



南京大學

研究生毕业论文 (申请工程硕士学位)

论 文 题 目 _____
(论文长标题第一行)

(论文长标题第二行)

作 者 姓 名 _____
张富利

学 科、专 业 名 称 _____
软件工程

研 究 方 向 _____
DevOps、区块链

指 导 教 师 _____
张贺 教授

2021 年 12 月 14 日

学 号：**MG1932016**

论文答辩日期：xxxx 年 xx 月 xx 日

指导教师： (签字)

by

Fuli Zhang

Supervised by

Professor He Zhang

A dissertation submitted to
the graduate school of Nanjing University
in partial fulfilment of the requirements for the degree of
MASTER OF ENGINEERING
in
Software Engineering



Software Institute
Nanjing University

December 14, 2021

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：_____

软件工程 专业 2019 级工程硕士生姓名：张富利
指导教师（姓名、职称）： 张贺 教授

摘 要

领域驱动设计是一种针对复杂业务流程的软件设计方法，它可以帮助架构师和软件开发人员提炼业务流程和构建复杂软件系统。近年来，越来越多的团队将领域驱动设计应用到大型分布式系统的设计与实现中。

然而，作为领域驱动设计的核心内容，战术建模的应用却存在诸多问题与挑战。其一，战术建模模式的实践过程仍缺少标准且统一的规范，使得实践者无法准确地使用这些模式建模业务流程；其二，战术建模实践过程所缺少的规范，也使得开发人员和架构设计人员在使用这些建模模式时具有不同的理解，从而带来了一系列项目团队沟通问题；其三，现有的大多数建模平台对领域建模过程的支持仍不完善，极大地阻碍了领域战术建模的应用。

为了解决上述问题与挑战，本文首先调研了现阶段有关领域驱动设计战术建模的理论文献，对具有相关实践经验的从业者进行访谈，提炼出一系列面向战术建模过程的实践规范作为战术建模指南。所提炼的建模指南包含一系列战术建模中使用到的战术模式、这些战术模式的属性、使用时机与实现技术。然后，基于 UML 扩展机制定义标准的领域驱动设计战术建模元模型作为战术建模语言。上述建模指南和战术建模语言共同组成本文所提出的战术建模支持方法。最后，基于所提出的战术建模支持方法，本文还实现了一个可视化战术建模工具 DDDD (Draw for Domain-Driven Design)，该工具支持标准化的战术建模、对建模结果的验证、复用以及扩展，帮助实践者更规范地进行战术建模。

本文采用案例研究对提出的建模支持方法及工具进行验证。结果表明本文提出的战术建模支持方法及工具，能够帮助领域驱动设计的实践者开展更规范化战术建模过程，从而解决战术建模实践过程缺少规范带来的一系列挑战，一定程度上促进了战术建模的应用。

关键词：领域驱动设计；战术建模；建模工具；特定领域建模语言

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Modeling Support Method and Tool for
Domain-Driven Design

SPECIALIZATION: Software Engineering

POSTGRADUATE: Fuli Zhang

MENTOR: Professor He Zhang

Abstract

As a software design approach for complex business process, Domain-Driven Design (DDD) enables architects and developers building complex software system. In recent years, more and more teams have applied DDD to the design and implementation of large-scale distributed systems.

However, as the core means of DDD, tactical modeling are facing many challenges in its application. Firstly, the practicing process of tactical modeling patterns still lacks standard and unified specifications, making it hard for practitioners to use these patterns to model business processes accurately; secondly, the lack of specifications also makes developers and architects have different understandings when using the modeling patterns, leading to a series of communication problems in project team; thirdly, most of the existing modeling platforms have incomplete support for the tactical modeling process, severely impeding the application of DDD.

In order to solve the aforementioned challenges, this thesis first reviews theoretical literature on tactical modeling, then interviews practitioners with relevant practical experience, and finally extracts a series of practical specifications for the tactical modeling process. As a guide for tactical modeling, the specifications contain a series of tactical patterns used in tactical modeling, their attributes, time of using, and implementation techniques. Furthermore, the standard tactical modeling metamodel is defined based on UML profile mechanism, as a tactical modeling language. The above modeling guide and tactical modeling language together constitute the tactical modeling support method proposed in this thesis. Based on the proposed method, this thesis also implements a visual tactical modeling tool called Draw for Domain-Driven Design, or DDDD. The tool

supports standardized tactical modeling, verification, reuse and extension of modeling results to help practitioners develop tactical modeling in a more standardized way.

This thesis uses case studies to verify the proposed modeling support method and tool. Results show that the proposed method and tool can help DDD practitioners to conduct a more standardized tactical modeling process, thus solving the challenges caused by lack of specification in the tactical modeling process, and promoting the application of tactical modeling to a certain extent.

keywords: Domain-driven design, Tactical modeling, Modeling tool, Domain-specific modeling language

目 录

目 录	iv
插图清单	vii
附表清单	ix
第一章 绪论	1
1.1 研究背景及意义	1
1.2 国内外研究现状	3
1.3 本文主要研究工作	5
1.4 本文组织结构	6
第二章 理论与技术支持	7
2.1 领域驱动设计	7
2.1.1 战略建模相关概念	7
2.1.2 战术建模相关概念	9
2.2 建模语言	11
2.3 特定领域建模语言构建方法	13
2.4 建模支持工具实现技术	15
2.5 本章小结	18
第三章 面向领域驱动设计的战术建模支持方法	19
3.1 研究方法概述	19
3.2 研究过程	20
3.2.1 文献综述	20
3.2.2 访谈	21
3.2.3 焦点小组	22
3.3 战术建模指南	23
3.3.1 战术建模模式及属性	23
3.3.2 战术建模模式使用时机与实现技术	25
3.3.3 战术建模指南构建结果	28
3.4 战术建模语言	29

3.4.1 战术建模语言元素集	30
3.4.2 战术建模元类	31
3.4.3 战术建模语言约束	31
3.5 本章小结	33
第四章 建模支持工具设计与实现	34
4.1 概述	34
4.2 需求分析	35
4.2.1 工具总体功能	35
4.2.2 可视化建模模块需求分析	36
4.2.3 模型校验模块需求分析	37
4.2.4 模型存储与转化模块需求分析	39
4.3 工具设计与实现	40
4.3.1 总体设计	40
4.3.2 可视化建模模块详细设计与实现	42
4.3.3 模型校验模块详细设计与实现	44
4.3.4 模型存储与转化详细设计与实现	47
4.4 本章小结	49
第五章 建模支持工具测试与案例研究	50
5.1 建模支持工具测试	50
5.1.1 可视化建模测试	50
5.1.2 建模约束校验测试	51
5.1.3 模型存储测试	52
5.1.4 生成项目测试	53
5.1.5 性能测试	53
5.2 案例研究	54
5.2.1 验证步骤	55
5.2.2 验证过程	55
5.3 本章小结	59
第六章 总结与展望	60
6.1 总结	60
6.2 展望	61
致 谢	62

参考文献	63
A 访谈问题	67
A.1 战术建模访谈问题	67
简历与科研成果	72

插图清单

1-1 战术建方法及支持工具	3
2-1 战术模式关系图 ^①	9
2-2 特定领域建模语言提出方法.....	13
2-3 MOF 四层架构与 UML 类图 ^④	14
2-4 UML profile 扩展类图实例	14
2-5 两种建模工具实施方式	15
2-6 响应式设计追踪变化	17
3-1 战术建模支持方法研究过程图	20
3-2 焦点小组实施过程	23
3-3 领域事件实现流程图 ^①	27
3-4 战术建模指南	29
3-5 战术建模扩展元类 profile	31
3-6 DomainEvent 约束实现	32
4-1 建模支持工具业务流程	35
4-2 工具总体功能	35
4-3 可视化建模模块用例图	36
4-4 模型校验模块用例图	38
4-5 模型存储与转化模块用例图	39
4-6 支持工具整体架构图	41
4-7 支持工具前端整洁架构图	42
4-8 可视化建模模块类图	43
4-9 可视化建模模块顺序图	43
4-10 初始化 ToolBar 代码	44
4-11 模型校验模块类图	45
4-12 模型校验部分实现代码	47

4-13 模型存储与转化类图	48
4-14 模型存储与转化顺序图	48
4-15 模型存储部分实现代码	49
5-1 图形化建模测试结果图	51
5-2 建模约束校验测试结果图	52
5-3 工具帮助页面图	55
5-4 工具主页	56
5-5 可视化建模过程	56
5-6 模型校验	57
5-7 保存导出模型	57
5-8 加载历史模型	58
5-9 建模结果图	58

附表清单

3-1 受访者信息	21
3-2 访谈关注问题	22
3-3 战术建模模式及属性	24
3-4 战术模式使用时机	25
3-5 战术模式包含内容	28
3-6 战术建模 profile 元素集	30
3-7 构造型约束规则	31
4-1 可视化建模用例表	37
4-2 模型校验用例表	38
4-3 模型存储与转化用例表	39
5-1 可视化建模测试用例	51
5-2 建模约束校验测试用例	52
5-3 模型存储测试用例	52
5-4 生成项目测试用例	53
5-5 测试环境	53
5-6 接口性能测试	54
5-7 案例验证结果	59

第一章 絮论

1.1 研究背景及意义

在应对大型复杂软件系统的设计与实现时，越来越多的企业选择借鉴领域驱动设计中的思想与方法。领域驱动设计（Domain-Driven Design, DDD）是一种针对大型复杂软件系统的分析与建模方法，由 Eric Evans 于 2004 年在其著作《领域驱动设计：软件核心复杂性应对之道》[1] 中首次提出。与传统的系统设计方式相比，领域驱动设计更加聚焦领域中的业务流程而非系统操作的数据，强调领域模型的重要性。领域驱动设计遵循关注点分离的原则，对领域对象进行了明确的策略和职责划分，可以将现实业务映射到领域对象上，为领域专家与开发人员搭建沟通的桥梁。通过领域驱动设计，开发团队和领域专家 [1] 深入交谈，对业务了解更加准确，领域专家借助领域驱动设计的方法与开发团队进行沟通，可以更好地对领域进行建模，并对建模的领域对象进行快速迭代修改，同时将修改同步到实现层次上，更加适应快速迭代的敏捷软件开发节奏。

领域驱动设计主要被分为战略建模（Strategic Modeling）和战术建模（Tactical Modeling）两个层次 [2]，战略建模层次指导架构师及领域专家构架大型复杂软件系统并进行拆分，战术建模层次指导开发者对拆分出的子系统进行落地。战略建模层次的主要思想与方法十分契合当今流行的微服务架构（Microservice Architecture, MSA）[3]，且使用者多为经验丰富的架构师及领域专家，所以战略建模能在高层次业务流程分析与系统划分时发挥作用；战术建模层次则更加侧重于单个微服务或微服务中某一模块的落地，并给出建模与实现过程中应该遵循的原则与约束。战术建模是基于战略建模的结果，进行进一步抽象设计，充当战略建模与最终技术实现的中间过渡阶段。战术模式（Tactical Patterns）是战术建模中常用的表达业务问题的一种方式。战术建模相比战略建模更加依赖领域模型（Domain Model）[4] 和通用语言（Ubiquitous Language）[4]，因为其需要通过战术模式将领域模型和通用语言中的概念映射到代码实现中，代码实现也会随着领域模型的变化而重构，体现领域模型的改变。战术建模还可以借助标准通用的战术模式，让开发人员甚至非技术人员与

架构师进行合作，参与到建模过程中来。

上述领域驱动设计的两个层次都强调领域模型的重要性。领域模型（Domain Model）是对领域中概念或对象的可视化表示，专注于领域本身，可以用来分析业务，发掘重要的业务领域概念，并建立业务领域概念之间的关系。领域模型通常以从业务流程中分析提炼出的通用语言为基础进行构建，并作为通用语言的核心。通用语言（Ubiquitous Language）是将团队沟通与软件实现紧密联系到一起的一种基于模型的语言，在应用通用语言进行领域建模时，领域专家和开发人员可以更好地讨论需求、制定开发计划和描述系统特性。

然而，领域驱动设计，特别是战术建模层次在领域建模实践时却面临许多挑战。第一，战术建模中每种模式都是知识高度凝练的结果，包含了复杂的概念知识，学习成本很高，没有大量的实践经验基本无法理解。在建模实践时，难以准确把握各种模式的规范和约束，无法正确地将战术建模模式与业务流程中的业务对象对应，导致战术建模结果不标准、不统一，甚至战术建模结果错误，难以发挥战术建模的优势；第二，开发人员和架构师在使用战术建模时对业务的理解程度不同，对模式的应用时机和实现技术看法不同，导致架构师和开发人员无法在建模时进行顺畅地沟通，不同开发人员的开发水平、经验也有差距，同一开发团队也往往很难让建模结果保持标准和统一；第三，实施战术建模没有规范化的标准流程指导，当前的一些建模工具（如 Enterprise Architect^①、PlantUML^②）对领域建模规则与约束的支持也不够完善，无法建立起足够通用的模型，最终导致战术建模的结果无法规范化、标准化地应用。上述挑战阻碍了开发者正确使用领域驱动设计战术建模。

为了解决上述挑战，本文提出了一种面向领域驱动设计的战术建模支持方法及工具，具体工作内容如图 1-1 所示。战术建模方法包括一套标准化的**战术建模指南**，用于帮助理解不同模式的特征属性、使用时机以及实现技术，作为领域建模的理论支撑指导建模过程；根据战术建模指南，本文还基于 UML 的 profile 扩展机制实例化**战术建模语言**，用于规范化使用战术建模模式进行领域建模的流程；基于所提出的战术建模方法，本文还实现了一种**战术建模支持工具**，支持可视化建模、验证建模结果以及存储和扩展建模结果，为战术建模支持方法提供可视化方式展现。

^①Enterprise Architect 官网：<https://sparxsystems.com>

^②plantUML 官网：<https://plantuml.com>

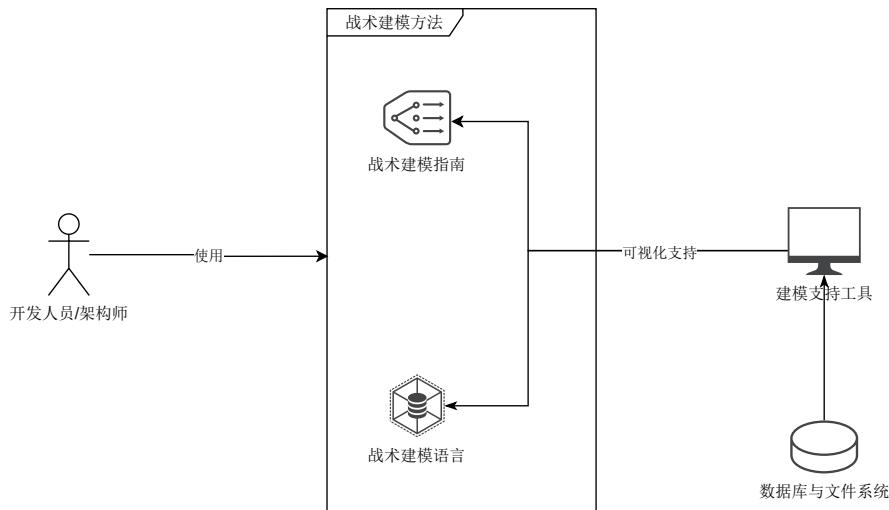


图 1-1: 战术建方法及支持工具

1.2 国内外研究现状

领域驱动设计自提出以来，受到了学术界与工业界的广泛关注。战略建模层次的实践与实现代码关联不紧密，更多关注的是顶层设计与架构搭建，实施者一般也拥有较多的开发经验与较高的建模水平，所以在国内外都能较好地进行落地。但战术建模层次更加侧重于建模的设计过程和具体实现，对于建模实施者的水平有较高的要求，还需要通过统一标准化的流程来达到建模结果的准确性，规范化的建模语言也是支持战术建模成功实施的必要条件。所以，一套完整的战术建模支持方法及工具就显得格外重要。

特定领域建模语言（Domain Specific Modeling Language）[5]是一种专注于某个特定领域，结合了特定领域知识和概念的建模语言。可以使用特定领域建模语言来进行战术建模，从而获得统一标准化且具有特定领域特征的建模结果。目前国内外研究人员对特定领域建模语言的定义方法了一些研究。Hao Wu 等人 [6] 指出，可以通过 UML 结合对象约束语言（Object Constraint Language, OCL）的方式来扩充医疗系统领域现有元模型，形成一种更为完整的建模语言及方法。Kühlwein 等人 [7] 通过一种分层架构的思想来打通物联网领域的多种元模型，构建了一种平台型的特定领域建模语言，其中每层都有针对不同设备的元模型，通过该平台进行组织和交互，这种定义方法需要依赖较为成熟的元模型；Jumagaliyev 等人 [8] 通过对云存储平台的通用存储类型进行分析，抽象出数据类型的特征属性，创建元模型并使用 EuGENia[9] 进行注释，最终借助 Eclipse IDE 将创建的元模型转换为具体的图形建模框架编辑器，不仅定义了特

定于云存储平台的元模型，还提供了建模工具编辑器。

国内外研究人员也对领域驱动设计建模语言的定义进行了一些研究。Florian Rademacher 等人 [10] 根据领域驱动设计实践经验提出使用 UML profile 定义元模型，并将其应用到微服务领域中去。Florian Rademacher 等人调研了大量领域建模的 UML 图，确定了可以描述领域模型的 UML 类图的构造方法。通过扩展元类的方式，达到了使用领域驱动设计战术建模实现微服务系统建模的目的，并为验证模型有效性和实现微服务代码奠定了基础。同样的，Hippchen 等人 [11] 也强调了微服务化拆分中应用面向领域驱动设计建模语言的好处。Florian Rademacher 等人强调了领域驱动设计概念在 UML 中缺少正式的定义，阻碍了模型的验证和转化，但同时 UML 又符合软件设计领域建模的要求，应用也十分广泛，故以 UML 为基础进行扩展，定义出了一套新的针对领域驱动设计相关规则和约束的建模语言。Andreas Diepenbrock 等人 [12] 提出使用本体论（Ontology）[13] 来定义针对微服务应用的领域驱动设计元模型，该研究主要使用了本体论在特定领域表达知识语义的功能，结合领域驱动设计实现了建模的目的。

以上工作说明任何特定领域建模语言的定义与提出都需要前期大量领域实践知识的总结，并且在已有工作成果的基础上进行元模型的定义效率更高。由于领域驱动设计战术建模理论较为成熟，目前定义的战术模式特征明确，所以，本文将基于领域驱动设计战术建模的现有研究成果，结合实际应用情况，进行修改和优化，定义一套新的领域驱动设计战术建模语言。

除了定义元模型来实现建模语言之外，提供支持建模语言的工具也十分重要。刘辉等人 [14] 提出元建模工具的实现主要分为两种途径。第一种是使用配置文件扩展通用建模工具，让通用建模工具支持特定领域的元模型从而支持特定领域建模语言。Guerriero 等人 [15] 通过 UML 的 profile 扩展机制，扩展了现有 UML 的语法元模型，达到创建新的元模型的目的，扩展 UML 语法元模型的实现方式只依赖于配置文档，有利于多种建模方法的集成，但需要借助统一建模语言的实现平台，如 MetaEdit+^③。第二种是通过建模工具生成器根据元模型直接生成相应的建模工具，La Fosse 等人 [16] 使用 GEMOC（一种基于 Eclipse 的特定领域语言创建平台）来创建扩展元模型，生成了新的建模语言及支持工具，这种方式定制化程度更高，可以提供独立的工具，但生成的建模工具依赖工具生成平台，如 EMF（Eclipse Modeling Framework）^④ 提供的一系列生成元

^③MetaEdit+ 首页：<https://www.metacase.com/products.html>

^④Eclipse Modeling Framework 主页：<https://www.eclipse.org/modeling/emf/>

模型的工具。

许多国内外的研究人员对领域驱动设计建模语言的支持工具开展了实践与探究工作。Duc Minh Le 等人 [17] 定义了一种名为 DCSL 的基于注释的特定领域建模语言，通过注释约束了领域模型的特征与行为，由于 Java 语言对注释的良好支持，采用 Java 开发了一款建模语言支持工具，摆脱了建模工具的平台依赖性，但注释对模型的表达不够直观，仅仅依靠注释来表达建模过程和结果远远不够；Kapferer 等人 [18] 定义了一种战略建模语言，并提供了相应的编辑、验证和转换工具，重点关注上下文映射工作，实现支持工具借助了 PlantUML 建模平台，达到了可视化建模支持。

虽然目前已有一些关于领域驱动设计建模的探索与研究，但仍然存在诸多问题。首先，许多建模语言的提出以元模型为基础，但元模型的定义缺少理论依据，导致元模型的定义不够规范化和标准化；其次，建模语言的提出即使调研过大量文献，也缺少对学术界与工业界实际应用差别的思考，导致最终定义的建模语言脱离实际应用场景；最后，战术建模的支持工具易用性不够，或者依赖太多额外平台，学习和使用成本过高，导致难以应用到实际生产中去。总的来说，战术建模流程缺少一套标准化的建模支持方法及工具。

1.3 本文主要研究工作

本文主要的研究工作分为以下三个方面：

1. 围绕领域驱动设计的战术建模过程展开了理论调研，从《领域驱动设计：软件核心复杂性应对之道》[1] 和《实现领域驱动设计》[4] 两本著作中抽取了八种战术建模模式及其重要特征。具体地，针对八种战术建模模式，设计了调查问卷，与工业界具有领域驱动设计实战经验的架构师和开发人员展开访谈；根据访谈结果，通过多次焦点小组讨论，对八种战术建模模式及其重要特征进行验证和完善，克服了理论脱离实际的问题。最终得出一套战术建模指南，该指南包括战术建模模式、模式属性、使用时机以及实现技术。

2. 基于上述理论基础，通过 UML profile 机制扩展 UML 元类，实例化战术建模语言。战术建模语言描述了战术模式的构造型、必要属性、关联关系以及重要约束。以 UML 中元类为基础，更符合软件设计中面向对象（Object-Oriented）的思想，也更易于软件从业者接受和学习。以该元模型为基础的建模语言，更关注战术建模，包含最贴合实践的规则和约束，建模效率更高。

3. 实现了一个战术建模支持工具，该工具对建模过程中使用的战术模式进行约束与规范性校验，对建模结果进行多种格式的转化与存储，还包含生成框架项目代码包等扩展功能。对战术建模支持工具进行了功能测试，并使用该工具进行了战术建模案例研究，结果表明该工具支持开发人员快速理解各种战术模式的重要特征和规则约束，降低了使用战术建模的学习成本；可以对建模结果进行验证并提示开发人员进行修改，规范化建模过程；还具有将建模结果转化为多种格式文件和框架项目代码的功能，使建模结果更具有通用性。

上述战术建模指南、战术建模语言以及战术建模支持工具共同组成了本文研究工作的战术建模支持方法及工具。

1.4 本文组织结构

本文组织结构如下：

第一章 绪论。介绍了本文的研究背景、国内外研究现状以及本文主要的研究工作；

第二章 理论与技术支持。介绍领域驱动设计尤其是战术建模设计相关理论和概念，对建模语言原理以及支持工具实现方法进行介绍；

第三章 面向领域驱动设计的战术建模支持方法。通过文献综述、访谈和焦点小组提出战术建模支持方法，并详细介绍了该战术建模支持方法；

第四章 建模支持工具设计与实现。介绍本文如何基于第三章得到的战术建模方法实现一个建模支持工具，介绍对该工具的需求分析，工具整体架构与各模块的设计和实现；

第五章 建模支持工具测试与案例研究。对建模支持工具进行功能测试、性能测试和案例分析研究，介绍了使用建模支持工具建模的流程和效果；

第六章 总结与展望。总结本文所做的研究工作和贡献，分析工作不足之处，并对后续研究进行了展望。

第二章 理论与技术支持

本章将介绍领域驱动设计战略建模和战术建模相关概念，为构建战术建模方法做理论支持；还将介绍建模语言相关概念以及特定领域建模语言构建方法，作为构建战术建模语言的参考；最后，将介绍建模支持工具实现技术，为实现支持工具做技术支撑。

2.1 领域驱动设计

领域驱动设计与传统的建模分析方法不同，领域驱动设计不再只关注业务数据，而是从领域中的重要概念出发，将业务流程中关键对象提炼成领域模型与关键逻辑，来解决复杂业务的本质问题。由于战术建模中的概念依赖于战略建模，并存在于某个战略建模的限界上下文（Bounded Context）中，所以战略建模是战术建模实施的支撑背景。本节将从战略建模和战术建模两个层次进行介绍，重点关注战术建模中的重要概念。

2.1.1 战略建模相关概念

战略建模是从宏观角度对领域业务进行建模的建模方法，强调领域内的业务特性。战略建模可以划分出业务的边界、组织团队结构以及系统架构，为后续的战术建模划定限界上下文。使用战略建模可以很好地进行微服务化拆分[19]与系统拆分。下面将对战略建模相关重要概念进行介绍。

通用语言（Ubiquitous Language）

通用语言是 Eric Evans 在《领域驱动设计：软件核心复杂性应对之道》[1] 中提出的术语，用于开发人员和用户建立通用、无歧义的语言。该语言的基础是软件设计中的领域模型，应严格按照领域模型进行定义，保证其严谨性。本文提出的战术建模语言及支持工具都使用通用语言进行表达。

子域（Subdomain）

领域可以进一步划分为核心域（Core Domain）、通用域（General Domain）和支撑域（Support Domain）。每一个子域对应更小的问题域或业务范

围，根据其自身的功能属性进行不同划分。

核心域决定业务的核心竞争力，是最重要的子域，包含主要的业务流程。通用域是被其他多个子域共同使用的一部分，没有定制化需求，包含通用功能。支撑域最关注业务，对应某个业务中的重要部分，具有业务特定性，在不同企业业务中不通用，但必不可少。本文提出的战术建模支持方法目的在于解决某个子域的建模问题。

限界上下文（Bounded Context）

限界上下文是一种概念性边界，限定了领域模型的工作范围。每一个模型概念和其中的属性与操作，在它所属的边界之内，都有特定的含义，通过之前约定的通用语言，参与建模的团队成员应该可以明确领域模型的具体含义。对于系统架构来说，限界上下文还确定了应用边界和技术边界，提供了解决特定领域问题的建模明确边界，同时也充当了问题空间与解空间之间的桥梁。本文提出的战术建模支持方法工作范围在某个限界上下文之内，可以聚焦于特定的业务领域，使建模中的交流成本变低。

上下文映射（Context Map）

上下文映射在项目团队中共享，并确保被每个团队成员理解。通过上下文映射，可以从宏观上看到每个上下文之间的关系，能够更好地指导后续的程序设计。上下文映射不拘泥于任何形式的文档，重点在于帮助团队成员理解不同上下文之间的关系。在不同限界上下文使用本文提出的建模支持方法进行战术建模后，可以通过上下文映射进行建模结果的交流。

防腐层（Anti-Corruption Layer）

防腐层可以根据领域模型为自身所在的限界上下文提供服务。该层通过与另一个系统进行通信，几乎不需要对自身限界上下文进行任何修改。防腐层需要在两个模型之间进行必要的双向转换 [1]。防腐层不仅防止内部代码被外部逻辑侵入，还在于分离不同的领域并确保它们在将来保持分离。通过防腐层可以在战术建模支持方法中体现战略建模的底层基础作用。

战略建模首先需要确定通用语言，将要解决的领域问题划分为更细粒度的子域，并通过划定限界上下文隔离不同领域模型的工作范围；然后通过通用语言进行建模，借助上下文映射来理解不同限界上下文之间的关系，最后还可以通过防腐层来与外部系统进行沟通。

2.1.2 战术建模相关概念

本小节将描述领域驱动设计战术建模的相关概念，战术建模是本文研究工作的重点内容，战术建模支持方法基于战术建模相关概念提出。战术建模是在特定的限界上下文中进行的更细粒度的建模，可以用来管理领域模型的复杂性并确保领域模型中行为的准确性。通用语言是将团队沟通与软件实现紧密联系到一起的一种基于模型的语言。战术模式是战术建模中常用的表达业务问题的一种方式，包含领域中的抽象概念、关系和约束规则，并将领域模型和通用语言中的概念映射到实现技术中。战术建模依赖通用语言和战术模式，如图 2-1 所示为模型驱动设计战术模式关系图，展示了主要的战术模式与它们之间的关系，下面将对战术模式相关概念进行详细介绍并举例说明。

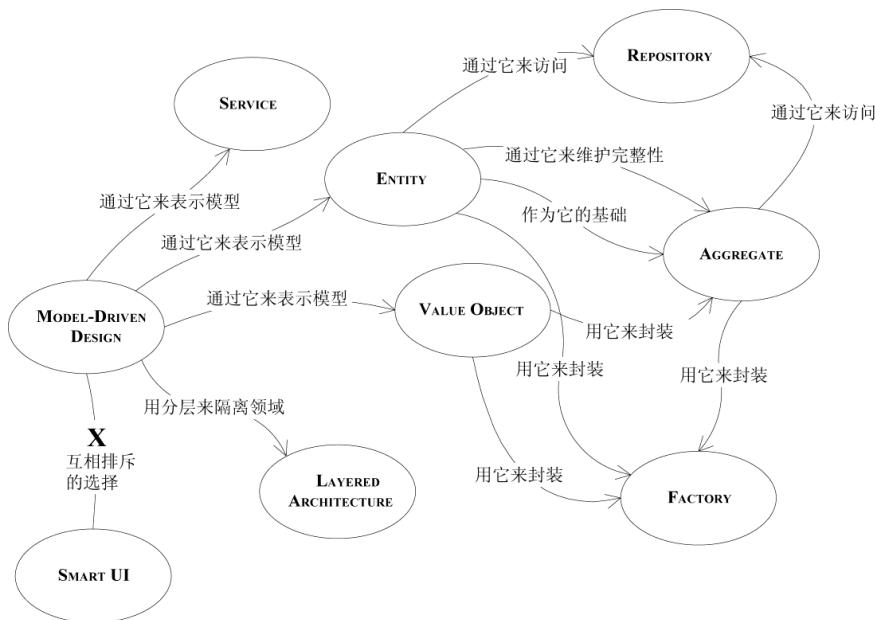


图 2-1: 战术模式关系图^①

实体 (Entity)

实体是一个由标识符定义的对象 [1]。当我们关注一个对象的个性特征，或者需要区分不同的对象时，我们使用实体这个概念，换句话说，有必要将该对象与其他对象区别开来确保其个性特征时，我们希望将其建模成实体 [4]。例如，每辆汽车都通过车牌号与其他汽车进行区分，在这种条件下汽车应该被建模为实体。

^①图片来源于《领域驱动设计：软件核心复杂性应对之道》

值对象（Value Object）

值对象的作用是度量和描述事物，值对象可以很方便地进行创建、测试、使用、优化和维护。值对象的使用比实体更加广泛，值对象一旦创建就不可变化，只可以进行替换，具有可比性 [4]。例如，一个人的家庭住址可以用来作为邮寄地址，当家庭住址改变时，可以直接将邮寄地址替换，在这种条件下家庭住址应该被建模为值对象。

领域服务（Domain Service）

领域服务是一个独立接口，负责承担不属于实体或值对象职责的操作或转换过程。领域服务可能关注一个显著的业务操作过程或领域对象的转换，在单个原子操作中处理多个领域对象，并且命名要与通用语言保持一致 [4]。例如，保险公司在计算赔偿金额时需要进行复杂的计算流程，而这种赔偿流程不是某个对象的职责，在这种条件下计算赔偿金额应该被建模为领域服务。

领域事件（Domain Event）

领域事件用来捕获发生在领域中的一些事情。领域事件记录了领域中已经发生的事情，无法改变。需要维护业务一致性时，也需要用到领域事件，该事件往往是需要发布到外部系统（如外部限界上下文）的，另外，领域事件还可以让远程依赖系统与本地系统保持一致 [4]。例如，在网购平台上订单已支付后，会触发商家发货等一系列流程，在这种条件下订单已支付应该被建模为领域事件。

聚合（Aggregate）

聚合是一个将实体和值对象聚类到一致性边界内的容器。聚合不仅仅是聚集了一些共享父类、密切关联的对象，更加重要的是其更关注内部的不变条件和整体的一致性边界 [4]。聚合根（Aggregate Root）是聚合用来标识自己的一个实体，需要追踪聚合变化时，就需要跟踪根实体，根实体也是代表聚合与外界交互的对象。例如，汽车是一种很复杂的机器，其中包含发动机、车轮、制动器等元件，这些元件统一协同通信才组成了整个汽车，在这种条件下，应该将这些元件组合到一起建模为聚合。

资源库（Repository）

资源库是一个用来存取领域对象的安全存储区域 [4]。可以通过资源库来安全地存储领域对象，并在需要时取出使用，资源库可以依据不同实现形式来存取不同的领域对象。

工厂（Factory）

工厂是一种具有创建复杂对象和聚合职责的单独对象，该对象并不承担领域模型中的职责，但是依然是领域驱动设计的一部分 [4]。创建复杂对象和聚合的逻辑经常封装在工厂中，与设计模式中的工厂相似，但不应该为创建每个对象都提供一个工厂。例如，组装汽车是一个复杂的过程，但最终产出仍然是汽车，在这种条件下，应该将创建汽车的工作建模成工厂。

模块（Module）

模块是一个命名的容器，用于存放内聚的类，并可以对不在同一个模块的类解耦 [4]。主要应用在技术层面，如代码组织，构建目录与打包。例如，汽车、飞机、轮船等交通工具在进行建模时，可以统一放在交通工具模块中。

2.2 建模语言

战术建模与实现层次关联较强，甚至有些团队直接采用代码形式来进行建模，用代码来表达领域模型，但这种方式不易于理解，在大型团队中无法很好地发挥领域模型的沟通作用。建模语言是一种描述信息或者数据模型概念的语言，可以作为模型与实现之间沟通的桥梁，UML 就是一种统一建模语言。构建元模型是实现建模语言的一种方式，元模型可以作为交换和存储数据的介质或支持特定方法的语言，UML 也是通过元模型进行实现的建模语言。通过对对象约束语言对 UML 元模型的额外约束，可以构建一种特定领域建模语言。战术建模这一特定领域长期以来缺少标准、规范的建模语言来对建模过程进行规范化和约束，随着战术建模应用越来越广泛，需要强大且合适的建模语言来进行支撑。

元模型（Metamodel）是描述模型的模型，在软件工程中使用模型来作为表达方式越来越普遍。模型的建立背后应该对应着一个元模型，模型是真实世界中现象的抽象，元模型是关注模型本身属性的一种抽象，所以可以把一个元模型看做对模型的抽象。元模型可以充当交换或存储语义数据的中间介质，可以作为支持特定方法或过程的语言，还可以作为扩展现有信息额外含义的语义语言 [20]。任何模型都应该服从其元模型的定义。目前模型驱动工程（Model Driven Engineering, MDE）最活跃的分支是 Object Management Group（OMG）^② 提出的模型驱动架构（Model Driven Architecture, MDA）解决方案 [21]。该解决方案描述了被称为元对象机制（Meta-Object Facility, MOF）的元模型结

^②OMG 组织官网：<https://www.omg.org/>

构。本文将采用元模型的方式来定义战术建模模型，战术建模语言也将以元模型为基础，元模型是战术建模支持方法中建模语言的组成元素。

面向对象建模方法是一种依靠现实生活中常用思维来认识、理解和描述客观事物的建模方法，强调最终建模的对象之间反映现实中的固有问题和关系。面向对象分析与设计方法的发展在 80 年代末 90 年代初出现了一个高潮，由此出现了许多面向对象的建模方法。其中具有代表性的如 Yourdan 等人 [22] 提出的面向对象分析（Object-Oriented Analysis, OOA）方法，Jacobson 等人 [23] 提出的面向对象软件工程（Object-Oriented Software Engineering, OOSE）方法等。UML 在此次高潮后，作为一种统一的建模语言产生。UML 是一种建模语言，而不是方法，它不包含对过程的描述，同时，UML 也是 OMG 提出的典型元模型之一。

UML 关注建模的普适性和可移植性，直接使用 UML 进行战术建模粒度不够，无法反映战术建模的一些模式及其规则和约束，会造成建模细节的损失。特定领域建模语言（Domain Specific Modeling Language）[5] 是一种专注于某个特定领域，结合了特定领域知识和概念的建模语言。分析领域的建模人员不必从头开始构建这些概念，可以直接使用具有特定领域知识和概念的建模语言来进行建模，所以，特定领域建模语言更加适合作为战术建模的基础语言。本文构建特定于领域驱动设计战术建模的建模语言，包含战术建模相关概念，使用该建模语言进行建模与战术建模目的相符。同时 UML 可以适配任何适用于自己项目类型的过程，并记录最终的分析和设计结果 [24]。因此，可以使用 UML 来作为建模结果的描述语言，使建模结果具有通用性和普遍性，UML 的 profile 扩展机制，也提供了通过元模型（Metamodel）来确定构造型、约束和特定语义的方法，从而支持特定领域的建模过程。针对于领域驱动设计战术建模，UML 元类可以通过 profile 扩展机制，表示特定的模式，如实体（Entity）、值对象（Value Object）等 [25]。本文将使用具有面向对象特征的 UML profile 扩展机制来作为元模型的实现方式，既符合领域驱动设计的领域特征，又能扩充 UML 元模型构建新的建模语言。

对象约束语言是一种施加在指定模型元素上的约束语言。对象约束语言最早是在 1995 年在 IBM^③ 内部开发的，1997 年被集成到 UML 标准中，最初的作用是对 UML 表示法的补充，OCL（Object Constraint Language）表达式能以附加在模型元素上的条件和限制来表现对该对象的约束，克服了 UML 表示法甚

^③IBM 公司主页：<https://www.ibm.com/>

至是任何图形表示法在系统详细设计时不够精确的局限性，OCL 充当了模型驱动工程（Model-Driven Engineering, MDE）技术的关键组成部分，成为各种元模型查询，操作和制定规格要求的默认语言 [26]。UML 图表示法不够完善，无法规范化所有约束和属性，如果直接使用自然语言来表示，会造成歧义，OCL 也很好地填补了这一部分的空白。本文提出的元模型经过 OCL 的附加约束，形成了完整的一套建模语言。

2.3 特定领域建模语言构建方法

本小节将介绍提出和验证特定领域建模语言的方法，具体方法过程如图 2-2 所示。首先对特定的领域进行分析，得出基础理论知识；然后根据知识设计元模型，进行细化；将创建的元模型扩展为建模语言和工具，实现建模语言；最后，对建模语言进行使用，验证效果 [27]。



图 2-2: 特定领域建模语言提出方法

1) 领域分析。这一阶段可以分为两种形式。第一种形式是由领域专家直接负责或参与，由于领域专家拥有特定领域的丰富领域知识与实践经验，可以利用现有知识对特定领域重要概念进行提取，抽象出一个初始的元模型，这种方法效率较高，适合于小范围领域，但最终建模语言质量依赖于领域专家的个人水平。第二种形式是领域专家间接参与或不参与，这种形式需要通过阅读特定领域自然语言文献、对领域专家进行访谈或直接基于前人工作进行扩展，得出基础理论知识或初始元模型，这种方法周期较长，适合比较成熟的领域。

2) 设计元模型。这一阶段也可以在领域分析阶段直接完成。如果仅得到特定领域理论知识，需要使用理论知识构建元模型。目前元模型的构建主要遵循的规则是元对象机制（Meta-Object Facility, MOF）定义的四层架构，UML 也是 MOF 的一个实例 [28]。如图 2-3 展示了 MOF 四层架构与 UML 类图之间的对应关系。M3 元元模型层（Meta-Metamodel）包含了定义建模语言所需的元素，作为整个框架的最高层，定义了最基本的元类（Metaclass），并且是自描述（Self-Descriptive）的；M2 元模型层（Metamodel）是 M3 元元模型的实例，定义了 M1 模型层需要使用的元素，如在类图中定义了“Operation”、“Class”、“Property”等元信息；M1 模型层（Model）是 M2 元模型层的具体实现。

体实例，是用户根据 M2 元模型层定义所创建的具体模型实例，如类图中定义的具体类，以及该类包含的具体属性和操作；M0 实例层（Instance）是 M1 模型层的具体实例，是用户真正使用的实例对象，如根据类图创建的具体对象。

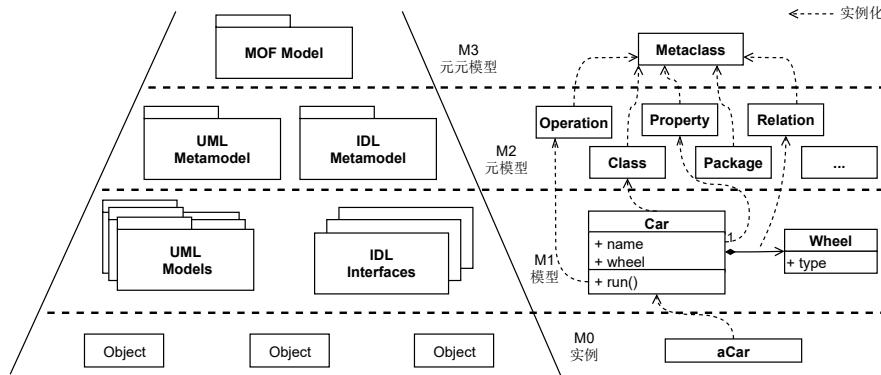


图 2-3: MOF 四层架构与 UML 类图^④

在 MOF 四层架构与 UML 类图对应的基础上，进一步细化元模型设计，将重要领域概念与元模型元素对应起来。UML 中使用 profile 对 M2 元模型层进行扩展定义，提供了三种扩展方式：构造型（Stereotype）是一个配置文件，可以通过构造型从现有的模型元素中派生出新的模型，新的模型通常具有更适合特定领域的属性；标记值（Tagged Values）用于扩展 UML 属性，可以在模型元素的规则中添加额外信息，允许使用键值对的方式以字符串形式呈现，标记在模型元素下方，通常强调模型的版本控制，著作权等信息；约束（Constraints）用于指定模型中必须始终满足的语义或条件，约束可以显示为字符串，并包含在关联元素附近的方括号中。通常来限制模型中属性的取值范围。如图 2-4 展示了 UML profile 扩展类图的一个实例。构造型 «Machine» 表示图中 Car 类为扩展的机器类型，标记值表示了该 Car 类的版本号，约束保证了该 Car 类的速度数值限制在 0 到 300 之间。

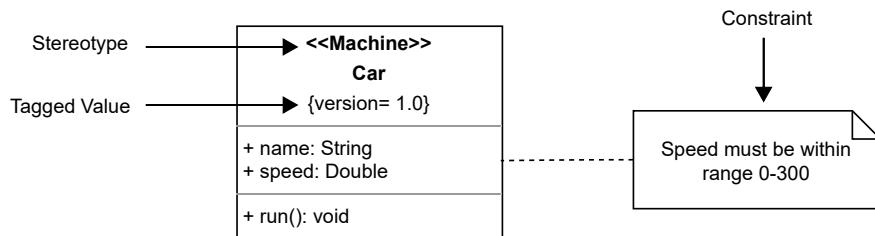


图 2-4: UML profile 扩展类图实例

^④图片来源于 OMG Unified Modeling Language (OMG UML), Superstructure

3) 实现。将设计完成的元模型结合理论知识运用到生产中，是这一阶段的主要任务。建模工具的实施方式分为两种，如图 2-5 展示了两种不同实施工具的方式。第一种方式通过扩展通用建模工具实施，这种方式依赖现有的支持通用建模语言的工具或平台，将特定领域的建模语言配置文件导入，来支持新的建模语言。该方式有利于多种建模语言的集成，但需要统一建模语言平台提供扩展功能。第二种方式通过建模工具生成器根据元模型生成相应的建模工具，这种方式可以给用户提供一个独立的工具，有利于对建模工具的定制、修改和优化，但添加额外功能困难，依赖于工具生成器的成熟度 [14]。

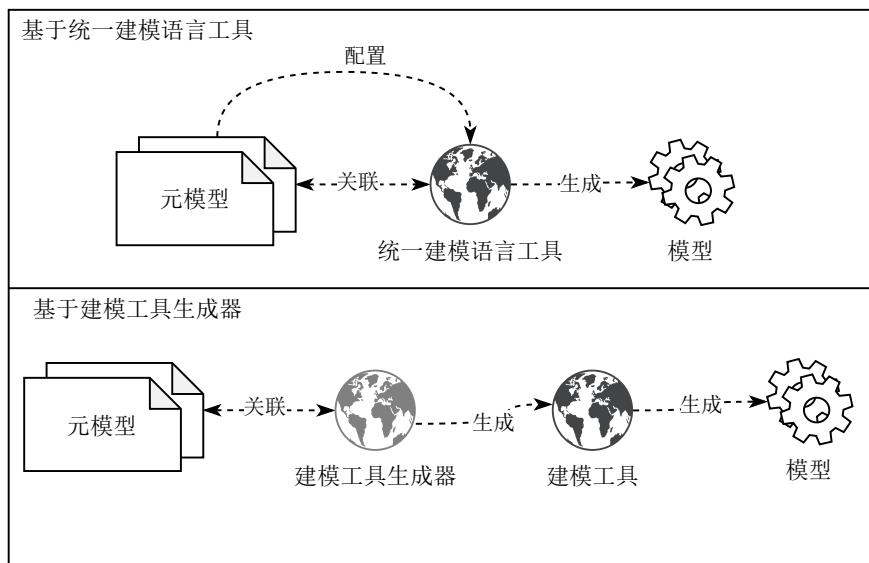


图 2-5: 两种建模工具实施方式

4) 验证。针对最终形成的特定领域建模语言，还可以进行使用效果的验证。包含定性和定量的分析方法，定性分析主要关注建模语言使用者的感受，从易用性、可靠性以及正确性等方面进行考察；定量分析主要关注使用新建模语言的效率，从建模各阶段耗费时间进行统计分析 [29]。

2.4 建模支持工具实现技术

本小节将对建模支持工具实现技术进行介绍。本文实现的建模支持工具所需技术包括可视化建模展示所用的 mxGraph，前端界面构建所用的 Vue.js 和 ElementUI；后端服务器所需技术包括 Spring Boot 和 ElasticSearch。

mxGraph

mxGraph^⑤ 是一个使用 SVG 和 HTML 进行渲染的 JavaScript 图表库客户端，支持交互式图形和图表的快速创建，在主流的浏览器中都能良好运行。支持在网页中设计和编辑工作流图、流程图、网络图和各种图表。**mxGraph** 库不使用任何第三方软件，不需要任何插件，并且可以集成在几乎任何框架中。**mxGraph** 拥有完善的绘图功能，许多大型产品底层都使用其进行绘图和渲染。不仅拥有 JavaScript 源码版本，还有 Java 源码版本，拥有便于移植和适配性强等特点，还支持以 XML 或图片形式进行展示等功能。本文使用 **mxGraph** 作为可视化建模的支持技术，通过在网页中运行 **mxGraph**，来达到拖拽绘制模型、点击连线、双击修改文字以及导出建模结果等功能。

Vue.js

Vue.js^⑥ 是一套用于构建用户界面的渐进式框架。与其它大型框架不同，**Vue.js** 可以自底向上逐层应用。**Vue.js** 的核心库只关注视图层，能方便地获取数据更新，并通过组件内部特定的方法实现视图与模型的交互。此外，**Vue.js** 容易上手，与第三方库或现有项目进行整合时方便，与主流的开发工具链以及各种支持类库结合使用时，**Vue.js** 也完全能够为复杂的单页应用提供驱动。得益于响应式设计，**Vue.js** 获取数据更新十分方便。响应式是指 MVC 模型中的视图随着模型变化而变化。在 **Vue.js** 中，开发者只需将视图与对应的模型进行绑定，**Vue.js** 便能自动观测模型的变动，并重绘视图，这一特性使得 **Vue.js** 的状态管理变得相当简单直观。**Vue.js** 构建的界面简洁美观，交互方式易于理解，交互响应速度快，本文使用 **Vue.js** 来实现前端的用户界面展示与交互。

组件（Componet）也是 **Vue.js** 的一大基础功能。组件可以被复用，封装可多次重用的代码逻辑，复用的组件能更加方便地进行管理、修改和优化。组件还可以以不同形式（如数组、树）组织起来，几乎任意类型的应用的界面都可以抽象为一个组件树，从而构建出大型复杂的应用。

如图 2-6 所示描述了响应式设计追踪变化的过程。当一个普通的 JavaScript 对象传入 **Vue** 实例作为 **data** 选项时，**Vue** 将遍历此对象所有的 **property**，并使用 **Object.defineProperty** 把这些 **property** 全部转为 **getter/setter**。这些 **getter/setter** 对用户来说是不可见的，但是在内部它们让 **Vue** 能够追踪依赖，在 **property** 被访问和修改时通知变更。每个组件实例都对应一个 **watcher** 实例，它会在组件渲染的过程中把“接触”过的数据 **property** 记录为依赖。之后当依赖项的 **setter** 触发时，会通知 **watcher**，从而使它关联的组件重新渲染。

^⑤**mxGraph** 项目地址：<https://github.com/jgraph/mxgraph>

^⑥**Vue** 官网 <https://cn.vuejs.org/>

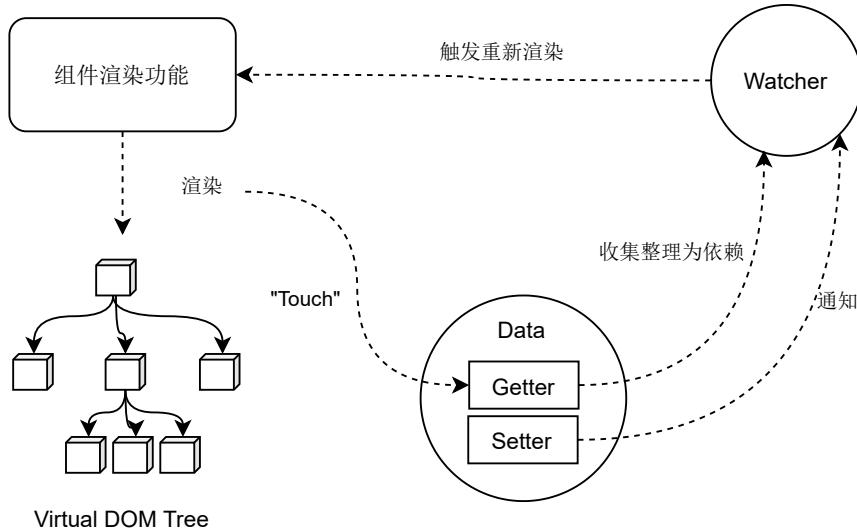


图 2-6: 响应式设计追踪变化

ElementUI

ElementUI^⑦ 是一套基于 Vue2.0 的桌面端组件库，包含了大量的页面设计组件与资源，秉承了一致、反馈、效率、可控的设计原则，帮助使用者快速搭建网站前端交互界面。ElementUI 强调界面元素一致及用户界面和生活场景一致，做到清晰的页面反馈和控制反馈，提供的组件表述清晰直白，使用方式简单高效，用户可以自由准确地使用这些组件进行自主性操作。本文使用 ElementUI 对前端组件进行统一管理，对数据和组件进行绑定和交互。

Spring Boot

Spring Boot^⑧ 是为了简化 Spring 应用搭建以及开发过程的一种框架。Spring Boot 提供起步依赖和自动依赖管理，以注解形式进行配置，代替了不易阅读的 XML 格式配置文件。集成的 Web 服务器不需要再次对网络请求进行复杂处理，可以快速构建后端服务器。本文使用 Spring Boot 开发后端服务，以服务器形式为前端提供接口服务。

ElasticSearch

ElasticSearch^⑨ 是一个基于 Lucene[30] 库的搜索引擎。ElasticSearch 可以用于搜索各种文档，具有接近实时的搜索效果，可以通过 JSON 和 Java API 提供服务，适合含有大量数据的文档搜索与存储。本文项目中的模型结果以文档形式存储在 ElasticSearch 中，能够支持快速检索。

^⑦ElementUI 主页: <https://element.eleme.io>

^⑧SpringBoot 主页: <https://spring.io/projects/spring-boot>

^⑨ElasticSearch 主页: <https://www.elastic.co/cn/>

2.5 本章小结

本章主要介绍了与本文工作相关的理论与支持技术。首先对领域驱动设计中的重要概念进行解释，主要介绍了战术建模包括的重要模式和一些相关概念；然后对建模语言进行了介绍，重点解释了元模型，UML profile 机制以及 OCL 等支持构建建模语言的方式；还介绍了构建特定领域建模语言的方法和具体过程；最后，介绍了支持工具需要用到的技术，包括可视化技术 mxGraph，前端支持工具 Vue.js 和 ElementUI，以及后端技术 Spring Boot 和 ElasticSearch。

第三章 面向领域驱动设计的战术建模支持方法

3.1 研究方法概述

领域驱动设计的概念已经在工业界得到运用，但在针对领域驱动设计进行建模时，尤其是战术建模时，没有一套标准的建模语言和流程规范进行指导和约束，导致战术建模过程难以实施，建模结果难以落地，落地后无法进行复用和扩展。本文工作的核心目标之一是提出一套针对战术建模的支持方法，包含战术建模指南和实例化的战术建模语言，用来支持战术建模工作。

本节将对提出战术建模支持方法的研究方法和各阶段产物进行概述，具体内容如图 3-1 所示。为了构建一套包含战术模式、战术模式重要属性、使用时机以及实现技术的战术建模指南，本文工作首先对战术建模和元模型领域进行调研，通过文献综述收集整合领域驱动设计权威著作中的理论知识。文献综述（Literature Review）是对某一研究领域的问题搜集大量相关资料，通过阅读、分析、提炼、整理该领域最新进展和学术成果，对其做出归纳性整理和综合性介绍的一种研究方式。其次，结合从文献中收集的知识，设计访谈问题，对工业界内战术建模的实践者进行访谈。访谈法（Interview）是以口头形式根据受访者的回答搜集客观事实材料的一种研究性交谈方式，通过访谈法不断迭代完善现有知识，形成最终的建模指南。

以调研的初步产物为基础，结合焦点小组法（Focus Group）[31]，构建战术建模支持方法。焦点小组是通过召集一组与研究主题相关的人员对同一议题进行讨论，并得出深入结论的定性研究方法，焦点小组产出的战术建模支持方法包括战术模式及属性列表、战术建模语言以及一套战术建模使用时机与实现技术指导。本文工作对战术建模语言进行了实例化和工具支持，主要包含建模语言元模型、对象约束语言（OCL）和具体表示方法。

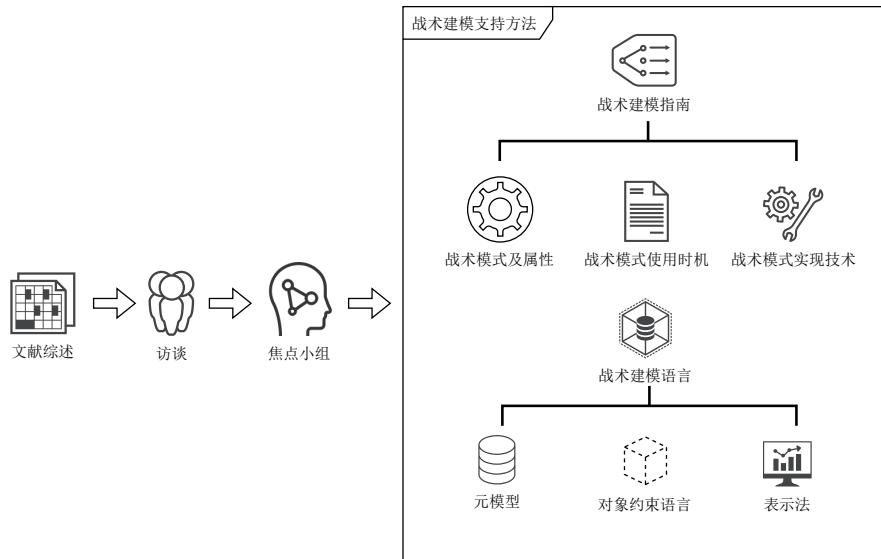


图 3-1: 战术建模支持方法研究过程图

3.2 研究过程

3.2.1 文献综述

本小节主要介绍文献综述相关工作。文献综述总结了前人的工作成果，从领域驱动设计相关著作中进行战术模式的收集与定义。本工作从战术模式中抽取了重要特征，提高实例化战术建模语言的效率与正确性。总结了战术建模的使用时机与实现技术，作为战术建模支持方法的一部分。本工作在调研阶段还参考了相关文献中提出元模型的流程方法，作为元模型构建的理论基础，对现有领域驱动设计建模元模型的特征进行研究和总结，吸取元模型构建经验。

作者从《领域驱动设计：软件核心复杂性应对之道》[1]、《实现领域驱动设计》[4]两本著作中抽取了八种战术建模模式以及它们的重要特征，并以此为基础，构建了一个特定于领域驱动设计战术建模的初始元模型。Evans 认为使用领域驱动设计的人员需要经常快速地浏览一些重要的模式，并掌握这些模式的实施要点和特征，并与实际业务进行结合使用 [32]。领域驱动设计中的概念知识，特别是更为分散的战术模式相关概念知识繁多复杂，需要在使用时经常进行学习和回顾，确保实践的准确性，这也是将战术模式属性、使用时机和实现技术纳入建模支持方法的一个重要原因。

在本小节总结的成果中，战术建模模式包括：实体（Entity）、值对象（Value Object）、领域服务（Domain Service）、领域事件（Domain Event）、

聚合（Aggregate）、资源库（Repository）、模块（Module）、工厂（Factory）。战术模式重要特征描述了战术模式应该具备的特点、约束条件以及它和不同模式间的关系。战术模式应用场景描述了每个模式的使用时机和实现时的核心问题。上述成果将作为访谈问卷的理论基础，经过访谈不断迭代完善，最终形成关于战术模式的使用指南。

3.2.2 访谈

通过分析多次访谈的结果，对文献综述阶段得到的战术模式进行验证和筛选，对战术模式重要特征、使用时机和实现技术进行确认和修改，总结出一套战术建模指南。

访谈活动共进行三次，每次访谈一名对象，访谈持续时间控制在一小时左右。表3-1展示了访谈对象的具体信息，三名访谈对象分别来自思特沃克、阿里巴巴和华为，其中有两名架构师和一名开发人员。

表 3-1: 受访者信息

业务领域	角色	从业经验及年限
软件咨询领域	架构师	就职于斯特沃克（ThoughtWorks），为国内外医疗、金融、通信、汽车等行业的客户提供软件咨询和交付服务。作为技术顾问参与和主导了多次领域驱动设计相关工作，是将领域驱动事件风暴引入国内的第一批技术人员。拥有 10 年左右的开发及架构设计经验。
通信领域	架构师	就职于华为，使用领域驱动设计相关思想与方法多次主导并参与过大型分布式项目的架构设计与微服务拆分。拥有 10 年以上的开发及架构设计经验。
电商领域	开发人员	就职于阿里巴巴，担任过项目经理，参与过微服务开发，具有战术建模的实践经验，对领域驱动设计有过一定了解。拥有 2 年左右的开发及项目管理经验。

访谈首先对访谈对象的个人信息进行了解，包括受访对象的职位、团队规模以及业务领域，方便对领域驱动设计战术建模实践场景进行评估；然后对领域驱动设计建模过程进行了解，包括建模使用的工具，建模工作的总体流程，用来和本文工作做对比，评估本文工作对战术建模领域的作用；最终聚焦战术建模中的战术模式，对文献综述阶段总结的成果进行评估。如表3-2所示，展示了访谈中主要关注的五个方面，并阐述了具体原因。具体访谈问卷问题见附录A。在访谈中，受访对象均对本文工作研究内容和最终产出的支持工具持积极态度。

表 3-2: 访谈关注问题

关注问题	原因
何时使用该模式	获取和验证战术模式的使用时机。
该模式应该包含哪些重要属性	获取和验证战术模式的属性。
实践该模式时有哪些好的实践	获取和验证战术模式的实现技术与细节。
实践该模式时有哪些不好的实践	排除不好的实现技术。
针对于特定模式的特殊问题	验证对应战术模式是否有必要加入建模支持方法。

3.2.3 焦点小组

本小节介绍研究过程中使用焦点小组形式进行战术建模支持方法构建的相关内容。在所有访谈结束后，组织焦点小组对文献综述与访谈结果进行分析和评估，目的是不断完善战术建模支持方法内容，包括迭代和实例化战术建模语言，总结整合战术模式、模式属性、使用时机以及实现技术。

每次焦点小组由包含作者在内的三名研究人员参与，作者为主持人兼记录员，负责主持会议和记录会议内容，其他与会人员均对领域驱动设计相关知识具有一定了解。

如图 3-2 所示展示了焦点小组的详细实施过程。在会议开始前，先将前期文献综述与访谈的成果作为资料展示给参与人员，并由主持人介绍访谈的初步成果。会议开始后，使用领域驱动设计战术建模中的重要概念以及会议资料对战术建模支持方法进行设想。首先，将文献综述和访谈结果中的战术模式与其重要特征进行整合，形成战术模式及其属性列表；在整合过程中，找出重复和互补的修改建议，使最终战术模式与特征列表更加准确完善，更好地充当构建战术建模语言时的附加属性；通过 UML profile 机制构建元模型，并设计对象约束语言和建模语言表示方法来实例化战术建模语言；最后，将战术模式使用时机与实现技术纳入到战术建模指南中；经过多次焦点小组的讨论和修改后，战术模式、战术模式的属性、使用时机以及实现技术共同组成了战术建模指南，汇总战术建模指南和实例化的战术建模语言，形成最终的战术建模支持方法，并对其进行评估。

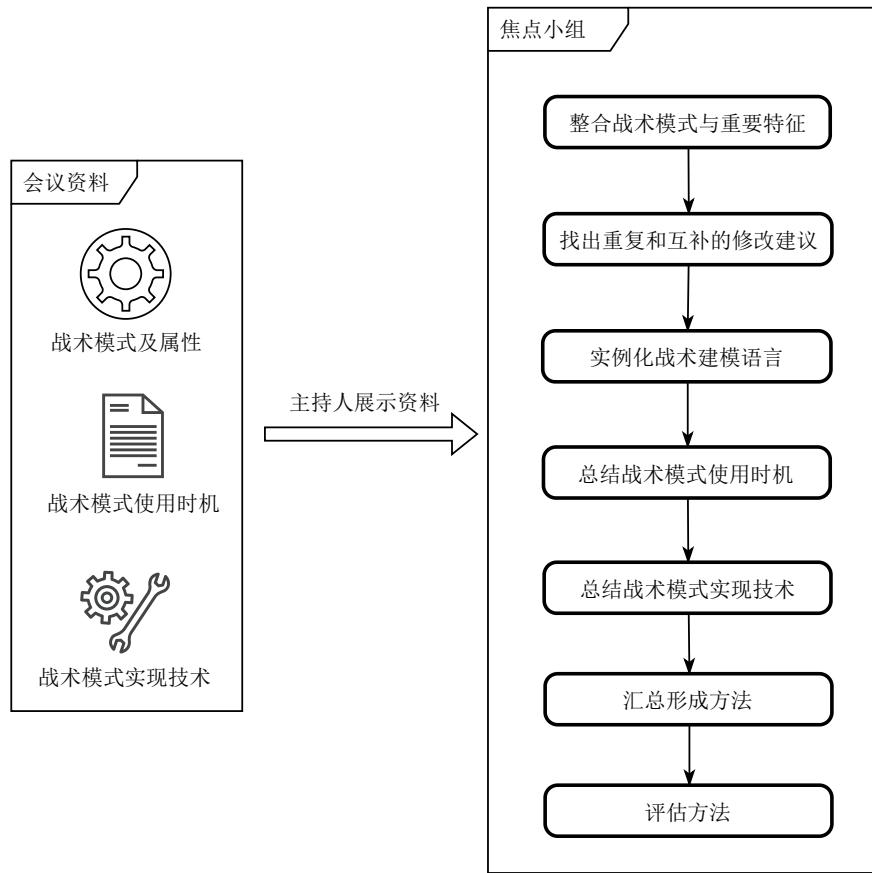


图 3-2: 焦点小组实施过程

3.3 战术建模指南

3.3.1 战术建模模式及属性

经过文献综述调研工作和对多次访谈的结果进行分析，总结出了八种战术模式及属性。作者发现有些原本属于战略建模设计的概念在战术建模时也会纳入考虑范围，例如，防腐层（Anti-Corruption Layer，ACL）往往会在战术建模时经常使用，用来隔离和代理对其他限界上下文的访问行为。此外，战术模式在实现时与理论概念稍有不同，在真正进行战术建模时，模块（Module）这一模式更多的是代码层面的组织和结构要求，而非建模时设计阶段需要考虑的因素；工厂（Factory）则与设计模式中的工厂区别不大，作为建模中的模式意义不大，完全可以作为实现的一种方式，故对这两种模式进行排除，表 3-3 展示了战术建模中的主要模式和它们的重要属性，其中战术模式的属性以英文单词

“Characteristic”的前两位大写字母“CH”结合序号进行编号，并对每个战术模式的属性进行描述。

表 3-3: 战术建模模式及属性

模式	属性	描述
实体	CH1: 唯一标识	实体是唯一的，需要借助唯一标识进行区分和跟踪变化。
	CH2: 可变性	实体在其生命周期中是可变的，除唯一标识外都可以进行修改，不管如何变化都可以通过唯一标识进行跟踪。
值对象	CH3: 不变性	值对象创建后不可改变。只有初始化时才能修改对象的属性状态。
	CH4: 可替换性	值对象无法继续正确表达状态时，度量和描述改变，可以用另一个值对象直接替换。
	CH5: 值对象相等性	系统中存在很多相等的值对象实例，但它们不是同一个值对象，相等性通过比较两个对象的类型和属性来决定。
	CH6: 无副作用行为	值对象的所有方法必须只产生输出，而无法改变对象状态，防止破坏不变性。
	CH7: 概念整体	值对象可以处理一组关联的属性，将它们视为整体。
领域服务	CH8: 无状态	服务本身不保存任何数据。
	CH9: 输入	领域服务的输入参数。
	CH10: 输出	领域服务的输出结果。
领域事件	CH11: 触发时间	领域事件是过去发生的，需要一个时间戳来记录发生事件。
	CH12: 事件发送方	产生该领域事件的对象和其他参与操作的对象。
	CH13: 不变性	领域事件携带的属性反映过去的信息，不应该再改变。
	CH14: 唯一标识	将领域事件建模成聚合，发布到外部限界上下文时，都需要唯一标识来进行区分、跟踪或去重。
聚合	CH15: 根实体	根实体是访问其他聚合的媒介，相当于聚合的“唯一标识”，尽量包含最少的属性。
	CH16: 一致性边界	聚合是一个原子性的整体，聚合边界内部的所有对象都应该保持一致性，聚合的所有操作都是事务的。
资源库	CH17: 为聚合根提供	理想条件下只为聚合根提供资源库，但有时也可为实体提供。
	CH18: 额外行为	不包含业务逻辑的额外行为，如计算总数，特殊化查询方法等。

3.3.2 战术建模模式使用时机与实现技术

结合前期文献综述和访谈结果，总结出了战术模式在实现时的技术经验与指导准则，并整合到战术建模支持方法中，从各种模式的使用时机和实现技术两个方面进行阐述，为战术建模使用人员界定模式使用场景和时间提供参考，为战术建模落地实现提供快速解决方案。

如表 3-4 所示为战术模式使用时机表，其中各个战术模式的使用时机以单词“Time”的首字母“T”结合序号进行编号，并进行描述。

表 3-4: 战术模式使用时机

模式	使用时机
实体	T1: 需要将对象和其他对象进行区分时 T2: 需要跟踪对象变化时 T3: 涉及到对象持久化操作时
值对象	T4: 该对象具有明显的可替换性 T5: 该对象具有不可变的特性
领域服务	T6: 一个显著的业务操作过程不属于实体或值对象职责时 T7: 一个含有业务含义的领域对象转换过程不属于实体或值对象职责时
领域事件	T8: 需要捕获发生在领域中的一些重要事件时 T9: 需要维护跨越不同聚合或限界上下文间事件的一致性时
模块	T10: 将内聚类放在相同模块进行管理 T11: 将没有联系的类放在不同模块进行解耦合
聚合	T12: 需要将实体和值对象聚类到一致性边界内时 T13: 需要保证密切关联的一组对象内部的不变条件时
资源库	T14: 聚合实例可以放在资源库中，保证存取同一个聚合 T15: 存储需要进行全局访问的对象 T16: 严格来说只为聚合提供，但有时也可以为实体提供

战术模式实现技术对各个模式进行实现时采用的技术进行了描述和编号，其中实现技术及方案以单词“Practice”的首字母“P”结合序号进行编号，并给出了实现时推荐的技术方案与具体细节。具体内容如下：

实体实现技术及方案

P1：生成唯一标识

- 用户提供唯一的一个或多个初始值作为输入的唯一标识。该方式简单直接，但有时保证用户提供的标识唯一很困难。
- 应用程序生成唯一标识。该方式能生成有明确意义的可读标识，且程序生成的方式保证了全局唯一性。

- 持久化机制生成唯一标识。该方式将生成标识的职责委派给持久化机制（如数据库），结果保证唯一，但耗时较长，成功率依赖持久化机制稳定性。

P2：唯一标识生成时间

- 创建对象时生成（及早生成）。如果需要发布领域事件，就要尽早地在发布前就生成，以便标识领域事件；在未持久化之前需要进行区分时，也要尽早生成唯一标识。
- 对象持久化时生成唯一标识。除上述情况外，都可以在对象持久化时再生成唯一标识。

P3：保持唯一标识稳定性的方法

- 将设置唯一标识的方法隐藏，让外部无法访问；设置唯一标识前询问是否存在，如存在则禁止更新。

值对象实现技术及方案

P4：保证值对象不变性

- 只允许初始化时设置值对象属性。

P5：存储值对象集合的方法

- 多个值对象序列化到单个列中。该方式将集合序列化为单列文本，序列化后的对象无法支持查询，且需要持久化机制支持较长单列长度。
- 将值对象转化为数据库实体。该方式将多个值对象共同关联的实体的唯一标识作为主键，相当于将值对象在持久化层次变成了“实体”。

领域服务实现技术及方案

P6：实现领域服务

- 抽象一个接口。该方式适合有多种实现类的领域服务。
- 直接定义实现类。该方式适合逻辑简单、无需定义接口的领域服务。
- 只关注业务逻辑。领域服务只关注业务逻辑和流程，不依赖特定实现技术或持久化机制。

领域事件实现技术及方案

P7：实现领域事件

- 聚合创建事件并发布事件。
- 事件订阅方直接转发或先存储再转发到远程订阅方，如果消息中间件

没有和模型共享数据存储，要使用两阶段提交（Two-Phase Commit, 2PC），确保最终一致性。

- 尽量避免过度暴露领域模型给消息设施。
- 保证订阅方在事件发布前完成订阅，确保能够接收事件。
- 资源库禁止删除已存储事件，因为事件表示已发生的既定事实。
- 如图 3-3 所示描述了实现领域事件过程的流程图。

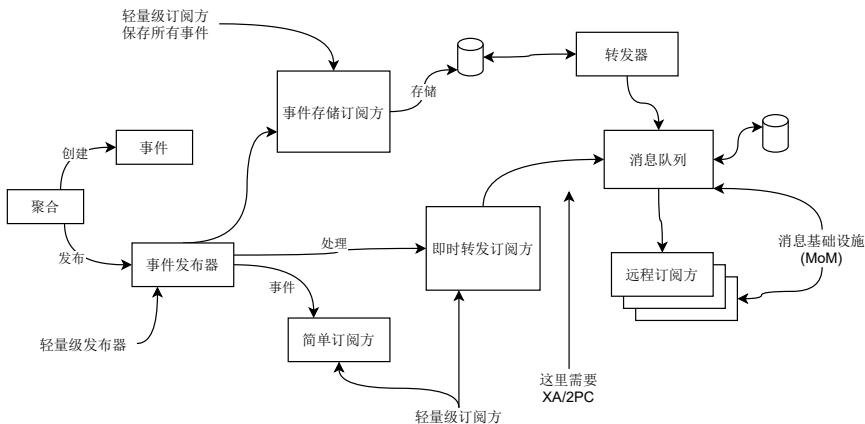


图 3-3: 领域事件实现流程图^①

P8：保持领域事件不变性

- 只允许领域事件全状态初始化，避免初始化后又进行修改。
- 避免领域模型暴露到消息设施，防止第三方进行修改。

P9：转发领域事件

- 通过 RESTful 接口发布。该方式适合多个客户同时通过相同 URL 请求来获取事件通知，不关注事件通知的顺序。
- 通过消息中间件方式发布。该方式依赖消息中间件省去细节处理，只需确保推送消息成功。

模块实现技术及方案

P10：组织模块

- 不要只根据通用组件所属战术模式类型来组织模块，即不要简单地将所有聚合放在同一个模块，所有领域服务放在同一个模块，要考虑业务逻辑与行为。

^① 图片来源于《实现领域驱动设计》

- 设计松散耦合的模块，有利于维护和重构模块。
- 同层模块出现耦合时，杜绝循环依赖。
- 模块与建模对象一同变化，模型中对象改变时，模块也应进行改变。

聚合实现技术及方案

P11：保证一致性边界

- 用事务来保证一致性边界。事务一致性要求立即性和原子性，提交单个事务时，聚合边界内的所有内容都必须保持一致。

P12：设计聚合的原则

- 设计小聚合。使用根实体代表聚合，只包含最小数量的属性（需要在领域内保持一致性的属性）。
- 聚合内部建模成值对象。值对象可以配合根实体一同序列化存储，不会带来不必要的操作。

资源库实现技术及方案

P13：实现资源库

- 为存储的聚合或实体提供访问的全局接口，确保存取的便捷性。
- 提供添加和删除方法。
- 提供顺序自增的唯一标识。

3.3.3 战术建模指南构建结果

本小节将介绍基于文献综述和访谈结果，通过焦点小组讨论构建的战术建模指南。指南中各模式包含的内容如表 3-5 所示。战术建模指南包含各个模式的名称、特征属性、使用时机和实现技术方案，为方便表示，以上内容均以编号代表。

表 3-5: 战术模式包含内容

模式	属性	使用时机	实现技术
实体	CH1 CH2	T1 T2 T3	P1 P2 P3
值对象	CH3 CH4 CH5 CH6 CH7	T4 T5	P4 P5
领域服务	CH8 CH9 CH10	T6 T7	P6
领域事件	CH11 CH12 CH13 CH14	T8 T9	P7 P8 P9
模块	无	T10 T11	P10
聚合	CH15 CH16	T12 T13	P11 P12
资源库	CH17 CH18	T14 T15 T16	P13

形成的具体战术建模指南如图 3-4 所示。建模指南按模式划分，每个模式包含了属性、实现技术及使用时机，每个属性使用虚线关联其对应的实现技术。为方便表示，以上内容均以编号代表。该建模指南可以作为战术建模全流程的标准化参考。

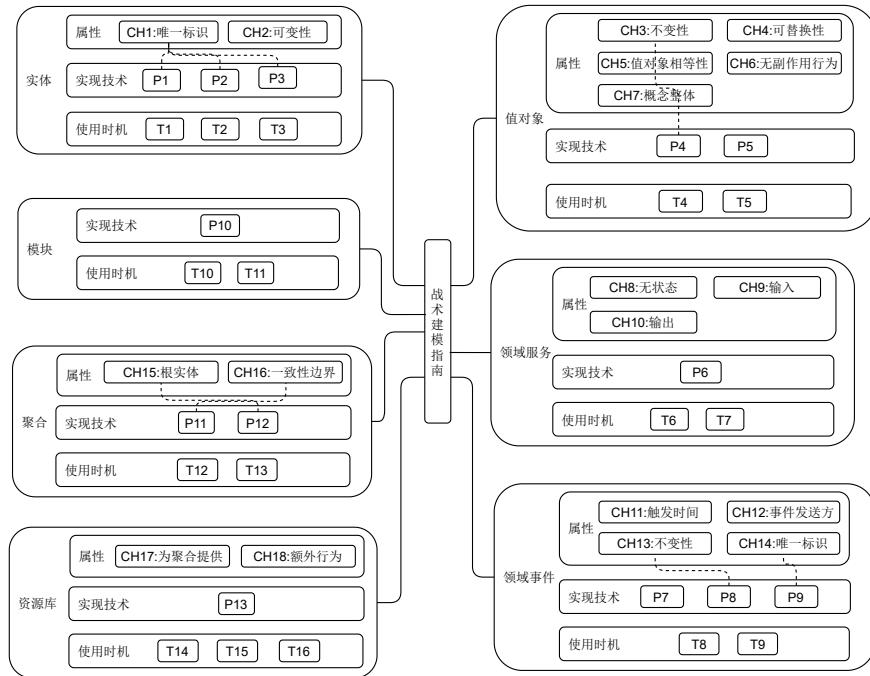


图 3-4: 战术建模指南

3.4 战术建模语言

本节将介绍基于焦点小组讨论实例化的战术建模语言。作者实例化建模语言的过程参考了 [33, 34] 文献中的流程方法，提出的元模型参考了 [10, 11, 17] 文献中的现有元模型。首先使用 UML profile 机制来进行战术建模元模型的构建，包括战术建模初始元素集的构建，初始元素集展示了建模语言中需要使用的重要概念，并与元类类型对应；还构建了战术建模元类，UML 中的元类是一种成熟的元模型实现方式，由于 UML 类图与软件设计实现关系密切，并且易于阅读理解，所以以其为重点进行 UML profile 的构建；然后使用对象约束语言（OCL）对约束条件进行实现，对象约束语言（OCL）是一种施加在指定模型元素上的约束语言，为元模型附加了更多约束条件使其更加完善。

3.4.1 战术建模语言元素集

根据表3-3定义战术建模 UML profile 元模型（Metamodel）需要的元素集。本文定义的元素集如表3-6所示，表中展示了元素集中元素的名称，代表的元类类型以及描述。

表 3-6: 战术建模 profile 元素集

元素名	元类类型	描述
Entity	Class	实体是一个由标识符定义的对象，通过标识符与其他对象区别开来确保其个性特征。
ValueObject	Class	值对象用来度量和描述事物，具有不可变性。
DomainService	Class	领域服务是一个独立接口或对象，负责承担不属于实体或值对象职责的操作或转换过程。
DomainEvent	Class	领域事件记录了领域中发生的事情，无法改变，还可以维护发布到外部上下文的业务一致性。
AggregateRoot	Class	聚合是一个将实体和值对象聚类到一致性边界内的容器，每个聚合内部有一个实体作为聚合根，聚合通过聚合根与外部进行交互。
Repository	Class	资源库是一个用来存取领域对象的安全存储区域，还可以提供对象的存取、删除和查询操作。
DefineIdentity	Property or Operation	唯一标识，用来区分其他对象和跟踪对象。
Module	Package	模块是用来存放内聚类的容器，主要用在技术层面的代码组织、构建目录与打包。
Input	Class and Property	领域服务对象或领域事件对象方法的输入。
Output	Class and Property	领域服务对象方法的输出。
OccurTime	Property	领域事件触发时间。
EventStep	Property or Operation	事件步骤，用来描述领域事件。
ACL	Class	防腐层，用来和其他上下文交互和隔离。

如表3-6所示，在重要战术模式表3-3的基础上，进行了优化与修改。直接用聚合根（AggregateRoot）和与聚合根直接或间接相连的所有对象表示聚合（Aggregate），为领域服务（Domain Service）引入必要属性输入（Input）和输出（Output），为领域事件（Domain Event）引入必要属性输入（Input）来代表发送方，还引入了触发时间（OccurTime）和领域事件步骤（EventStep）属性。引入严格意义上不是战术模式但可能会用到的概念元素防腐层（Anti-Corruption Layer， ACL）。

3.4.2 战术建模元类

一个 UML profile 包含构造型 (Stereotypes) 形式的 UML 元类的扩展，还包含元类中类 (Class)、属性 (Property) 和操作 (Operation) 的映射关系 [35]，使用定义好的配置文件构造型，可以对元类进行扩展，丰富其语义。如图 3-5 描述了扩展元类的战术建模 UML profile，包含所有的构造型，以及各构造型之间的包含关系，采用标记值 (Tagged Values) 进行表示。其中，箭头表示构造型扩展自元类。

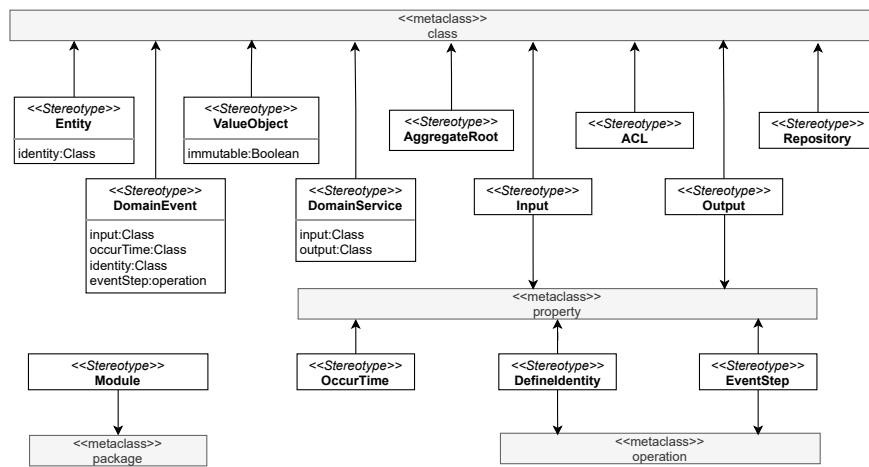


图 3-5: 战术建模扩展元类 profile

3.4.3 战术建模语言约束

为元类添加约束可以防止不规范或错误使用元类，为确保上述扩展元类正常使用以及建模过程的准确性，每个元类都被添加了相应约束，具体约束如表 3-7 所示，使用英文单词“Constraint”的大写首字母“C”结合序号对约束进行编号，并通过对对象约束语言对约束进行标准化的实现。

表 3-7: 构造型约束规则

构造型	约束
Entity	C1: 实体必须拥有一个方法或属性定义唯一标识。 C2: 领域服务类不能具有其他构造型。
DomainService	C3: 领域服务必须拥有输入值。 C4: 领域服务必须拥有输出值。

DomainEvent	C5: 领域事件类不能具有其他构造型。 C6: 领域事件必须记录被触发时间。 C7: 领域事件必须拥有事件发送方作为输入。 C8: 领域事件必须拥有唯一标识来跟其他对象进行区分。
AggregateRoot	C9: 只有实体能充当聚合根并与其他聚合根连接。 C10: 一个聚合根只能由一个实体代表。
Repository	C11: 资源库类不能具有其他构造型。 C12: 只为实体或聚合根提供资源库。 C13: 资源库只定义方法，且至少定义一个方法。
ValueObject	C14: 值对象不可变。
DefinesIdentity	C15: 唯一标识可以以属性或方法返回值的形式存在于对象中。 C16: 唯一标识必须在实体或领域事件中使用。
Input	C17: 输入以属性形式存在于对象中，且必须指定输入代表的对象。
Output	C18: 输出以属性形式存在于对象中，且必须指定输出代表的对象。
OccurTime	C19: 触发时间必须作为属性在领域事件中使用。

对象约束语言（OCL）可以施加在指定元类上，为元类附加更多约束条件，使其在使用过程中更加标准规范，避免错误地使用元类。由于本文篇幅有限，仅选择领域事件的约束实现进行展示，如图 3-6 所示，展示了用 OCL 实现的领域事件约束。

```

-- 领域事件不能有其他构造型
context DomainEvent inv:
    self.base_Class.extension_Entity = null and
    self.base_Class.extension_ValueObject = null and
    self.base_Class.extension_DomainService = null and
    self.base_Class.extension_AggregateRoot = null and
    self.base_Class.extension_Repository = null and
    self.base_Class.extension_DomainEvent = null

-- 领域事件必须记录被触发时间
context DomainEvent inv:
    self.occurTime->notEmpty()

-- 领域事件必须拥有事件发送方作为输入
context DomainEvent inv:
    self.in->notEmpty()

-- 领域事件必须拥有唯一标识来跟其他对象进行区分
context DomainEvent inv:
    self.identity->notEmpty()

```

图 3-6: DomainEvent 约束实现

3.5 本章小结

本章介绍了提出战术建模支持方法的研究过程及其结果。其中研究过程包括文献综述、访谈和焦点小组，文献综述部分对前期收集理论知识的目的和过程进行介绍，并为访谈和构建支持方法做准备工作；访谈部分依据文献综述的成果，设计了问卷并对工业界内的领域驱动设计实践者进行访谈；访谈后的初步产物作为焦点小组的输入，通过多次讨论，构建了战术建模支持方法。研究结果包括由文献综述和访谈得到的战术建模指南，即战术模式及属性、战术模式使用时机和实现技术；还包括焦点小组讨论得出的战术建模语言。上述战术建模指南和战术建模语言共同组成了本文提出的战术建模支持方法。

第四章 建模支持工具设计与实现

4.1 概述

战术建模在分析与设计阶段一直缺少一个标准化、规范化的流程指导，许多设计过程停留在使用纸和笔简单勾画草稿的阶段，即使是建模经验丰富的架构师，也无法保证战术建模结果的统一性。开发人员缺少对领域的理解，甚至对战术模式的使用也不够熟悉，导致投入战术建模的精力无法得到相应回报。

因此，一个支持标准化、规范化战术建模的工具十分重要。本章基于第三章提出的战术建模支持方法中的战术建模语言，提供配套的可视化建模工具 DDDD（Draw for Domain-Driven Design）来支撑实现战术建模。建模工具 DDDD 应具备简单的建模知识介绍，方便使用者快速获取战术建模知识，并运用到分析设计中；还应具有“可视化建模”模块，该模块支持依据本文提出的战术建模元模型进行可视化建模，通过灵活的拖拽和点击，即可完成战术建模过程，帮助架构师和开发人员更加灵活高效地进行战术建模；还应具有“模型校验”模块，该模块支持对建模结果的验证，用于检测建模的结果是否符合第三章中提出的战术建模元模型，并给出修改意见，保证建模结果的规范性和正确性；最后，还应具有“模型存储与转化”模块，该模块支持将建模结果进行存储，方便后续访问和修改，保证了建模结果的可复用性，支持对建模结果进行扩展，生成符合领域驱动设计标准的框架项目代码包，为实现阶段提供良好开端。

本章实现的建模支持工具使用流程如下图 4-1 所示。开发人员或架构师使用支持工具开始建模，首先可以通过“模型存储与转化”模块的模型加载功能继续从上一次建模保存的结果开始继续建模；使用“可视化建模”模块进行建模时，开发人员或架构师可以随时使用“模型校验”模块进行对建模结果的验证，确保建模的正确性和规范性，工具也将对不符合规范的建模结果给出提示和警告，帮助开发人员和架构师进行修改；如果建模结果符合规范和约束，工具将通过“模型存储与转化”模块保存建模结果到数据库，或者以 XML、图片或 JSON 格式导出，还可以根据建模结果生成框架项目文件。

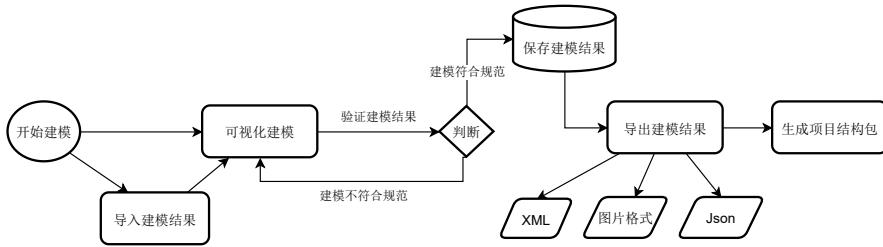


图 4-1: 建模支持工具业务流程

本章的剩余部分将对建模支持工具进行需求分析，根据分析得到的结果对工具进行设计与实现。首先对工具按功能模块进行划分，按模块介绍功能需求与非功能性需求；然后对工具进行总体设计；最后介绍各模块详细设计与实现。

4.2 需求分析

4.2.1 工具总体功能

DDDD (Draw for Domain-Driven Design) 建模支持工具总体功能规划如图 4-2 所示，主要包括“可视化建模”模块、“模型校验”模块以及“模型存储与转化”模块，各模块详细功能需求描述如下：

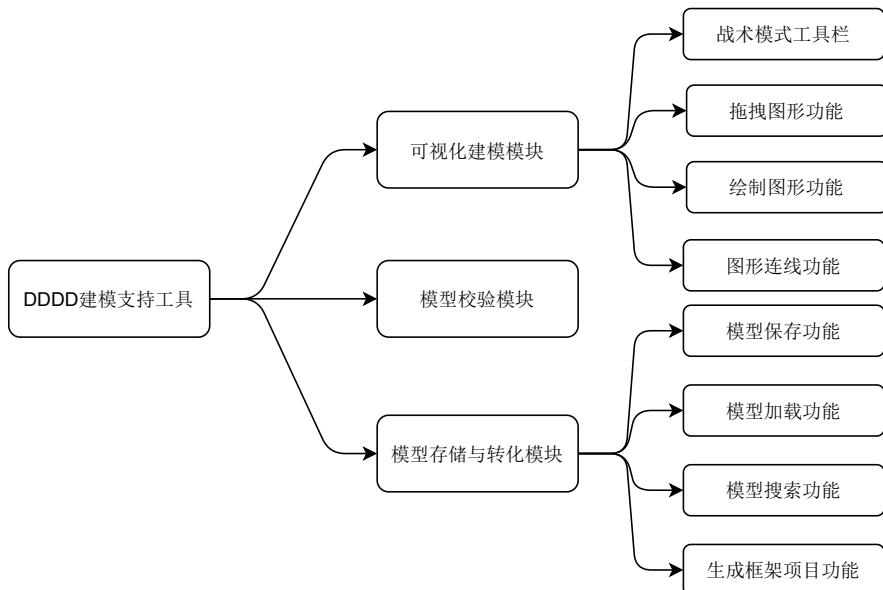


图 4-2: 工具总体功能

- “可视化建模”模块：支持开发人员从工具栏中选择需要使用的战术模式，并灵活地将模式图形拖拽到建模绘图面板上进行绘制，还支持根据建模需要对不同的模式进行连线，表达模式间的关系，完成可视化建模过程。
- “模型校验”模块：在建模过程中或建模完成时，可以依据定义的战术建模元模型中的约束对建模的结果进行验证，并给出修改意见。
- “模型存储与转化”模块：建模完成后，将建模结果进行存储或导出到本地，也可以从数据库或本地读取模型文件，根据建模结果，生成符合领域驱动设计规范的框架项目代码包。

4.2.2 可视化建模模块需求分析

功能性需求

“可视化建模”模块是本建模支持工具的核心功能模块，如图 4-3 所示，开发人员或架构师可以通过“可视化建模”模块“选择战术建模模式”、“拖拽图形”、“绘制图形”以及进行“图形间连线”。

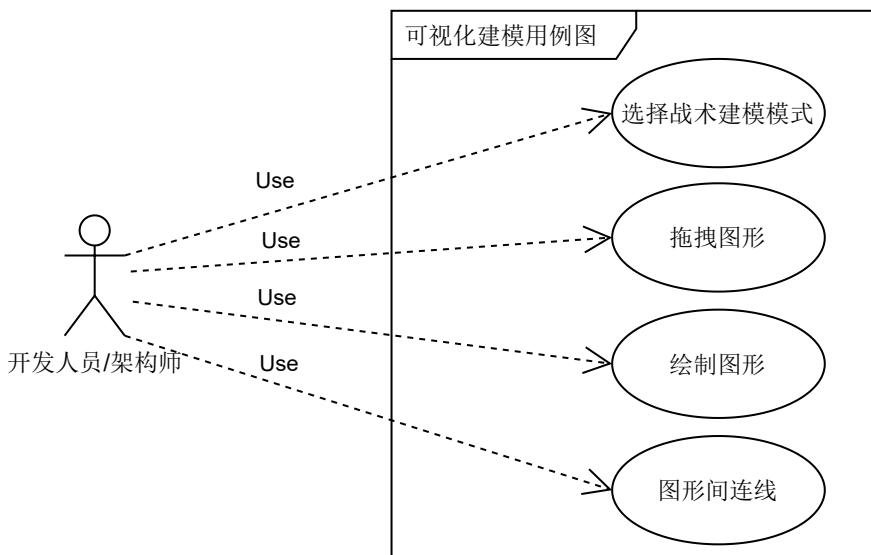


图 4-3: 可视化建模模块用例图

表 4-1 展示了“可视化建模”模块的用例描述，“可视化建模”为开发者提供灵活方便的可拖拽式建模方式，可以自由选择需要使用的战术模式，拖拽到绘制面板中进行绘制，对绘制的模型对象可以双击修改文字信息，并将当前模型对象与其他对象进行连线，以表示它们之间的关系，在绘制面板上方还有以按钮形式提供的绘图操作菜单。

表 4-1: 可视化建模用例表

ID	UC1
名称	可视化建模用例
描述	开发者使用工具可以拖拽式地进行建模，选择战术模式相应图形化模型，拖拽到建模绘制面板中，并修改模型对象信息、对象间关系，达到灵活可视化建模。
触发条件	从主页点击“开始建模”按钮
前置条件	用户浏览器支持 JavaScript
后置条件	无
正常流程	(1) 用户点击“开始建模”按钮； (2) 用户点击展开战术模式列表，拖拽需要的模式进入绘图面板； (3) 松开鼠标，模型对象绘制完成，双击模型对象文字块，修改文字信息； (4) 当鼠标悬停在模型对象上且边框变绿，按住鼠标左键拖动绘制箭头，连接到其他模型对象上； (5) 点击绘图板上方按钮可对模型进行验证、组合、分解、删除、撤销以及缩放画布操作。
异常流程	无

非功能性需求

“可视化建模”模块的非功能性需求主要包括易用性、可靠性和可扩展性，具体需求如下：

- 易用性：可视化的建模过程易于学习和使用，用户仅需进行简单的拖拉拽操作即可完成图形化建模，速度快，图形变换灵活简单，建模结果清晰明确，符合用户建模的期望。
- 可靠性：在“可视化建模”过程中，绘图逻辑大多都在前端完成，即使网络连接断开也不影响建模过程，保障了建模过程的连贯性。
- 可扩展性：“可视化建模”的战术模式工具栏可以根据需求添加和删除战术模式，适应未来战术建模理论的更新和发展。

4.2.3 模型校验模块需求分析

功能性需求

“模型校验”模块用例图如图 4-4 所示，该模块帮助开发人员或架构师根据元模型和约束规则对建模结果进行验证和修改。

表 4-2 展示了“模型校验”功能模块的用例描述，“模型校验”根据本文定义的战术建模元模型约束规则和图形绘制规则对图形化的建模结果进行验证，对

不符合规范和约束的模型进行警告和给出修改提示，对验证通过的模型进行命名并保存到数据库中。

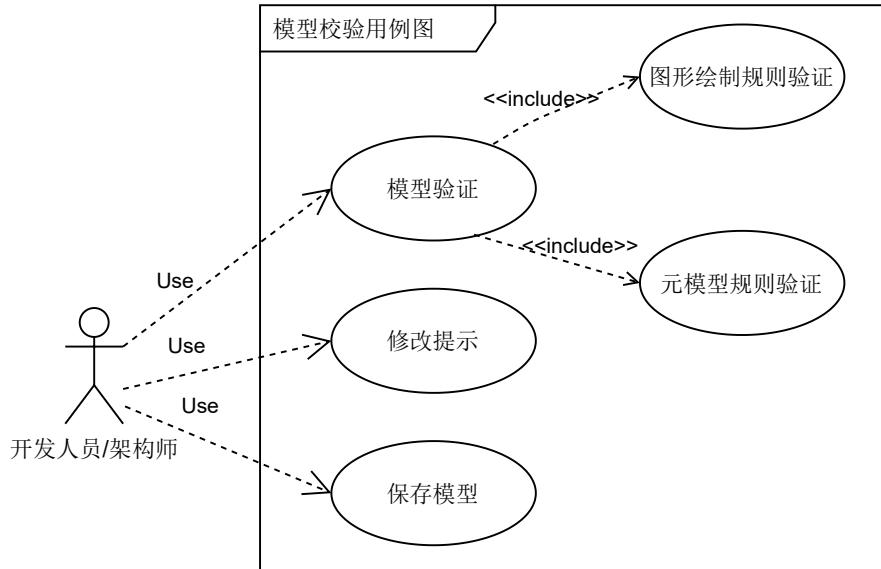


图 4-4: 模型校验模块用例图

表 4-2: 模型校验用例表

ID	UC2
名称	模型校验
描述	根据本文提出的战术建模元模型及约束规则对建模结果进行验证，并给出修改意见
触发条件	绘图界面点击“验证”按钮
前置条件	建模进行中或已经完成
后置条件	无
正常流程	(1) 当建模面板不为空时，开发者点击“验证”按钮； (2) 前端界面弹出验证通过或验证失败及修改提示信息； (3) 如果验证通过，将建模结果命名并自动保存在数据库中。
异常流程	保存建模结果时未填写名称，提示填写

非功能性需求

“模型校验”模块的非功能性需求主要包括易用性、健壮性和可靠性，具体需求如下：

- 易用性：模型的验证结果以弹窗形式展示，指出建模不规范的地方并给出修改建议，让用户明确需要修改哪些建模结果。
- 健壮性：验证操作能够应对各种异常情况并作出响应。对于错误输入、非

法字符和误操作等能够规避和纠正，有效防止建模过程中工具的崩溃。

- 可靠性：支持每次验证操作都将建模结果自动保存到数据库，保障了建模结果不丢失和及时更新持久化内容。

4.2.4 模型存储与转化模块需求分析

“模型存储与转化”模块用例图如图 4-5 所示，开发人员或架构师通过“模型存储与转化”模块对模型进行导入和导出，对数据库中存储的模型进行管理，即搜索和查看模型，还能根据模型生成框架项目。

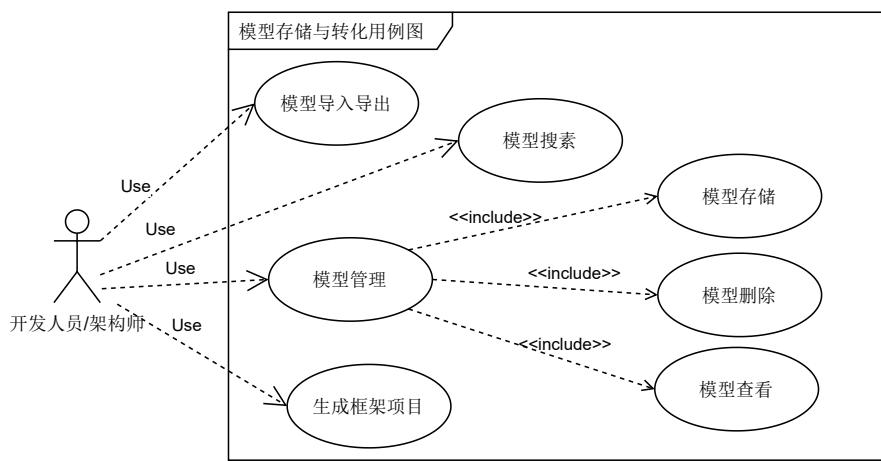


图 4-5: 模型存储与转化模块用例图

表 4-3 展示了“模型存储与转化”模块的用例描述，“模型管理”可以将通过验证的建模结果保存到数据库中，“模型导入导出”可以将模型以 XML、图片等文件格式导出到本地，也可以导入本地模型文件或搜索数据库中现有模型进行展示和修改，通过“生成框架项目”可以得到符合领域驱动设计规范的框架项目代码包。

表 4-3: 模型存储与转化用例表

ID	UC3
名称	模型存储与转化
描述	将图形化建模结果存储到数据库，或直接以 XML、图形等格式导出到本地，也可以从本地导入 XML 模型文件，根据完整的建模结果，自动生成符合领域驱动设计结构规范的框架项目代码
触发条件	绘图界面点击“导入”、“导出”、“验证”、“保存”或“生成项目”按钮
前置条件	建模进行中或已经完成
后置条件	无

正常流程	(1) 进入绘图界面时, 可以选择导入本地模型文件或搜索数据库中模型文件进行继续建模; (2) 模型验证通过后, 选择导出建模结果到本地或存储到数据库中; (3) 点击绘图面板上方“生成项目”按钮可以生成框架项目文件。
异常流程	保存建模结果时未填写名称, 提示填写

非功能性需求

“模型存储与转化”模块的非功能性需求主要包括可扩展性、可靠性和性能, 具体需求如下:

- 可扩展性: 工具采用前后端分离的结构, 可以对前端或后端组件进行替换或扩展。满足存储持久化设施多样性, 为后续存储更多模型提供保障。
- 可靠性: 工具支持本地文件系统的导入导出, 不强依赖远程数据库, 满足各种网络条件下的建模需求。
- 性能: 对于数据存储和检索使用了 ElasticSearch, 基于倒排索引能达到高效的全文搜索 [36]。满足搜索模型的即时性, 让工具使用速度提升。

4.3 工具设计与实现

4.3.1 总体设计

本工具的架构根据典型的四层架构 [37] 进行设计和划分, 整体架构图如图 4-6所示。第一层表现层负责向用户展示业务流程和数据, 包含可视化建模与验证修改模型等功能; 第二层服务层, 包含对模型的存储以及读取、对模型的二次校验和框架项目文件生成功能; 第三层持久层主要保存了业务模型对象的数据, 即对实体、值对象和领域服务等对象数据进行保存; 第四层数据层主要负责建模结果的保存和读取。

- 表现层: 该层为用户提供访问入口, 通过该层用户可以直观地使用“可视化建模”、“模型校验”等工具核心功能。工具前端采用 Vue.js 框架结合 ElementUI 组件进行开发, 使整个页面整洁美观, 交互良好; 对于图形的拖拽和绘制, 采用开源框架 mxGraph 作为底层支持, mxGraph 拥有优秀的交互和卓越的性能, 已经在许多商业软件中被使用, 能有效支持灵活的图形编辑; OCL.js 主要用来支持模型规范和标准的校验, 充当对象约束语言和前端 JavaScript 之间的媒介, 可以直接使用 JavaScript 完成 OCL 的功能。

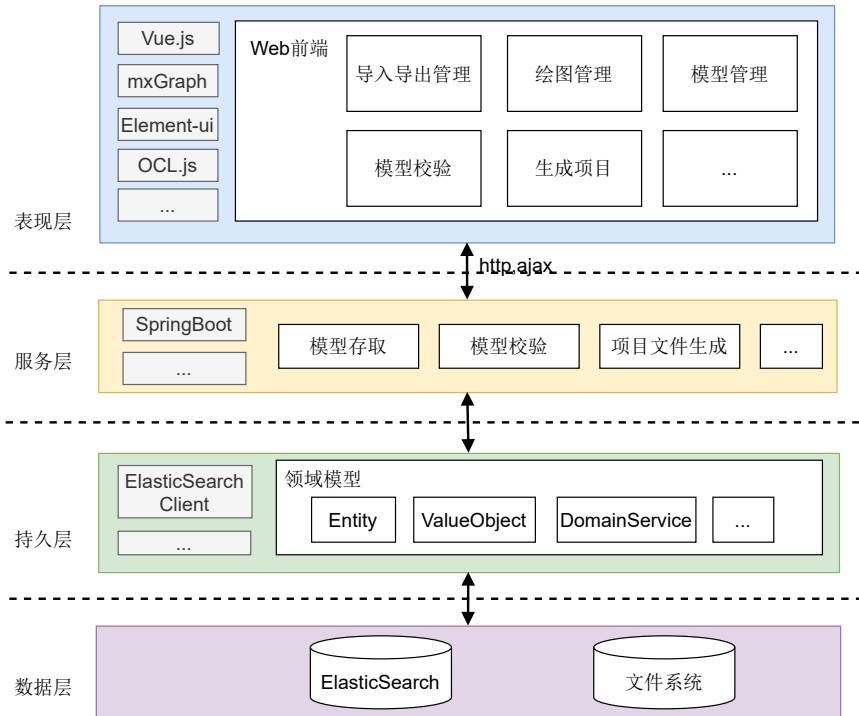


图 4-6: 支持工具整体架构图

- 服务层：该层主要提供后端接口服务和对建模逻辑的二次验证。后端整体采用 Spring Boot 框架进行开发，封装抽象了核心的业务逻辑模型（实体、值对象等），提供了模型存取与框架项目文件生成的业务逻辑。
- 持久层：该层主要充当数据层与服务层间交互的媒介，将业务逻辑模型对象转化为底层数据。持久化存储采用 ElasticSearch Client 与数据层进行通信，通过 RESTful 接口进行写入和读取。
- 数据层：该层是底层的存储数据依赖，模型底层存储格式为 XML 文件，文件篇幅较长，但对读写并发要求不高。采用两种数据基础设施进行实现，采用 ElasticSearch 支持大篇幅文档型数据存储，提高模型搜索的检索效率，也可根据用户需要直接存储到本地文件系统。

建模支持工具 DDDD 是一个前后端分离的 Web 应用。前端采用 mxGraph 结合 Vue.js 进行开发实现，后端采用 Spring Boot 框架，以 ElasticSearch 作为持久化机制进行实现，前后端采用 RESTful 风格进行交互 [38]。

由于建模过程的可视化描述和模型校验反馈是工具的核心内容，负责与用户交互和展示模型的 Web 前端的架构也有一定复杂度，故采用如图 4-7 所描述的整洁架构 [39] 对前端进行划分。整洁架构是一种与数据库、客户端接口以

及框架都无关的架构，可以用来构建复杂的前端项目，将业务逻辑抽离，但又不依赖后端，保证项目的可扩展性。本工具使用整洁架构将前端划分为三层，包括表现层、领域层和数据层。表现层包含绘图、校验、导入导出以及菜单等与用户交互的模块；领域层包含展示模型和校验反馈的底层业务逻辑，如拖拽绘制、校验模型、导入导出模型、画布管理、路由管理以及生成项目等功能；数据层包含调用后端 API 连接的远程数据源和本地数据源，负责存储和获取模型。

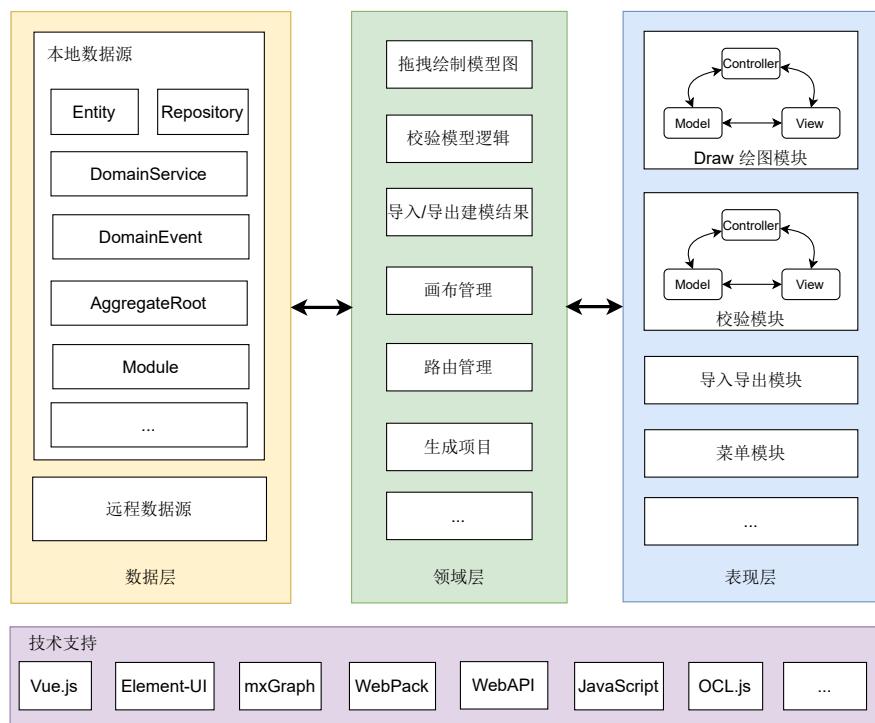


图 4-7: 支持工具前端整洁架构图

4.3.2 可视化建模模块详细设计与实现

本小节将介绍“可视化建模”模块的详细设计与实现。详细设计部分，通过类图展现“可视化建模”模块的静态详细设计，通过顺序图展现“可视化建模”模块的动态业务流程详细设计。实现部分，通过部分核心功能的实现代码来进行展现。“可视化建模”模块具体的设计如下：

可视化建模类图

“可视化建模”模块类图如图 4-8 所示。本工具提供对建模过程可视化的支持，主要由 Canvas 类对 mxGraph 中封装好的画图工具类接口进行调用，并确

保工具类是单例的，**Canvas** 类还负责工具运行时对画布的初始化，保证画图时的线条、箭头和颜色符合预期，最后，还包含绘制新图形时的判断逻辑。**Toolbar** 类主要提供了建模时对模式的选择支持，可以直接从中点击拖动实现选择战术模式。**Menu** 类包含了对画布操作的一系列接口。**Container** 类负责统一管理 **Toolbar**、**Canvas** 和 **Menu** 类，并收集反馈信息与用户进行交互。

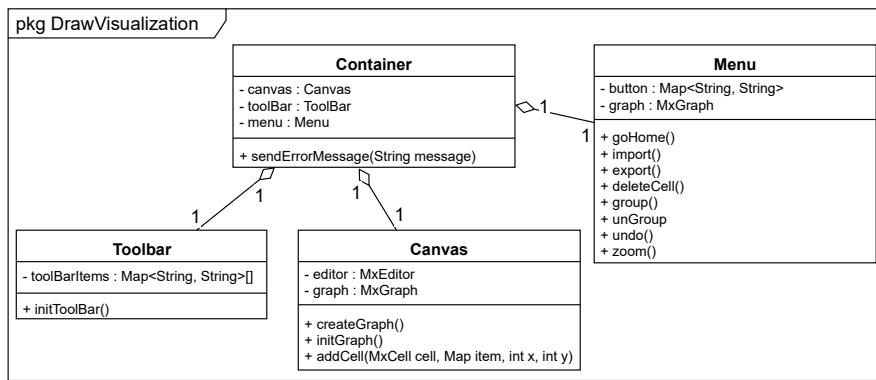


图 4-8: 可视化建模模块类图

可视化建模顺序图

“可视化建模”模块的示例过程如图 4-9 所示。首先开发人员或架构师通过使用 **Container** 类初始化画布、工具栏等可视化建模必要元素，通过 **Toolbar** 类实现拖拽图形化模式对象，并在合适的位置由 **Canvas** 类绘制，在建模过程中，通过菜单可以实现撤销、重做、组合、删除、缩放画布等一系列操作，辅助完成建模过程，在建模过程中画布会实时反馈信息给用户。

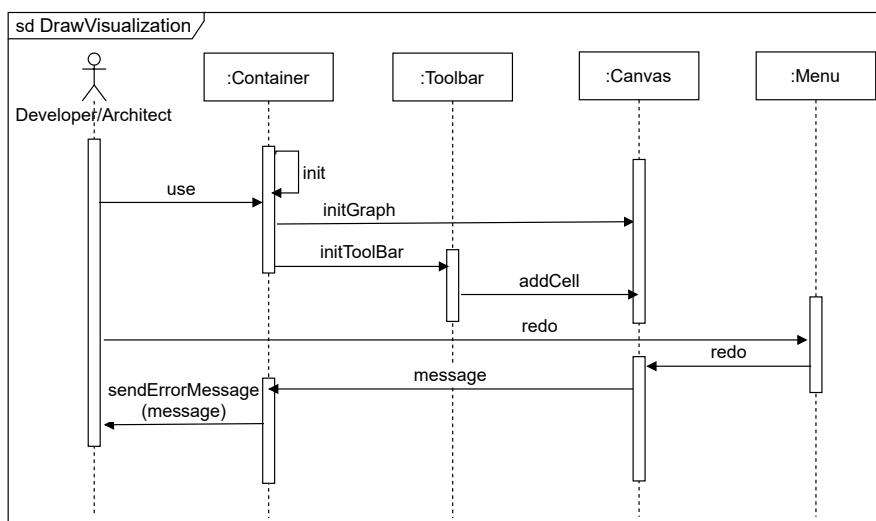


图 4-9: 可视化建模模块顺序图

可视化建模实现代码

如图 4-10 所示为“可视化建模”核心功能工具栏的初始化代码。代码描述了获取图形对象数组、拖动图形、绘制图形的逻辑。

```
//核心方法，让工具箱项目可以被拖拽，放手时进行绘图
initToolbar() {
    //获取dom数组
    const domArray = this.$refs.toolbarItems;
    if (!(domArray instanceof Array) || domArray.length <= 0) {
        return;
    }

    domArray.forEach((dom, domIndex) => {
        //获取dom数组中每个item
        const toolbarItem = this.toolbarItems[domIndex];
        const {width, height} = toolbarItem;
        //释放对象图形时调用addCell函数
        const dropHandler = (graph, evt, cell, x, y) => {
            this.addCell(graph, toolbarItem, x, y);
        }
        //创建拖动时的预览图像
        const createDragPreview = () => {
            const elt = document.createElement('div');
            elt.style.border = '2px dotted black';
            elt.style.width = `${width}px`;
            elt.style.height = `${height}px`;
            return elt;
        }
        // 获取绘制cell位置的信息，如果释放位置有cell，判断该cell是否允许子元素加入
        const getDropTarget = (graph, x, y) => {
            const cell = graph.getCellAt(x, y);
            return this.R.propOr(null, 'dropAble', cell) ? cell : null;
        }
        //使用mxUtils类使工具栏选项可以被拖动
        mxUtils.makeDraggable(dom, this.graph, dropHandler,
            createDragPreview(), 0, 0, false, true,
            true, getDropTarget);
    })
}
```

图 4-10: 初始化 ToolBar 代码

4.3.3 模型校验模块详细设计与实现

本小节将介绍“模型校验”模块的详细设计与实现。详细设计部分，通过类图展现“模型校验”模块中各战术模式的静态详细设计；实现部分，通过校验算法展现“模型校验”模块的验证逻辑的设计，通过部分实现代码展示具体实现。“模型校验”模块具体的设计如下：

模型校验类图

“模型校验”模块类图如图 4-11 所示。本工具提供对建模结果的校验，战术模式统一继承自 Pattern 类，Pattern 类描述了模式的名称、类型、连接输入以及连接输出，并在 validation 方法中调用 OCLEngine 对象实例完成对象约束语言的验证。PatternData 用来记录所有 Pattern 子类实例的数据细节，辅助校验过程。

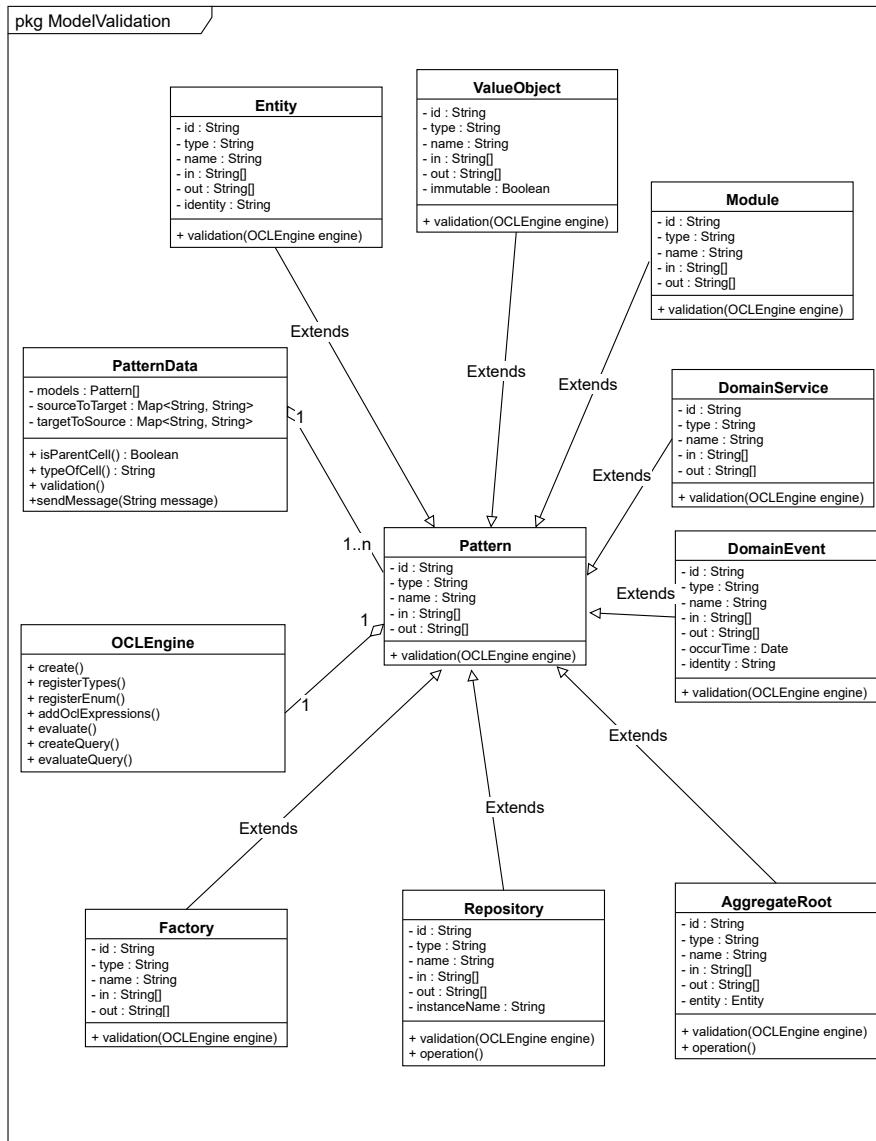


图 4-11: 模型校验模块类图

模型校验算法

“模型校验”功能的具体实现逻辑如算法 4.1 所示。该算法首先通过遍历所有图形化对象节点，建立两种连接关系双向映射，第一种描述源对象到目标

对象映射关系，第二种描述目标对象到源对象映射关系；然后进行第二次遍历，首先判断该对象节点是否为顶级节点，只有顶级节点才作为战术模式元素，如果为顶级节点，则调用 `PatternData` 类中的 `validation` 函数进行模式属性以及 OCL 约束验证，验证通过将该对象以具体模式（实体、值对象等）添加到 `patternData` 的 `models` 数据结构中，若验证失败，将布尔型成功标识变量标记为否，并退出循环体。无论验证是否通过，都将非顶级节点的子节点跳过，节省验证时间；如果不是顶级节点，则继续执行循环体；最终返回验证结果及具体建模结果。

算法 4.1 模型校验算法

```
Input: mxCells:HTMLCollectionOf<Element>
Input: patterns:PatternData
Output: models:Map<String, Pattern>
Boolean success = true;
for mxCell in mxCells do
    patterns.setSourceToTarget(mxCell.getId, mxCell.getTarget);
    patterns.setTargetToSource(mxCell.getSource, mxCell.getId);
end for
for mxCell in mxCells do
    if patterns.isParentCell(mxCell) then
        if patterns.validation(mxCell) then
            newPattern = new Pattern(mxCell);
            patterns.models.add(mxCell.getId, newPattern);
        else
            success = false;
            break;
        end if
        skip loopStep;
    else
        continue loop;
    end if
end for
if success then
    sendSuccessMessage();
    return patterns.models;
else
    sendErrorMessage();
end if
```

模型校验实现代码

如图 4-12 所示是“模型校验”功能部分实现代码。主要描述 `validation` 方法中对 `DomainService` 对象的验证过程，包括从图形化对象转化为抽象模式类的过程以及对象约束语言验证与属性验证。

```
if(typeOfCell.replace(/\s*/g,"") == 'DomainService'){

    //从图形节点获取对象的各种参数
    let id = mxCells[i].getAttribute('id');
    let name = mxCells[i+1].getAttribute('value');
    let inId = mxCells[i+2].getAttribute('id');
    let outId = mxCells[i+3].getAttribute('id');

    //创建新 DomainService对象时，内部会进行OCL验证
    let domainService = new DomainService(typeOfCell, id, name);

    //获取 入参对应的对象Id 和 出参对应的对象Id
    let inObjectId = this.targetToSource.get(inId);
    let outObjectId = this.sourceToTarget.get(outId);
    if(inObjectId != null && outObjectId != null) {
        domainService.in.push(inObjectId);
        domainService.out.push(outObjectId);
        this.models.set(id, domainService);
    }else if(inObjectId == null) {
        success = false;
        this.sendErrorMessage("领域服务没有连接输入对象!");
    }else {
        success = false;
        this.sendErrorMessage("领域服务没有连接输出对象!");
    }
}
```

图 4-12: 模型校验部分实现代码

4.3.4 模型存储与转化详细设计与实现

本小节将介绍“模型存储与转化”模块的详细设计与实现。详细设计部分，通过类图展现“模型存储与转化”模块的静态详细设计，通过顺序图展现“模型存储与转化”模块的动态业务流程详细设计。实现部分，通过部分核心功能实现代码进行展现。“模型存储与转化”模块具体的设计如下：

模型存储与转化类图

“模型存储与转化”模块的类图如图 4-13 所示。本工具提供对建模结果的存储与转化，由 **File** 类负责收集和组织图形化模式的具体信息，可以直接在浏览器客户端导出建模结果的 XML 格式文件和图像格式文件，也可以导入模型的 XML 文件，直接将模型绘制在画布上继续进行编辑；当与后端进行通信时，会将模型信息抽象为 **Model** 类，由 **ModelRepository** 负责模型的存储、修改、删除以及检索，使用 **ElasticSearch** 提供的客户端进行持久化支持，可以极大提升检索模型文件的效率。

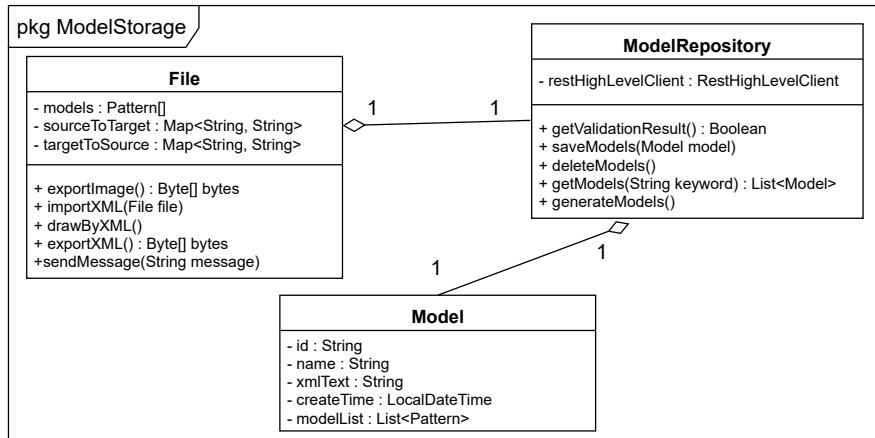


图 4-13: 模型存储与转化类图

模型存储与转化顺序图

“模型存储与转化”功能的示例过程如图 4-14 所示。开发人员或架构师使用浏览器客户端将 XML 模型文件导入画布，可以直接进行编辑，编辑完成后，可以选择导出为 XML 文件或图像格式，也可以将模型存储到数据库中并生成符合领域驱动设计规范的框架项目。

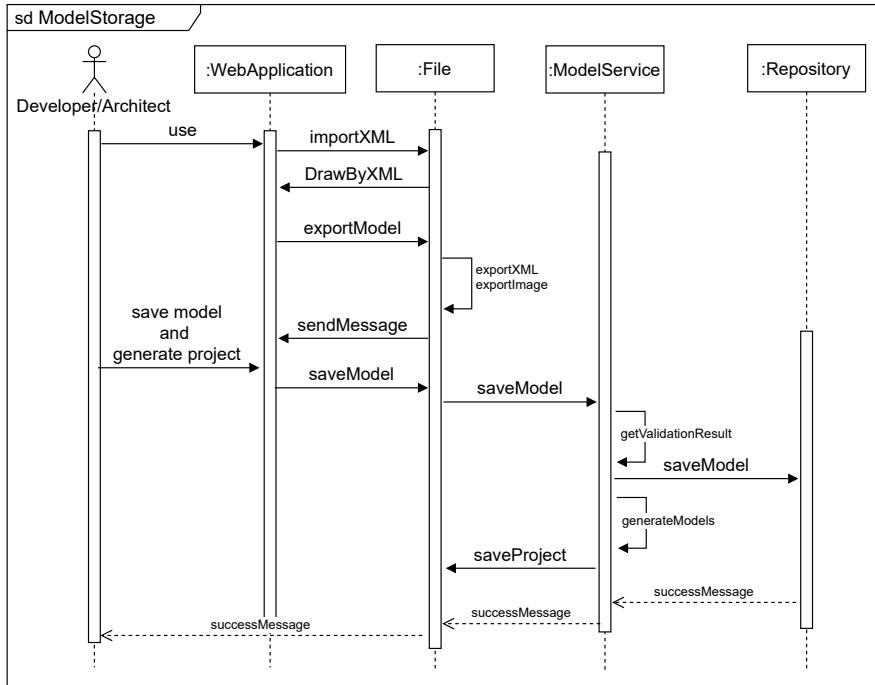


图 4-14: 模型存储与转化顺序图

模型存储部分实现代码

如图 4-15 所示是“模型存储”部分实现代码。主要描述将模型存储进 Elasticsearch 中的过程，包括对模型对象的构建、存储格式的转化、发起存储请求与接收响应。

```
public String saveModel(String name,
                        String xmlText,
                        List<PatternDTO> modelList) throws IOException {

    //日期格式化
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-
dd hh:mm:ss");

    //构建model对象
    DDDModel model = DDDModel.builder().name(name)
        .xmlText(xmlText).modelList(modelList)
        .createTime(LocalDateTime.now())
        .format(formatter)
        .build();

    //创建请求
    IndexRequest request = new IndexRequest("dddmodels");

    //规则 PUT /index/_doc/
    request.timeout(TimeValue.timeValueSeconds(1));

    //数据放入请求
    request.source(JSON.toJSONString(model), XContentType.JSON);

    //客户端发送请求，获取响应的结果
    IndexResponse response = restHighLevelClient.index(request,
    RequestOptions.DEFAULT);

    return response.status().toString();
}
```

图 4-15: 模型存储部分实现代码

4.4 本章小结

本章介绍了建模支持工具的设计与实现。首先通过对建模场景中的用例分析，获取具体需求，包括“可视化建模”、“模型校验”以及“模型存储与转化”三个主要模块。然后对工具进行了功能性与非功能性需求的分析与展示，通过需求对工具进行总体设计与详细设计。最后通过各模块功能类图、顺序图、功能算法逻辑和实现代码来展示工具的具体实现过程。

第五章 建模支持工具测试与案例研究

本章将介绍建模支持工具 DDDD 的测试与案例研究工作。介绍了“可视化建模”、“模型校验”以及“模型存储与转化”三个模块的主要功能单元测试；还以案例的形式演示了工具的完整使用流程，验证了本文提出的建模支持工具满足开发人员和架构师的战术建模需求。

5.1 建模支持工具测试

本小节主要介绍对建模支持工具的功能测试和性能测试。主要对工具的三个模块进行测试，测试的功能主要包括：“可视化建模”、“建模约束校验”、“模型存储”以及“生成项目”。对“可视化建模”功能进行单元测试，验证建模支持工具是否支持灵活易用的建模过程；对“建模约束校验”功能进行单元测试，验证建模支持工具是否支持对建模结果正确性和规范性的保障；对“模型存储”和“生成项目”功能进行单元测试，验证建模支持工具是否支持模型的保存、复用和扩展使用，最后对支持主要功能的后端接口进行了性能测试。

进行单元测试时，主要针对实现功能的关键接口进行测试。涉及前后端交互的部分均以 Postman^①工具为辅助，采取 JSON 格式进行数据传输，代替实际项目中以抽象类封装的数据。进行性能测试时，以脚本形式对接口进行多次访问，统计接口的平均响应时间。

5.1.1 可视化建模测试

表 5-1 是对“可视化建模”进行测试的测试用例具体描述。通过输入一系列鼠标拖拽操作进行测试，测试的接口是实现绘制图形的核心接口“Draw.addCell”。测试目的是验证建模支持工具的“可视化建模”功能。单元

^①Postman 工具主页：<https://www.postman.com>

测试首先使用拖拽图形的方式进行“实体类”的绘制，并双击“实体类”修改名称；然后拖拽图形进行“值对象类”的绘制，并双击修改名称；最后，进行“实体类”和“值对象类”之间的连线绘制。

测试结果为建模结果的图像表示。对“可视化建模”进行测试的结果输出如图 5-1 所示，该图展示了成功绘制“实体类”和“值对象类”的图形建模结果，结果表明“可视化建模”可以支持正常建模。

表 5-1: 可视化建模测试用例

ID	TC1
测试目标	可视化建模实体与值对象，并建立连接
测试接口	Draw.addCell
前置条件	工具画布初始化正常
输入	(1) 拖拽“实体类”，绘制在画布中，并修改实体名为“Student”，唯一标识为“ID:String”； (2) 拖拽“值对象类”，绘制在画布中，并修改值对象名为“Book”； (3) 建立实体到值对象的连接。
预期输出	包含实体和值对象的图形化建模结果

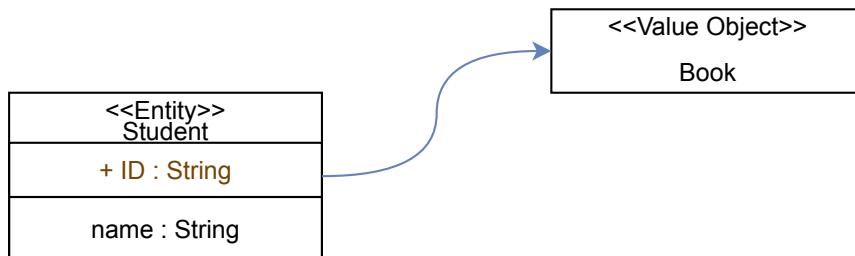


图 5-1: 图形化建模测试结果图

5.1.2 建模约束校验测试

表 5-2 是对“建模约束校验”进行测试的测试用例具体描述。输入为包含未连接任何对象的“资源库类”的建模结果，测试的接口是实现“建模约束校验”的核心接口“PatternData.validatio”。测试目的是验证对“资源库类”的“建模约束校验”功能。

由于表 3-7 中的约束 C12 规定，本次测试结果为校验失败，具体的结果如图 5-2 所示，以弹窗形式输出校验警告信息和修改提示。结果表明“建模约束校验”功能正常实现。

表 5-2: 建模约束校验测试用例

ID	TC2
测试目标	正确检测图形化建模约束问题，给出警告提示
测试接口	PatternData.validation
前置条件	工具画布初始化正常
输入	包含未连接任何对象的“资源库类”的建模结果
预期输出	校验失败信息“请给 Repository 连接一个它所存储的实体（或聚合根）”

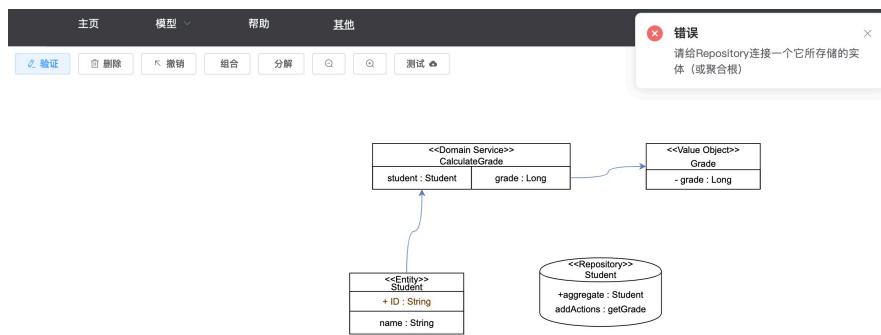


图 5-2: 建模约束校验测试结果图

5.1.3 模型存储测试

表 5-3 是对“模型存储”进行测试的测试用例具体描述。输入为模型名称以及模型 XML 格式文件，测试的接口是实现“模型存储”的核心接口“ModelRepository.saveModels”。测试目的为验证模型的“保存”功能。

数据库初始化正常且网络连接通畅，模型存储测试成功，并可以从数据库中检索出模型，绘制在画布上。结果表明“模型存储”功能实现正常。

表 5-3: 模型存储测试用例

ID	TC3
测试目标	正确存储模型，并能从模型数据库中检索重建模型
测试接口	ModelRepository.saveModels
前置条件	数据库初始化正常，网络连接通畅
输入	模型名称
预期输出	保存成功提示，成功从数据库中检索模型并重建模型

5.1.4 生成项目测试

表5-4是对“生成项目”进行测试的测试用例具体描述。输入为图形化建模结果，测试的接口是实现“生成框架项目”功能的核心接口“`ModelRepository.generateModels`”。测试目的为验证根据建模结果“生成框架项目”的功能。

测试结果为弹出生成项目成功的提示窗口，并可以在指定文件目录下访问项目文件，项目文件结构符合领域驱动设计战术建模规范。结果表明“生成项目”功能实现正常。

表 5-4: 生成项目测试用例

ID	TC4
测试目标	根据图形化建模结果生成框架项目
测试接口	<code>ModelRepository.generateModels</code>
前置条件	无
输入	图形化建模结果
预期输出	指定文件路径下的框架项目文件包

5.1.5 性能测试

性能测试部分主要针对“模型存储与转化”功能中的核心接口进行测试。测试环境如表5-5所示，描述了测试环境服务器的具体配置，将ElasticSearch部署至该测试环境中进行测试。

表 5-5: 测试环境

配置项	描述
操作系统	CentOS 7.6 64 位
CPU	Intel(R) Xeon(R) Platinum 8269CY, 4 核, 2.5 GHz/3.2 GHz
内存	8G
Java 版本	OpenJDK 1.8.0 181
ElasticSearch 版本	7.11.1

建模支持工具的所有模块中，“模型存储与转化”模块前后端交互频繁，需要经常请求持久化设施，而其他模块的前后端交互很少，几乎不需要考虑性能问题。故主要对“模型存储与转化”模块的接口进行性能测试。请求持久化设施会在网络上发生耗时的数据传输操作，所以响应时间（Response Time, RT）是衡量接口性能的核心指标，性能测试通过编写脚本对每个核心接口进行50次请求，并统计平均响应时间。具体测试结果如表5-6所示，展示了

测试接口的平均响应时间，测试的接口包括保存模型的“`saveModel`”接口，查询模型的“`getModels`”接口，删除模型的“`deleteModel`”接口，以及搜索模型的“`getModelByKeyword`”接口。表中平均响应时间说明后端接口反馈迅速，基本满足性能需求。

表 5-6: 接口性能测试

接口名	请求类型	平均响应时间
<code>saveModel</code>	PUT	50ms
<code>getModels</code>	GET	21ms
<code>deleteModel</code>	DELETE	13ms
<code>getModelByKeyword</code>	GET	18ms

5.2 案例研究

本小节对本文提出的战术建模支持方法及工具进行案例研究。通过案例演示使用支持工具进行建模的完整过程，来验证该建模支持方法和工具是否能够提供灵活易用的建模支持，提高建模效率；是否能够保证建模结果的标准化、规范化；以及是否能够满足建模结果的持久性、可复用性和扩展应用。

为验证本工具能够支持战术建模整个流程，本文以 Vaughn Vernon 的《实现领域驱动设计》[4] 中的企业协作软件系统 CollabOvation 为研究案例展开验证。

SaaSOvation 是一家企业办公软件提供商，CollabOvation 是由 SaaSOvation 公司开发的一款企业协作（Collaboration）软件，并且加入社交网络的功能。该产品的功能包括论坛、共享日历、博客、即时消息、wiki、留言板、文档管理、通知和提醒、活动跟踪和 RSS 等。所有协作工具都旨在满足企业服务的需求，帮助他们在项目中提高效率。CollabOvation 项目启动后，项目中的一些有经验的开发人员采用领域驱动设计将项目划分为协作上下文、身份与访问上下文和敏捷项目上下文，具体项目代码已发布到 GitHub^① 代码仓库。案例研究将 CollabOvation 的身份与访问上下文作为业务范围，进行战术建模。案例研究验证过程包括使用建模支持工具进行标准化战术建模的五个具体步骤，并给出平台截图展示。

^①IDDD-Samples 代码仓库地址：<https://github.com/VaughnVernon/IDDDSamples/>

5.2.1 验证步骤

本小节将对建模支持方法及工具进行验证，首先验证可视化建模能否支撑含有复杂设计的战术建模，满足图形化的建模基本需求；其次验证对精细化建模的支持程度，能否正确识别建模中不规范的操作，并给予相应的提示；最后验证建模结果的存储与转化使用，能否正确存储建模结果并生成框架项目指导编码工作。

Vaughn Vernon 在《实现领域驱动设计》[4] 中已经详细介绍了项目的背景和需求，但没有进一步细化划分出的限界上下文，以下的验证步骤的目的（也是本案例研究的目的）在于探究本文所提出的建模支持方法及工具能否支持对所选案例进行战术建模，具体验证工作步骤如下：

- 步骤一：**开发人员通过工具快速学习战术建模相关概念知识；
- 步骤二：**开发人员进入可视化建模页面，进行建模；
- 步骤三：**对于不符合战术建模规范的操作，没有通过战术模式约束校验的属性弹出警告提示；
- 步骤四：**建模结果校验通过，将模型导出为 XML 文件和图片，保存到数据库中，并生成框架项目文件；
- 步骤五：**读取保存的模型文件或数据库中存储的模型，进行二次建模。

5.2.2 验证过程

本小节将依据上一节所述的步骤对支持工具进行演示和验证。

步骤一：点击工具平台首页的“Can I DDD”按钮，进入展示战术建模相关概念知识的帮助页面，如图 5-3 所示为帮助页面。该页面帮助开发人员快速了解战术建模相关概念与使用场景，回顾战术建模知识，降低使用工具的门槛。



图 5-3: 工具帮助页面图

步骤二：开发人员或架构师进入可视化建模页面，根据项目背景需求进行战术建模，由于整个战术建模过程时间跨度长，模型复杂，故仅选择战术建模的一部分进行展示：

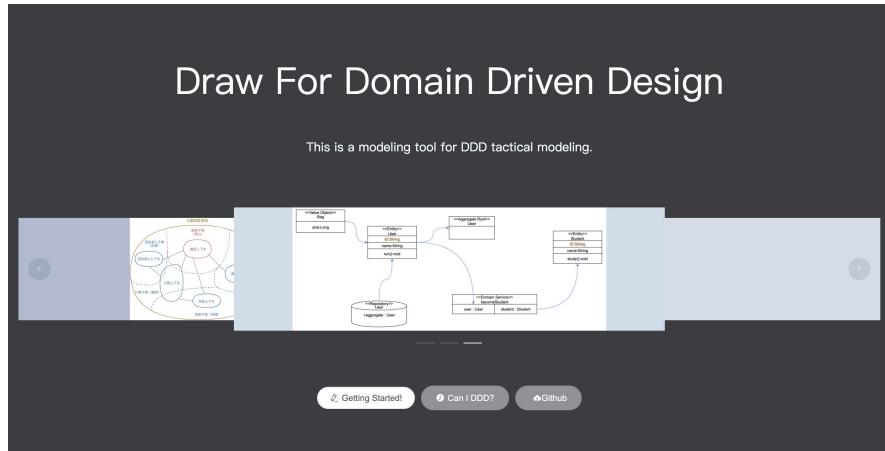


图 5-4: 工具主页

1. 工具主页如图 5-4 所示，点击“Getting Started!”按钮进入建模页面；
2. 进入建模页面后，拖动左侧工具栏实体、值对象以及领域事件模式图，放在右侧画布合适位置，进行绘制；
3. 双击需要修改的模式属性，输入文字进行修改；
4. 点击对象，按住拖动进行对象间连线；
5. 点击右侧画布菜单按钮，可以对建模操作进行撤销，对画布进行缩放，将建模对象进行组合和删除。

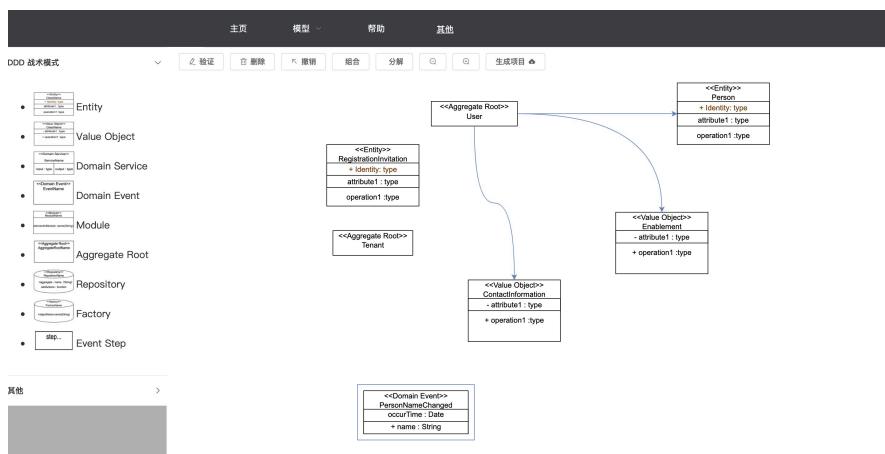


图 5-5: 可视化建模过程

如图 5-5 所示，展示了上述步骤二的建模过程最终结果。

步骤三：建模时可以对建模结果进行实时校验，校验规则严格按照本文提出的战术建模支持方法要求，并给出校验结果展示：

1. 进行可视化建模时，点击右侧画布菜单按钮“验证”；
2. 如图 5-6 所示，根据当前图形化建模结果，进行约束和规则校验，弹出不符合规范和约束的警告信息。

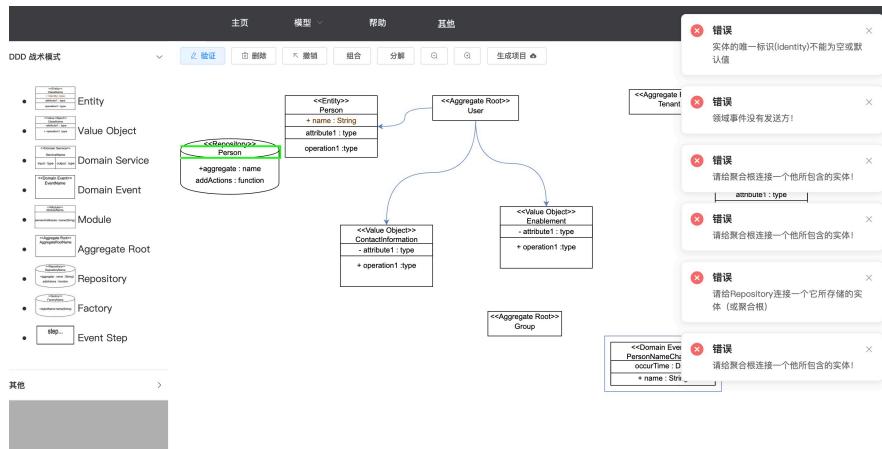


图 5-6: 模型校验

步骤四：选择菜单如图 5-7 所示，建模完成后，经过校验模型符合规范，可以将模型以 XML、图像形式导出，并存储到数据库中。

1. 建模完成，校验通过；
2. 在菜单中依次选择“模型”、“导出”、“XML 格式”，选择保存文件夹；
3. 在菜单中依次选择“模型”、“导出”、“图像格式”，选择保存文件夹；
4. 在菜单中依次选择“模型”、“保存模型到数据库”，将模型命名为“userAccess”，点击确定；
5. 在画布菜单中点击按钮“生成项目”，框架项目文件在指定文件目录中生成。



图 5-7: 保存导出模型

步骤五：再次使用工具时，可以使用“导入”和“加载历史模型”功能，来重现建模结果，并进行二次建模。

- 在菜单中依次选择“模型”、“加载历史模型”，持久化设施中保存的模型文件列表如图 5-8 所示，展示了模型名称、创建时间以及操作按钮；

历史模型		
模型名称	创建时间	操作
testModel3	2021-03-10 02:33:24	查看详情 加载模型 删除
userAccess	2021-03-12 06:49:13	查看详情 加载模型 删除
testModel1	2021-03-12 06:49:39	查看详情 加载模型 删除
student	2021-03-12 06:49:50	查看详情 加载模型 删除
car	2021-03-12 06:50:01	查看详情 加载模型 删除

图 5-8: 加载历史模型

- 选择想要加载的模型，点击“加载模型”按钮，画布中将绘制出模型；
- 加载的历史模型如图 5-9 所示，可以对历史模型进行新的操作。

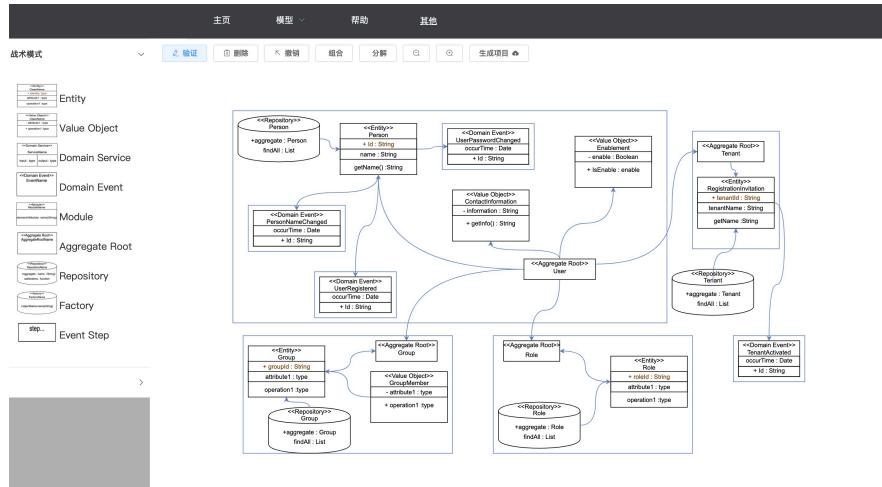


图 5-9: 建模结果图

通过上述五个步骤的验证，得到验证结果如表 5-7 所示，表格展示了每个步骤对应的功能需求和验证的结果。以上步骤和得到的结果表明，提出的建模

支持方法及工具达到了以下效果。第一，实现了可视化战术建模，使建模过程更加高效灵活；第二，对建模结果进行约束验证，来判断是否符合战术建模需要遵守的规则，解决了战术建模阶段流程化与标准化的规范性问题；第三，可以将建模结果进行存储，方便下次访问，并提供根据建模结果生成框架项目的扩展功能，提高了建模结果的可复用性和扩展性。

表 5-7：案例验证结果

步骤	需求	验证结果
步骤一	帮助开发者了解或回顾战术建模概念知识	将来自书籍和调研的经验理论总结成体系，供开发者学习，验证结果表明实现的工具满足开发者了解和回顾知识的需求
步骤二	通过拖拉拽图形和连线、双击修改文本，达到可视化建模	通过工具对“User”、“Person”和“Tenant”等领域概念进行建模，实现的工具达到了可视化建模的需求
步骤三	对建模结果进行校验，并给出修改提示	点击工具中“验证”按钮，对画布中现有的对象进行校验，弹出警告提示，实现的工具达到了验证建模结果标准性与规范性的需求
步骤四	将建模结果转化为多种形式保存到本地，同时将建模结果保存到远程数据库	通过工具保存了建模结果的 XML、图像和框架项目文件到本地，同时也保存了建模结果到远程数据库，实现的工具达到了存储模型的需求
步骤五	将建模结果再次从本地导入或从远程数据库加载	通过工具导入了本地 XML 格式的模型文件，也可以加载远程数据库的模型，实现的工具达到了再次编辑已有模型的需求

综上所述，本文提出的战术建模支持方法及工具满足战术建模的功能性需求，可以支撑战术建模全流程的实践。

5.3 本章小结

本章介绍了对战术建模支持方法及工具的测试和案例研究。首先设计测试用例对建模支持工具的“可视化建模”、“建模约束校验”、“模型存储”以及“生成项目”功能进行了测试，还通过在具体案例中执行建模的五个步骤对建模支持方法及工具进行了验证。

第六章 总结与展望

6.1 总结

领域驱动设计作为一种针对复杂业务流程的分析与建模方法，正在成为大型分布式系统设计与实现的最佳解决方案，其中的战术建模层次可以帮助更快地进行建模设计与实现代码的落地。然而，战术建模在应用时仍面临着许多挑战，由于对战术模式的规范和约束不明确，往往会导致战术建模结果不够标准和规范；开发人员和架构师对战术建模理解程度不同，无法统一和复用建模结果；领域建模过程缺少相应的支持平台和工具。为了解决这些挑战，对该领域的研究进展与工业界实践经验进行了调查与总结，提出了一套战术建模支持方法及工具。

具体来说，本文所提出的战术建模支持方法及工具包括以下三个具体贡献。其一，通过对领域驱动设计战术建模过程的理论调研和对工业界从业人员的访谈，本文总结得出一套战术建模指南，包括八种战术模式、这些模式的重要属性、使用时机以及实现技术，该建模指南经过工业界实践经验认证，可以作为战术建模实践时的指导。其二，通过开展焦点小组讨论，本文构建了一种战术建模语言，该战术建模语言作为使用战术建模支持方法的支撑，提升了建模的效率与准确性。以上两点构成了战术建模支持方法的主要内容。其三，以所提出的战术建模语言为基础，本文还实现了一个建模支持工具，实现了灵活易用的可视化建模过程，保证了扩展性与通用性，并避免了对特定平台的依赖。

本文提出的战术建模支持方法和工具，支持简单了解领域驱动设计战术建模基本概念，通过灵活的可视化建模来实现建模过程，并校验建模结果的标准性与规范性，为战术建模提供了一套可靠的流程。此外，还支持将建模结果以多种格式导出和存储，还能为实现阶段构建框架项目，提高了战术建模结果的复用程度。总体而言，本文提出的工具也降低了使用战术建模的最低要求，对领域驱动设计在实践中发展具有积极意义。

6.2 展望

本文提出的领域驱动设计战术建模支持方法及工具可以支持战术建模实践，但仍有很多方面可以进一步提升。首先，战术建模的属性和实现技术在实践时还应该做到根据业务特性进行变通，对不同开发团队的不同业务，战术建模工具应该有不同的个性化配置；其次，战术建模工具对于战术建模的结果利用程度还不够高，除了生成框架项目之外，未来该工具还将支持更多多种形式与更细粒度的扩展结果；最后，战术建模工具在多人协同建模方面支持度不够，仅能通过共享建模结果来进行协同，未来该工具还将支持通过在线协作实现即时协同建模，提升建模过程的体验并提高建模效率。

致 谢

在本篇论文最后，我想真诚地感谢在我撰写论文过程中指导和帮助我的老师、同学们，还有支持我、鼓励我的家人和朋友们。

首先，我要感谢在我毕设研究工作中，张贺教授对我的悉心指导和帮助，在跟随您的两年研究生学习工作中，您总是教导我们要多去实践，要以严谨认真的态度踏踏实实地完成科研学习工作，还要努力阅读大量学术文献，提高自己看问题的层次，还为我们提供了很多与工业界交流的机会，这些都为我的毕设工作和日常工作学习带来了很大的好处。

然后，我要感谢实验室的师兄师姐和同学们，感谢你们这两年中在学习、工作和生活中带给我的帮助与鼓励，特别是指导我毕设工作的师姐，对我的问题一一解答，对我的论文工作耐心指导，启发我进行深度的思考，反复与我评估和讨论论文研究的问题，得益于她的帮助，我的论文才能保质保量地顺利完成。

最后，还要感谢我的父母和家人，感谢你们在背后默默支持和陪伴我。很幸运，在研究生阶段遇到了相互扶持，互相鼓励的伴侣，在生活和学习上我们互相帮助，给予彼此很大的动力与希望。也要感谢我的舍友们，让我的生活中充满了乐趣。

在论文工作中，访谈和焦点小组中付出了大量的精力时间，实现工具时也遇到过困难与问题，但我依然坚持尽力做到最好，马上就要从学校步入社会，我将坚持“诚朴雄伟，励学敦行”的精神，不畏艰难，努力奋斗出属于自己的一片天地。

参考文献

- [1] EVANS E. Doain-design - tackling complexity in the heart of software[M]. [S.l.] : Addison-Wesley, 2004.
- [2] MILLETT S, TUNE N. Patterns, principles, and practices of domain-driven design[M]. [S.l.] : John Wiley & Sons, 2015.
- [3] NADAREISHVILI I, MITRA R, MCLARTY M, et al. Microservice architecture: aligning principles, practices, and culture[M]. [S.l.] : O'Reilly Media, Inc., 2016.
- [4] VERNON V. Implementing domain-driven design[M]. [S.l.] : Addison-Wesley, 2013.
- [5] FRANK U. Domain-specific modeling languages: requirements analysis and design guidelines[G] // Domain engineering. [S.l.] : Springer, 2013 : 133 – 157.
- [6] WU H, HINSBERGER L, TIMONEY J. A workflow for healthcare systems via OCL and SMT solving: short paper[C] // Proceedings of the International Workshop on Software Engineering in Healthcare Systems. 2018 : 42 – 45.
- [7] KÜHLWEIN A, PAULE A, HIELSCHER L, et al. Firmware Synthesis for Ultra-Thin IoT Devices Based on Model Integration[C] // 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). 2019 : 339 – 346.
- [8] JUMAGALIYEV A, ELKHATIB Y. A Modelling Language to Support the Evolution of Multi-Tenant Cloud Data Architectures[C] // 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS). 2019 : 139 – 149.
- [9] KOLOVOS D, ROSE L, PAIGE R, et al. The epsilon book[J]. Structure, 2010, 178 : 1 – 10.

- [10] RADEMACHER F, SACHWEH S, ZÜNDORF A. Towards a uml profile for domain-driven design of microservice architectures[C] // International Conference on Software Engineering and Formal Methods. 2017 : 230 – 245.
- [11] HIPPCHEN B, SCHNEIDER M, GIESSLER P, et al. Systematic Application of Domain-Driven Design for a Business-Driven Microservice Architecture[J]. International Journal on Advances in Software, 2019, 12(3–4).
- [12] DIEPENBROCK A, RADEMACHER F, SACHWEH S. An ontology-based approach for domain-driven design of microservice architectures[J]. INFORMATIK 2017, 2017.
- [13] SMITH B. Ontology[J], 2003.
- [14] 刘辉, 麻志毅, 邵维忠. 元建模技术研究进展 [D]. [S.I.] : Citeseer, 2008.
- [15] GUERRIERO M, NESTA A, DI NITTO E. Streamgen: a UML-based tool for developing streaming applications[C] // Proceedings of the 10th International Workshop on Modelling in Software Engineering. 2018 : 57 – 58.
- [16] LA FOSSE T B, TISI M, BOUSSE E, et al. Towards platform specific energy estimation for executable domain-specific modeling languages[C] // 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). 2019 : 314 – 317.
- [17] LED M, DANG D-H, NGUYEN V-H. On domain driven design using annotation-based domain specific language[J]. Computer Languages, Systems & Structures, 2018, 54 : 199 – 235.
- [18] KAPFERER S, ZIMMERMANN O. Domain-specific Language and Tools for Strategic Domain-driven Design, Context Mapping and Bounded Context Modeling.[C] // MODELSWARD. 2020 : 299 – 306.
- [19] MERSON P, YODER J W. Modeling Microservices with DDD[C] // 2020 IEEE International Conference on Software Architecture Companion, ICSA Companion 2020, Salvador, Brazil, March 16-20, 2020. [S.I.] : IEEE, 2020 : 7 – 8.

- [20] EMERSON M, SZTIPANOVITS J. Techniques for metamodel composition[C] // OOPSLA–6th Workshop on Domain Specific Modeling. 2006: 123–139.
- [21] SOLEY R, OTHERS. Model driven architecture[J]. OMG white paper, 2000, 308(308): 5.
- [22] COAD P, YOURDON E, COAD P. Object-oriented analysis : Vol 2[M]. [S.l.] : Yourdon press Englewood Cliffs, NJ, 1991.
- [23] JACOBSON I. The use-case construct in object-oriented software engineering[G] // Scenario-based design: envisioning work and technology in system development. 1995: 309–336.
- [24] 王文玲, 金茂忠. UML 模型及其应用 [D]. [S.l.] : [s.n.], 1999.
- [25] PAHL C, JAMSHIDI P. Microservices: A Systematic Mapping Study.[C] // CLOSER (1). 2016: 137–146.
- [26] CABOT J, GOGOLLA M. Object constraint language (OCL): a definitive guide[C] // International school on formal methods for the design of computer, communication and software systems. 2012: 58–90.
- [27] SOBERNIG S, HOISL B, STREMBECK M. Extracting reusable design decisions for UML-based domain-specific languages: A multi-method study[J]. Journal of Systems and Software, 2016, 113: 140–172.
- [28] BÉZIVIN J. In search of a basic principle for model driven engineering[J]. Novatica Journal, Special Issue, 2004, 5(2): 21–24.
- [29] KARDAS G, TEZEL B T, CHALLENGER M. Domain-specific modelling language for belief–desire–intention software agents[J]. IET Software, 2018, 12(4): 356–364.
- [30] BIAŁECKI A, MUIR R, INGERSOLL G, et al. Apache lucene 4[C] // SIGIR 2012 workshop on open source information retrieval. 2012: 17.
- [31] FATTABI E, BIDAR M, KANAN H R. Focus group: an optimization algorithm inspired by human behavior[J]. International Journal of Computational Intelligence and Applications, 2018, 17(01): 1850002.

-
- [32] EVANS E. Domain-Driven Design Reference: Definitions and Pattern Summaries[M]. [S.l.]: Dog Ear Publishing, 2014.
 - [33] GIACHETTI G, MARÍN B, PASTOR O. Integration of domain-specific modelling languages and UML through UML profile extension mechanism.[J]. IJCSA, 2009, 6(5): 145 – 174.
 - [34] GIACHETTI G, MARÍN B, PASTOR O. Using UML as a domain-specific modeling language: a proposal for automatic generation of UML profiles[C] // International Conference on Advanced Information Systems Engineering. 2009 : 110 – 124.
 - [35] FILES A N M C. Omg unified modeling language tm (omg uml)[J]. Object Management Group, 2013 : 1 – 752.
 - [36] DIVYA M S, GOYAL S K. ElasticSearch: An advanced and quick search technique to handle voluminous data[J]. Compusoft, 2013, 2(6) : 171.
 - [37] 王君. 基于 Struts+ Spring+ Hibernate 的企业级 WEB 应用框架的研究 [D]. [S.l.]: 合肥: 合肥工业大学, 2007.
 - [38] RICHARDSON L, RUBY S. RESTful web services[M]. [S.l.]: O'Reilly Media, Inc., 2008.
 - [39] MARTIN R C. Clean architecture: a craftsman's guide to software structure and design[M]. [S.l.]: Prentice Hall, 2018.

附录 A 访谈问题

A.1 战术建模访谈问题

一、个人信息

1. 您的职位是?

- A. 开发人员
- B. 架构师
- C. 项目管理者
- D. 运维人员
- E. 其他

2. 您所在团队的人员规模?

- A. 5 人以下
- B. 5-10 人
- C. 10-20 人
- D. 20-50 人
- E. 50 人以上

3. 您所在团队业务关注的领域?

- A. 软件/IT 领域
- B. 通信领域
- C. 软件提供商
- D. 电商领域
- E. 银行和金融领域
- F. 其他

二、DDD 建模总体实践

1. 您的团队按照领域驱动设计进行领域建模的流程是什么?

- a) 分析建模、设计建模和实现建模的顺序
- b) 领域模型图和代码实现之间的关系

2. 您的团队是否有使用某种建模工具进行领域建模? 是什么建模工具? 它的优点缺点是什么?

- a) DDD 专用的建模套件工具
- b) 传统的类图工具, 如 EA、PlantUML、visual paradigm
- c) 使用纸和笔直接进行建模

三、DDD 战术建模模式实践

1. 关于实体

- a) 您认为何时应该使用实体?
- b) 您认为实体应该包含哪些重要属性?
 - i. 实体应该拥有唯一标识, 方便跟踪和与其他实体区分
 - ii. 实体在其生命周期内可以连续变化
- c) 除上述属性外, 还应该包括哪些属性?
- d) 您的团队在实现实体时有哪些最佳实践?
 - i. 如何建模唯一标识
 - ii. 唯一标识通常由哪些组成部分
 - iii. 如何为标识生成唯一值, 即如何生成唯一标识
 - iv. 什么时候生成唯一标识
 - v. 如何存储唯一标识
 - vi. 如何保证唯一标识的稳定性
- e) 您认为在建模实体时有哪些不好的实践是需要避免的?

2. 关于值对象

- a) 您认为何时应该使用值对象?
- b) 您认为值对象应该包含哪些重要属性?
 - i. 不变性
 - ii. 可替换性
 - iii. 值对象相等性
 - iv. 无副作用行为
 - v. 是一个概念整体
- c) 除上述属性外, 还应该包括哪些属性?
- d) 您的团队在实现值对象时有哪些最佳实践?
 - i. 如何保证值对象的不可变性
 - ii. 如何实现无副作用行为

iii. 如何存储值对象集合

e) 您认为在建模值对象时有哪些不好的实践是需要避免的?

3. 关于服务

a) 您认为何时应该使用服务?

b) 您认为领域服务应该包含哪些重要属性?

i. 无状态

c) 除上述属性外, 还应该包括哪些属性?

d) 您的团队在实现领域服务时有哪些最佳实践?

i. 如何组织领域服务, 领域服务放在项目中的什么位置

e) 您认为在建模领域服务时有哪些不好的实践是需要避免的?

4. 关于领域事件

a) 您认为何时应该使用领域事件?

b) 您认为领域事件应该具有哪些属性

i. 事件发生时间

ii. 事件的发送方或造成事件的对象

iii. 不变性

iv. 唯一标识

c) 除上述属性外, 还应该包括哪些属性?

d) 是否应该将领域事件建模成聚合?

e) 您的团队在实现领域事件时有哪些最佳实践?

i. 如何命名领域事件

ii. 如何存储事件发生的时间

iii. 如何保证领域事件的不变性

iv. 如何避免将领域模型暴露给消息设施

v. 如何保证在事件发布前完成订阅

vi. 如何保证领域模型和事件存储的一致性

vii. 存储的领域事件有何作用

viii. 如何转发领域事件

ix. 何时应该把领域事件建模成聚合

f) 您认为在建模领域事件时有哪些不好的实践是需要避免的?

5. 关于模块

a) 您认为何时应该使用模块?

- b) 您认为模块应该具有哪些属性?
 - c) 您的团队在实现模块时有哪些最佳实践?
 - i. 如何命名模块
 - ii. 模块的表现形式, Java 中的包, C++ 命名空间
 - iii. 如何组织模块
 - d) 您认为在建模模块时有哪些不好的实践是需要避免的?
6. 关于聚合
- a) 您认为何时应该使用聚合?
 - b) 您认为聚合应该包含哪些重要属性?
 - i. 边界内保持一致性
 - ii. 根实体
 - iii. 值对象/其他实体
 - c) 除上述属性外, 还应该包括哪些属性?
 - d) 您的团队在实现聚合时有哪些最佳实践?
 - i. 如何设计一个聚合
 - ii. 聚合中如何使用其他聚合或对象 (对象的导航性)
 - iii. 如何保证一致性边界
 - iv. 何时允许在单个事务中修改多个聚合
 - v. 如何隐藏聚合内部信息
 - vi. 如何引用其他聚合
 - e) 您认为在建模聚合时有哪些不好的实践是需要避免的?
7. 关于资源库
- a) 您认为何时应该使用资源库?
 - b) 您认为资源库应该具有哪些属性?
 - i. 只为聚合提供资源库 (一种和其他 pattern 的关系, 能否算作属性)
 - c) 您的团队在实现资源库时有哪些最佳实践?
 - i. 如何设计资源库, 面向集合还是面向持久化
 - d) 资源库经常与聚合相关, 是否应该将领域对象的关联关系作为属性?
 - e) 您认为在建模资源库时有哪些不好的实践是需要避免的?
8. 关于工厂
- a) 您认为何时应该使用工厂?
 - i. 委托其他对象创建复杂对象对性能的影响可以忽略时

- ii. 需要减轻客户端在创建聚合实例的负担
- iii. 确保创建实例处于正确状态时
- b) 您认为工厂应该具有哪些属性?
- c) 您的团队在实现工厂时有哪些最佳实践?
- i. 工厂应该放在何处，作为聚合的一部分还是独立成领域服务
- d) 您认为在建模工厂时有哪些不好的实践是需要避免的?

简历与科研成果

基本信息

李质颖，男，汉族，1997年2月出生，山东省济宁人。

教育背景

2015年9月 – 2019年6月 重庆大学大数据与软件学院

本科