从面向 MBSE 模型传递的理论基础进行研究，给出了在 MBSE 领域模型传递的定义与内涵，并给出了模型传递的场景与范围。其次，分析了模型传递的互操作特征与自动化能力特征。基于这两个特征进行技术选型，进一步提出了面向 MBSE 模型的模型传递研究的技术框架，作为下文第三章、第四章技术研究的框架指导。

## 第 3 章 基于服务的架构-仿真模型一体化表达方法

### 3.1 引言

针对于模型传递的互操作性特征，研究工具级的模型数据互操作方法，使得架构与仿真模型在语义层与数据层实现一体化表达。标准的数据集成规范是实现产品生命周期中模型传递与数据集成的基础。针对产品生命周期中由跨领域跨阶段跨工具导致的异构模型传递与数据集成困难，研究基于开放生命周期协作服务（Open Service for Lifecycle Collaboration, OSLC）的架构-仿真模型一体化表达方法。首先，研究开放生命周期协作服务（OSLC）的技术规范与元数据模型，为后续架构与仿真模型一体化表达提供理论基础；其次，基于 OSLC 这一数据集成规范将架构与仿真模型进行统一描述，建立架构、仿真模型与 OSLC 元数据模型之间的映射关系，消除架构与仿真模型的语义异构，为后续实现设计信息在架构模型与仿真模型之间端到端传递提供理论基础；最后，基于上述映射方法，开发架构与仿真工具适配器，为实现设计信息在架构模型与仿真模型之间端到端的传递提供实际的操作基础。

### 3.2 开放生命周期协作服务元数据模型

在使用 MBSE 方法进行复杂装备系统开发时，系统架构设计与仿真验证阶段需要分别基于不同的建模语言规范与领域工具进行开发。由于架构模型与仿真模型语法语义异构，导致设计参数难以跨过建模与仿真工具实现端到端的传递。为了应对上述难题，开放生命周期协作服务规范作为可以应用于 MBSE 全生命周期工具集成与数据互操作的新标准，为实现模型传递的互操作性特征提供了一个新的方案。

#### 3.2.1 开放生命周期协作服务规范

开放生命周期协作服务规范（Open Service for Lifecycle Collaboration, OSLC）是由 IBM 提出的一套用于集成 MBSE 生命周期内工具的技术规范<sup>[59]</sup>。它规定以最少量的协议支持工具之间的无缝集成与数据互操作。OSLC 旨在提供生命周期管理工具的无缝集成，并使其能够在早期开发阶段的数据之间建立明确的关系<sup>[60]</sup>。

OSLC 为实现互操作性使用了两个基本技术。其一是使用基于资源描述框架（Re

source Description Framework, RDF)<sup>[61]</sup>的核心模型对开发信息以及技术资源(模型, 数据, 操作接口等)以及进行统一表达, 并为其生成特定的统一资源标识符(Uniform Resource Identifier, URI)。资源描述框架(RDF)是一个用于承载或交换元数据的基础框架, 可以描述资源的特性, 及资源与资源之间的关系, 表现形式为“资源-属性-属性值/资源”的三元组。由于 OSLC 规范是基于 RDF 定义的, 其核心模型包括服务提供方目录、服务提供方、服务、资源等相关概念都要求具有 RDF 的表达形式。通过将开发信息与技术资源统一描述为 RDF 资源, 可以解决语义层的异构性, 通过 RDF 三元组中的“属性”实现数据链接。其次, OSLC 使用 HTTP 协议, 支持以发送 HTTP 请求的方式来操作(创建、读取、更新、删除)这些资源。基于网端的 HTTP 通信协议, 可以对 OSLC 规范进行实例化并对实例进行操作。OSLC 服务可以通过服务提供者的资源访问, 服务提供者会描述提供的服务。每个服务都可以提供资源创建的创建工厂、资源查询的查询功能和委派的 UI 对话框, 以使客户端能够通过 Web UI 创建和选择资源。查询功能和创建工厂可以提供资源形状来描述由服务管理的资源的属性。基于 HTTP 标准实现模型资源网端的操作, 可以使模型资源不受地域、组织、物理载体的限制, 解决开发过程中产生的信息孤岛的局面。

OSLC 规范由核心规范和领域规范组成<sup>[62]</sup>。核心规范用于对核心的集成技术及通用概念进行描述, 使用标准的术语列出每个 OSLC 服务可以支持的通用特性。生命周期或领域的每个部分都有各自的规范, 例如变更管理、质量管理、估计和测量等, 每个领域的规范都建立在核心规范之上。

OSLC 核心规范介绍了 OSLC 的概况、OSLC 如何定义资源以及 OSLC 如何提供服务等内容。核心规范指定了 OSLC 服务必须、应该和可以满足的功能。核心规范根据允许的和要求的属性名称和值类型建立了用于定义资源的术语和规则, 其中核心术语包括服务提供方目录(ServiceProviderCatalog)、服务提供方(Service Provider)、服务(Service)、资源(Resource)、资源形状(ResourceShape)、属性(Property)、查询功能(Query Capability)、创建工厂(Creation Factory)等相关概念, 这些概念也称作 OSLC 的元数据模型, 将在 3.2.2 节详细介绍。

领域规范是基于核心规范的基础上定义的。根据具体的工程领域进行划分, 如需求管理、配置管理、变更管理、质量管理、资产管理、架构管理等领域。OSLC 领域

规范使用 OSLC 核心规范中的规则和术语来描述各自领域中的资源，建立在 OSLC 核心规范之上，允许实现 OSLC 核心规范的部分规则而非全部规则，因此可以避免承担实现不需要功能的额外成本，为 OSLC 规范的实现提供了灵活性。此外，OSLC 领域规范的定义，使得该规范在特定领域的应用更符合领域特征，减少了工程师的使用成本，更有利于提高特定域互操作性的效率。例如，在需求管理领域，相比于核心规范，主要表现在术语定义与规则制定两个方面。首先，在术语定义方面，需求管理领域的规范在核心规范的基础上，扩展了术语及功能。其次，在规则制定方面，需求管理领域的规范在核心规范的基础上增加了一些特定于需求管理领域的附加要求。

### 3.2.2 开放生命周期协作服务元数据模型

OSLC 的核心规范规定了 OSLC 的元数据模型。OSLC 元数据模型可以将系统全生命周期中产生的模型或数据资源（例如架构模型与仿真模型）在概念层进行统一表达，从而解决异构模型或数据语义异构的问题，为实现工具集成与数据互操作提供基础。OSLC 元数据模型包括服务提供方目录（ServiceProviderCatalog）、服务提供方（Service Provider）、服务（Service）、资源（Resource）、资源形状（ResourceShape）、属性（Property）、查询功能（Query Capability）、创建工厂（Creation Factory）等相关概念，概念以及概念与概念之间的关系如图 3.1 所示。

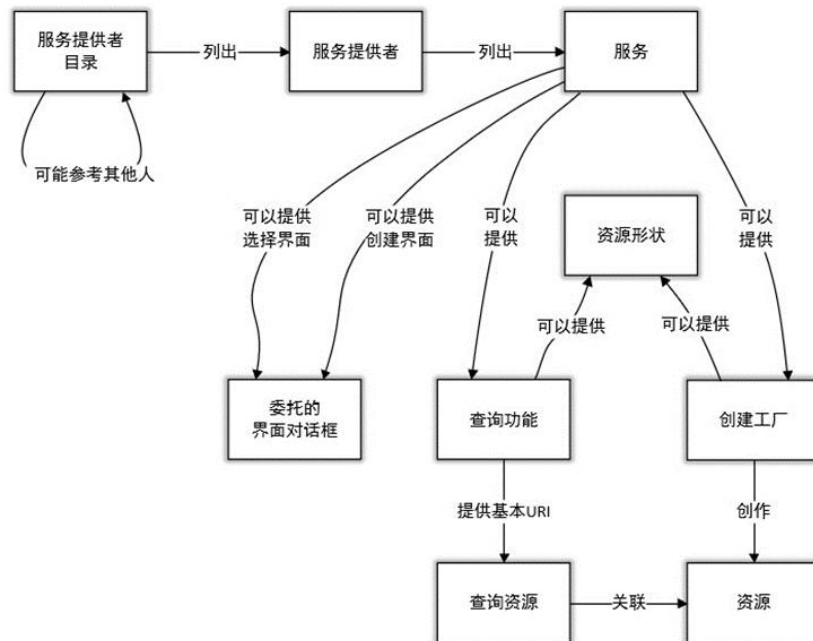


图 3.1 OSLC 元数据模型

- OSLC服务提供方目录是指提供一个或多个服务提供方资源的OSLC实现。
- OSLC服务提供方是指能提供一个或多个OSLC服务的OSLC实现。
- OSLC服务是一些功能的集合，能够查询、创建、更新和删除OSLC资源。
- OSLC资源是指由OSLC服务管理的资源，是OSLC服务查询、创建、更新和删除的对象。
- OSLC资源形状：是资源所包含的属性集，其属性值是OSLC服务查询、创建、更新和删除的最小单元。
- OSLC定义的属性：由OSLC规范定义或由OSLC资源形状定义的资源属性。
- OSLC查询功能：是指查询OSLC资源、资源形状即属性集，例如查询资源的属性以及属性值等信息。
- OSLC创建工厂：是通过为资源类型以及资源形状资源属性赋值指，创建一个新的OSLC资源实例。

以 Excel 需求表格为例，解释 OSLC 元数据模型中的各个概念及概念之间的关系，如表 3.1 所示。假定该需求表格保存在本地文件夹中，当使用该表格对系统需求进行捕捉时，需要对 Excel 表格中的行、列、单元格内容进行查询、修改、新增、删除等活动。

表 3.1 Excel 表格对应的 OSLC 元数据模型

OSLC 元数据模型	Excel 需求表格组成
服务提供方目录	需求表格所在文件夹
服务提供方	需求表格
服务	对需求表格、行以及单元格进行增删改查
资源	需求表格，需求表格中的行或行中的单元格
查询功能	查询需求表格、行或单元格
创建工厂	创建需求表格、行或单元格

### 3.3 架构模型与 OSLC 元数据模型的映射方法

#### 3.3.1 基于多架构建模语言的架构模型构建方法

多架构建模语言 KARMA 是一个基于 GOPPRR-E 元模型的文本式形式化建模语言<sup>[63]</sup>。其中，GOPPRR-E 是一种面向对象的架构建模方法，通过六种基本的元元模型

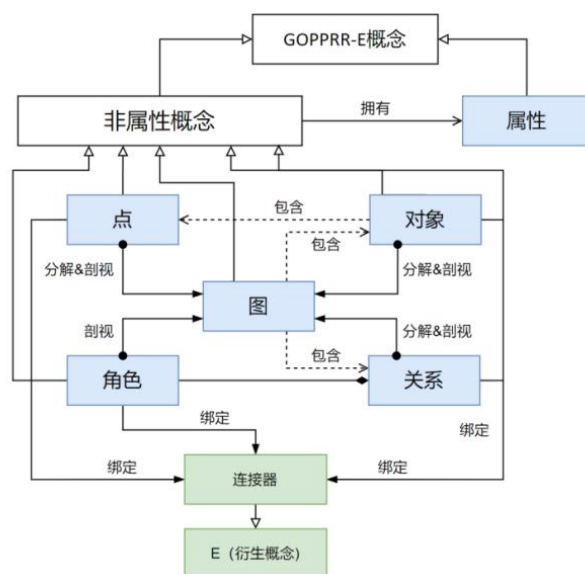


图 3.2 GOPPRR-E 元元模型及其之间的关系

- **图（Graph）**：表示模型的某一类视图，是对象、关系的集合。图可以拥有属性。
- **对象（Object）**：对象是对图中一类实体（例如，使用SysML建模语言描述的一个电池模块即为一个实体）的抽象描述，可以单独存在。对象可以包含点，且可以拥有若干个属性。
- **点（Point）**：描述对象中与关系（或角色）连接的端口，点不能够单独存在，必须通过依附在对象的上形式存在。
- **关系（Relationship）**：是对象之间的关联关系的抽象描述，通过角色与对象或者对象上拥有的点相连。关系不能单独存在，其两端必须与对象或者对象上的点相连才具有意义。



- **角色 (Role)**: 描述对象或对象上的点如何与关系相连接。角色依附在关系的两端, 不能脱离关系单独存在。角色可以用于表示方向, 例如始端和终端。
- **属性 (Property)**: 用于描述其他五个元元模型 (图、对象、角色、关系和点) 的特性。

GOPRR-E 元元模型作为构建领域建模语言的基础, 也遵循一定的语法以描述六种元素之间的关联规则。在 GOPRR-E 元建模框架中, 这些关联规则是通过扩充的衍生概念描述的, E (Extension) 即扩充的衍生概念, 主要包括:

- (1) **包含**: 描述元元模型之间的存在的包含关系, 图可以包含对象、关系, 对象可以包含点。
- (2) **属性所属**: 描述图、对象、点、关系、角色五种元素和属性之间的拥有关系。
- (3) **剖视**: 描述一个对象 (或关系、点、角色) 元模型的实例可以从不同视角映射为多个图实例, 表示图实例是对对象 (或关系、点、角色) 实例某一视角剖面的详细化描述。
- (4) **分解**: 描述一个对象 (或关系、点) 元模型的实例可以分解为一个图实例。表示图实例是对对象 (或关系、点) 实例内部组成的详细化描述。
- (5) **绑定**: 描述对象、点、角色、关系元元模型之间存在的约束方式, 通过绑定可以定义元建模语言中元素之间的连接规则。例如, 描述何种类型对象实例或点实例可以与何种类型角色实例进行绑定, 何种关系实例两端能连接何种类型的角色实例。
- (6) **跨模型关联**: 描述属于两个不同图实例的元素之间的关联关系, 元素可以是对象、点、角色实例。

此外, KARMA 语言还具有模型库 (Package) 与模型 (Model) 概念, 它们分别是指:

- **模型库 (Package)**: 包含一系列模型的集合, 用于描述特定领域的问题和解决方案。
- **模型 (Model)**: 在模型库中定义的特定问题领域的抽象表示, 用于描述该领域中的实体和关系。

GOPRR-E 元建模方法以统一的底层数据描述多领域建模语言的元模型和型，实现了多领域架构模型的统一表达。使用 GOPRR-E 元建模方法构建系统架构模型的过程可以分为元模型构建阶段和模型构建阶段。如图 3.3 所示，元模型构建阶段的步骤包括：（1）确定元模型：建模工程师根据所建模目标对象所涉及的领域，总结领域相关的知识、标准规范以及约束，然后抽象分析得出所有类型的元模型。确定元模型具有两种方式：依据成熟建模语言构建元模型和特定域建模。例如 SysML 中的需求即为一种元模型。（2）建立所确定的元模型与 GOPRR 六种元素之间的映射关系，即确定使用 GOPRR 中的哪种元元模型来表示所确定的元模型。例如可以使用对象元元模型进行实例化来表示 SysML 中的需求图元模型。（3）基于 GOPRR 六种元元模型构建元模型。添加领域建模语言元模型所包含的语法和语义，构建 GOPRR 表征的领域建模语言元模型，并构建这些元模型之间的关联约束。例如对象元元模型添加“需求”的语义、及其拥有的属性和其图形的具体形状，则变成一种格式良好的需求对象元模型。

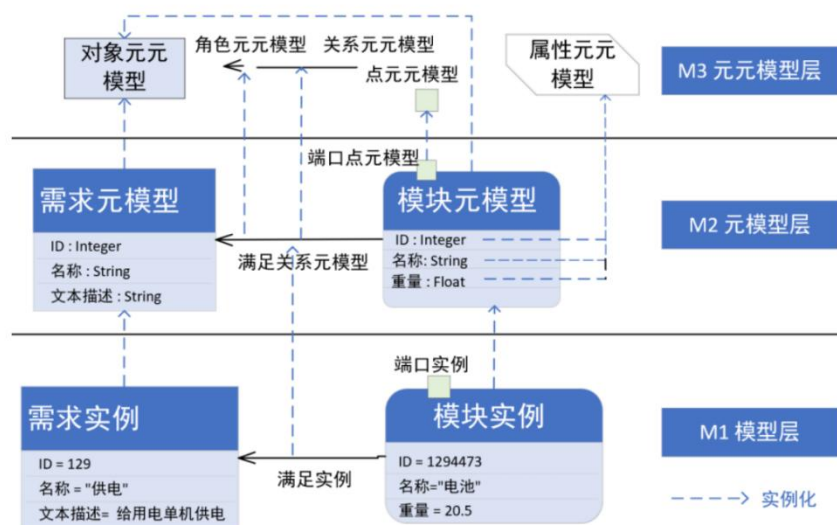


图 3.3 基于 GOPRR-E 元建模方法的模型构建流程

### 3.3.2 GOPRR-E 元模型与 OSLC 元数据模型映射方法

通过分析架构模型的模型库（Package）、模型（Model）、图（Graph），对象（Object），属性（Property），点（Point），关系（Relationship），角色（Role）等模型要素的涵义，根据 3.2.2 章节介绍的 OSLC 元数据模型，在概念上将 GOP

PRR-E 元模型抽象表达为 OSLC 元数据模型，提出 GOPRRR-E 元模型与 OSLC 元数据模型之间的映射方法，如图 3.4 所示。



图 3.4 GOPRRR-E 元模型与 OSLC 元数据模型映射方法

**规则 1:** KARMA 语言定义模型库 (Package) 映射为 OSLC 元数据模型中的服务提供方目录 (ServiceProviderCatalog)。

模型库和服务提供方目录都是用于组织和管理模型或服务的集合。它们都提供了一个入口点，用于访问和查找相关的模型或服务。

**规则 2:** KARMA 语言定义的模型 (Model) 映射为 OSLC 元数据模型中的服务提供方 (ServiceProvider)。

模型是描述特定领域的抽象表示，与服务提供方类似，都可以提供模型及其包含的 GOPRRR-E 元模型的查询、修改、新增和删除的相关服务。

**规则 3:** 对模型库 (Package)、模型 (Model) 概念以及图 (Graph)、对象 (Object)、点 (Point)、关系 (Relationship)、角色 (Role)、属性 (Property) 的查询、修改、新增和删除操作。

OSLC 服务是一些功能的集合，支持对资源的查询、修改、新增和删除，因此可以将对模型以及模型要素的查询、修改、新增和删除操作映射为服务。架构模型的查询、修改、新增和删除：它是指对架构模型文件 (.karma) 进行模型文件名称的查询与修改，例如查询 test.karma 文件的名称，将 test.karma 模型文件名称修改为 test2.karma；新增是指新增一个新的内容为空的架构模型文件，删除是指删除一个架构模型文件。

**规则 4：** GOPPRR-E 元模型中的图（Graph）映射为 OSLC 元数据模型中的资源（Resource）。

图是模型某一类视图，在模型中支持对图进行查询、修改、新增和删除操作。因此从大的颗粒度来看，图可以认为是服务可以操作的最大单元，即图是服务操作的对象，即资源。架构模型图（Graph）的查询、修改、新增和删除：查询是指对架构模型图（Graph）所包含的图本身的信息（例如图的名称）、图所包含的对象以及关系实例、图的拓扑结构进行查询；修改是指对上述查询到的图本身的信息、图中的对象及关系实例、图的拓扑结构进行修改；新增是指新增一个图元模型，并将该图元模型进行实例化，例如为图元模型命名实例化为图模型；删除是指在模型文件中删除整个图（Graph）的实例。

**规则 5：** GOPPRR-E 元模型中的对象（Object）映射为 OSLC 元数据模型中的资源（Resource）。

对象是图中特定的概念或实体，在图中支持对对象进行查询、修改、新增和删除操作。因此从细的颗粒度来看，对象可以认为是服务操作的对象，即资源。架构模型对象（Object）的查询、修改、新增和删除：它是指对架构模型对象（Object）及其所包含的点（Point，例如输入输出端口）、属性（Property，例如对象名称）进行查询与修改；新增是指在图中新增一个新的未经实例化的架构模型对象，删除是指删除图中已有的一个对象实例。

**规则 6：** GOPPRR-E 元模型中的点（Point）映射为 OSLC 元数据模型中的资源（Resource），两者都是图中的基本元素。

点是存在与对象上，是描述对象中与关系（或角色）连接的端口，在图中支持对点进行查询、修改、新增和删除操作。因此从细的颗粒度来看，点可以认为是服务操作的对象，即资源。架构模型点（Point）的查询、修改、新增和删除：查询是指查询图中对象所包含的点的属性，例如点的名称、点的输入与输出，点的形状、位置等；修改是指对上述查询到的点的信息进行修改，例如修改点的名称；新增是指在图中新增一个新的未经实例化的点元模型；删除是指删除图中已有的一个点实例。

**规则 7：** GOPPRR-E 元模型中的关系（Relationship）映射为 OSLC 元数据模

型中的资源（Resource），两者都是描述实体之间连接或相互作用的元素。

关系描述对象或点之间的连接或者相互作用，支持对关系进行查询、修改、新增和删除操作。因此从细的颗粒度来看，关系可以认为是服务操作的对象，即资源。架构模型关系（Relationship）的查询、修改、新增和删除：查询是指查询图中对象与对象之间已有的关联关系，包括关系的名称、关系传递的信息（例如传递力、扭矩），关系两端的角色等；修改是指修改上述查询到的关系的信息，例如修改关系的名称等；新增是指在图中现有的对象或端口之间新增关联关系，并为新增的关联关系赋值；删除是指删除图中现有的一条关联关系。

**规则 8：** GOPPRR-E 元模型中的角色（Role）映射为 OSLC 元数据模型中的资源（Resource），两者都是定义资源之间行为或属性的元素。

角色用于描述对象或对象上的点如何与关系相连接，在图中支持对角色进行查询、修改、新增和删除操作。因此从细的颗粒度来看，角色可以认为是服务操作的对象，即资源。架构模型角色（Role）的查询和修改：查询是指查询关系两端已有的角色信息，包括角色的名称、形状等；修改是指修改上述查询到的角色的信息，例如修改角色的名称等。

**规则 9：** GOPPRR-E 元模型中的属性（Property）映射为 OSLC 元数据模型中的资源（Resource），两者都是描述资源的特征或属性的元素。

属性是用于描述其他五个元元模型（图、对象、角色、关系和点）的特性。在图中支持对属性进行查询、修改、新增和删除操作。因此从细的颗粒度来看，属性可以认为是服务操作的对象，即资源。架构模型属性（Property）的查询、修改、新增和删除：查询是指查询模型中图、对象、点、关系、角色所包含的所有属性信息；修改是指对上述查询到的属性信息进行修改，包括修改属性的名称以及属性的值等；新增是指为模型中图、对象、点、关系、角色新增一条属性，并为属性赋值；删除是指删除模型中现有的一条属性信息。

基于上述定义的 GOPPRR-E 元模型与 OSLC 元数据模型的映射规则，为每一个 OSLC 资源生成一个统一资源标识（Uniform Resource Identifier, URI），映射规则与标识符的生成框架整理成表 3.2 如下：

表 3.2 架构模型映射规则及统一资源标识

OSLC 元模型 概念	架构模型概念	OSLC 资源统一标 识	标识输入参 数	资源标识含义
服务提供方目录 (Service Provider Catalog)	架构模型库 (Package)	http://localhost:909 2/catalogs/model	——	某指定的架构模型 库，一般为服务器下 的架构模型文件夹
服务提供方 (Service Provider)	架构模型 (Model)	http://localhost:909 2/providers/model/ {karmaFile}	架构模型文 件名称	某架构模型可以提 供所包含的所有模 型要素服务
	图 (Graph)	http://localhost:909 2/providers/model/ {karmaFile}/model Content	架构模型文 件名称	某个架构模型的拓 扑图
	对象 (Object)	http://localhost:909 2/providers/model/ {karmaFile}/model Object/{objectId}	架构模型文 件名称、对象 名称	某架构模型包含的 某个对象实例
	点 (Point)	http://localhost:909 2/providers/model/ {karmaFile}/model Point/{pointId}	架构模型文 件名称、点名 称	某架构模型包含的 某个点实例
资源 (Resource)	关系 (Relationship)	http://localhost:909 2/providers/model/ {karmaFile}/model Relationship/{relati onship}	架构模型文 件名称、关系 名称	某架构模型包含的 某个组件所包含的 某条关系实例
	角色 (Role)	http://localhost:909 2/providers/model/ {karmaFile}/model Role/{role}	架构模型文 件名称、角色 名称	某架构模型包含的 某个角色实例
	属性 (Property)	http://localhost:909 2/providers/model/ {karmaFile}/model Property/{property}	架构模型文 件名称、属性 名称	某架构模型包含的 某个属性实例

### 3.4 仿真模型与 OSLC 元数据模型的映射方法

#### 3.4.1 基于多架构建模语言的架构模型构建方法

Modelica 是一种面向对象、声明式的多领域建模语言，可用于基于组件的复杂系统建模，此复杂系统是指包含机械、电气、液压、热力、控制、电力或过程控制等相关领域子系统的多领域耦合系统<sup>[64]</sup>。Modelica 建模语言根据不同领域子系统自身的物理本质，对子系统的各个构件采用数学方程的形式进行统一表达，从而实现不同物理子系统间的多领域集成和信息传递，以满足不同应用场景的建模要求<sup>[65]</sup>。首先对 Modelica 语言的语法进行分析。

##### (1) Modelica 类

Modelica 类是构成 Modelica 模型的基本单元，其包含三种类型的成员：变量、方程和成员类<sup>[65]</sup>。变量表示类的属性，通常代表物理量，方程指定了类的行为，描述了变量之间的数值约束关系。类可以作为其他类的成员，并且可以通过继承从基类中获取成员。Modelica 语言有详细的语法规则，但是并没有对语法进行抽象表达。

因此，本文根据 Modelica 语法规则以及规定的 Modelica 类的类型，提出一种 Modelica 元模型的抽象方法。由于元模型的抽象不是唯一的，因此本文的定义方法只是其中之一。Modelica 元模型包括：模型库（Package），模型（Model），组件（Component/Block），连接器（Connector），连接线（Connection），参数（Parameter），修改语句（Modification）等，如图 3.5 所示。操作接口包括：模型及各模型要素的增（Create），删（Delete），改（Update），查（Retrieve）等操作。

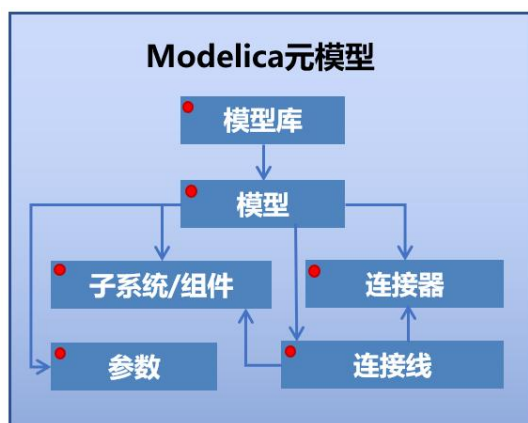


图 3.5 Modelica 元模型

- **模型库 (Package)**: 模型库是用于组织和管理模型的一种结构。它类似于文件系统中的文件夹, 用于将相关的模型组织在一起, 以便更好地管理和组织模型。
- **模型 (Model)**: 模型是描述系统的抽象, 包括组件、连接器以及连接关系。模型可以包含子系统模型, 完整且可以立即仿真。
- **子系统模型 (Subsystem)**: 子系统模型是模型中的一部分, 由组件或其他子系统模型组成。子系统模型有助于模块化系统, 并允许对系统进行层次化建模。
- **组件 (Component)**: 组件是模型中的基本构建块, 用于描述系统的功能和行为。组件可以是原子组件, 也可以是复合组件, 可以包含连接器、参数、方程等。
- **连接器 (Connector)**: 连接器描述了组件之间的连接方式, 包括变量的传递、能量流或信息传递。连接器定义了组件之间的接口, 使得不同组件可以相互连接并进行信息交换。
- **连接线 (Connection)**: 用于表示系统模型中子系统、组件连接器之间的连接关系。
- **参数 (Parameter)**: 参数是组件或模型中的变量, 用于描述模型的特性或初始值。参数通常在模型实例化时被指定, 可以在模拟过程中保持不变。
- **修改语句 (Modification)**: 修改语句用于在模拟过程中修改模型的属性或参数值。修改语句允许用户在模拟过程中对模型进行动态调整, 以适应不同的仿真场景或需求。

## (2) 组件连接机制

Modelica 语言的软件组件模型具有与硬件组件系统同等的灵活性和重用性。其核心概念包括组件、连接机制和组件构架。组件通过连接机制进行交互连接, 而组件构架则负责实现组件和连接, 以确保约束和通讯的稳定可靠性。

在 Modelica 语言中, 组件的接口被称为连接器, 而建立在连接器之上的耦合关系称为连接。这些连接可以分为因果连接和非因果连接, 具体取决于其表达的是因果关系还是非因果关系。模型类必须具有明确的接口, 即连接器, 以实现组件与外界的通讯。模型类应该定义为环境无关的, 这样才能实现组件的可重用性。这意味着在类定



义中只能包含方程，并且只能使用局部变量和连接器变量，而组件与外界通讯必须通过组件连接器进行<sup>[66]</sup>。连接器是连接器类的实例，其主要作用是定义组件接口的属性与结构。连接器类中定义的变量可以划分为两种类型：流变量和势变量。流变量是一种“通过”型变量，如电流、力、力矩等，由关键字 `flow` 限定。而势变量则是一种“跨越”型变量，如电压、位移、角度等。如图 3.6 所示，连接器类 `Pin` 定义了电路元件的接口，其中包含两个变量：电压 `v` 为势变量，电流 `i` 为流变量。这种明确定义的接口有助于组件的独立性和可重用性。

```
01. connector Pin
02. Real v;
03. flow Real i;
04. end Pin;
```

图 3.6 Modelica 连接器类 `Pin`

在研究了 `Modelica` 语言语法与组件连接机制之后，研究 `Modelica` 语言的面向对象、基于组件的建模，非因果、陈述式建模以及多领域建模的物理建模方式。

#### (1) 面向对象、基于组件的建模方式

`Modelica` 语言将面向对象的思想引入到建模过程中，实现了模型建立过程中的模块化分解。以类为基本单元，对系统各个独立功能模块进行数据封装，利用组件连接机制实现模块之间的信息交换，同时通过继承机制实现模型的复用<sup>[67]</sup>。

以使用 `Modelica` 语言构建一个简单的单摆系统为例，如图 3.7 所示，通过将重力场、转动副、阻尼、摆杆等基本机械元件封装成独立的模块，每个模块内部包含描述其物理特性的数学方程。基于 `Modelica` 内部的组件连接机制，将各个模块的接口连接，即可实现信息的相互传递。这种模块化的建模思想使得复杂系统的建模过程更加清晰和灵活。通过将系统分解为独立的模块，可以更好地理解系统的行为，并且可以在需要时轻松地替换或调整特定模块<sup>[68]</sup>，从而实现快速的模型迭代和验证。

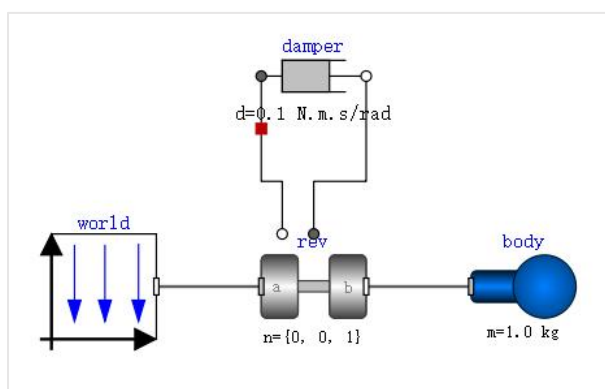


图 3.7 单摆系统的 Modelica 模型

## (2) 非因果、陈述式建模

Modelica 语言建立系统模型的过程是将物理系统抽象成一个类，其中系统的物理本质被转换成相应的数学模型，并利用 Modelica 语言对其进行描述，以表达系统的行为规律<sup>[69]</sup>。例如对于简单的单摆系统，Modelica 语言在描述阻尼时，依赖于数学方程而不是赋值语句或者过程式算法。这种建模方式聚焦于系统的物理规律，而不关注因果关系，从而大大降低了公式推导的复杂性和错误率。

```

1. model Damper //阻尼的 Modelica 模型
2.   parameter SI.RotationalDampingConstant d(final min=0, start=0);
3. equation
4.   tau = d*w_rel;
5.   lossPower = tau*w_rel;
6.   annotation ( ... );
7. end Damper;

```

## (3) 多领域建模

系统的多领域建模涉及对系统的物理本质规律进行辨识，并通过建模参数实现数学方程对系统物理基本规律的描述<sup>[70]</sup>。Modelica 语言的模型库包含各领域的基本元件，例如在数控机床的各个领域子系统中，功能部件是由领域内基本元件组合并扩展而来。Modelica 语言内在的数学描述一致性行为，使得其支持将不同领域的元器件模型融合到一个多领域系统模型中表示，不同领域模型之间的信息交互或能量传递通过能量转换器实现<sup>[71]</sup>。例如在一个涵盖机械与电气领域的耦合系统中，组件模型包含简单的电阻、电感等电气元器件，以及简单的机械元器件。其中电动势组件实现了机械元器件与电气元器件之间的信息流通，将电能转化为机械能。根据不同领域的物理规律，可

以通过定义不同领域之间的变换器实现多领域物理系统之间的信息交互，从而构建多领域耦合机制模型<sup>[72]</sup>。

### 3.4.2 Modelica 元模型与 OSLC 元数据模型映射方法

通过分析 Modelica 元模型，即模型库（Package）、模型（Model）概念以及子系统模型（Subsystem）、组件（Component）、连接器（Connector）、连接线（Connection）、参数（Parameter）、修改语句（Modification）的要素内涵，根据 3.2.2 章节介绍的 OSLC 元数据模型，在概念上将 Modelica 元模型抽象表达为 OSLC 元数据模型，提出 Modelica 元模型与 OSLC 元数据模型之间的映射方法，如图 3.8 所示。

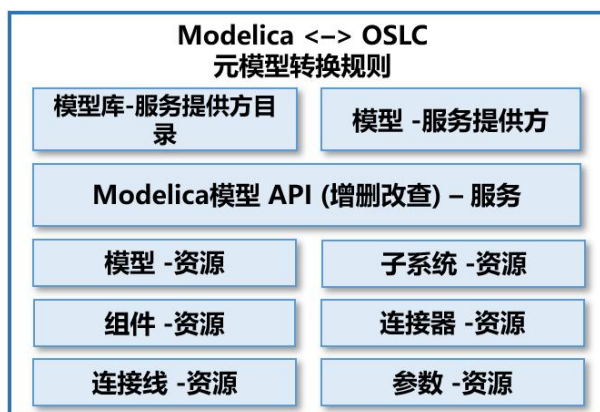


图 3.8 Modelica 元模型与 OSLC 元数据模型映射方法

**规则 1：**Modelica 元模型中的模型库（Package）映射为 OSLC 元数据模型中的服务提供方目录（ServiceProviderCatalog）。

模型库和服务提供方目录都是用于组织和管理模型或服务的集合。它们都提供了一个入口点，用于访问和查找相关的模型或服务。

**规则 2：**Modelica 元模型中的模型（Model）映射为 OSLC 元数据模型中的服务提供方（ServiceProvider）。

模型是描述多领域系统的抽象表示，与服务提供方类似，都可以提供模型及其包含的其它 Modelica 元模型实例的查询、修改、新增和删除的相关服务。

**规则 3：**对模型库（Package）、模型（Model）概念以及子系统模型（Subsystem）、组件（Component）、连接器（Connector）、连接线（Connection）、参数（Parameter）、修改语句（Modification）的查询、修改、新增和删除操作。

OSLC 服务是一些功能的集合，支持对资源的查询、修改、新增和删除，因此可以将对模型以及其它 Modelica 元模型要素的查询、修改、新增和删除操作映射为服务。

**规则 4：**Modelica 元模型中的模型（Model）概念映射为 OSLC 元数据模型中的资源（Resource）。

在 OSLC 中，资源是描述服务和信息的基本单位。从大的颗粒度来看，模型可以认为是服务可以操作的最大单元，即模型是服务操作的对象，即资源。仿真模型（Model）的查询、修改、新增和删除：查询是指对仿真模型（Model）所包含的本身的信息（例如模型的名称）、模型所包含的组件以及连接器、模型的拓扑结构进行查询；修改是指对上述查询到的模型本身的信息、模型中的组件及连接器、模型的拓扑结构进行修改；新增是指新增一个模型，并将该模型进行实例化，例如为模型命名；删除是指在模型文件中删除整个模型（Model）的实例。

**规则 5：**Modelica 元模型中的子系统模型（Subsystem）映射为 OSLC 元数据模型中的资源（Resource）。

子系统模型（Subsystem）在 Modelica 模型中是描述模型或者模型要素的基本元素。因此，将其映射为资源，能够使得其在 OSLC 中被统一管理和操作，同时体现了其基本性质和功能。

**规则 6：**Modelica 元模型中的组件（Component）映射为 OSLC 元数据模型中的资源（Resource）。

仿真模型组件（Component）的查询、修改、新增和删除：它是指对仿真模型组件（Component）及其所包含的连接器（Connector，例如输入输出端口）、参数（Parameter，例如弹簧组件包含的弹性系数  $k$ ）进行查询与修改；新增是指在图中新增一个新的未经实例化的仿真模型组件，删除是指删除模型中已有的一个组件实例。

**规则 7：**Modelica 元模型中的连接器（Connector）映射为 OSLC 元数据模型中的资源（Resource）。

仿真模型连接器（Connector）的查询、修改、新增和删除：查询是指查询模型中组件所包含的连接器的属性，例如连接器的名称、输入连接器的流变量与势变量，连接器的形状、位置等；修改是指对上述查询到的连接器的信息进行修改，例如修改连接器的名称；新增是指在图中新增一个新的未经实例化的连接器模型；删除是指删除

模型中已有的一个连接器实例。

**规则 8:** Modelica 元模型中的参数 (Parameter) 映射为 OSLC 元数据模型中的资源 (Resource)。

仿真模型参数 (Parameter) 的查询、修改、新增和删除：查询是指查询模型、组件所包含的参数名称及参数值；修改是指对上述查询到的参数值进行修改；新增是指为模型新增一个参数，并为其赋值；删除是指删除模型中现有的一个参数。

**规则 9:** Modelica 元模型中的子修改语句 (Modification) 映射为 OSLC 元数据模型中的资源 (Resource)。

仿真模型修改语句 (Modification) 的查询、修改、新增和删除：查询是指查询模型、组件所包含的修改语句；修改是指对上述查询到的修改语句进行修改；新增是指为模型新增一个修改语句，并为其赋值；删除是指删除模型中现有的一个修改语句。

基于上述定义的 Modelica 元模型与 OSLC 元数据模型的映射规则，为每一个 OSLC 资源生成一个统一标识，映射规则与标识符的生成框架整理成表 3.3 如下。

表 3.3 仿真模型映射规则及统一资源标识

OSLC 元模型概念	仿真模型概念	OSLC 统一资源标识	标识输入参数	资源标识含义
服务提供方 目录 (Service Provider Catalog)	仿真模型库 (Package)	http://localhost:8084/catalogs/modelicaLib	——	某指定的仿真模型库，一般为服务器下的 Modelica 模型文件夹
服务提供方 (Service Provider)	仿真模型 (Model)	http://localhost:8084/providers/modelica/{Modelica File}	仿真模型文件名称	某仿真模型可以提供所包含的所有模型要素服务
资源 (Resource)	仿真模型 (Model)	http://localhost:8084/providers/modelica/{Modelica File}/modelicaModel	仿真模型文件名称	某个仿真模型
	子系统模型 (Subsystem)	http://localhost:8084/providers/modelica/{Modelica File}/modelicaSubsystem /{Subsystem}	仿真模型文件名称、子系统名称	某仿真模型包含的某个子系统

表 3.3 仿真模型映射规则及统一资源标识（续表）

OSLC 元模型概念	仿真模型概念	OSLC 统一资源标识	标识输入参数	资源标识含义
资源 (Resource)	组件 (Component)	http://localhost:8084/providers/modelica/{ModelicaFile}/modelicaComponent/{Component}	仿真模型文件名称、组件名称	某仿真模型包含的某个组件
	连接器 (Connector)	http://localhost:8084/providers/modelica/{ModelicaFile}/modelicaConnector/{Component}.{Connector}	仿真模型文件名称、组件名称、连接器名称	某仿真模型包含的某个组件所包含的连接器名称
	连接线 (Connection)	http://localhost:8084/providers/modelica/{ModelicaFile}/modelicaConnection/{Connection}	仿真模型文件名称、连接线名称	某仿真模型包含的某个连接线
	参数 (Parameter)	http://localhost:8084/providers/modelica/{ModelicaFile}/modelicaParameter/Parameter	仿真模型文件名称、参数名称	某仿真模型包含的某个参数
	修改语句 (Modification)	http://localhost:8084/providers/modelica/{ModelicaFile}/modelicaModification/{Component}_{Modification}	仿真模型文件名称、组件名称、修改语句名称	某仿真模型包含的某个组件中的某个修改语句

### 3.5 基于服务的工具适配器构建方法

在使用 OSLC 元数据模型建立起对异构模型（包含表格）的映射规则、将异构模型进行一体化表达后，基于该映射规则开发适配器。该技术采用面向服务的体系结构以及基于 Web 的服务技术开发领域工具适配器，将异构模型转化成标准化的数据服务。通过在领域工具中调用数据服务的网端链接实现对于异构模型数据的动态查询与修改。

基于服务的工具适配器构建技术包括数据解析、服务语义生成、基于模板的可视化生成三部分。其中，数据解析是指使用编程手段，提取异构模型中有关的数据信息，

并转化为便于计算机可读的数据格式。服务语义生成是指在基于映射规则生成 OSLC 服务语义框架的基础上，将解析后的模型数据信息填充至语义框架中，生成完整的 OSLC 服务语义。基于模板的可视化生成是指在生成完整的 OSLC 服务语义的基础上，提取重要的模型信息填充至前端可视化的模板中，形成完整的 OSLC 模型数据服务，例如 OSLC 模型拓扑查询服务，以及模型属性值修改服务。

### （1）基于 ANTLR 的异构模型数据解析方法

研究采用基于 ANTLR (ANother Tool for Language Recognition, ANTLR) 的语法树解析方法，实现模型语义分析与数据生成。ANTLR (是一个根据输入自动生成语法树并可可视化的显示出来的开源语法分析器，用于读取、处理、执行和翻译结构化文本或二进制文件。它通过下列方式支持多领域模型的数据信息的读取、处理与转换：首先，根据特定域建模语言 (Domain specific Language, DSL) 的语法规则定义语法及词法解析文件.g4 文件，并通过该 g4 文件生成特定的语法分析器词法解析器与语法解析器。然后，根据这些词法解析器与语法解析器，将输入的非结构化的模型文本处理为可视化的抽象语法树；使用树分析器遍历抽象语法树节点，可获取到任意处的模型信息，例如可获取 Modelica 模型.mo 文件中模型名称、类名，继承，修改语句，标注，参数绑定方程，初始化方程，连接语句等信息；将获取到的模型数据信息存储为便于计算机理解与传输的数据格式，例如 JSON 格式。

### （2）OSLC 服务语义生成

OSLC 资源具有多种表现形式，常见的有 RDF、Turtle、JSON 等数据格式。由于 OSLC 规范建立在标准的 RDF 数据模型之上，因此基于 RDF 的资源序列化方式是实现资源一致性表达的基础。服务生成可以概括为：通过 RDF 序列化的方式，将 OSLC 资源统一表达为标准的资源—属性—属性值 / 资源的三元组描述框架，并使用 URI 进行统一标识，将提取到的资源信息映射生成符合 OSLC 规范的 RDF 语义信息。

在建立异构模型与 OSLC 核心模型的映射规则、以及异构模型的数据解析之后，将提取到的模型数据信息填充至 OSLC RDF 语义框架中，实现从模型语义到 OSLC 服务语义的映射。以 KARMA 图 (Graph) 模型的服务语义生成为例，根据映射规则，KARMA 图模型映射为 OSLC 核心概念中的资源 (Resource)，由于图 (Graph) 模型除包含图属性 (Property) 外，是对象 (Object) 及关系 (Relationship) 模型要素

的集合，因此提取图模型、图模型属性（例如图模型名称）以及图模型中包含的对象（Object）及关系（Relationship）等数据集信息，填充至由映射规则定义好的语义模板中，为该图模型之生成与之对应的 URI 以供全局标识。通过 RDF 的属性实现数据层面的链接，OSLC 可以进行服务发现，例如通过机械臂系统模型，通过其对应的 RDF 属性信息链接到控制器子系统模型，再通过子系统发现组件以及属性等。通过为每个模型组成元素与操作接口（指对模型组成元素的查询、创建、更新和删除）生成特定的统一资源标识符（URI），基于 HTTP 标准实现模型资源网端的操作，通过发送网络请求就可以访问到模型资源。

### （3）基于模板的模型可视化

在完成异构模型数据解析与 OSLC 服务语义生成后，研究基于模板的模型可视化。本节所介绍到的模型可视化，为模型拓扑图的可视化，图中包括该模型所包含要素（例如对象、关系等）、各要素属性（例如对象名称、对象属性、属性值）以及各要素之间的位置关系等信息，通过 OSLC 服务也可以进行各要素之间的关联查询。步骤包括：将所提取到的关于模型库、模型、模型要素的语义信息及其相应的位置信息存储为便于网端传输与解析、符合 OSLC 规范的 JSON 数据格式。以此数据为基础，将其填充为基于前端可视化框架的模板中，实现从 OSLC 数据语义到前端可视化框架所需的数据语义的映射。在上述映射完成之后，前端可视化框架将加载模型数据将模型要素及其位置关系等进行渲染重绘，从而实现模型的可视化生成。

## 3.6 本章小结

本章研究了基于服务的架构-仿真模型一体化表达方法。首先介绍了 OSLC 的基本概念和技术，阐述了其在促进模型在不同工具之间传递与数据互操作方面的作用。其次，研究了 OSLC 元数据模型的含义。然后研究多架构建模语言 KARMA 的模型构建方法，提出了 KARMA 架构模型与 OSLC 元数据模型之间的映射关系。同理，研究多领域仿真建模语言 Modelica 的建模方法，提出了 Modelica 仿真模型与 OSLC 元数据之间的映射方法。在语义层实现了架构模型与仿真模型的一体化表达，并在数据层，实现了架构模型与仿真模型数据格式的统一，并支持以标准的网络协议对架构、仿真模型及其要素进行演化活动即增、删、改、查等操作，实现了模型传递过程的工具集



成与数据互操作，并且为下文模型进行持续集成与传递，即实现模型传递的自动化特征提供统一的操作基础。

## 第 4 章 面向模型持续集成的自动化 workflow 设计方法

### 4.1 引言

在使用开放生命周期协作规范实现了 MBSE 模型传递的工具集成与互操作特征的基础上,针对 MBSE 模型传递具有的第二个特征,自动化能力特征的实现技术进行研究。**面向 MBSE 的持续集成**是指:以由需求、架构、仿真以及其它分析工具组成的工具链牵引形成的 MBSE 工作流为集成对象,对 MBSE 工作流中所包含的与模型有关的活动(例如模型及模型要素的演化,模型的检查、转换、仿真等活动)进行频繁、自动化的集成,从而支持模型在系统全生命周期中的动态闭环传递。

面向 MBSE 领域的持续集成包括以下内容:

(1) 模型驱动技术的集成:将模型驱动的技术例如代码生成、基于仿真的验证与确认、模型检查等手段进行集成,以支持数字孪生体<sup>[73]</sup>的实施。通过模型更新可以自动触发代码生成,通过代码生成技术可以生成部署在目标系统的源代码,或生成用于虚拟仿真的模型。仿真用于对 CPS 系统设计早期进行验证至关重要,协同仿真通过标准如功能模型接口(FMI<sup>[74]</sup>),为互连模型和相关仿真环境提供标准化接口,支持将一系列可执行的模型进行联合仿真。除此之外,对使用特定的建模语言来支持显式地定义持续集成/持续部署流水线提出了要求。

(2) 异构资源的集成:即跨阶段、跨工具、跨环境的不同资源之间数据的集成与互操作,例如系统架构模型 SysML 模型,与支持嵌入式软件和硬件建模的 AADL 模型,与部署至嵌入式系统中的 C 代码或多领域仿真模型 Modelica 模型之间的数据集成。

本章对面向 MBSE 模型持续集成的自动化 workflow 设计方法展开研究,分析 workflow 中各活动的模型传递的输入与输出,建立 workflow 与 workflow 执行脚本又称为流水线脚本之间的映射关系,定义 workflow 各活动与执行逻辑的语义模板,并基于第三章的 OSLC 技术来统一执行各活动,为模型进行跨阶段、跨工具的自动化传递与集成提供了解决方案。

## 4.2 架构与仿真过程中的 workflow 活动分析

首先分析由架构、仿真工具组成的工具链牵引形成的 MBSE 工作流的活动组成，然后明确执行各活动所需的 OSLC 服务技术，同时从时间与空间维度分析各活动节点上实现的模型传递。根据 MBSE 模型持续集成的定义，将 MBSE 工作流中包含的架构工具与仿真工具活动分为两类。一类是模型及模型元素演化活动，这类活动是对模型及模型要素进行增删改查的数据操作，强调模型在同一种表达形式上的变更与传递；另一类是模型检查、转换与仿真活动，即一类模型驱动技术的工具层面的活动，强调模型在不同种表达形式上的变更。下面分别对这两类活动进行分析。

### 4.2.1 模型及模型元素演化活动分析

对于模型及模型元素演化活动，根据 OSLC 服务能够提供的模型操作功能，这类活动又分为两小类，分别为查询活动与修改活动。其中，架构工具中，模型及模型元素演化的活动是指对架构模型图、对象、点、关系、属性的查询与修改活动，如 3.3.2 节所介绍的；仿真工具中，模型及模型要素的演化活动是指对仿真模型、组件、子系统、连接器、连接线、参数等的查询与修改活动，如 3.4.2 节所介绍。

首先，1) 对查询活动与修改活动的实现方式进行分析：针对于同一个待查询或修改的模型或模型元素，查询与修改活动的对象的 OSLC 服务标识是相同的，只是对 OSLC 服务标识的请求方法不同。

其次，2) 对查询与修改活动节点模型的传递进行分析：对于**查询活动节点**，**模型传递的输入**是原始版本的架构与仿真模型或模型元素，**模型传递的输出**是未经更改的架构与仿真模型或模型元素，设计信息在其中未发生变化，只用于表达和呈现系统的结构和行为状态；对于**修改活动节点**，**模型传递的输入**是原始版本的架构与仿真模型或模型元素，**模型传递的输出**是经过更改的新版本的架构与仿真模型或模型元素。在这其中，设计信息的变更发生在同一模型对象上，在**时间维度**上实现了以模型为载体进行传递的过程，如表 4.1 所示。

表 4.1 架构及仿真工具活动 OSLC 服务标识及模型传递

模型活动分类	模型活动分类	模型活动分类	执行活动的 OSLC 统一资源标识	模型传递输入	模型传递输出
模型及模型元素演化活动	查询	查询架构模型及模型元素	见表 3.2 架构模型资源的 OSLC 统一资源标识	架构模型	未变化的架构模型
		查询仿真模型及模型元素	见表 3.3 仿真模型资源的 OSLC 统一资源标识	仿真模型	未变化的仿真模型
	修改	修改架构模型及模型元素	见表 3.2 架构模型资源的 OSLC 统一资源标识	架构模型	变更后的架构模型
		修改仿真模型及模型元素	见表 3.3 仿真模型资源的 OSLC 统一资源标识	仿真模型	变更后的仿真模型

#### 4.2.2 模型检查、转换与仿真活动分析

第二类是模型检查、转换与仿真活动。这类活动主要基于模型驱动的技术来实现，具体分为以下三类：模型检查、模型转换、模型仿真。分别从架构模型与仿真模型工具活动进行分析。

##### (1) 架构模型检查、转换和验证活动

① 架构模型静态模型检查：用于检查架构模型是否符合语法规则，例如关键字、标识符、注释、表达式、语句等。将模型中的概念与（真实）系统的给定设计规范进行比较，其内涵包括模型中包含正确的元素及元素之间的正确关系、模型不违反建模规则和惯例、模型正确地描述了所建模的领域、模型内部以及模型之间不存在矛盾。

② 架构模型代码生成：是指基于代码生成转换脚本，将源模型转换生成目标代码或者文本的过程。在具体开发过程中，工程师依据各自领域语言建立领域模型，应用已定制好的代码生成脚本，代码生成引擎将自动转换为目标代码。以智能驾驶系统的内部模块图为例，代码生成器将读取该模型，并按照规定生成 Modelica 模型格式文件，以支持后续仿真验证。

根据上述活动，基于开放生命周期协作规范拓展架构工具适配器的功能，将架构模型检查、架构模型到仿真模型代码生成转化为通过发送网络请求可执行的服务。目前适配器已经实现架构模型到仿真模型代码生成的操作转化为工具适配器服务，其对应的统一资源标识为：<http://localhost:9092/providers/model/{架构模型文件名称}/modelContent/codeGenerate>。例如对于机电传动系统的架构模型 VehicleDriveCycle.karma 执

行架构模型转化为仿真模型操作对应的适配器服务为：<http://localhost:9092/providers/model/VehicleDriveCycle.karma/modelContent/codeGenerate>。

## （2）仿真工具中关于仿真模型分析、转换或验证的工具活动

① 仿真模型检查：仿真模型检查包括语法检查、类型检查、约束检查。1) 语法检查：检查模型是否符合仿真语言的语法规则，例如关键字、标识符、注释、表达式、语句等。2) 语义检查：验证仿真模型的结构和语义是否正确，例如检查模型中的方程、变量和组件是否按照正确的语义规则进行定义和连接。3) 类型检查：检查仿真模型中的变量、方程、函数等是否具有正确的数据类型，例如验证模型中的变量和方程的数据类型是否匹配。4) 约束检查：检查仿真模型中的方程是否满足一定的约束条件，例如平衡方程数和未知数，保证方程组有唯一解。

② 仿真执行：仿真包括参数设置、仿真执行两个步骤。其中参数设置包括设置仿真初始条件例如开始结束时间、仿真步长、仿真算法等，仿真执行是指使用仿真工具执行模型，通过数值方法求解模型的微分方程或代数方程，对虚拟系统的动态行为进行模拟、分析和展示。

③ 仿真模型代码生成：是指基于代码生成转换脚本，将仿真模型转换生成目标代码或者文本的过程。在具体开发过程中，工程师使用仿真建模工具中内置的代码生成引擎，可以将 Modelica 模型的文本自动转换为目标代码，例如符合 FMI 规范的 XML 格式的文本，或 C 代码，以支持后续的联合仿真或将控制程序部署至硬件环境中。

然后，根据上述活动，基于开放生命周期协作规范拓展仿真工具适配器的功能，将仿真模型检查、仿真模型仿真转化为可执行的服务，并为其生成统一的资源标识，如对机电传动系统的架构模型 VehicleDriveCycle.mo 执行仿真模型检查对应的适配器服务为：<http://localhost:8084/providers/VehicleDriveCycle.mo/modelCheck>。

在上述对这类活动的实现方式进行总结为：通过对 OSLC 核心规范进行拓展，并对工具适配器的功能进行拓展，将模型分析、转换与仿真活动的工具操作转化为标准的 OSLC 服务，并为其生成统一的资源标识。

然后，对各个节点的模型传递进行分析。其中模型检查活动又包括架构模型的模型检查和仿真模型的模型检查，该节点**模型传递的输入**是架构模型与仿真模型，**模型传递的输入**是语义与语法检查的数据结果；模型转换中包括架构模型通过代码生成的

方式转换生成仿真模型的源码，该节点**模型传递的输入**是架构模型与代码生成执行脚本，**模型传递的输入**是生成的仿真模型的源代码；模型仿真主要是针对于仿真模型而言，通过对模型进行仿真获取系统虚拟运行下的结果，该节点**模型传递的输入**是仿真模型及对应的仿真配置，**模型传递的输入**表现为图形、表格或其它格式的代码。在这其中，设计信息的变更发生在**不同模型和数据对象**上，在**空间维度**上实现了以模型为载体进行传递的过程，如表 4.2 所示。

表 4.2 模型工具活动 OSLC 服务标识及模型传递

模型活动 分类	模型活动 分类	模型活动 分类	执行活动的 OSLC 服务标识	模型传递 输入	模型传递 输出
模型分析、 转换、仿真 活动	模型检查	仿真模型 检查	http://localhost:8084/providers/modelica/{仿真模型文件名称}/modelicaModel/modelCheck	仿真模型	检查数据
	模型转换	架构模型 代码生成 仿真模型	http://localhost:9092/providers/model/{架构模型文件名称}/modelContent/codeGenerate	架构模型	仿真模型
	模型仿真	仿真模型 执行仿真	http://localhost:8084/providers/modelica/{仿真模型文件名称}/modelicaModel/{仿真参数}	仿真模型	仿真图形、表格、.mat 格式

### 4.3 workflow活动与流水线脚本元素映射方法

本文的流水线（Pipeline）是指自动化集成平台 Jenkins 中的流水线概念，它是自动化集成平台中对 workflow 的一种形式化表达与实现。流水线（Pipeline）脚本是一种形式化的文本文档，它通过预定义的语法和语义规则，详细描述了持续集成 workflow，包括 workflow 包含的各活动、执行逻辑以及执行条件等。通过自动化集成平台 Jenkins，集成已规定好的流水线脚本，能够实现 workflow 的自动执行。本节通过分析 workflow 的形式化定义脚本即流水线脚本的组成元素，建立流水线脚本元素与 workflow 活动之间的映射关系，为下文构建 workflow 各活动的语义模板提供理论基础。

### 4.3.1 流水线脚本定义及组成元素分析

在工作流的自动化执行中，流水线脚本是至关重要的组成部分，它通过预定义的语法和语义规则，详细描述了持续集成工作流，包括工作流包含的各活动、执行逻辑以及执行条件等。流水线脚本通常由多个元素组成，包括阶段（Stages）、步骤（Steps）、脚本（Scripts）和参数化配置（Parameterization），这些元素共同定义了工作流过程。以下是对流水线脚本元素的分析：

首先，基于自动化集成平台构建工作流项目，需要进行项目配置。项目配置包含以下元素：

（1）源（Source）：作为流水线的输入，一般为一个在 Git 版本管理下的仓库（当仓库中文件发生变更后，可自动触发流水线构建，实现一系列自动化的操作）；

（2）参数（Parameterization）：允许用户在执行流水线时提供不同的参数，以满足不同场景的需求；

其次，对流水线脚本元素进行分析。

（1）流水线（Pipeline）：流水线是整个流程的顶层结构，在流水线中定义了一系列阶段，每个阶段包含一组相关的活动。流水线的声明通常位于脚本的顶部，并指定了流水线的名称和其他全局属性。流水线（Pipeline）的语义框架如下：

```
1. pipeline{ //流水线脚本
2.     stages {
3.         stage('阶段一'){
4.             steps{ //阶段一中包含的步骤
5.                 script{ /*脚本*/ }
6.             }
7.         }
8.         stage('阶段二'){
9.             steps{
10.                script{ /*脚本*/ }
11.            }
12.        }
13.        ...
14.    }
15. }
```

(2) 阶段 (Stage): 每个阶段代表流水线中的一个主要阶段或任务, 可以包含一个或多个步骤。阶段的目的是将流水线的执行过程划分为逻辑上相关的部分, 便于管理和监控。阶段 (Stage) 的语义框架如下:

```
1. stage('阶段'){
2.     steps{
3.         //阶段中包含的步骤1 ...
4.         //阶段中包含的步骤n
5.     }
6. }
```

(3) 步骤 (Steps): 步骤是流水线中的最小执行单元, 代表一个具体的操作或任务。每个步骤都是流水线执行过程中的一个环节, 可以是命令、函数调用、脚本或插件的调用等。步骤 (Steps) 语义框架如下:

```
1. steps{
2.     //执行命令1
3.     //调用函数2
4.     //运行脚本3
5. }
```

(4) 脚本 (Script): 脚本是步骤执行操作或任务的实现方式, 脚本是步骤的具体执行内容, 可以是任何合法的脚本语言代码, 例如 Groovy、Shell、Python 等, 每个步骤通常由一个或多个脚本组成。脚本 (Script) 语义框架如下:

```
1. script{
2.     //执行脚本具体内容
3. }
```

#### 4.3.2 workflow活动与流水线元素映射方法

在 4.2 节架构与仿真过程中的 workflow 活动分析中, 将 MBSE workflow 活动分为了两类, 一类是模型及模型元素演化活动, 一类是模型分析、转换与仿真活动, 并对各个节点对象的 OSLC 技术服务以及模型传递进行了分析。构建流水线脚本文件, 对上述 workflow 的活动以及执行顺序进行形式化表达, 是实现自动化 workflow 的基础。

首先, 梳理 MBSE workflow 过程, 包括的 workflow 输入、各阶段涉及的工具活动、输出, 以及针对各活动之间模型/文件流的输入与输出以及实现该活动所需要的条件, 构建出 workflow 过程与项目配置及流水线脚本之间的映射关系:



**规则 1：**将工作流的输入映射为脚本项目配置中的源（Source）。

将工作流的输入数据或文件源与项目配置中的源关联，确保流水线能够获取到所需的输入数据。

**规则 2：**将工作流过程映射为脚本文件中的流水线（Pipeline）。

将工作流的整体流程以流水线脚本的形式定义在二进制文件中。

**规则 4：**将工作流各活动映射为脚本文件中的阶段（Stages）。

将工作流中的每个活动转化为流水线中的一个阶段。例如将仿真模型检查活动映射为脚本文件中的第一个阶段（Stage）；

**规则 5：**将实现各活动的过程映射为脚本文件中的步骤（Steps）。

将每个活动的执行过程分解为流水线中的一系列步骤。例如仿真模型检查活动包含两个步骤：①将仿真模型导入仿真建模工具，②仿真建模工具执行模型检查功能；

**规则 6：**将实现各步骤的方式映射为脚本文件中的脚本（Script）。

在流水线的每个步骤中，编写相应的脚本来实现具体的操作或任务。例如将实现仿真模型检查这一活动的两个步骤，通过调用仿真工具适配器的服务来实现，具体表现为将http://localhost:8089/.../VehicleDriveCycle.mo/modelCheck写入脚本文件中的Script部分；

**规则 7：**将实现各活动所需的条件映射为项目配置中的参数（Parameterization）：

将执行每个活动所需的参数或配置项作为流水线项目的参数，在执行时动态传递。例如将执行仿真模型检查这一活动所需的仿真模型文件路径作为参数项输入流水线项目。

将上述 workflow 组成与流水线脚本元素之间的映射规则整理成 0 如下：

表 4.3 工作流组成与流水线脚本元素的映射关系

工作流组成	流水线脚本语义模块	描述
工作流的输入	源（Source）	将工作流的输入数据或文件源与项目配置中的源关联
工作流过程	流水线（Pipeline）	将工作流的整体流程以流水线脚本的形式定义在二进制文件中
工作流各活动	阶段（Stages）	将工作流中的每个活动转化为流水线中的一个阶段

表 4.3 workflow组成与流水线脚本元素的映射关系（续表）

workflow组成	流水线脚本语义模块	描述
实现各活动的过程	步骤（Steps）	将每个活动的执行过程分解为流水线中的一系列步骤
实现各步骤的方式	脚本（Script）	在流水线的每个步骤中，编写相应的脚本来实现具体的操作或任务
实现各活动所需的条件	参数（Parameterization）	将执行每个活动所需的参数或配置项作为流水线项目的参数，在执行时动态传递

## 4.4 workflow的语义模板设计

根据基于映射关系与流水线的语法语义规则，从模型及模型元素演化活动、模型分析、转换或仿真活动这两类来定义workflow各活动与执行逻辑的语义模板，支持workflow执行脚本的可重用设计：通过在语义模板中采用第三章定义的OSLC服务来统一执行各活动，为模型持续workflow的执行脚本设计提供了可复用的框架，支持模型持续集成workflow的自动执行。

### 4.4.1 模型及模型元素演化活动的流水线语义模板定义

根据workflow与流水线脚本之间的映射规则，整个流水线脚本是workflow的形式化表达，流水线脚本中的不同阶段（Stage）对应workflow中包含的各活动，是workflow最重要的组成模块。由于架构工具与仿真工具驱动的MBSE workflow活动分为了两类，一是模型及模型元素演化活动，二是模型分析、转换与仿真活动。因此，在流水线脚本的顶层框架中，分别对这两类活动的形式化定义展开研究，构建这两类活动的语义模板，从而以模块化的方式，支持不同业务场景下workflow形式化的定义。

结合上文流水线脚本元素与workflow活动的映射规则，定义符合上述规则的模型及模型要素的流水线语义模板。根据OSLC核心规范所支持的OSLC服务技术实现，将对模型及模型要素的演化活动分为查询与修改活动两类。

#### （1）模型或模型元素的查询活动的语义模板定义

首先是模型或模型元素的查询活动的语义模板定义如下。在模型及模型要素查询活动的语义模板中，workflow中一个查询活动对应流水线中的一个阶段，例如，一个仿

真模型的参数查询活动，对应于流水线中的一个阶段；查询活动对应的阶段中，包含了一些实现查询活动的步骤，这些步骤是由脚本来实现的，脚本中包含两个部分，一是执行方法，二是执行逻辑。例如，针对仿真模型的参数查询活动，首先实现查询功能，其对应的执行方法为：针对仿真模型参数的OSLC资源统一标识，使用GET的请求方法发送HTTP请求，期望获得参数资源信息；然后判断执行逻辑：根据HTTP请求的响应状态，判断是否成功查询到参数资源，如果成功，则进入下一个执行阶段，若失败，则流水线终止执行。

在下列语义模板中，`${模型或模型要素名称}`与`${OSLC统一资源标识}`作为参数输入，是模板中支持自定义的部分，也是实现模板可重用的基础。对于查询活动而言，只需通过对OSLC统一资源标识使用GET的方式发送HTTP请求，即可获得对象的OSLC查询服务。因此查询活动只有查询的模型或模型要素及其对应的OSLC统一资源标识不同，活动的执行方法与执行逻辑都相同。因此，`${OSLC统一资源标识}`作为参数输入该查询活动的语义模板，它来源于表 3.2 与表 3.3 表格中定义的各架构与仿真模型资源对应的OSLC统一资源标识，即符合HTTP规范的URL。`${模型或模型要素名称}`参数则用于补全该阶段的名称。

```

1. stage('${模型或模型要素名称}查询') { //模型或模型要素名称占位符
2.     steps {
3.         script{
4.             def query = httpRequest(
5.                 httpMode: "GET", //查询活动的请求方式
6.                 url: '${OSLC 统一资源标识}', //资源统一标识占位符
7.             )
8.             def result = query.content
9.             if(result.status == 200){ //判断是否查询成功
10.                 println(result)
11.                 echo "查询成功" //成功则执行下个阶段
12.             }else{
13.                 echo "查询失败" //失败则流水线执行中断
14.                 exit 1
15.             }
16.         }
17.     }
18. }

```

## (2) 模型或模型元素修改活动的语义模板定义

根据OSLC服务规范，模型及模型元素的修改与查询活动相比，只是发送统一资源标识的请求方法不同，即由HTTP GET请求，换成HTTP PUT请求，因此语义模板中的脚本在执行方法和执行逻辑上发生的变化如下：

```

1.  stage('${模型或模型要素名称}修改') { // 模型或模型要素名称占位符
2.      steps {
3.          script{
4.              def update = httpRequest(
5.                  httpMode: "PUT", // 修改活动的请求方式
6.                  url: '${OSLC 统一资源标识}', // 资源统一标识占位符
7.                  customHeaders: [ /*修改内容*/ ]
8.              )
9.              def result = update.content
10.             if(result.status == 200){ // 判断是否修改成功
11.                 println(result)
12.                 echo "修改成功" // 成功则执行下个阶段
13.             }else{
14.                 echo "修改失败" // 失败则流水线执行中断
15.                 exit 1
16.             }
17.         }
18.     }
19. }

```

针对修改活动的语义模板中定义的活动执行方法与执行逻辑举例如下。例如，针对仿真模型的参数修改活动，首先实现修改功能，其对应的执行方法为：针对仿真模型参数的OSLC资源统一标识，使用PUT的请求方法发送HTTP请求并带有修改后的内容，期望更新参数资源信息；然后判断执行逻辑：根据HTTP请求的响应状态，判断是否成功修改了参数资源，如果成功，则进入下一个执行阶段，若失败，则流水线终止执行。

修改活动语义模板中的\${OSLC统一资源标识}参数，同样来源于表 3.2 与表 3.3 中定义的各架构与仿真模型及模型元素对应的OSLC统一资源标识，是执行活动的数据操作基础。

#### 4.4.2 模型分析、转换或仿真活动的脚本定义

在实现了模型及模型元素演化活动的语义模板定义后，针对MBSE工作流中包含的第二类活动，即模型分析、转换与仿真活动，结合现有架构与仿真工具适配器能够实现的OSLC服务，分别对第二类活动下的架构模型代码生成、仿真模型检查与仿真模型执行仿真这三个活动进行语义模板定义。

##### (1) 架构模型代码生成活动的语义模板定义

针对架构模型代码生成活动，从活动执行方法与执行逻辑上进行分析。首先实现架构模型代码生成的功能，其对应的执行方法为：针对实现架构模型代码生成功能的OSLC资源统一标识，使用GET的请求方法发送HTTP请求，期望将架构模型转换成其它格式的模型或代码；然后判断执行逻辑：根据HTTP请求的响应状态，判断是否成功执行，如果成功，则打印出转换生成的代码，并进入下一个执行阶段，若失败，则流水线终止执行。

语义模板中的第5行{架构模型名称}作为重要的参数输入，与模板中预先内置的部分OSLC统一资源标识即URL，形成一个完整的活动执行链接，该URL的定义来源于表4.2 架构模型代码生成活动的OSLC服务标识。

```

1. stage('${架构模型名称}代码生成') { //模型名称占位符
2.     steps {
3.         script{
4.             def codeGenerate = httpRequest(
5.                 url:'http://localhost:9092/providers/model/${架构模型名
6.                 称}/modelContent/codeGenerate') //架构模型名称占位符
7.             println(codeGenerate.status)
8.             def result = codeGenerate.content
9.             if(result.contains("completed successfully")){
10.                 println(result)
11.                 echo "架构模型代码生成成功"
12.             }else{
13.                 echo "构模型代码生成失败"
14.                 exit 1
15.             }
16.         }
17.     }

```

## (2) 仿真模型检查活动的语义模板定义

针对于仿真模型检查的语义模板如下。针对仿真模型检查活动，从活动执行方法与执行逻辑上进行分析。首先实现仿真模型检查的功能，其对应的执行方法为：针对实现仿真模型检查功能的OSLC资源统一标识，使用GET的请求方法发送HTTP请求，期望将仿真模型进行语法及语义上的检查；然后判断执行逻辑：根据HTTP请求的返回结果，判断仿真模型是否符合语法规则，如果成功，则进入下一个执行阶段，若失败，则流水线终止执行。

语义模板中的第5行{仿真模型名称}作为重要的参数输入，与模板中预先内置的部分OSLC统一资源标识即URL形成一个完整的活动执行链接，该URL的定义同样来源于表4.2 仿真模型检查活动的OSLC服务标识。

```

1. stage('${仿真模型名称}模型检查') { //模型或模型要素名称占位符
2.     steps {
3.         script{
4.             def check = httpRequest(
5.                 url:'http://localhost:8084/providers/modelica/${仿真模
   型名称}/modelicaModel/modelCheck')
6.             println(check.status)
7.             println(result)
8.             def result = check.content
9.             if(result.contains("completed successfully")){
10.                 echo "模型符合语法规则"
11.             }else{
12.                 echo "模型不符合语法规则"
13.                 exit 1
14.             }
15.         }
16.     }
17. }
```

## (3) 仿真模型执行仿真活动的语义模板定义

针对于仿真模型执行的语义模板如下。针对仿真模型执行活动，从活动执行方法与执行逻辑上进行分析。执行方法：针对实现仿真模型执行功能的OSLC资源统一标识，使用GET的请求方法发送HTTP请求，期望将仿真模型执行仿真；然后判断执行逻辑：根据HTTP请求的返回结果，判断仿真模型是否符合语法规则，如果成功，

则进入下一个执行阶段，若失败，则流水线终止执行。

语义模板中的第 5 行{仿真模型名称}与{仿真参数配置}作为重要的参数输入。其中，仿真参数配置包括：设置仿真初始条件（例如开始结束时间、仿真步长、仿真算法等）、设置仿真结果输出格式、选择参数进行绘图等。它与模板中预先内置的部分 OSLC 统一资源标识即 URL 形成一个完整的活动执行链接，该 URL 的定义同样来源于表 4.2 仿真模型执行仿真活动的 OSLC 服务标识。

```

1. stage('${仿真模型名称}仿真') {
2.     steps {
3.         script{
4.             def simulate = httpRequest(
5.                 url:'http://localhost:8084/providers/modelica/${仿真模
   型名称}/modelicaModel/simulate/${仿真参数配置}')
6.             def result = simulate.content
7.             println(simulate.status)
8.             println(simulate)
9.             if(result == 'true'){
10.                 echo "模型仿真执行成功"
11.             }else{
12.                 echo "模型仿真执行失败"
13.                 exit 1
14.             }
15.         }
16.     }
17. }
```

上述步骤根据工作流与流水线脚本的映射规则，基于流水线脚本的语法语义，针对 MBSE 工作流中模型及模型元素演化活动（例如查询与修改）以及模型分析、转换与仿真活动，进行了流水线语义模板的定义，模板从活动的执行方法与执行逻辑上进行分析，设定了模板的可变部分与不变部分，其中可变部分即模型的参数输入部分，其中包括支持活动执行的 OSLC 统一资源标识，不变部分即执行逻辑。基于上述定义的模板，可以支持特定场景 MBSE 工作流的模块化定义。

在完成上述工作流的语义模板的设计后，工作流的自动执行是通过自动化集成平台 Jenkins，集成已规定好的流水线脚本。Jenkins 是一种开源的持续集成和持续交付的软件，可以实现对软件项目的自动化构建、测试和部署等操作，支持多种编程语言、

开发框架和部署环境。执行过程需要利用Jenkins提供的功能或者插件，根据脚本文件中定义的工作流节点信息和顺序，通过HTTP请求或者其他方式调用相应的适配器服务，并获取服务返回的结果。通过Jenkins集成已规定好的流水线脚本，可以自动按顺序调用OSLC服务统一执行工作流，实现模型持续集成工作流的自动执行。

## 4.5 本章小结

本章研究了面向模型持续集成的自动化工作流设计方法。首先，分析架构与仿真过程中工作流所包含的活动分析，以及各活动节点的模型传递，并将各活动转化成了标准的OSLC服务，为模型自动化传递提供操作基础。然后，分析了流水线脚本的定义与组成元素，建立起了工作流活动与流水线脚本元素之间的映射关系。其次，设计了工作流的流水线语义模板，用于定义工作流各活动与执行逻辑，并基于OSLC技术来统一执行工作流，为模型持续工作流的执行脚本设计提供了可复用的框架，实现了模型持续集成工作流的自动执行，验证了模型传递的自动化特征。



## 第 5 章 面向航空发动机系统的 MBSE 模型传递案例验证

### 5.1 引言

本节以航空发动机系统为对象，根据前文所提出的面向 MBSE 的模型传递框架，验证基于服务的架构-仿真模型一体化表达方法、模型持续集成 workflow 设计方法的有效性。首先介绍航空发动机的设计背景，然后基于 RFLP 建模方法论构建航空发动机的系统架构模型，基于 Modelica 建模语言构建航空发动机物理仿真模型。（1）针对构建的航空发动机系统架构模型与仿真模型，研究架构模型与仿真模型在工具级的集成与互操作，对基于服务的架构-仿真模型一体化表达方法进行验证；（2）选择航空发动机系统开发过程的具体业务场景，构建航空发动机设计的持续集成 workflow 模型，并基于 OSLC 技术执行 workflow，以验证模型持续集成 workflow 设计方法的有效性。

### 5.2 航空发动机系统设计背景

航空发动机作为飞机动力的直接来源，涉及到机、电、液、控制等多个学科与多个领域，是一个典型的复杂系统。随着航空发动机系统的复杂性的增加，系统研发过程的复杂性也随之增加，系统的研发模式也逐渐由基于文档的系统工程转向基于模型的系统工程。在使用 MBSE 方法完成航空发动机的系统架构设计之后，需要通过仿真模型或原型样机等进行测试和验证。使用系统仿真或测试原型样机对系统架构模型进行验证，可以在投入实际生产之前发现系统设计存在的问题，从而缩短设计周期，加快设计效率，减少设计成本。然而，在使用 MBSE 方法进行航空发动机系统设计过程中依然存在以下问题：（1）在系统设计阶段构建的架构模型，与仿真验证阶段构建的仿真模型之间语法与语义不一致，导致设计信息（例如设计参数）在模型之间传递存在困难，难以实现端到端的有效传递与集成；（2）在 MBSE 设计阶段与仿真验证阶段，存在模型转换、参数配置等大量手动操作，手动操作在无形中提高了设计过程的错误率，严重降低了设计与验证的信息一致性。

## 5.3 面向航空发动机系统架构设计与仿真验证过程的模型传递

### 5.3.1 航空发动机系统架构建模与仿真建模

在验证 MBSE 架构模型与仿真模型传递技术框架,验证航空发动机系统架构模型与仿真模型一体化表达方法与模型持续集成 workflow 设计方法之前,针对航空发动机系统的特点,首先基于需求-功能-逻辑-物理 (Requirement-Function-Logic-Physics, RFLP) 建模方法论与多架构建模语言 KARMA 对航空发动机系统进行架构建模;然后根据 RFLP 设计阶段的第四阶段物理架构建模阶段的模型,例如航空发动机系统的内部模块图与参数图,并考虑到航空发动机系统多领域、多学科耦合的特点,采用多领域建模语言 Modelica 与仿真建模语言 Simulink 构建出航空发动机的物理仿真模型,为后续进行架构模型与仿真模型一体化集成与设计过程持续集成提供模型基础。

#### 5.3.1.1 基于 RFLP 方法的航空发动机系统架构建模

针对航空发动机系统的特点,采用 RFLP 建模方法论以及 KARMA 多架构建模语言,以航空发动机系统为研究对象,分别在需求分析阶段、功能分析阶段、逻辑设计阶段以及物理架构设计阶段构建相应的模型。由于本案例设计过程主要目的在于通过 RFLP 正向设计方法从需求得到航空发动机系统的物理架构,从而分析物理架构模型与仿真模型之间的设计信息传递,因此在完成 RFLP 中的需求分析、功能分析、逻辑设计阶段后,重点研究航空发动机的物理架构建模阶段。其中,物理架构建模阶段是指构建物系统的物理架构模型,表达系统的物理组成与系统内外部的信息交互。

根据逻辑设计阶段已完成的系统功能组成与各组成之间的活动交互,将航空发动机系统分为结构系统、控制系统、供电系统以及 PHM 系统四部分。以结构系统的物理架构设计为例,采用内部模块图表示结构系统的具体组成以及信息传递路径。

首先对航空发动机结构系统的工作原理进行分析。结构系统的具体组成如下图所示。

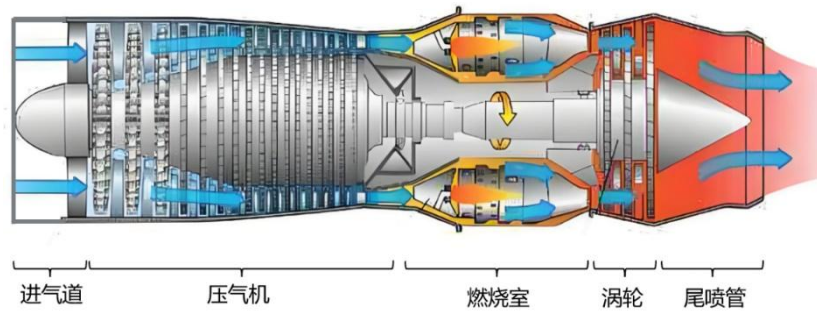


图 5.1 航空发动机组成结构

(1) 首先，航空发动机从进气道吸入空气。由于飞行速度是变化的，而压气机对进气速度有严格要求，因此进气道用于将进气速度控制在合适的范围内。

(2) 吸入的空气进入压气机。压气机用于提高吸入空气的压力，主要为扇叶式，通过转动叶片对气流做功，从而使气流的压力与温度升高。

(3) 高温高压气流进入燃烧室。燃烧室的燃油喷嘴射出油料，与空气混合后点火，产生高温高压燃气，向后排出。

(4) 高温高压燃气向后流过高压涡轮与动力涡轮，部分内能在涡轮中膨胀转化为机械能，一部分用于驱动涡轮旋转。由于高压涡轮、动力涡轮与压气机装在同一根转轴上，因此由内能转化的另一部分机械能也用于驱动压气机旋转，从而反复压缩吸入的空气。

(5) 从高温涡轮中流出的高温高压燃气，在尾喷管中继续膨胀，以高速从尾部喷口向后排出。这一速度比气流进入发动机的速度大得多，从而产生了对发动机的反作用推力，驱使飞机向前飞行。

根据上述发动机的工作过程，将发动机结构系统的组成（进气道、压气机、燃烧室、高压涡轮、动力涡轮及尾喷管），以及各组成之间的物质、能量交互关系（空气、高温高压空气、高温高压燃气、机械能）用内部模块图进行表达，构建结构系统的物理架构模型，如图 5.2 所示。

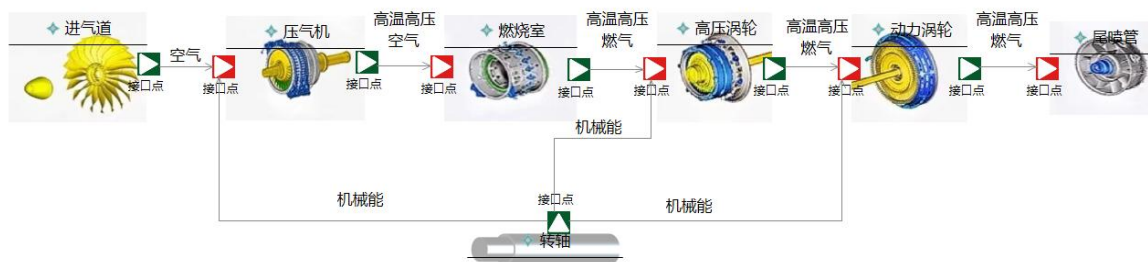


图 5.2 航空发动机结构系统内部模块图

其中，以压气机对象的属性参数设计为例。压气机的主要性能参数有转速、流量、空气流量、增压比和效率等。其中，增压比是指压气机出口气流的压强与其进口气流的压强之比。增压比的高低，在设计时根据发动机的需要来选定，它是影响涡轮喷气发动机工作性能的一个重要的循环参数，对发动机的单位推力和耗油率有较大的影响。根据需求航空发动机的设计需求，在内部模块图中，初步设定压气机对象的增压比属性的值为 20。构建结构系统的内部模块图使用到的 KARMA 模型实例如表 5.1 所示。

表 5.1 架构模型元模型实例汇总

架构模型实例	模型实例	模型名称
航空发动机结构 系统内部模块图	图（Graph）元模型实例	1
	对象（Object）元模型实例	7
	点（Point）元模型实例	11
	关系（Relationship）元模型实例	8
	角色（Role）元模型实例	15
	属性（Property）元模型实例	15

### 5.3.1.2 基于 Modelica 语言构建航空发动机仿真模型







根据上述航空发动机的物理结构组成以及各组成之间的物质、能量交互关系，使用 OpenModelica 多领域建模与仿真工具，根据第 3.4.1 节介绍的多领域仿真模型构建方法，基于 Modelica 建模规范定义航空发动机结构系统的连接器模型以及组件模型，最后建立结构系统的仿真模型。

首先，根据结构系统的工作场景中进气道、压气机、燃烧室、涡轮、收敛尾喷管、发动机转轴之间的物质、能量交换，定义航空发动机的连接器（Connector）元模型。按照已知的物质、能量交换包括：空气、高温高压空气、高温高压燃气、机械能，将

连接器首先分为两个大类，分别是流体接口库与机械接口库。

在 OpenModelica 多领域建模与仿真工具中构建的连接器模型如表 5.2 所示。其中流体接口库包含流体接口（FluidPort）、流体输入接口（FluidPort\_a）与流体输出接口（FluidPort\_b）三个连接器元模型，它们都包含三个参数，其中  $m\_flow$  代表**接口质量流量**，即单位时间内通过该接口横截面的气体的质量； $p\_t$  代表**接口总压**，即通过该流体接口的总压强； $T\_t$  代表**接口总温**，即通过该流体接口的总温度，单位为摄氏度。机械接口库中包含功率接口（PowerPort）、功率输入接口（PowerPort\_a）、功率输出接口（PowerPort\_b）三个连接器元模型，它们都包含两个参数，其中  $P$  代表**接口传递功率**； $N$  代表**接口传递的实际转速**。

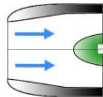
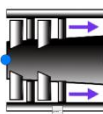
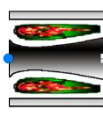
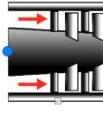
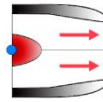

表 5.2 Modelica 连接器元模型及所包含参数

所属 Modelica 元元模型	接口库	所属 Modelica 元模型	对应模型	包含参数 (Parameter)
连接器 (Connector)	流体接口库	流体接口 (FluidPort)		$m\_flow$ 、 $p\_t$ 、 $T\_t$
		流体输入接口 (FluidPort_a)		$a.m\_flow$ 、 $a.p\_t$ 、 $a.T\_t$
		流体输出接口 (FluidPort_b)		$b.m\_flow$ 、 $b.p\_t$ 、 $b.T\_t$
	机械接口库	功率接口 (PowerPort)		$P$ 、 $N$
		功率输入接口 (PowerPort_a)		$a.P$ 、 $a.N$
		功率输出接口 (PowerPort_b)		$b.P$ 、 $b.N$

然后，根据航空发动机结构系统的工作场景，分析各系统组成部分以及经过该组成部分的物质或能量流向，定义航空发动机结构系统的组件（Component）元模型及其包含的连接器元模型与参数，如表 5.3 所示。航空发动机结构系统包含进气道、压

气机、燃烧室、涡轮、收敛尾喷管、发动机转轴六类组件，在 OpenModelica 工具中构建的组件元模型如下表所示。以压气机组件为例，它具有一个流体输入接口，以接受进气道进入的空气，具有一个流体输出接口，以向燃烧室输出经过压缩的高温高压空气，具有一个功率输入接口，以接收来自发动机转轴传递的机械能从而反复地吸入与压缩空气。

表 5.3 Modelica 连接器元模型及所包含参数

所属 Modelica 元元模型	所属 Modelica 元模型	对应模型	包含连接器 (Connector) 元模型	包含参数 (Parameter)
组件 (Component)	进气道 (Air_Inlet)		流体输出接口	b.m_flow、b.p_t、b.T_t
	压气机 (Compressor)		流体输入接口	a.m_flow、a.p_t、a.T_t
			流体输出接口	b.m_flow、b.p_t、b.T_t
			功率输入接口	a.P、a.N
	燃烧室 (Chamber_TLA)		流体输入接口	a.m_flow、a.p_t、a.T_t
			流体输出接口	b.m_flow、b.p_t、b.T_t
	涡轮 (Turbine)		流体输入接口	a.m_flow、a.p_t、a.T_t
			流体输出接口	b.m_flow、b.p_t、b.T_t
			功率输出接口	b.P、b.N
	尾喷管 (Nozzle)		流体输入接口	a.m_flow、a.p_t、a.T_t
	转轴 (Shaft)		功率输入接口	a.P、a.N
			功率输出接口	b.P、b.N

最后，基于上述定义的连接器和组件元模型，结合航空发动机结构系统的工作场景，构建航空发动机结构系统的仿真模型，如图 5.3 所示。该模型具有 6 个组件 (Component)，12 个连接器 (Connector)，6 条连接线 (Connection)，32 个参数 (Parameter)。

ter) 等元模型实例。

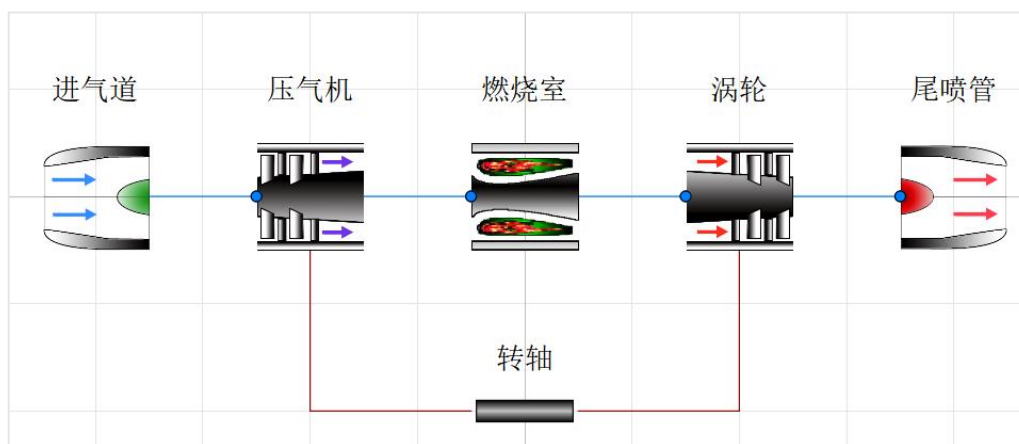


图 5.3 航空发动机 Modelica 仿真模型图

### 5.3.2 基于服务的航空发动机系统架构-仿真模型一体化表达

#### 5.3.2.1 基于服务的系统架构模型一体化表达

根据 3.3 节中架构模型与 OSLC 元数据模型的映射方法, 开发多架构建模工具 MetaGraph 适配器, 将架构工具中对模型及模型要素演化的工具活动, 以及对架构模型进行的分析、检查、转换等活动, 转化为标准的网络服务。

##### (一) 仿真工具中对模型及模型要素演化的工具活动服务化

将航空发动机架构模型库映射为服务提供方目录, 如图所示, 它包含多个 RFLP 阶段构建的系统模型即服务提供方, 例如 InternalBlockDiagram\_engine.karma 即为发动机结构系统的内部模块图, 用以描述架构系统的物理组成及交互关系。查询该服务提供方目录的统一标识为 <http://localhost:9092/catalogs/model>, 如图 5.4 所示。

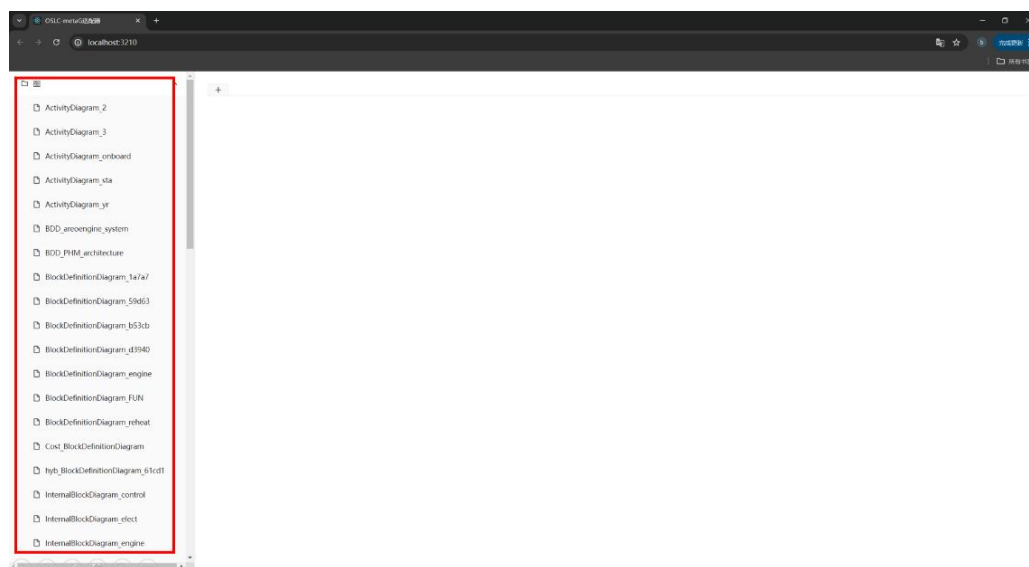


图 5.4 模型库映射为服务提供方目录页面

以上述架构模型库所在的服务提供方目录为入口，通过将架构模型映射为服务提供方，用于提供与模型及模型要素有关的服务。例如发动机结构系统的内部模块图 `InternalBlockDiagram_engine.karma` 即为一个服务提供方，在模型数据操作层面，它能够提供对于对象、点、关系、角色、属性的五种服务，如图 5.5 所示，查询该服务提供方的统一标识为 `http://localhost:9092/providers/model/InternalBlockDiagram_engine.karma`。

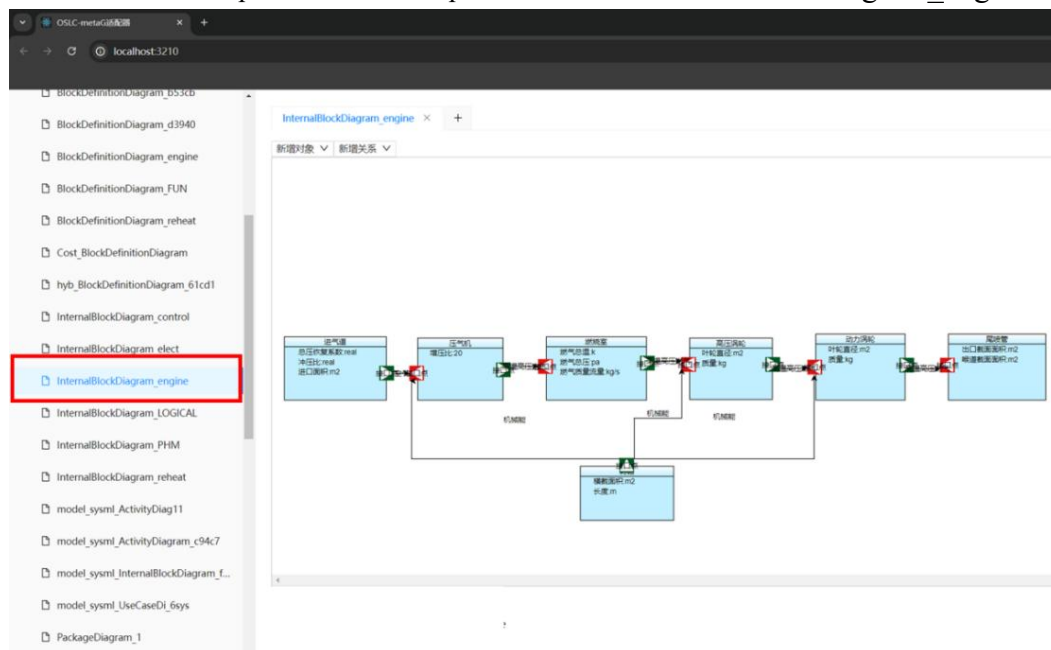


图 5.5 图映射为服务提供方页面



以架构模型所对应的服务提供方为入口，包含以图、对象、点、关系、角色、属性的六种模型及模型元素所代表的资源的服务。以对图资源的查询服务为例，例如，将发动机结构系统物理架构模型 `InternalBlockDiagram_engine.karma` 即视为资源，即服务可以操作的单元，实现模型查询服务的统一标识为 `http://localhost:9092/providers/model/InternalBlockDiagram_engine.karma/modelContent`，如图 5.6 所示。

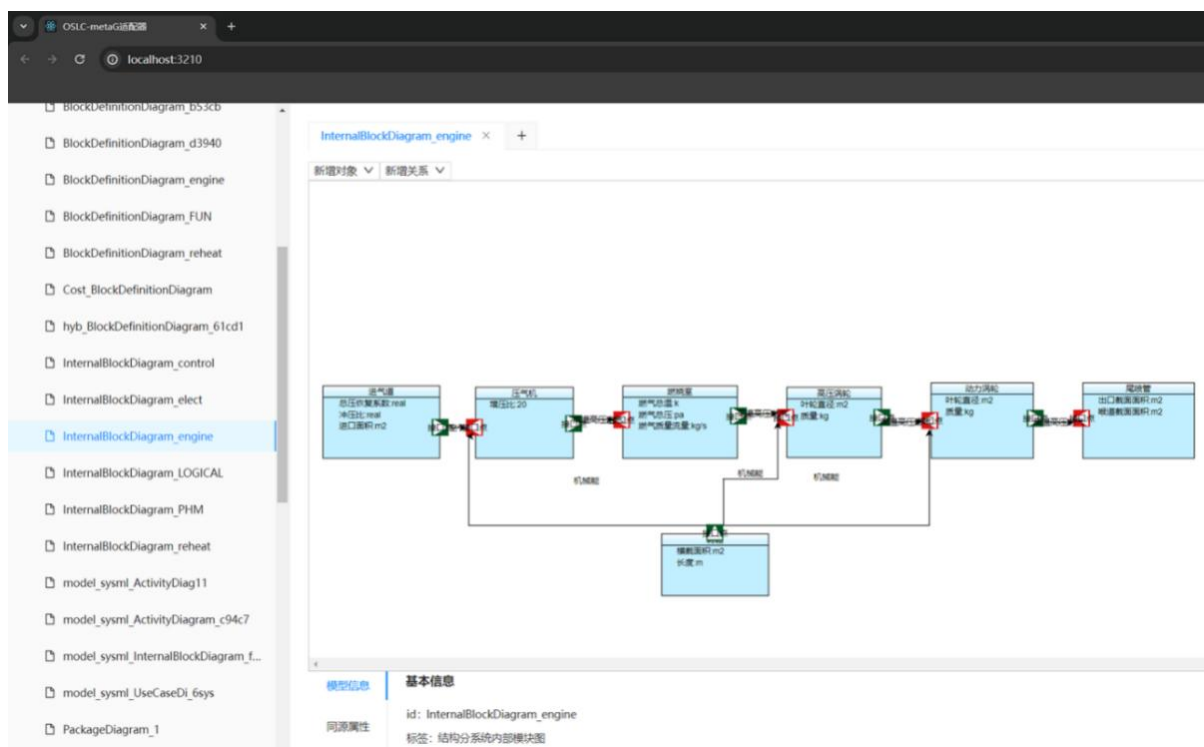


图 5.6 架构模型映射为资源的服务查询页面

其中，在以架构模型为资源的服务查询页面，通过数据链接原理，支持对对象、点、关系、角色、属性的五种模型元素所代表资源进行服务操作。以通过架构模型资源链接到对象资源为例，通过点击上述架构模型 `InternalBlockDiagram_engine.karma` 中的压气机对象，即可查询到该压气机对象资源的信息。通过点击该对象查询服务的统一标识，即可链接到对象资源的服务查询页面，如图 5.7 所示，可以查询到对象资源的名称与该对象所包含的点与属性资源信息。

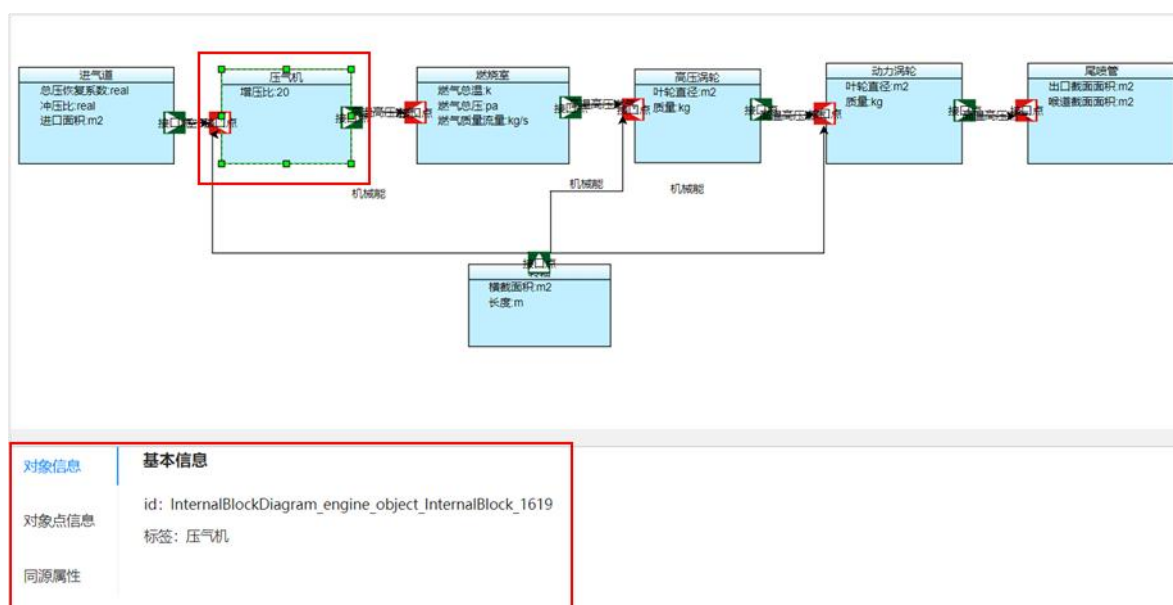


图 5.7 对象模型映射为资源的服务查询页面

同理，基于数据链接原理，在以模型为资源的服务查询页面，通过数据链接原理，支持对点与属性资源进行服务操作，如图 5.8 点资源服务页面与图 5.9 参数资源服务查询页面所示。

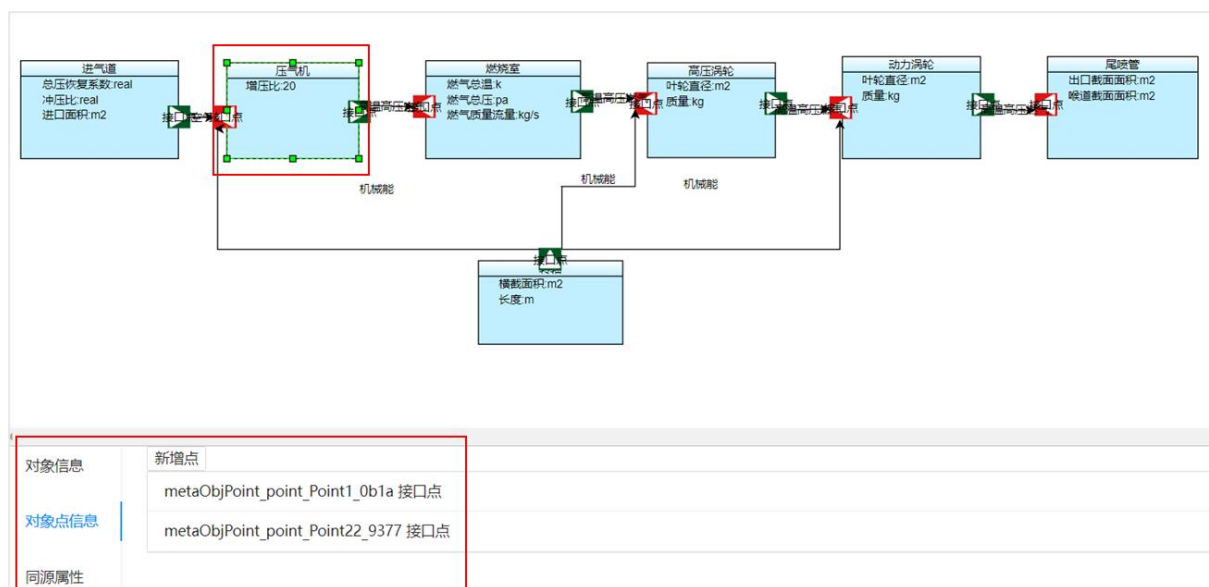


图 5.8 对象资源链接到点资源的服务查询页面

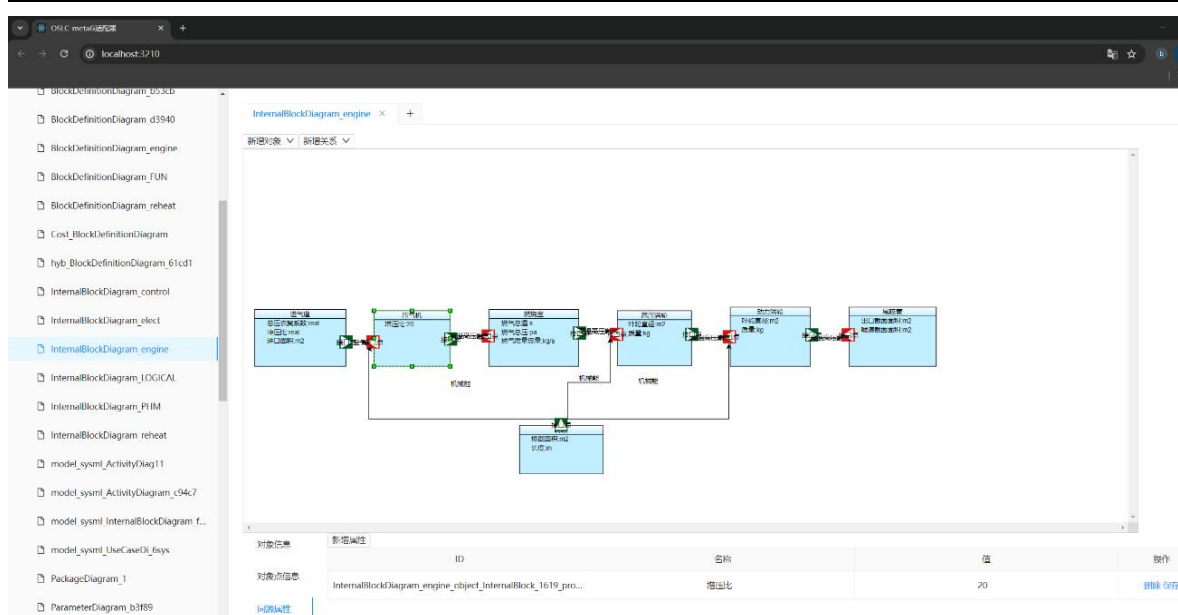


图 5.9 对象资源链接到参数资源的服务查询页面

除查询服务外，适配器支持对对象资源所包含的参数资源进行修改操作。例如，将压气机对象所包含的增压比属性值参数由 20 修改为 22，如图 5.10 所示，实现参数资源修改服务的统一标识为 <http://localhost:8084/providers/modelica/TurboJet.mo/modelicaModel>。图 5.11 为压气机增压比参数由 20 修改为 22 后的结果。

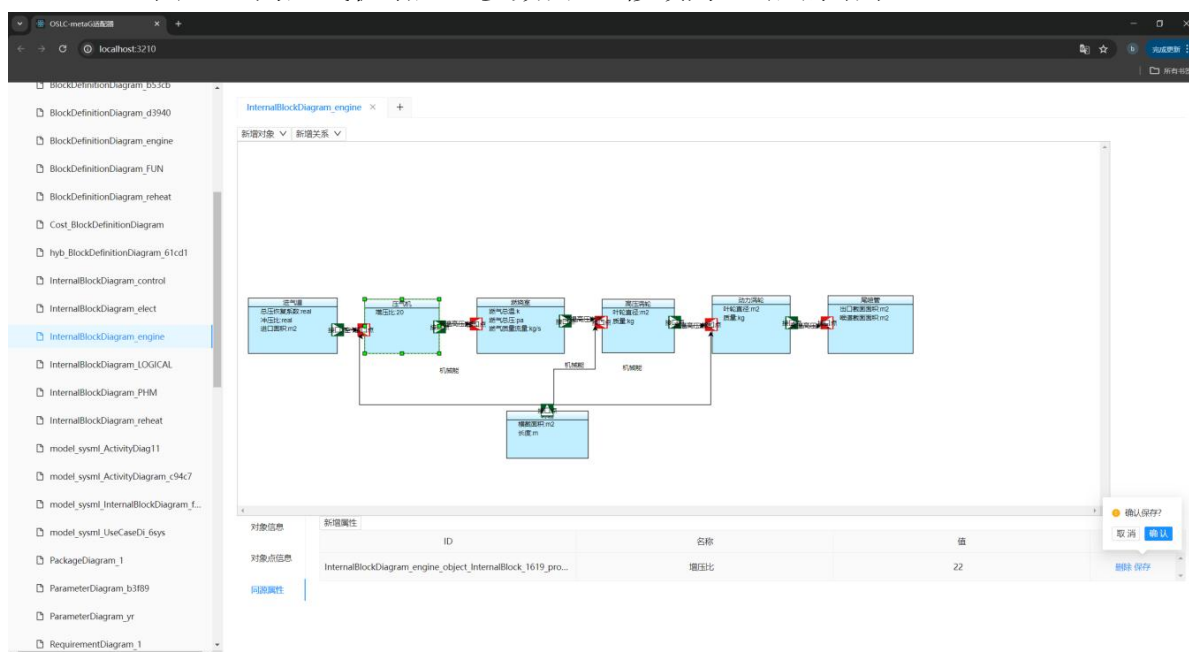


图 5.10 参数资源的修改服务页面

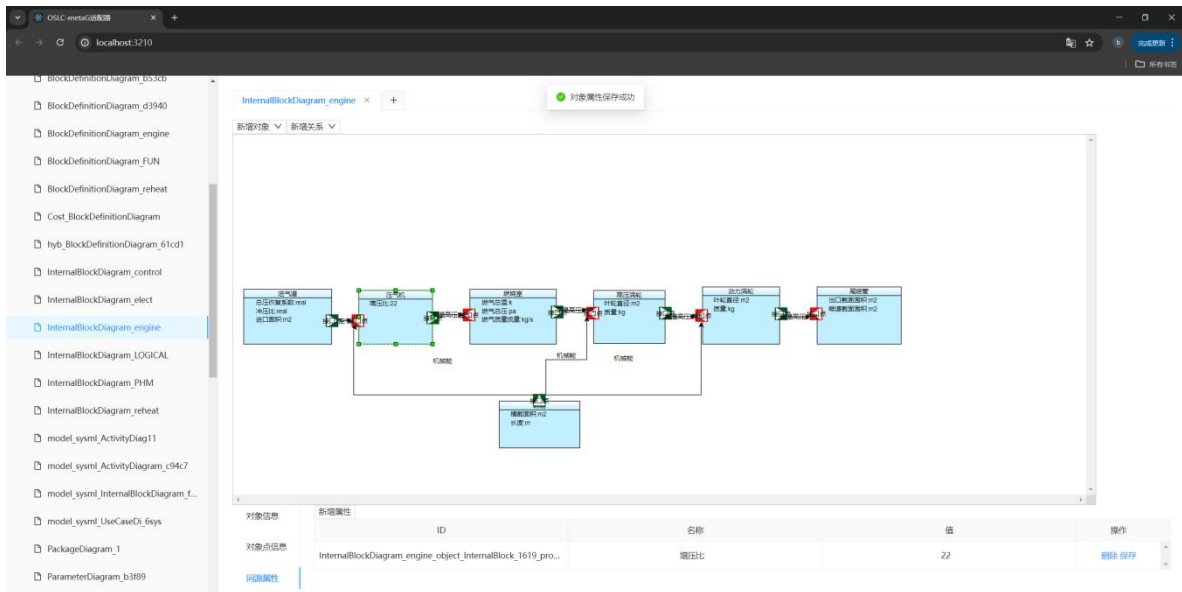


图 5.11 参数修改后结果

## (2) 架构模型分析、检查、转换等工具活动服务化

适配器在实现对架构模型及模型元素操作的工具活动之外，支持对架构模型进行分析、检查、转换等工具活动转化为统一的操作接口。架构代码生成：对结构系统内部模块图 `InternalBlockDiagram_engine.karma` 根据已定义的代码生成脚本，执行代码生成操作，自动转化为目标模型或代码。实现代码生成活动的统一标识为 `http://localhost:9092/providers/model/InternalBlockDiagram_engine.karma/modelContent/codeGenerate`，如图 5.12 所示。

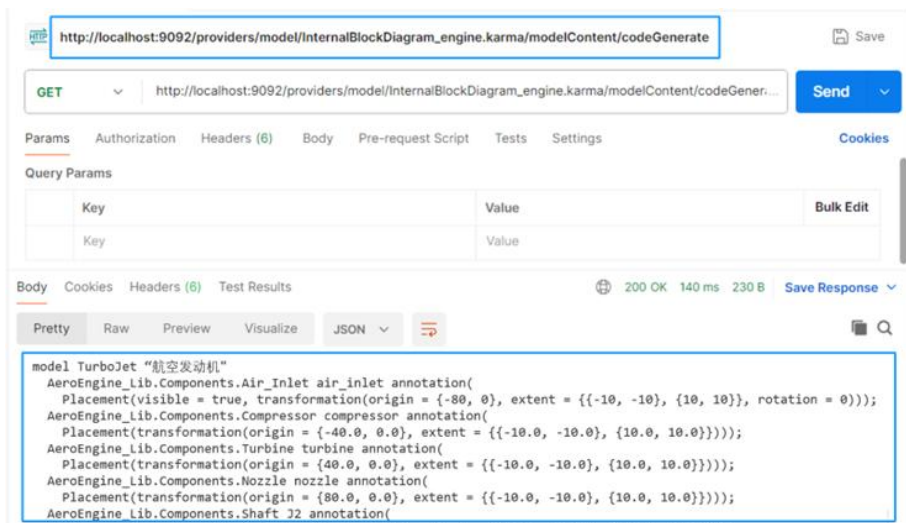


图 5.12 架构模型代码生成仿真模型代码

### 5.3.2.2 基于服务的系统仿真模型一体化表达

根据 3.4 节中仿真模型与 OSLC 元数据模型的映射方法及适配器实现方法，开发仿真建模工具 OpenModelica 适配器，将仿真工具中对模型及模型要素演化的工具活动，以及对仿真模型进行的分析、检查、转换或执行仿真等活动，转化为标准的网络服务。

#### (1) 仿真工具中对模型及模型要素演化的工具活动服务化

将航空发动机结构系统的仿真模型库映射为服务提供方目录，它包含多个仿真模型即服务提供方，例如 TurboJet.mo 即为发动机结构系统仿真模型，如图 5.13 所示，查询该服务提供方目录的统一标识为 <http://localhost:8084/catalogs/modelicaLib>。

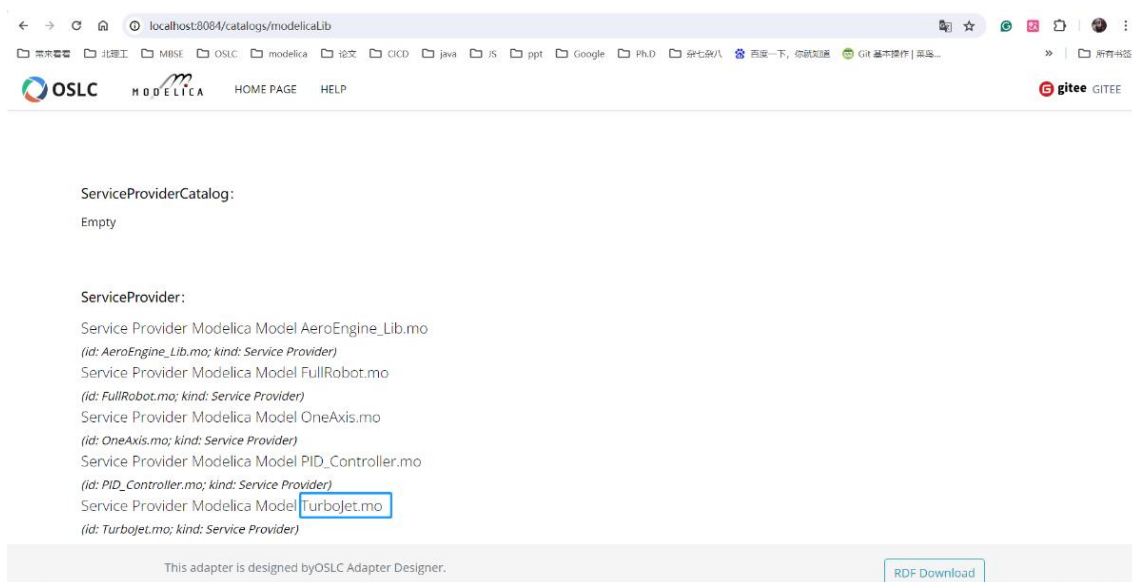


图 5.13 仿真模型库映射为服务提供方目录

以上述仿真模型库所在的服务提供方目录为入口，通过将仿真模型映射为服务提供方，用于提供与模型及模型要素有关的服务。例如 TurboJet.mo 即发动机结构系统仿真模型作为一个服务提供方，在模型数据操作层面，它能够提供对于仿真模型、子系统、组件、连接线、连接器、参数与修改语句的六种服务。以仿真模型所对应的服务提供方为入口，能够实现对以仿真模型、子系统、组件、连接线、连接器、参数与修改语句的六种模型及模型元素所代表的资源进行操作。例如，将仿真模型 TurboJet.mo 即发动机结构系统仿真模型视为资源，即服务可以操作的单元，实现模型查询服

务的统一标识为 <http://localhost:8084/providers/modelica/TurboJet.mo/modelicaModel>，如图 5.14 所示。

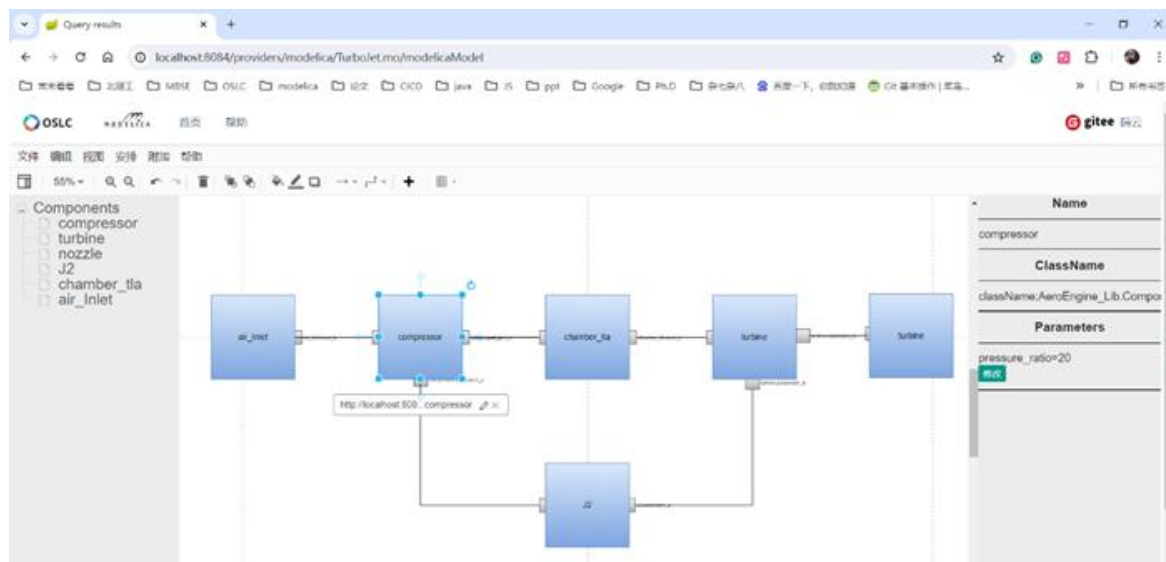


图 5.14 仿真模型映射为资源的服务查询页面

其中，在以仿真模型为资源的服务查询页面，通过数据链接原理，支持对子系统、组件、连接线、连接器、参数与修改语句的其它五种模型元素所代表的资源进行服务操作。以通过仿真模型资源链接到组件资源为例，通过点击上述仿真模型 TurboJet.mo 中的压气机组件，即可查询到该组件资源的信息。通过点击该组件查询服务的统一标识 <http://localhost:8084/providers/modelica/TurboJet.mo/modelicaComponent/compressor>，即可链接到组件资源的服务查询页面，如图 5.15 所示，与在图 5.14 中模型资源包含的组件查询结果相同，可以查询到组件资源的名称与该组件所包含的连接器与参数资源信息。



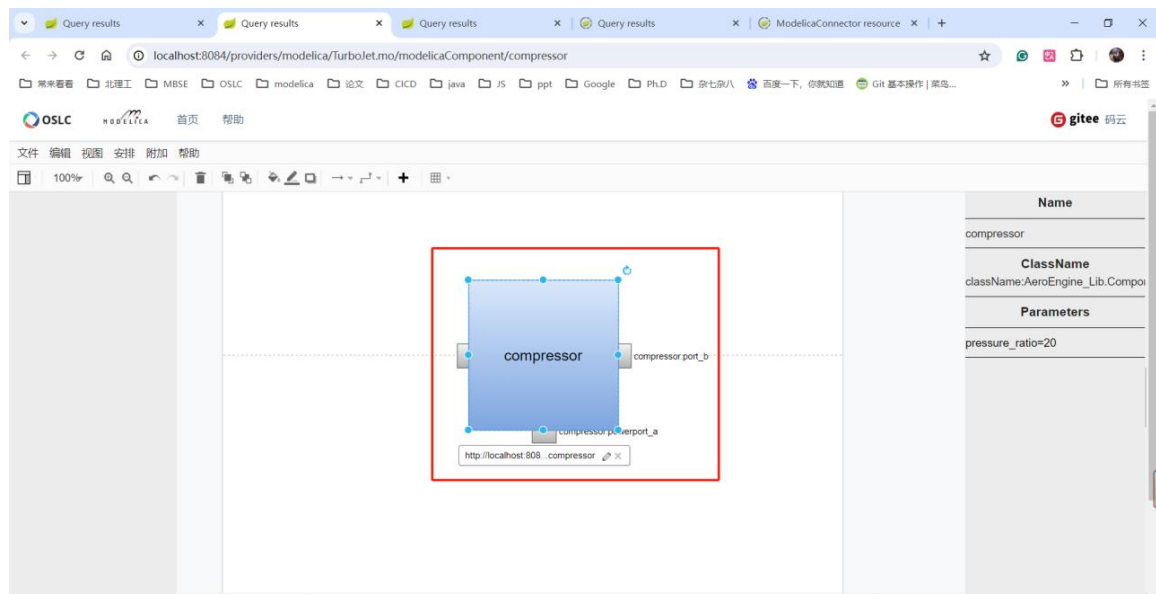


图 5.15 组件模型映射为资源的服务查询页面

同理，基于数据链接原理，在以组件为资源的服务查询页面，通过数据链接原理，支持对连接器、参数与修改语句这三种模型元素所代表的资源进行服务查询，图 5.16 是参数资源的查询服务，图 5.17 是连接器资源的查询服务。

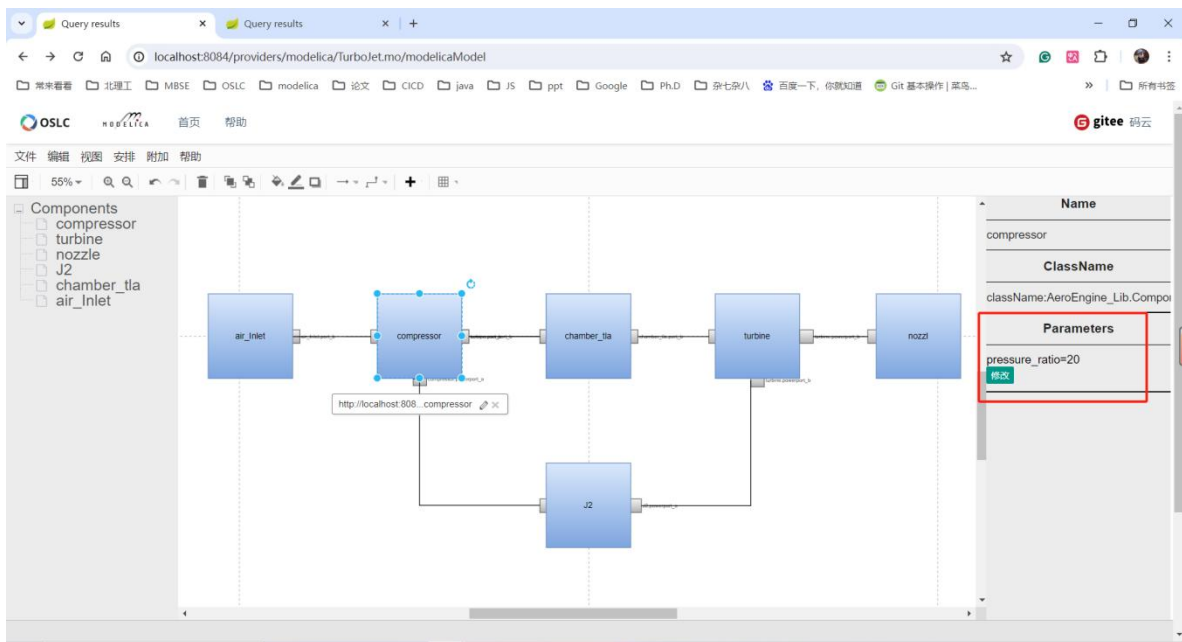


图 5.16 组件资源链接到参数资源的服务查询页面

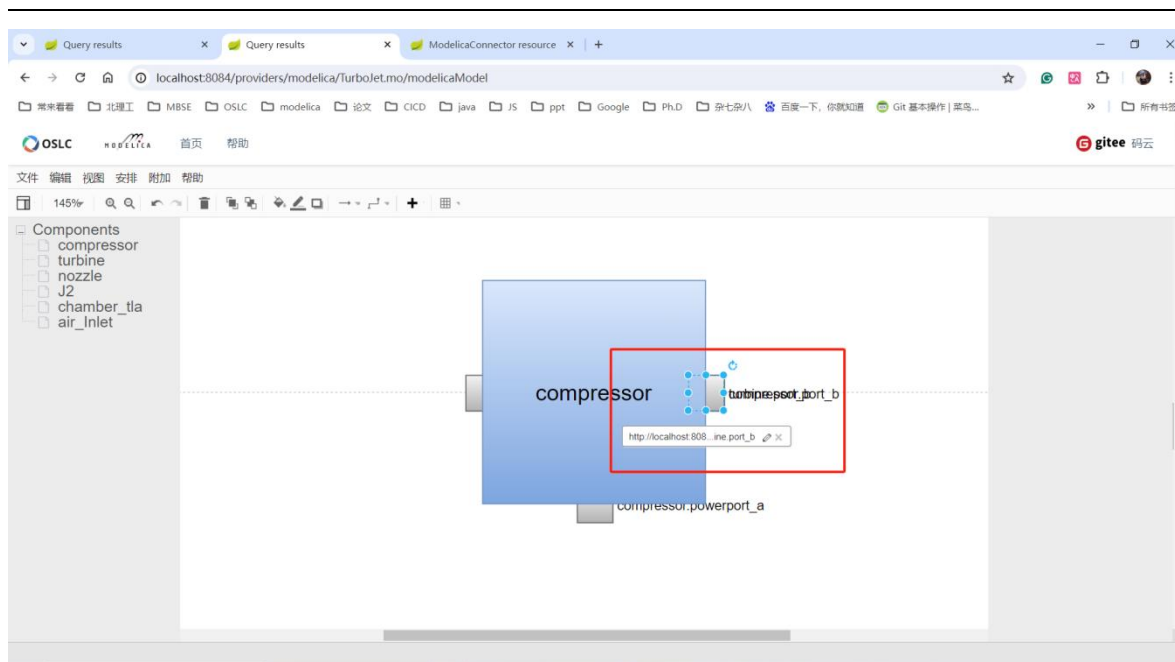


图 5.17 组件资源链接到连接器资源的服务查询页面

除查询服务外，适配器支持对组件资源所包含的参数资源进行修改操作。例如，将压气机组件所包含的增压比参数由 20 修改为 22，如图 5.18 所示，实现参数资源修改服务的统一标识为 `http://localhost:8084/providers/modelica/TurboJet.mo/modelicaModification/compressor_pressure_ratio`。

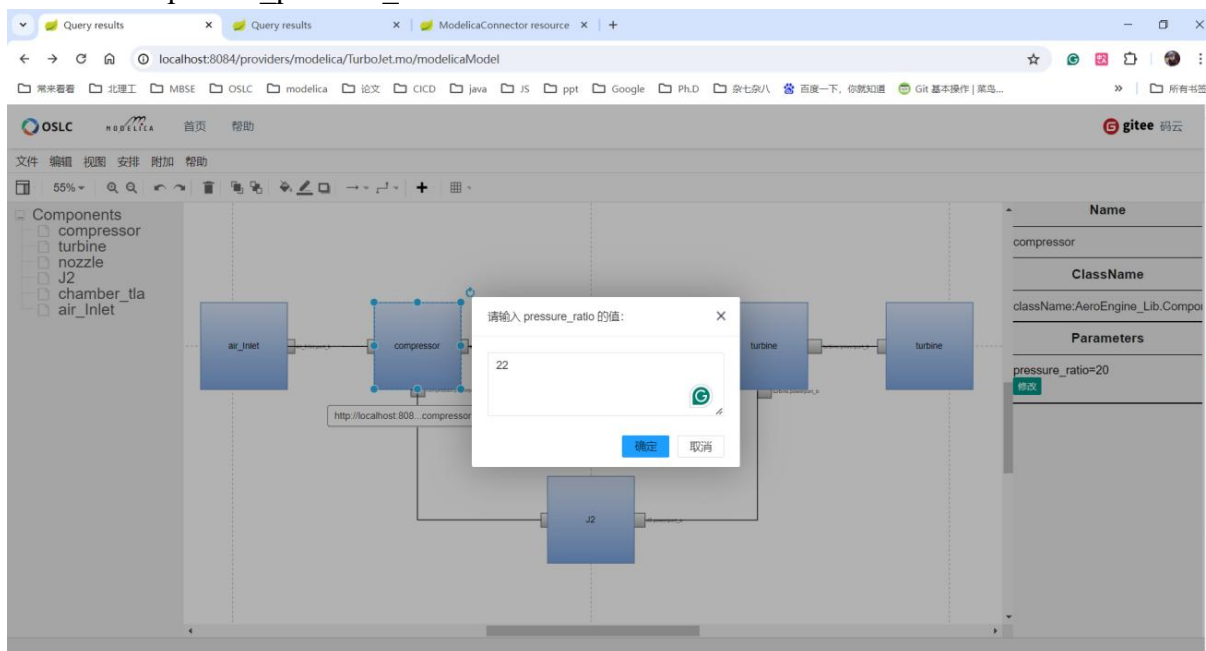


图 5.18 参数资源的修改服务页面



## (2) 仿真模型分析、检查、转换或仿真等工具活动服务化

适配器在实现对仿真模型及模型元素操作的工具活动之外，支持对仿真模型进行分析、检查、转换或执行仿真等工具活动转化为统一的操作接口。

仿真模型检查：对发动机结构系统仿真模型 TurboJet.mo 执行模型检查操作，实现仿真模型检查活动的统一标识为：<http://localhost:8084/providers/modelica/TurboJet.mo/modelicaModel/modelcheck.>，如图 5.19 所示。

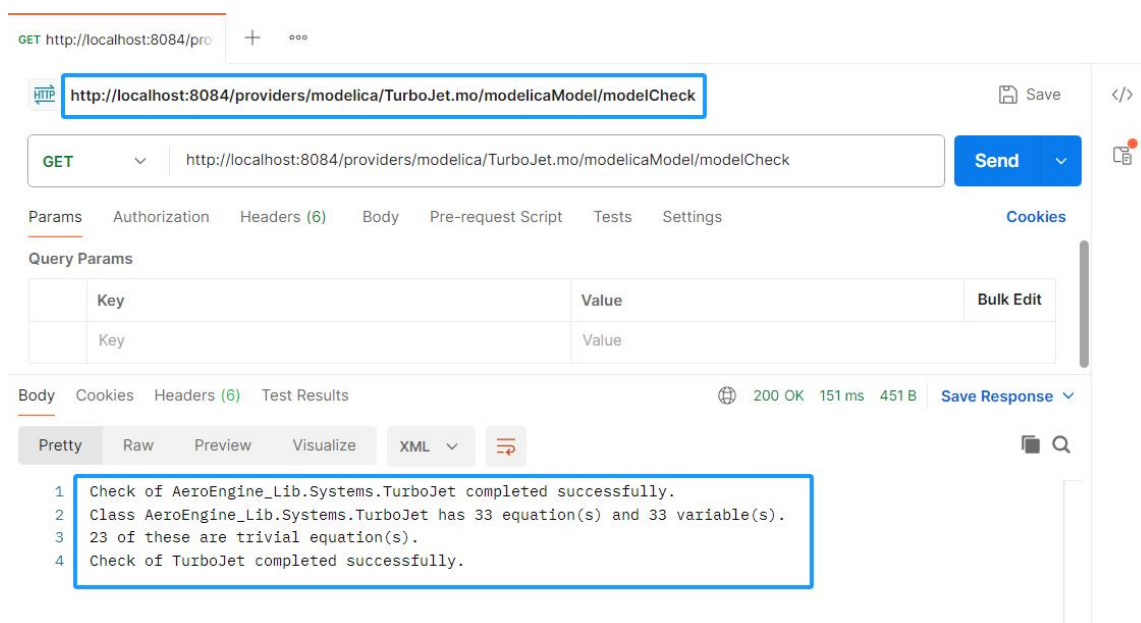


图 5.19 仿真模型检查服务页面

模型仿真：对发动机结构系统仿真模型 TurboJet.mo 执行仿真，支持设置仿真时间、步长、算法，仿真结果输出、选择参数绘制仿真图形等，实现模型仿真活动的统一标识为：[http://localhost:8084/providers/modelica/TurboJet.mo/modelicaModel/simulate/compressor.powerport\\_a.N](http://localhost:8084/providers/modelica/TurboJet.mo/modelicaModel/simulate/compressor.powerport_a.N)，如图 5.20 所示。

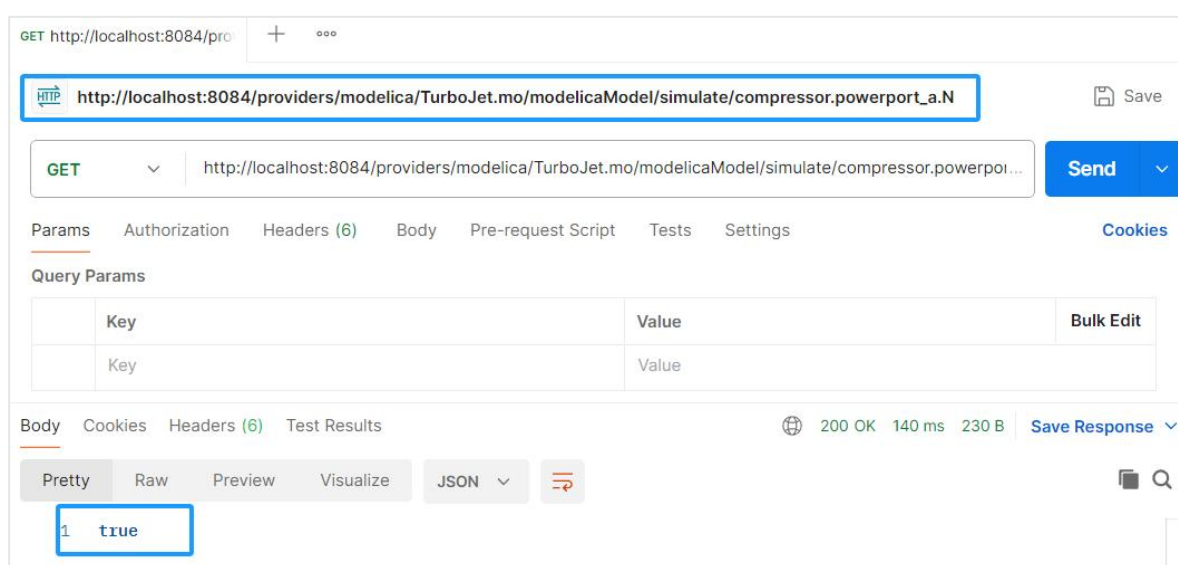


图 5.20 模型仿真执行页面

### 5.3.3 航空发动机系统架构与仿真持续集成 workflow 设计

在解决了上述对航空发动机系统进行物理架构设计与仿真的过程中架构与仿真模型数据与操作接口异构的问题后，设计过程存在大量的手动操作，会使设计信息的传递不一致，例如在内部模块图中定义的压气机对象的增压比属性参数初始设定为 20，对应到仿真模型中压气机组件的增压比设计也为 20。当设计需求发生变化或者由于仿真参数迭代，需要对架构模型中的设计参数进行修改时，设计参数的同步过程通常是由人工手动进行的。手动识别的过程包括工程师人眼甄别需要对哪个架构模型中哪个对象包含的参数进行修改，对应到仿真模型中，需要对哪个仿真模型及仿真模型中的参数进行修改，修改完成后进一步进行仿真，人眼根据仿真结果判断该架构是否满足系统需求，并判断是否需要再进行一轮仿真与设计迭代。

由上述架构与仿真的验证场景可以看出，大多数设计与验证的闭环反馈过程都是由工程师人脑做出判断与手工进行执行。这个过程有大量人在环中的设计，不可否认会导致设计效率打折与错误率攀升：从效率的角度来说，人手工操作工具难以实现系统快速地验证与设计；从设计一致性与正确性的角度来说，人脑在频繁的反馈与迭代过程中甄别设计信息，容易出现错误导致设计信息传递不一致的问题。存在针对上述存在的设计场景，进行航空发动机系统架构与仿真持续集成 workflow 设计，针对模型传递的自动化能力特征，验证该方法的有效性。

### 5.3.2.1 发动机系统架构与仿真持续集成 workflow 分析

首先，根据航空发动机的架构设计与仿真验证场景，识别出一条 MBSE 工作流。该工作流以航空发动机结构系统物理架构设计阶段构建的内部模块图 `InternalBlockDiagram_engine.karma` 作为工作流的输入，经过架构模型代码生成、仿真模型检查、仿真模型组件参数查询、仿真模型执行仿真这几个活动，各活动的具体内容与执行逻辑如下：

- 1) 架构模型代码生成活动：根据架构模型代码生成的规则脚本，将架构模型 `InternalBlockDiagram_engine.karma` 转化为仿真模型 `TurboJet.mo` 的源代码；`TurboJet.mo` 的源代码生成成功则进入下一阶段，失败则流水线终止执行。
- 2) 仿真模型检查活动：对仿真模型 `TurboJet.mo` 进行语法检查、类型检查、约束检查，判断模型中变量和方程数是否相等；相等则检查成功、执行下一阶段，失败则流水线终止执行。
- 3) 仿真模型组件参数查询活动：查询代码生成的仿真模型 `TurboJet.mo` 压气机的增压比参数值是否与架构模型 `InternalBlockDiagram_engine.karma` 中定义的压气机增压比的属性值是否相同；相同则证明设计参数在架构与仿真模型之间端到端的传递正确、执行下一阶段；不同则证明在代码生成过程中设计参数的传递不一致，流水线终止执行。
- 4) 仿真模型执行仿真活动：在预定义的架构模型的对象属性值与通过代码生成手段得到的仿真模型的组件参数值相同后，对仿真模型执行仿真，并进行仿真参数配置，包括：开始时间、结束时间、间隔数量、仿真积分算法、积分误差、仿真结果输出格式等；若正常的输出仿真结果，则整条流水线执行成功，并在此节点结束执行，若未正常输出仿真结果，则流水线执行失败，并终止此次执行。

然后，根据上述对工作流输入以及各活动的执行内容与执行逻辑的分析，分别查询上述工作流活动与 4.4 节设计的工作流语义模板的对应关系，例如本工作流中架构模型代码生成仿真模型活动的语义模板对应 4.4.2 节（1）架构模型代码生成活动的语义模板定义，其余三个活动的语义模板对应关系如表 5.4 所示。然后，明确各活动需要输入语义模板的参数以及具体的参数值，例如模型名称，根据模型名称的值可以 O

SLC 统一资源标识，进而可以基于语义模板完成这四个活动对应的脚本构建。

表 5.4 基于语义模板的航发结构系统 workflow 设计

工作流活动	对应的流水线语义模板	输入语义模板的参数	具体的参数值	OSLC 统一资源标识
架构模型 代码生成 仿真模型	见 4.4.2 节 (1) 架构模型代码生成活动的语义模板定义	架构模型文件名称	InternalBlockDiagram_engine.karma	http://localhost:9092/providers/model/InternalBlockDiagram_engine.karma/modelContent/codeGenerate
仿真模型 检查	见 4.2.2 节 (2) 仿真模型检查活动的语义模板定义	仿真模型文件名称	TurboJet.mo	http://localhost:8084/providers/modelica/TurboJet.mo/modelicaModel/modelCheck
仿真模型 参数查询	见 4.4.1 节 (1) 模型或模型元素查询活动的语义模板定义	仿真模型文件名称、组件名称、参数名称	TurboJet.mo, compressor, pressure_ratio	http://localhost:8084/providers/modelica/TurboJet.mo/modelicaModification/compressor_pressure_ratio
仿真模型 执行仿真	见 4.2.2 节 (3) 仿真模型执行仿真活动的语义模板定义	仿真模型文件名称、仿真参数配置	TurboJet.mo, compressor.powerport_a.N	http://localhost:8084/providers/modelica/TurboJet.mo/modelicaModel/simulate/compressor.powerport_a.N

### 5.3.2.2 基于语义模板的架构与仿真集成 workflow 定义

根据上述表 5.4，填充各活动对应的语义模板如下。

(1) InternalBlockDiagram\_engine.karma 模型代码生成活动

```

1. stage('InternalBlockDiagram_engine.karma 模型代码生成') {
2.     steps {
3.         script{
4.             def codeGenerate = httpRequest(
5.                 url: 'http://localhost:9092/providers/model/InternalBlockDiagram_engine.karma/modelContent/codeGenerate')
6.             println(codeGenerate.status)
7.             def result = codeGenerate.content
8.             if(result.contains("completed successfully")){
9.                 println(result)
10.            echo "架构模型代码生成成功"

```

```

11.         }else{
12.             echo "架构模型代码生成失败"
13.             exit 1
14.         }
15.     }
16. }
17.}

```

## (2) TurboJet.mo 模型检查活动

```

1.stage('TurboJet.mo 模型检查') {
2.     steps {
3.         script{
4.             def check = httpRequest(
5.                 url:'http://localhost:8084/providers/modelica/TurboJet.mo/modelicaModel/modelCheck')
6.             println(check.status)
7.             def result = check.content
8.             if(result.contains("completed successfully")){
9.                 println(result)
10.                echo "模型符合语法规则"
11.            }else{
12.                cho "模型不符合语法规则"
13.                exit 1
14.            }
15.        }
16.    }
17.}

```

## (3) TurboJet.mo 模型的 compressor 组件的 pressure\_ratio 增压比参数查询活动

```

1.stage('TurboJet.mo 模型的 compressor 组件的 pressure_ratio 增压比参数查询') {
2.     steps {
3.         script{
4.             def query = httpRequest(
5.                 url:'http://localhost:8084/providers/modelica/TurboJet.mo/modelicaModification/compressor_pressure_ratio',
6.             )
7.             def result = query.content
8.             if(result.status == 200){
9.                 println(result) //打印出查询到的参数结果

```

```

10.             echo "查询成功"
11.         }else{
12.             echo "查询失败"
13.             exit 1
14.         }
15.     }
16. }
17. }

```

#### (4) TurboJet.mo 模型仿真执行活动

```

1.stage('TurboJet.mo 模型仿真执行') {
2.    steps {
3.        script{
4.            def simulate = httpRequest(
5.                url:'http://localhost:8084/providers/modelica/TurboJet.mo/
modelicaModel/simulate/compressor.powerport_a.N',
6.            )
7.            def result = simulate.content
8.            println(simulate.status)
9.            println(simulate)
10.           if(result == 'true'){
11.               echo "模型仿真执行成功"
12.           }else{
13.               echo "模型仿真执行失败"
14.               exit 1
15.           }
16.        }
17.    }

```

### 5.3.2.1 架构与仿真持续集成工作流自动化执行

在上述使用流水线语义模板对结构系统设计与仿真工作流所包含的四个活动定义的基础上，补充工作流源和配置项的定义，对航空发动机结构系统设计与仿真工作流的流水线脚本进行完全的定義。通过在自动化集成平台 Jenkins 创建流水线项目，进行项目配置，导入并执行流水线脚本，在上述操作后对应的 Jenkins 工具页面如图 5.21 所示。其中，该流水线包括四个部分，按顺序自动调用 OSLC 数据服务执行。



图 5.21 基于 Jenkins 工具的流水线执行页面

对于第一个阶段，InternalBlockDiagram\_engine.karma 模型代码生成活动，其执行结果如图 5.22 所示。通过发送 OSLC 服务请求 [http://localhost:9092/providers/model/InternalBlockDiagram\\_engine.karma/modelContent/codeGenerate](http://localhost:9092/providers/model/InternalBlockDiagram_engine.karma/modelContent/codeGenerate)，根据代码生成的定义规则，由 InternalBlockDiagram\_engine.karma 模型代码生成 TurboJet.mo 仿真模型的源代码，并将生成结果打印在控制台。TurboJet.mo 的源代码生成成功则进入下一阶段，失败则流水线终止执行。

图中打印成功了 TurboJet.mo 模型的源代码，因此判断该步骤正常执行，并自动进入流水线的下一个阶段。

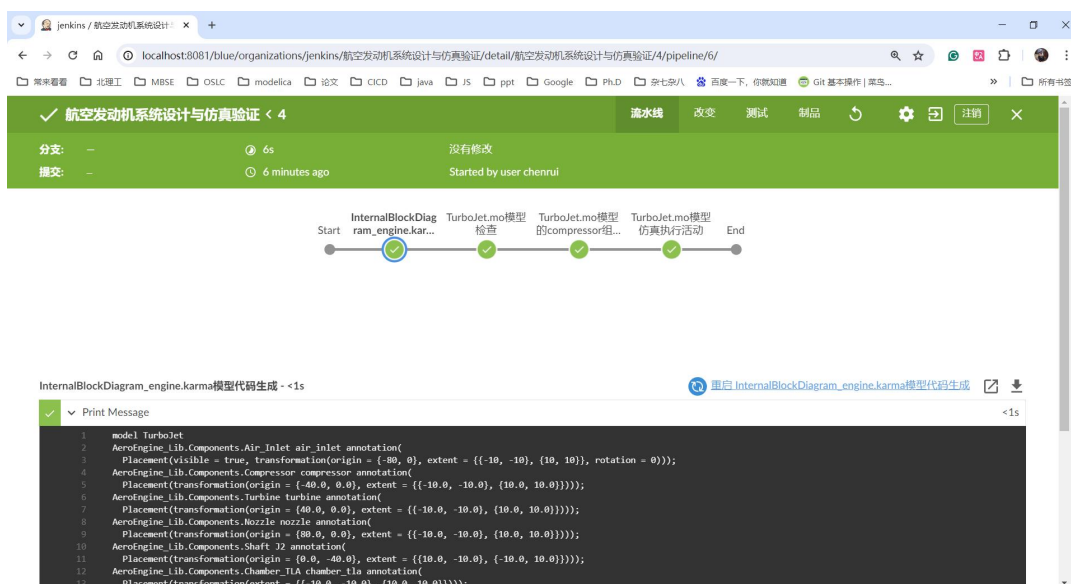


图 5.22 InternalBlockDiagram\_engine.karma 模型代码生成执行结果



对于第二个阶段，对由代码生成的 TurboJet.mo 模型进行仿真模型检查，其执行结果如图 5.23 所示。通过发送 OSLC 服务请求 <http://localhost:8084/providers/modelica/TurboJet.mo/modelicaModel/modelCheck>，对 TurboJet.mo 进行模型检查，并判断模型中变量与方程的数是否相等，并将结果打印在控制台。变量与方程的数相等则检查成功、执行下一阶段，失败则流水线终止执行。

图中第一列表示 OSLC 服务发送的结果，显示发送成功；第二列表示 TurboJet.mo 仿真模型检查结果：“Check of AeroEngine\_Lib.Systems.TurboJet completed successfully. Class AeroEngine\_Lib.Systems.TurboJet has 33 equation(s) and 33 variable(s). 23 of these are trivial equation(s). Check of TurboJet completed successfully.”显示变量与方程的数相等。第三列是本阶段执行的判定结果，显示模型符合语法语义规则。因此，该阶段执行成功，自动进入下一环节的执行。



图 5.23 TurboJet.mo 仿真模型检查执行结果

对于第三个阶段，对 TurboJet.mo 模型的 compressor 组件的 pressure\_ratio 增压比参数查询，其执行结果如图 5.24 所示。通过发送 OSLC 服务请求 [http://localhost:8084/providers/modelica/TurboJet.mo/modelicaModification/compressor\\_pressure\\_ratio](http://localhost:8084/providers/modelica/TurboJet.mo/modelicaModification/compressor_pressure_ratio)，对 TurboJet.mo 模型的 compressor 组件的 pressure\_ratio 增压比参数进行查询，并判断参数值是否与在架构模型压气机对象的增压比属性初始值定义是否相同，并将结果打印在



控制台。相同则证明设计参数在架构与仿真模型之间端到端的传递正确、进入下一阶段执行；不同则证明在代码生成过程中设计参数的传递不一致，则流水线终止执行。

图中第一列表示 OSLC 服务发送的结果，显示发送成功；第二列显示 compressor 组件的 pressure\_ratio 增压比参数与预定义的架构模型压气机对象的增压比初始值相等，皆为 20。因此该阶段执行成功，自动进入下一环节的执行。



图 5.24 TurboJet.mo 模型的 compressor 组件的 pressure\_ratio 增压比参数查询执行页面

对于第四个阶段，对 TurboJet.mo 模型执行仿真，其执行结果如图 5.25 所示。通过发送 OSLC 服务请求 `http://localhost:8084/providers/modelica/TurboJet.mo/modelicaModel/simulate/compressor.powerport_a.N`，对 TurboJet.mo 模型执行仿真，并进行仿真参数配置，包括：开始时间、结束时间、间隔数量、仿真积分算法、积分误差、仿真结果输出格式等；若执行的结果为 `true`，并生成规定格式的仿真结果文件，则整条流水线执行成功，并在此节点结束执行；若未正常输出仿真结果，则流水线执行失败，并终止此次执行，并将此次结果打印在控制台。

图中第一列与第二列分别显示 OSLC 服务请求的 URL 与请求状态，显示 OSLC 服务发送成功；第三列表示仿真执行是否完成、仿真结果是否以规定的格式输出。结果显示仿真执行成功，此阶段是流水线的最后一个活动，因此判定流水线执行成功。

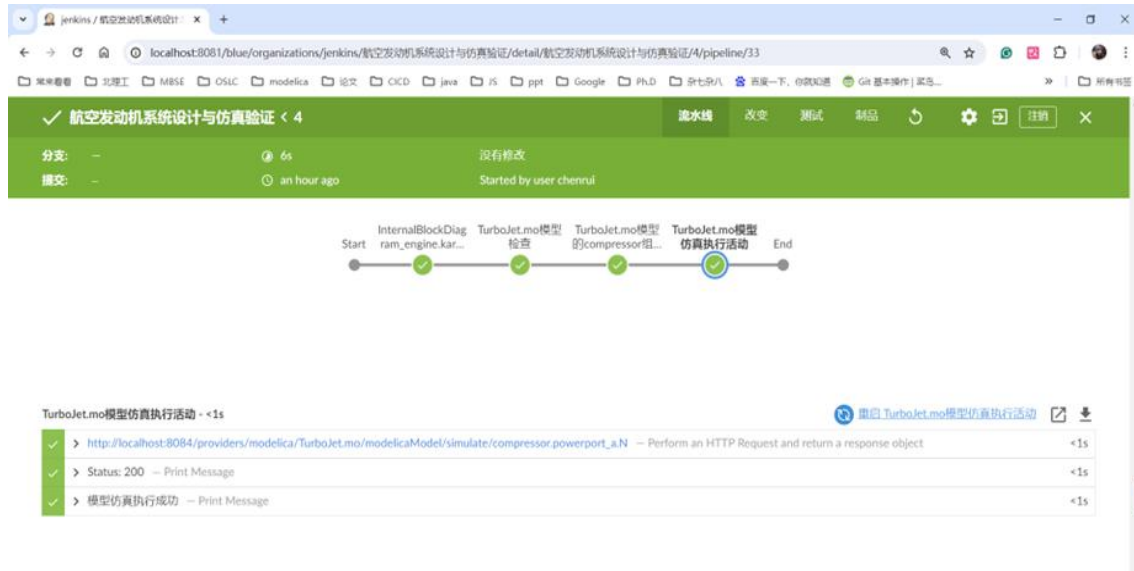


图 5.25 TurboJet.mo 模型的 compressor 组件的 pressure\_ratio 增压比参数查询执行页面

Jenkins 支持对流水线的源文件进行变更的轮询监控, 每当检测到流水线的源文件发生变更时, 就会自动触发流水线执行。此外, 基于 Jenkins 平台的插件功能, 支持对流水线脚本配置邮件通知的功能。在流水线每次执行结束之后, 将流水线的执行情况, 包括流水线的执行时间、执行状态、执行日志等, 以邮件或消息的形式自动通知给构建人, 以确保捕捉到每一次源模型文件变更导致的流水线执行结果的变更。

体现在本案例中, 以航空发动机结构系统的物理架构模型为流水线的源文件输入, 每当该系统架构模型发生变更后, 会自动触发流水线、按照顺序调用 OSLC 服务执行架构模型代码生成、仿真模型检查、仿真模型的组件参数查询以及仿真模型执行仿真。对串行的工作流活动而言, 流水线执行成功的条件是每个活动都成功执行, 一直执行到最后一个活动为止。在本案例中, 若仿真模型仿真执行成功之后, 构建人会及时收到流水线构建成功的通知, 从而及时对仿真结果进行查看与分析。当系统架构模型发生变化时, 会自动触发流水线的执行。设计信息以变更后的架构模型为载体, 在这条流水线上经过持续、自动化的传递, 它由架构模型开始、经过仿真模型, 并最终体现在对仿真结果的影响上。上述实现的 MBSE 模型传递过程, 有效验证与支持本文所提的面向复杂系统架构与仿真集成工作流设计方法。

## 5.4 案例总结与分析

本案例以航空发动机系统为对象，开展航空发动机系统的架构设计与仿真验证过程的 MBSE 模型传递案例验证。

首先，构建了航空发动机系统架构模型与仿真模型。在物理架构设计阶段，选取结构系统为研究对象，分析了结构系统的工作原理及物理组成，使用内部模块图对结构系统的组成及组成之间的物质能量交换进行了定义。然后，基于 Modelica 建模语言构建了航空发动机结构系统的物理仿真模型：定义了航空发动机的连接器元模型以及组件元模型，基于元模型与航空发动机的工作原理，构建了航空发动机结构系统的仿真模型。

然后，使用基于服务的架构-仿真一体化表达方法，将航空发动机的架构与仿真模型在语义层与数据层进行了统一表达。根据发动机结构系统的架构模型、仿真模型与 OSLC 元数据模型的映射规则，开发了多架构建模工具 MetaGraph 适配器以及多领域仿真建模工具 OpenModelica 适配器，将架构工具与仿真工具中对模型及模型要素的演化活动（例如查询与修改），以及模型的检查、转换与执行活动，转化为了标准的网络服务，为架构模型与仿真模型跨工具的集成提供了统一的操作基础。分析这两类活动模型传递的输入与输出，分别从时间上与空间上验证了 MBSE 模型传递的互操作性特征。

最后，设计了航空发动机系统架构与仿真持续集成 workflow。识别了航空发动机系统架构与仿真过程中的一条 workflow，分析了该条 workflow 包含的活动以及执行逻辑；基于流水线语义模板，完善了航空发动机 workflow 包含各活动的流水线脚本；通过自动化集成平台 Jenkins 集成了已完成的流水线脚本，并通过 OSLC 技术统一执行了流水线中各活动，实现了模型在 workflow 所代表的链路中进行频繁、自动化的集成，从而验证了航空发动机系统架构模型与仿真模型传递的自动化特征。

## 5.5 本章小结

本案例以航空发动机系统为研究对象，采用本文提出的 MBSE 模型传递技术，对系统架构设计与仿真验证过程进行了模型传递的案例验证。首先介绍了航空发动机的系统设计背景，交代了航空发动机系统设计过程存在的 MBSE 模型传递问题。然后，

针对航空发动机系统，构建了系统架构模型与仿真模型，描述了系统的结构特征与行为特征。然后，开发架构与仿真工具适配器，将架构模型、仿真模型以及工具操作都转化成了统一的 OSLC 服务，实现了模型跨工具的集成。然后，设计了航空发动机系统架构与仿真持续集成工作流，基于语义模板完成了工作流对应的脚本定义。通过 Jenkins 工具集成流水线脚本，并通过 OSLC 服务统一执行工作流，支持了航空发动机系统架构设计与仿真验证过程的 MBSE 模型传递。

## 第 6 章 总结与展望

### 1 全文总结

在面向复杂装备系统架构设计与仿真验证一体化的背景下，设计信息以模型为载体在架构设计与仿真验证等环节中闭环传递。模型传递保证了模型在设计上下游阶段之间信息的一致性，模型传递的效率直接决定架构设计与仿真验证两个环节的协同效率。面向架构设计-仿真验证一体化过程中，MBSE 模型传递仍存在多源异构模型集成困难与模型传递自动化水平低等问题，开展基于服务的 MBSE 模型传递技术研究。论文的主要研究工作总结如下：

（1）构建了面向 MBSE 模型传递的理论框架：研究明确了 MBSE 模型传递的概念及内涵，分析了模型传递的关键特征，特别是互操作性和自动化能力，为后续技术框架的构建提供了理论支撑。基于互操作性和自动化能力特征，提出了模型传递的技术框架，提供了架构设计与仿真验证集成的技术选型。

（2）研究了基于服务的架构-仿真模型一体化表达方法：研究了一种利用开放生命周期协作服务规范（OSLC）的元数据模型来实现架构模型与仿真模型的统一表达方法。首先研究了 OSLC 元数据模型，探索了其在架构模型与仿真模型一体化表达中的应用潜力。定义了架构模型、仿真模型及模型操作与 OSLC 元数据模型之间的映射规则，实现了模型信息在概念层的服务化表达。开发了工具适配器，实现了模型信息及模型操作在数据层的服务化表达。

（3）提出了一种面向模型持续集成的自动化 workflow 设计方法。研究了 workflow 定义规范及其组成元素，为后续 workflow 设计提供了理论基础。分析了架构设计-仿真验证一体化过程中的 workflow 活动，并建立了活动与模型操作服务之间的关联关系。构建了 workflow 活动定义与执行的语义模板，为特定场景下的 workflow 设计提供了可复用的框架。

（4）模型传递案例验证：选择了航空发动机系统作为案例，验证了所提出方法的有效性。应用提出的模型传递框架和方法，实现了航空发动机系统架构模型与仿真模型的有效传递和集成，验证了互操作性特征。设计并实施了航空发动机系统架构与仿真的持续集成 workflow，验证了自动化特征。

基于上述总结，本文的主要创新点有以下几点：

(1) 提出了一种基于服务的架构-仿真模型一体化表达方法。针对架构与仿真模型在语法和语义上的异构性问题,结合模型传递的互操作性关键特征,提出了一种创新的一体化表达方法。该方法通过引入开放生命周期协作服务规范(OSLC)的元数据模型,建立了架构模型、仿真模型与 OSLC 元数据模型之间的映射关系,实现了模型信息在概念层的服务化表达。这一方法不仅解决了架构模型与仿真模型之间的语义异构问题,而且通过开发工具适配器,实现了模型信息在数据层的服务化表达,从而为架构设计-仿真验证一体化过程中的模型传递提供了一种新的解决方案。

(2) 提出了一种面向模型持续集成的自动化 workflow 设计方法。该方法基于 workflow 定义规范,结合模型传递的自动化能力关键特征,为架构设计-仿真验证一体化过程提供了一套完整的工作流设计理论。通过形式化定义 workflow,建立起活动与模型操作服务之间的关联关系,并基于服务统一执行 workflow 活动,为自动化提供了统一的操作基础。此外,论文还构建了 workflow 活动定义与执行的语义模板,为特定场景下的 workflow 设计提供了可复用的框架,显著提高了模型传递的自动化水平。

## 2 工作展望

论文的研究工作针对复杂装备系统的 MBSE 模型传递技术研究,进行展开,它包括互操作性与自动化能力两个特征,基于这两个特征,进行基于服务的架构-仿真一体化表达方法研究,以及面向 MBSE 模型持续集成的自动化 workflow 设计方法研究,在此基础上还可以从两个方面进行完善:

(1) 增加模型驱动技术的集成。对于异构模型资源的集成,是指模型以及对模型的演化操作(例如查询与修改)的集成,可以通过 OSLC 集成规范对模型和模型操作接口进行统一描述,在数据层实现了异构模型之间的数据互操作,这种方法具有较好的扩展性与可操作性;但是对于模型驱动的技术集成,例如模型检查、模型转换、仿真执行这些需要领域工具支持的活动集成,基于 OSLC 规范采用开发点到点的工具接口进行集成与调用,只支持模型数据点到点的传递。本文只面向系统的架构设计与仿真验证阶段,讨论了系统架构建模与仿真建模工具的模型驱动技术的集成,后续为将 MBSE 模型传递扩展至全生命周期中的多个阶段,需要研究实现集成更多模型驱动的技术。

（2）研究 workflow 活动元模型的构建。本研究在基于流水线脚本规范，定义了 workflow 各活动的语义模板后，通过明确语义模板的参数输入可以形成流水线脚本。结合基于模型的系统工程思想，后续将研究如何将 workflow 各活动的语义模板封装为各活动的元模型，通过对元模型进行参数赋值，可以得到各活动对应的完整流水线脚本代码。在此基础上，研究并构建 workflow 建模平台，支持使用拖拽式的建模方式显式地表达 workflow 模型。开发 workflow 模型到流水线脚本的转换引擎，建模完成后支持模型自动转化为 workflow 的执行脚本。这个方法可以大大降低了流水线脚本的撰写难度，实现从基于文本的设计方式向基于模型的设计方式转型。

## 参考文献

- [1] 陶飞, 刘蔚然, 刘检华, 等. 数字孪生及其应用探索[J]. 计算机集成制造系统, 2018, 24(01): 1-18.
- [2] 王崑声, 袁建华, 陈红涛, 等. 国外基于模型的系统工程方法研究与实践[J]. 中国航天, 2012, 13(11): 52-57.
- [3] 朱静, 杨晖, 高亚辉, 等. 基于模型的系统工程概述[J]. 航空发动机, 2016, 42(04): 12-16.
- [4] 韩凤宇, 林益明, 范海涛. 基于模型的系统工程在航天器研制中的研究与实践[J]. 航天器工程, 2014, 23(03): 119-125.
- [5] Friedenthal S, Griego R, Sampson M. INCOSE model based systems engineering (MBSE) initiative[C]//INCOSE 2007 symposium. 2007: 11-15.
- [6] 蒋彩云, 王维平, 李群. SysML:一种新的系统建模语言[J]. 系统仿真学报, 2006, 22(06):1483-1487+1492.
- [7] 赵建军, 丁建完, 周凡利, 等. Modelica 语言及其多领域统一建模与仿真机理[J]. 系统仿真学报, 2006, (S2): 570-573.
- [8] 吴颖, 刘俊堂, 郑党党. 基于模型的系统工程技术探析[J]. 航空科学技术, 2015, 26(09): 69-73.
- [9] 卢志昂, 刘霞, 毛寅轩, 等. 基于模型的系统工程方法在卫星总体设计中的应用实践[J]. 航天器工程, 2018, 27(03): 7-16.
- [10] Lu J. Research survey on model-based systems engineering tool-chain[R]. Technical Report, 2019.
- [11] 贾晨曦, 王林峰. 国内基于模型的系统工程面临的挑战及发展建议[J]. 系统科学学报, 2016, 24(04): 100-104.
- [12] 毕强, 朱亚玲. 元数据标准及其互操作研究[J]. 情报理论与实践, 2007, (05): 666-670.
- [13] Chen R, Wang G, Wu S, et al. A Service-oriented Approach Supporting Model Integration in Model-based Systems Engineering[C]//2023 IEEE International Systems Conference (SysCon). Vancouver, BC, Canada, 2023: 1-7.



- [14]Badache N, Roques P. Capella to SysML bridge: A tooled-up methodology for MBSE interoperability[C]//9th European Congress on Embedded Real Time Software and Systems (ERTS 2018). 2018.
- [15]E. Huang, R. Ramamurthy, and L. F. McGinnis. System and simulation modeling using sysml[C]//2007 Winter Simulation Conference. 2007: 796–803.
- [16]R. Chen, Y. Liu, and X. Ye, Ontology based behavior verification for complex systems[C]//2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, 2018: 51739.
- [17]J. Lu, J. Ma, X. Zheng, et al. Design ontology supporting model-based systems engineering formalisms[J]. IEEE Systems Journal, 2021: 1-12.
- [18]S. Kwon, L. V. Monnier, R. Barbau, et al. Enriching standards-based digital thread by fusing as-designed and as-inspected data using knowledge graphs[J]. Advanced Engineering Informatics, 2020, 46: 101-102.
- [19]N. S. Zadeh, L. Lindberg, J. El-Khoury, et al. Service oriented integration of distributed heterogeneous it systems in production engineering using information standards and linked data[J]. Modelling and Simulation in Engineering, 2017.
- [20]Wu Y, Mao Y, Xu L (2022) FMI-based co-simulation method and test verification for tractor power-shift transmission. PLoS ONE 17(2): e0263838.
- [21]Riccobene E, Scandurra P. Integrating the SysML and the SystemC-UML Profiles in a Model-Driven Embedded System Design Flow[J]. Design automation for embedded systems, 2012, 16: 53-91.
- [22]Carpanzano, Andrea. Propulsion Subsystem Design through Model-Based and Data-Driven System Engineering automated toolchain. 2022.
- [23]J. Lu, J. Wang, D. Chen, et al. A Service-Oriented Tool-Chain for Model-Based Systems Engineering of Aero-Engines[J]. IEEE Access, 2018, 6: 50443-50458.
- [24]Vepsäläinen T, Kuikka S. Integrating model-in-the-loop simulations to model-driven development in industrial control[J]. Simulation, 2014, 90(12): 1295-1311.

- [25]Lu J, Chen D, Wang G, et al. Model-based systems engineering tool-chain for automated parameter value selection[J]. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2021, 52(4): 2333-2347.
- [26]J. Lu, G. Wang and M. Törngren. Design Ontology in a Case Study for Cosimulation in a Model-Based Systems Engineering Tool-Chain[J]. IEEE Systems Journal, 2020, 14(1): 1297-1308.
- [27]Shani U, Franke M, Hribernik K A, et al. Ontology mediation to rule them all: Managing the plurality in product service systems[C]//2017 Annual IEEE International Systems Conference (SysCon). 2017: 1-7.
- [28]D. Bilic, E. Brosse, A. Sadovykh, et al. An Integrated Model-Based Tool Chain for Managing Variability in Complex System Design[C]//2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Munich, Germany, 2019: 288-293.
- [29]Pavalkis S. Towards Industrial Integration of MBSE into PLM for Mission-Critical Systems[C]//INCOSE International Symposium. 2016, 26(1): 2462-2477.
- [30]Behrouz Alizadeh Mousavi, Cathal Heavey, Radhia Azzouz, Hans Ehm, et al. Use of Model-Based System Engineering methodology and tools for disruption analysis of supply chains: A case in semiconductor manufacturing[J]. Journal of Industrial Information Integration, 2022, 28: 100335.
- [31]徐州. 基于 MBSE 方法进行民机设计的工具链建设[J]. 航空制造技术, 2017(05):100-104.
- [32]朱睿, 胡峻豪, 李荣强, 等. 航空产品 MBSE 研发模式及平台框架研究[A]. 中国仿真学会, 第三十三届中国仿真大会论文集[C]. 航空工业成都飞机设计研究所;: 2021: 9-14.
- [33]陈国栋, 罗省贤.Scrum 敏捷软件开发方法实践中的改进和应用[J].计算机技术与发展,2011,21(12):97-99+104.
- [34]李强, 成俊峰. 持续集成在汽车软件开发环境中的应用[J]. 汽车电器, 2023(12): 44-45+48.

- [35]王帅, 蓝启亮, 陈聪, 等. 基于汽车嵌入式软件的持续集成和持续测试分析[J]. 汽车实用技术, 2023, 48(10): 156-162.
- [36]单立彬, 张浩, 张宏扬, 等. 基于 Jenkins 的高铁列控仿真自动测试系统设计与实现[J/OL]. 铁道标准设计: 1-8[2024-03-25].
- [37]杨文远, 黄卫, 赵鑫. 面向移动平台软件开发的持续集成系统设计与实现[J]. 无线互联科技, 2023, 20(22): 34-37.
- [38]郭永和, 闫龙川, 白东霞, 等. 面向电力信息系统的分布式自动化测试集成平台[J]. 网络安全技术与应用, 2022, (07): 103-106.
- [39]尹伟, 韩光辉, 肖前远, 等. 面向民用飞机的复杂航电系统软件研制与管理方法[J]. 航空工程进展, 2023, 14(04): 158-167.
- [40]李昌, 蒋友毅, 宋雁翔, 等. 航空机载软件测试工具链设计与应用[J]. 计算机测量与控制, 2019, 27(06): 55-61.
- [41]张健, 周乃春, 李明, 等. 面向航空航天领域的工业 CFD 软件研发设计[J]. 软件学报, 2022, 33(05): 1529-1550.
- [42]赵辉, 王开阳, 江云松, 等. 基于工厂模式的 OSLC 数据集成接口设计与实现[J]. 空间控制技术与应用, 2021, 47(02): 73-79.
- [43]B. Combemale and M. Wimmer, Towards a model-based DevOps for cyber-physical systems[C]//Proc. 2nd Int. Workshop Softw. Eng. Aspects Continuous Develop. New Paradigms Softw. Prod. Deployment, 2019: 1-11.
- [44]Sanford Friedenthal and Christopher Oster. Architecting Spacecraft with SysML: A Model-based Systems Engineering Approach[M]. Createspace Independent, 2017.
- [45]Thomas Vosgien et al. A Federated Enterprise Architecture and MBSE Modeling Framework for Integrating Design Automation into a Global PLM Approach. In 14th IFIP Int. Conf. on Product Lifecycle Management (PLM), July 2017.
- [46]Jeremiah Crane et al. Mbse for sustainment: A case study of the air force launch and test range system (ltrs). In AIAA SPACE and Astronautics Forum and Exposition, page 5302, 2017.

- [47]Awele I Anyanahun and William W. Edmonson. Inter-satellite communication MBSE design framework for small satellites[C]//Annual IEEE International Systems Conference. SysCon, Montreal, QC, Canada, 2017: 1-7.
- [48]Christophe Ponsard. Assessing IT Architecture Evolution using Enriched Enterprise Architecture Models[OL]. [http://soft.vub.ac.be/benevol2019/papers/BENEVOL\\_2019\\_paper\\_17.pdf](http://soft.vub.ac.be/benevol2019/papers/BENEVOL_2019_paper_17.pdf), 2019.
- [49]D'Ambrosio J, Adiththan A, Ordoukhanian E, et al. An MBSE Approach for Development of Resilient Automated Automotive Systems[J]. Systems, 2019, 7(1): 1.
- [50]Benoit Combemale, Manuel Wimmer. Towards a Model-Based Devops for Cyber-Physical Systems[C]//Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment. Switzerland, 2020: 84-94.
- [51]Viktor Kravchenko. An example of model-centric engineering environment with Capella and CI/CD. Capella Days Conference, 2021.
- [52]Sébastien Dupont, Guillaume Ginis, Mirko Malacario, et al. Incremental common criteria certification processes using devsecops practices[C]//European Symposium on Security and Privacy Workshops, Vienna, Austria. 2021: 12-23.
- [53]Andrey Sadovykh, Gunnar Widforss, Dragos Truscan, et al. Veridevops: Automated Protection and Prevention to Meet Security Requirements in Devops[C]//Design, Automation & Test in Europe Conference & Exhibition, Grenoble, France. 2021: 1-5.
- [54]马世骏, 王如松. 社会-经济-自然复合生态系统[J]. 生态学报, 1984, (01):1-9.
- [55]常绍舜. 从经典系统论到现代系统论[J]. 系统科学学报, 2011, 19(03): 1-4.
- [56]张结. 应用 IEC61850 实现产品互操作性的思考[J]. 电力系统自动化, 2005, (03): 90-94.
- [57]李军怀, 周明全, 耿国华, 等. XML 在异构数据集成中的应用研究[J]. 计算机应用, 2002, (09): 10-12.
- [58]高静, 段会川. JSON 数据传输效率研究[J]. 计算机工程与设计, 2011, 32(07): 2267-2270.

- [59] Mehrdad Saadatmand, Alessio Bucaioni. OSLC Tool Integration and Systems Engineering -- The Relationship between the Two Worlds[C]//2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications. Verona, Italy, 2014: 93-101.
- [60] 王开阳. 基于工厂模式和 OSLC 的数据集成技术研究与应用[D]. 陕西: 西安电子科技大学, 2021.
- [61] 杜方, 陈跃国, 杜小勇. RDF 数据查询处理技术综述[J]. 软件学报, 2013, 24(06): 1222-1242.
- [62] Ryman A G, Le Hors A, Speicher S. OSLC Resource Shape: A language for defining constraints on Linked Data[J]. LDOW, 2013, 996: 96.
- [63] 鲁金直, 王国新, 阎艳, 等. 基于多架构建模语言的系统工程建模方法[J]. 系统工程学报, 2023, 38(02).
- [64] Fritzson P, Engelson V. Modelica -- A Unified Object-Oriented Language for System Modeling and Simulation[C]//ECOP'98 Object-Oriented Programming: 12th European Conference Brussels. Belgium, 1998: 67-90.
- [65] 王春晓. 基于数字孪生的数控机床多领域建模与虚拟调试关键技术研究[D]. 山东大学, 2019.
- [66] 赵建军, 丁建完, 周凡利, 等. Modelica 语言及其多领域统一建模与仿真机理[J]. 系统仿真学报, 2006, 18(z2): 570-573.
- [67] 魏永利. 数控机床数字孪生虚实一致性模型构建与使用方法[D]. 山东: 山东大学, 2022.
- [68] 吴义忠, 刘敏, 陈立平. 多领域物理系统混合建模平台开发[J]. 计算机辅助设计与图形学学报, 2006, (01): 120-124.
- [69] 黄华, 周凡利. Modelica 语言建模特性研究[J]. 机械与电子, 2005: 62-65.
- [70] 于涛, 曾庆良. 基于仿真建模语言 Modelica 的多领域仿真实现[J]. 山东科技大学学报(自然科学版), 2005: 13-16.
- [71] Fritzson P. Introduction to modeling and simulation of technical and physical systems with Modelica[M]. John Wiley & Sons, 2011.

- [72]Wetter M. Modelica-based Modelling and Simulation to Support Research and Development in Building Energy and Control Systems[J]. Journal of Building Performance Simulation, 2009, 2(2): 143-161.
- [73]Jones D, Snider C, Nassehi A, et al. Characterising the Digital Twin: A systematic literature review[J]. CIRP journal of manufacturing science and technology, 2020, 29: 36-52.
- [74]Walach H, Buchheld N, Bütünmüller V, et al. Measuring mindfulness—the Freiburg mindfulness inventory (FMI)[J]. Personality and individual differences, 2006, 40(8): 1543-1555.

## 攻读学位期间发表论文与研究成果清单

### 发表论文

- [1] 本人一作. A Service-oriented Approach Supporting Model Integration in Model-based Systems Engineering[C]//2023 IEEE International Systems Conference (SysCon), Vancouver, BC, Canada, 2023: 1-7. (EI 收录)

### 科研项目

- [1] 复杂产品设计制造服务一体化建模理论 (科技部项目)
- [2] 国家人工智能产教融合创新平台“揭榜挂帅”项目

### 科研竞赛

- [1] 第三届启航杯 MBSE 建模大赛 高校组一等奖
- [2] 第十三届“挑战杯”中国大学生创业计划竞赛 校级金奖
- [3] 第八届中国国际“互联网+”大学生创新创业大赛 校级银奖
- [4] 2021 年地区高校大学生优秀创业团队一等奖

### 国际研讨会

- [1] OSLC FEST 2022 (OSLC 最具影响力年会, 本人汇报)

## 致谢

回首在北理工的三年，是我收获巨大成长的三年。借此机会，我想向一直帮助我、鼓励我的各位老师，同学，朋友表示衷心的感谢。

首先感谢我的导师王国新老师，感谢王老师在我学习和生活上的耐心指导。每当我学业与生活出现压力时，王老师总能用他过来人的经验与智慧帮我拨开乌云。王老师博学多识，在科研与治学上追求严谨与平实，给我树立了好的榜样。其次感谢鲁金直博士对我科研道路上的帮助，是鲁博士教会了我如何按时按质按量完成一项任务。

然后，感谢我的师兄吴绶玄师兄。作为我们组研究 OSLC 方向的唯二两人，是师兄带我走进了 OSLC 的大门，培养了我学术写作、代码、绘图等各方面的能力。和师兄每一次的开会交流都让我学到新的知识、拓宽新的思路、解决旧的疑惑。如果没有吴绶玄师兄，我将无法完成会议论文的写作，也无法在三年之后顺利毕业，感谢你！

感谢系统工程小组的各位师兄师姐师弟师妹，马君达师兄、靳益利师姐、栗寒师兄、陈婧琦师姐、丁洁师姐、李子航师兄、龚逸辉师兄，他们榜样的力量和对我的帮助是我科研路上前进的动力。感谢董梦如、刘智清师妹、李嘉伟师弟、张臻师弟、张浩轩师弟，他们在我学习和生活中的帮助让我感到无比温暖。与他们的朝夕相处给我留下了宝贵的回忆。

感谢我的室友房浩楠、张梦迪、李媛媛，他们善良、可爱、活泼、宽容，谢谢你们！感谢我实验室课题组的同门，于敬丹、魏竹琴、张楚雷、闫愿、李冠楠、孙延劭、欧阳晗青、赵玄德，谢谢你们给我鼓励、带给我欢乐！感谢姜凌波同学，陪我度过研究生期间最后的半年时光。感谢北京理工大学心理咨询中心张颖老师，是您每一次的倾听与包容让我觉得安心与安全，期待与您的每一次相遇。感谢我的家人，没有他们的一路陪伴和感情支持，我无法顺利的完成学业。感谢课题组、学校和国家发的补助，让我可以没有后顾之忧安心地读书。

谨此感谢一路上所有砥砺我前行的亲人、老师、同学和朋友们！谢谢大家！