内蒙古大学

硕士学位论文

SysModeler：一个SysML的建模工具的研究与实现

姓名：鲍鹏丽

申请学位级别：硕士

专业：管理科学与工程

指导教师：马浩海

20080530

SysModeler：一个 SysML 的建模工具的研究与实现

摘 要

出于满足系统工程的实际需要，国际系统工程学会 INCOSE(International Council on Systems Engineering)和 OMG 决定在对 UML2.0 的子集进行复用和扩展的基础上，于 2003 年提出了一种新的系统建模语言——SysML(Systems Modeling Language)，作为系统工程的标准建模语言。SysML（系统建模语言 Systems Modeling Language）是一种多用途建模语言，用于由软硬件、数据和人综合而成的复杂系统的集成体系结构说明、分析、设计及校验。

在应用 SysML 对系统工程建模时，需要相应的 SysML 建模环境。目前单独支持 SysML 的建模环境尚不存在，同时 UML 的建模环境又无法满足 SysML 建模的实际需求。因此本文提出了基于 OMGSysML1.0 规范的建模工具 SysModeler，来满足系统工程师建模的实际需求，推动系统工程理论和实践的发展。SysModeler 按照系统工程项目为中心的文档组织方式，囊括了系统工程建模时所需的若干个完整的工具，有助于整合系统工程过程。SysModeler 可以使系统工程师在直观环境中，用建模语言管理复杂系统，从而提升系统设计开发质量。

本文以 SysModeler 的设计与实现为主线，首先介绍了系统建模语言 SysML 的提出，以及 SysML 的语义和表示法，帮助进一步了解和掌握 SysML，其次分析了 SysModeler 所使用的开发平台插件 GEF 的工作机制，从而为了解软件的设计方案打下坚实的基础。然后阐述基于系统建模语言 SysML 的建模环境

SysModeler 的实现目标、体系框架的设计以及实现和使用的应用场景。

最后对系统的设计与实现的整个过程作了总结，提出了改进建议。

关键词：SysML，UML，建模工具，模型

RESEARCH AND IMPLEMENTATION OF A SYSML MODELING TOOL SYSMODELER

ABSTRACT

For meeting the actual needs of systems engineering, International Council on Systems Engineering and OMG decise to introduce a new Modeling Language - SysML (Systems Modeling Language) base on the UML2.0 in 2003, as a systems engineering standard modeling language. SysML is a general-purpose modeling language, and supports the specification, analysis, design, verification and validation of a broad range of complex systems. These systems may include hardware, software, information, processes, personnel, and facilities.

In the application of SysML to Model for systems engineering, it needs the corresponding SysML modeling environment. At present separating support SysML modeling environment does not exist, while the UML modeling environment can not meet the actual demand for SysML modeling. Therefore this paper introduce SysModeler based on standardized modeling tools OMGSysML1.0 to meet the actual demand of systems engineering, and promote systems engineering theory and practice of development. SysModeler, according to the organization way of document whose center is system engineering project, incorporates some integrity tools needed when system engineer sets up models, and it helps to integrating system engineering process. SysModeler can make the system engineers manage

complex systems using modeling language in a intuitionistic entironment , which can improve the qualities of system design and development.

The main line of this paper is SysModeler Design and Implementation.Firstly introduce the proposed of SysML, and the semantics and representation of SysML, to help further understand SysML. Secondly analys the work Mechanism of GEF which is the development platform of SysModeler, so as to understand the software design.And then elaborate the achievement aim ,system design and framework and using scenes of SysModeler.

Finally, concluded the system design and implementation of the entire process and made recommendations for improvement.

KEYWORD： SysML, UML, Modeling Tool, Model

图表目录

图 2.1 SysML 元模型体系结构.....6
图 2.2 SysML 对 UML 的扩展[10].....7
图 2.3 SysML 的包结构[10].....7
图 2.4 SysML 和 UML2.0 的关系[10].....8
图 2.5 SysML 图结构[10].....9
图 3.1 GEF 结构图[21].....14
图 3.2 GEF 的模型图[21].....15
图 3.3 EditPart 对象[21].....16
图 3.4 Draw2D 图形[22].....16
图 3.5 GEF 的视图[22].....17
图 3.6 GEF 工作机制[22].....18
图 4.1 SysModeler 体系结构.....21
图 4.2 SysModeler 的用户界面.....22
图 4.3 SysModeler 包结构图.....24
图 5.1 SysModeler 的工作平台.....25
图 5.2 模块定义图的抽象语法和具体语法.....29
图 5.3 RSW 的环境图.....43
图 5.4 RSW 的用例图.....44
图 5.5 RSW 的模块定义图.....45
图 5.6 RSW 的顺序图.....47
图 5.7 RSW 的状态机图.....48
图 5.8 RSW 的活动图.....49

原创性声明

本人声明：所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得内蒙古大学及其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

学位论文作者签名：鲍鹏刚　指导教师签名：马浩渊

日　　期：2008.5.29　日　　期：2008.5.30

在学期间研究成果使用说明书

本学位论文作者完全了解内蒙古大学有关保留和使用学位论文的规定，即：内蒙古大学研究生在校攻读学位期间论文工作的知识产权单位属内蒙古大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。作者今后使用涉及在学期间主要研究内容或研究成果，须征得内蒙古大学就读期间导师的同意；若用于发表论文，版权单位必须署名为内蒙古大学方可投稿或公开发表。

学位论文作者签名：鲍鹏刚　指导教师签名：马浩渊

日　　期：2008.5.29　日　　期：2008.5.30

第一章 绪论

1.1 课题研究背景

当前，系统工程师使用的建模语言、工具和技术种类很多，如行为图、IDEF0、N2 图等，但这些建模方法使用的符号和语义不同，彼此之间不能互操作和重用。系统工程正是由于缺乏一种强壮的“标准的”建模语言和建模工具，从而限制了系统工程师和其他学科之间的有效通信，影响了系统工程过程的质量和效率。

统一建模语言 UML(United Modeling Language)是面向对象的标准建模语言，自 1997 年 11 月 17 日被 OMG(ObjectManagement Group)批准为标准以来，UML 已经获得工业界、科技界和应用界的广泛支持。同时 UML 自身也在不断地发展和完善，目前的最新版本是 UML2.0。虽然 UML 的设计初衷是为软件开发提供一种标准建模语言，但 OMG 标准化过程也支持为特殊领域定制 UML，如系统工程领域。但是，系统工程师在使用 UML 建模的过程中，逐渐感觉到了 UML 用于系统工程的缺陷。

为了将 UML 转换成适合于系统工程的语言，OMG 和 INCOSE（International Council on Systems Engineering）发布了 UML 向系统工程扩展的提案请求，2003 年 5 月成立的 SysML 合作组正是响应于此。该组织汇集了众多工业界、政府界以及知名工具厂商的支持，经过三年的研究，定义了一种基于 UML2.0 的系统建模语言 SysML (Systems Modeling Language)。

从 2004 年 1 月发布最初版本以来，SysML 一直处于制定发展中，2005 年 1 月发布 SysML v.0.9 版本，之后根据用户反馈，于 2005 年 11 月发布并向 OMG 提交 1.0 版。SysML v.1.0a 是系统建模语言的第一个完整版本，相对于之前的版本降低了复杂度并进一步增强了语言的描述能力。2006 年 3 月 SysML 组织发布了 OMGSysML1.0 规范，2006 年 4 月 26 日在美国圣路易召开对象管理组织技术会议上被正式接受，2006 年 7 月 6 号 OMG 组织正式宣布采用 OMGSysML1.0 规范，2007 年 3 月 23 号 OMG 组织发布了 OMGSysML1.0 规范的改进版本，2007 年 9 月 1 日 OMG 发布了 OMGSysML1.0。SysML 作为系统工程的标准建模语言，和 UML 中用来统一软件工程中使用的建模语言一样，SysML 的目的是统一系统工程中使用的建模语言。SysML 是一种标准建模语言，能够支持各种复杂系统的详细说明、分析、设计、验证和确认，这些系统可能包括硬件、软件、信息、过程、人员和设备等。

在应用 SysML 对系统工程问题建模时，需要相应的 SysML 建模环境。目前单独支持

SysML 的建模环境尚不存在，同时 UML 的建模环境又无法满足 SysML 建模的实际需求。因此本文提出了基于 OMGSysML1.0 规范的建模工具 SysModeler，来满足系统工程师建模的实际需求，推动系统工程理论和实践的发展。

1.2 国内外研究现状

目前国内对 SysML 语言的研究成果不多，其中《SysML：一种新的建模语言》是 SysML0.9 规范的一个综述，文献《基于 SysML 的系统体系结构产品设计》是基于 SysML0.9 的 DoDAF（Dept . of Defense Architecture Framework）体系框架产品设计。本文是基于 SysML1.0 规范对 SysML 建模工具的研究。

同时国内对 SysML 建模工具的研究成果尚未存在，国外 Telelogic 公司新发布的 Telelogic TAU 和 Rhapsody V7.0 产品开始支持系统建模语言标准，在这些工具中集成了 SysML 的模型支持环境，但这些工具同时还支持 UML，DoDAF 等建模语言，所以说这些工具并不是专门为系统工程师量身定制的，SysML 支持环境只是其中的一个子集。而本软件 SysModeler 是专门为系统工程师设计的建模工具，系统工程师可以使用本软件对系统工程的各种问题进行建模。

1.3 课题解决的关键技术

本课题解决的关键问题为：

1) 基于 SysML 的元模型的建立。虽然 SysML 是基于 UML2.0，但是是 UML2.0 的扩展。在 UML 的基础上新增了需求图和参数图；扩展了活动图和模块图（包括模块定义图和内部模块图）；复用了用例图、顺序图、状态机图和包图。因此要建立起元模型不但要对 SysML 语义和表示法进行全面的掌握，还应了解 UML 的语义和表示法。
2) SysModeler 体系结构的确立。SysModeler 选用的开发工具是 Eclipse 的 GEF 插件，目前 GEF 框架的中文资料很少，同时 GEF 是使用的是 MVC 框架。MVC 掌握起来比较有难度。

1.4 本论文的安排

第一章，主要分析了类似的课题在国内外研究现状，研究背景以及对本课题解决的关键技术进行了简单的介绍。

第二章，从 SysML 的提出、SysML 的语义以及 SysML 的表示法等方面简单的介绍了

SysML（一种新的系统建模语言）。

第三章，简要介绍了 SysModeler 的开发平台，包括 Eclipse 工具、以及开发框架 GEF 的工作机制。

第四章，详细描述了 SysModeler 的实现目标、功能设计、体系结构以及软件的包结构。

第五章，简要介绍了 SysModeler 各个模块的实现，同时讲述了 SysModeler 的应用场景。

第六章，总结与展望，对本论文作了简单总结，并提出了今后工作的思路和意见。

第二章 SysML 简介

SysML（系统建模语言 Systems Modeling Language）是 UML 在系统工程应用领域的延续和扩展，是近年提出的系统体系结构设计的多用途建模语言，用于由软硬件、数据和人综合而成的复杂系统的集成体系结构说明、分析、设计及校验。SysML 定义为 UML2.0 外廓的通用建模语言，它复用了 UML2.0 的相对成熟的语义和表示法，并且扩展了 UML2.0。SysML 既不是一种方法学，也不是一种独立的工具。

2.1 SysML 的提出

长期以来，系统工程师都希望能够寻找一种通用的建模语言，以统一系统工程领域[1]纷繁众多的系统描述形式。而 UML(Unified Modeling Language，是对象管理组织 OMG (Object Management Group)于 1997 年采纳的，用于软件系统的可视化、详述、构造和文档化的统一建模语言[2])作为软件工程领域的标准建模语言，已经在软件界取得了巨大的成功。

但是，系统工程师在使用 UML 建模的过程中，逐渐感觉到了 UML 用于系统工程的缺陷。早在 2000 年，Ingmar Ogren[3]就探讨了定制 UML 满足系统工程需要的可能性，提出把 UML 的子集(主要是组件图)和编程语言 Ada95 的伪代码子集结合起来创建一种系统工程建模语言 SEML(Systems Engineering Modeling Language)。随后，Jakob Axelsson[4]提出扩展 UML 使之能对具有连续时间行为的物理组件建模。Terry Bahill 和 Jesse Daniels[4]提出在非软件领域如系统、硬件和算法的设计中应用 UML 工具。Conrad Bock[6]把 UML2.0 活动图与系统工程中广泛使用的增强功能流块图 EFFBD(Enhanced Functional Flow BlockDiagram)进行了比较，提出修改或扩展 UML2.0 满足系统工程的功能流建模需求。

后来，为了将 UML 转换成适合于系统工程的语言，OMG 和 INCOSE（International Council onSystems Engineering）发布了 UML 向系统工程扩展的提案请求，2003 年 5 月成立的 SysML 合作组正是响应于此[7]。该组织汇集了众多工业界、政府界以及知名工具厂商的支持，经过三年的研究，定义了一种基于 UML2.0 的系统建模语言 SysML(Systems Modeling Language)。

SysML 支持由软硬件、数据和人综合而成的复杂系统集成体系结构的说明、分析、设计及校验。它的主要目的是提高系统体系结构设计工具间的信息交互能力，建立统一的系统设计构架，实现硬件与软件等异构组件系统的有效集成，以提高系统间的互操作性水平。

SysML 是基于 UML2.0 的，在一定程度上重用了 UML 部分元模型，同时针对系统工程对 UML 进行扩展，增加了诸如需求、块、限制之类描述系统的元素和相关图形支持，最终确保它支持诸如 DoDAF/C4ISR 在内的体系结构框架标准。同时它也通过包含 AP-233（系统工程数据交换标准）提高了其与系统开发及测试工具的兼容性[8]。

从2004年1月发布最初版本以来，SysML一直处于制定发展中，2005年1月发布SysML v.0.9版本[9]，之后根据用户反馈，于2005年11月发布并向OMG提交1.0版[10]。SysML v.1.0a是系统建模语言的第一个完整版本，相对于之前的版本降低了复杂度并进一步增强了语言的描述能力。2006年3月SysML组织发布了OMGSysML1.0规范，2006年4月26日在美国圣路易召开对象管理组织技术会议上被正式接受，2006年7月6号OMG组织正式宣布采用OMGSysML1.0规范[11]，2007年3月23号OMG组织发布了OMGSysML1.0规范的改进版本，2007年9月1日OMG发布了OMGSysML1.0。SysML作为系统工程的标准建模语言[12]。和UML中用来统一软件工程使用的建模语言一样，SysML的目的是统一系统工程中使用的建模语言。

SysML 是一种标准建模语言，能够支持各种复杂系统的详细说明、分析、设计、验证和确认，这些系统可能包括硬件、软件、信息、过程、人员和设备等。SysML 的定义包括 SysML 语义和 SysML 表示法两个部分。SysML 的语义用于对现实世界进行抽象和描述，而 SysML 的表示法定义了 SysML 符号的表示方法，为开发者或开发工具使用这些图形符号和文本语法进行系统建模提供了标准，在语义上它是 SysML 元模型的实例。

2.2 SysML 的语义

SysML为系统的结构模型、行为模型、需求模型和参数模型定义了语义。结构模型强调系统的层次以及对象之间的相互连接关系，包括类和装配。行为模型强调系统中对象的行为，包括它们的活动、交互和状态历史。需求模型强调需求之间的追溯关系以及设计对需求的满足关系。参数模型强调系统或部件的属性之间的约束关系。SysML为模型表示法提供了完整的语义。

2.2.1 元模型理论

SysML的语义是基于SysML元模型的，它同时支持对元模型的扩展定义。在说明SysML的语义时，用元模型来说明建模概念的语义[12]。元模型（metamodel）由描述模型、构建模型、并为模型的实例化和自定义机制提供必要支持的元信息组成。简单地说，元模型是用

来描述模型的模型，定义了建模中可用的构造物及其性质。和 UML 一样，SysML 语言的结构也是基于四层元模型结构：元-元模型、元模型、模型和用户对象。体系结构如图 2.1 所示：

层次	描述	举例
元—元模型	元模型结构的基础 定义元模型描述语言的模型	MetaClass MetaOperation
元模型	元-元模型的实例 定义元模型描述语言的模型	Block、Ports and Flows、Rationale、Problem
模型	元模型的实例	实际应用领域的模型中的模块、属性、操作等。
用户对象	模型的实例	系统工程中的对象结构及相互交互。

图 2.1 SysML 元模型体系结构

Figure 2.1 SysML Model Architecture

元-元模型层具有最高抽象层次，是定义元模型描述语言的模型，为定义元模型的元素和各种机制提供最基本的概念和机制。

元模型是元-元模型的实例，定义模型描述语言的模型。元模型提供了表达系统的各种包、模型元素的定义类型、标记值和约束等。

模型是元模型的实例，定义特定领域描述语言的模型。由SysML模型组成，这一级的每个概念都是元模型层的概念的实例，这一抽象层是用来形式化概念，并根据给定个体定义表达沟通的语言。

用户对象是模型的实例。任何复杂系统在用户看来都是相互通信的具体对象，目的是实现复杂系统的功能和性能。

2.2.2 语言组织结构(包结构)

SysML语言重用和扩展了UML的很多包(如图2.2所示)[11]，使用的扩展机制包括模型元素的定义类型(stereotype)、元类(metaclass)和模型库(model library)。SysML的用户模型是通过实例化模型元素的定义类型stereotype和元类metaclass以及构造模型库中类的子类来创建的。

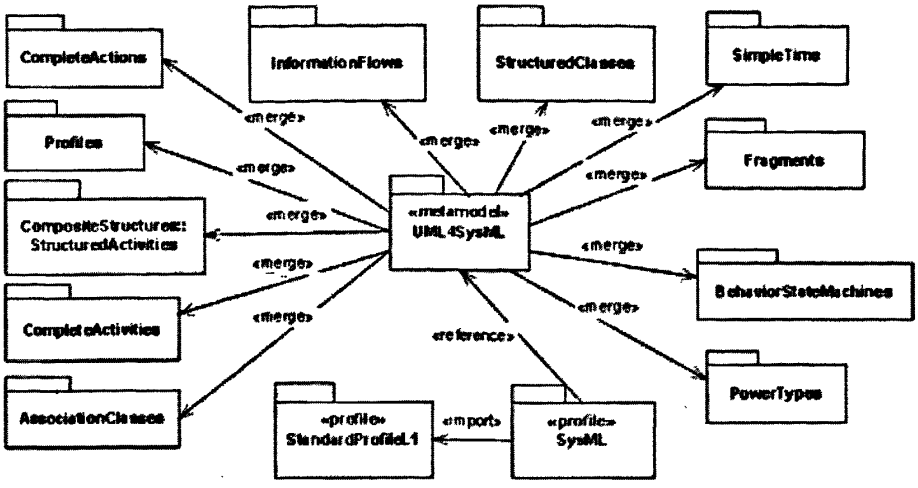

图 2.2 SysML 对 UML 的扩展[10]

Figure2.2 the SysML expansion of UML

SysML 的包结构（如图 2.3 所示）包括了符合扩展机制的 SysML 概念域的包的集合[11]。复用 UML 的部分没有被扩展，它们被归入 UML4SysML 包中，包括交互，状态机，用例和外廓。一些 UML 包没有被重用，因为它们在系统工程中不再需要；在 SysML 的活动包、类包和辅助包中增加了一些新的扩展；另外还增加了一些 UML 中没有的新包，如需求包、参数包等。

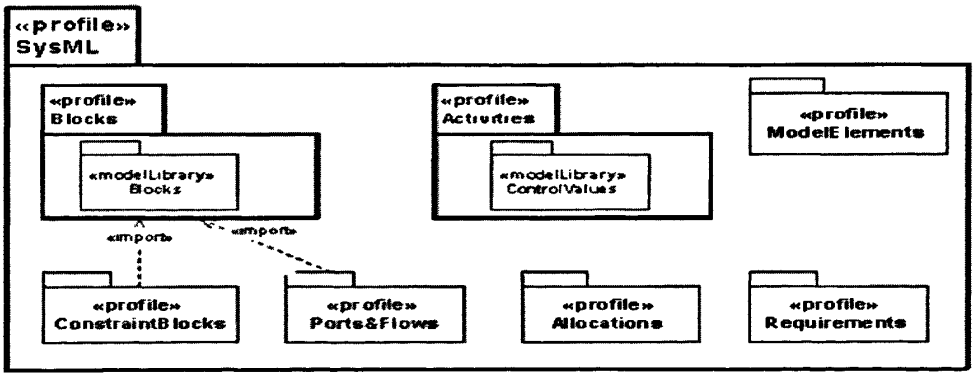

图 2.3 SysML 的包结构[11]

Figure2.3 SysML Package Structure

2.2.3 语言形式

形式化表示方法的好处有：(1)提高了描述的正确性；(2)减少了描述的二义性和不一致性；(3)增强了描述的可读性[12]。但语言的完全形式化是极为复杂的，因此，为了保持描述

的清晰易懂，SysML 和 UML 一样在给出自身的语义说明时采用了半形式化的描述方法。

SysML 建立在元模型基础之上的。元模型本身在 SysML 中表达，是 SysML 的一个子集，它定义了 SysML 中的各种图，而用户借助 SysML 提供的表示法定义自己系统的元模型，这是一个循环解释的过程。SysML 模型不太精确，特别是用可视元素表示模型元素时，其语义解释不够准确。因此，为了保持描述的清晰易懂，在给出自身语义说明时和描述模型时都采用了半形式化的的描述方法，这种半形式化的语义降低了 SysML 模型的精确性，给模型的自动分析和验证带来了困难。总的来说 SysML 的语义存在如下缺点：

1) 用自然语言描述的语义不够精确，不能满足对 SysML 模型进行严格分析的要求。
2) 元素的语法和语义导致冗余和不一致性，不具备良好的模块性。
3) 目前 SysML 并没有提供一个对 SysML 模型进行推理的合理机制。
4) 所以，到了系统设计的后期，要求所有的设计越来越精确，SysML 的不足也就显露出来了。因此，有必要为 SysML 的语义进行进一步形式化以增强 SysML 模型的精确性。

2.3 SysML 的表示法

SysML 的图形表示是 SysML 的可视化表示，是用来为系统建模的工具。SysML 定义了九种基本图形来表示模型的各个方面。在九种图形中需求图和参数图是 UML2.0 中没有的新图；活动图和模块图（包括模块定义图和内部模块图）来自于 UML2.0,并在 SysML 中进行了扩展；用例图、顺序图、状态机图和包图都来自于 UML2.0 的复用，没有进行修改[15]。SysML 与 UML2.0 的关系图如图 2.4 所示。

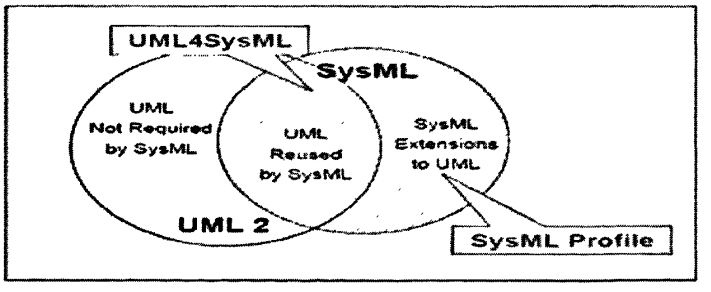

图 2.4 SysML 和 UML2.0 的关系[10]

Figure2. 4 Relationship of SysML and UML2.0

SysML图结构[12]（如图2.5所示）中定义了九种基本图形，同时将其分成四类：结构图(Structure Diagram)、参数图(Parametric Diagram)、需求图(Requirement Diagram)和行为图(Behavior Diagram)。

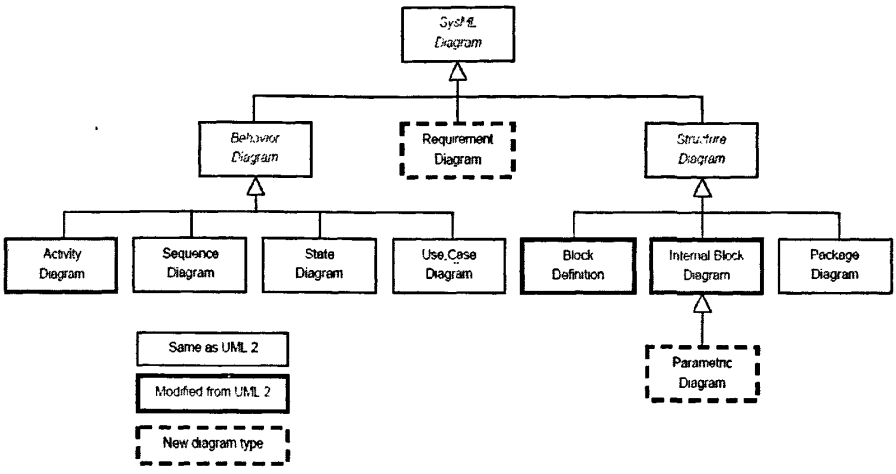

图 2.5 SysML 图结构[10]

Figure2.5 SysML Picture Structure

- 结构图包括模块定义图(Block Definition) 和内部模块定义图(Internal Block Definition)以及包图（Package Diagram）。模块《block》是 SysML 的基本结构单元，用来描述硬件，软件，设备，人员以及其它系统元素。
  1) 模块定义图定义了模块的属性以及模块之间的相互关系。这类关系包括关联、泛化和依赖。模块定义图是基于 UML 的类和 UML 的复合结构扩展而来的。它既可以描述系统的结构特征也可以描述系统的行为特征。模块定义图有唯一的名字，封装了若干属性和操作。操作的每个参数都有名称和类型。
  2) 内部模块定义图用属性和连接子定义了一个模块的内部结构，它描述了部件间的交互。内部模块定义图是在 UML 的复合结构图的基础上进行了约束和扩展而得到的。
  3) 包图描述了将模型元素组成组的机制，包图用于组织模型。

- 需求图是一种新的SysML图形，能够描述需求和需求之间以及需求和其他建模元素之间的关系。需求是指系统必须满足的能力或条件，一个需求能够分解成多个子需求。SysML用《requirement》说明需求，它也是一个类，有两个属性：text和id，前者是需求的文本描述，后者是需求的标识符。用户可以定义需求的子类，如操作需求、功能需求、接口需求、性能需求等等。使用导出关系《derive》表示一个需求可以从另一个需求产生或导出，使用满足关系《satisfy》表示一个需求能被其他的

模型元素实现，使用验证关系《verify》表示一个需求能被测试例子验证。SysML的《derive》、《satisfy》和《verify》都是继承UML的《trace》。SysML用《rationale》表示基本原理注释元素，能够附在任何模型元素上，用来说明建模决策如分析决策或设计决策的原理或原因。

- 参数图是一种新的SysML图形，定义了一组系统属性以及属性之间的参数关系。参数关系说明了一个属性值的变化怎样影响其他的属性值，参数关系是没有方向的。参数模型是分析模型，把行为模型和结构模型与工程分析模型如性能模型和可靠性模型等结合在一起，能用来支持权衡分析，评价各种备选的解决方案。参数约束是一个装配，用《paramConstraint》表示，通常和SysML装配图结合起来使用。《paramConstraint》声明一个装配用作参数约束，它的端口定义了参数约束的参数，《paramConstraint》装配的唯一有效用法是用值绑定约束绑定参数和装配中的其他属性。参数约束内部可以包含其他参数约束，但不能包含其他部件或属性。参数约束关系用来表示系统的结构模型中属性之间的依赖关系，可以是基本的数学操作符，也可以是和物理系统的性质有关的数学表达式如F = m * a和a = dv/dt等《valueBindingConstraint》从UML的Connector扩展而来，它声明两个相连的属性有相同的值，用虚线表示。

- 行为图包括用例图、活动图、顺序图以及状态机图。用例图提供了一种系统之间以及系统内部之间的较高层次的功能性描述。活动图描述了活动之间的数据流和控制流。顺序图描述了系统中相互协作的部件之间的交互。状态机图描述了系统或者其部件在完成事件响应时的状态转换和行为。

  1） 用例图描述了外部参与者对系统的使用，这是通过系统向参与者提供一系列服务来实现的。用例图包括用例、参与者以及它们之间的通讯，参与者可能是用户、外部系统或其他环境实体，它们和系统直接或间接交互。用例图由行为图中的状态机图、顺序图和活动图来实现。用例和用户之间的关系包括包含关系（include）、扩展关系（extend）和泛化关系（generalization）。包含关系提供一种分离出多个用例中共享的共同功能作为基本用例执行的机制；扩展关系提供了可选的功能，在定义的扩展点和指定的条件下扩展基本用例；泛化关系提供了基本用例的特殊化机制。SysML 用例图重用了 UML 用例图。

2） 活动图强调活动的输入输出、顺序和条件。SysML 活动图和系统工程领域常用的增强功能模块图 EFFBD（Enhanced Functional Flow Block Diagrams）类似，只是采用不同的术语和符号。SysML 活动图对 UML 活动图进行了扩展，包括把控制作为数据、表示连续的物质或能量流、引入概念等等，在 UML 活动图中，控制只能使动作开始。在 SysML 活动图中，控制既能使动作开始，还有能使正在执行的动作终止功能。

3） 顺序图根据控制流指定了一连串的交互，控制流是通过对象生命线之间发送和接收消息来定义的。消息包含了控和数据流，它发起接收消息的对象的行为，并把输入传给行为，消息的时间顺序和它在顺序图上的垂直位置相关。复杂的顺序可以抽象成引用顺序图。顺序图上可以包含条件逻辑，表示选择、连续流和循环等。对象生命线可以进一步分解为它的组成部分。SysML 顺序图重用了 UML 顺序图。

4） 状态机图通过状态以及状态之间的转移对离散行为建模，它把行为表示为对象的状态历史。在状态的转移、进入和退出过程中会调用活动，并指定相关的事件和守卫条件。在状态中调用的活动称为“do”活动，可以是连续的，也可以是离散的。一个组合状态有嵌套的状态，嵌套的状态以是顺序的，也可以是并发的。SysML 状态机图重用了 UML 状态机图。

第三章 SysModeler 的开发框架

SysModeler 与 UML 类图编辑器、ROSE 等图形化建模工具一样需要提供一个编辑器和一个工具条，用户能够在工具条里选择需要的工具，以拖动或单击的方式将节点放置在编辑区域中，同时能够查看和修改某个节点或连接的属性等等。目前流行软件 Eclipse 的 GEF（Graphical Editor Framework）插件提供了标准的 MVC（Model-View-Control）结构能够使开发者不需重新设计就能完美的实现上述功能，因此在实现 SysModeler 时选用了 Eclipse 平台，使用了 GEF 插件。

3.1 Eclipse 简介

Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台[20]。就其本身而言，它只是一个框架和一组服务，用于通过插件组件构建开发环境。Eclipse 附带了一个标准的插件集，包括 Java 开发工具（Java Development Tools，JDT），插件开发环境（Plug-in Development Environment，PDE），这个组件主要针对希望扩展 Eclipse 的软件开发人员，因为它允许他们构建与 Eclipse 环境无缝集成的工具。

Eclipse 就像软件开发者的『打铁铺』，它一开始备有火炉、铁钻与铁锤。就像铁匠会用现有的工具打造新的工具，也能用 Eclipse 打造新工具来开发软件-这些新工具可扩充 Eclipse 的功能。(Eclipse 其中一个卖点就是它的扩充性)。

3.1.1 历史背景

Eclipse 这样功能完整且成熟的开发环境，是由蓝色巨人 IBM 所释出[21]。IBM 花了 4 千万美金来开发这个 IDE(Integrated Development Environment)。第一版 1.0 在 2001 年 11 月释出，随后逐渐受到欢迎。

Eclipse 已经成为开放原始码计划(Open Source Project)，大部分的开发扔然掌握在 IBM 手中，但是有一部份由 eclipse.org 的软件联盟主导。Eclipse 项目由 Project Management Committee(PMC)所管理，它综观项目全局，Eclipse 项目分成 3 个子项目：

- 平台-Platform
- 开发工具箱-Java Development Toolkit(JDT)
- 外挂开发环境-Plug-in Development Environment(PDE)

这些子项目又细分成更多子项目。例如 Platform 子项目包含数各组件，如 Compare、Help 与 Search。JDT 子项目包括三各组件：User Interface(UI)、核心(Core)及除错(Debug)。PDE 子项目包含两各组件：UI 与 Core。

3.1.2 开放原始码软件

Eclipse 是一个开放源代码的项目，任何人都可以下载 Eclipse 的源代码，并且在此基础上开发自己的功能插件。也就是说未来只要有人需要，就会有建立在 Eclipse 之上的 COBOL，Perl，Python 等语言的开发插件出现。同时可以通过开发新的插件扩展现有插件的功能，比如在现有的 Java 开发环境中加入 Tomcat 服务器插件。可以无限扩展，而且有着统一的外观，操作和系统资源管理，这也正是 Eclipse 的潜力所在。

3.1.3 跨语言、跨平台

多数人认为 Eclipse 是 Java IDE，不过，其实 Eclipse 除了有 Java IDE(就是 JDT)，还有 PDE。并且 Eclipse 是万用工具平台。JDT 实际上是 Eclipse 的添加品，也就是外挂程序。Eclipse 本身实际上是指 Eclipse 平台(Eclipse Platform)，除了能取得 Java 工具集以外，还提供各种工具的支持，所以平台本身只是相当小的一组软件。

如果想开发 Java 程序，用的是 Eclipse 随附的 JDT 外挂程序。如果想开发其它语言的程序，就需要拿到其它外挂程序，诸如 CDT(C Development Toolkit)就可以开发 C/C++程序。

Eclipse 跨计算机语言，也跨人类的语言。相同的外挂机制可用来增加对不同语言的支持，这里使用一种特殊的外挂，叫做外挂程序片断(plug-in fragment)。IBM 以捐出一个语言套件，支持中文(繁体与简体)、法文、德文、意大利文、日文、韩文、葡萄牙文(巴西)与西班牙文。

照理说 Eclipse 用 Java 写成，应该可以在任何的平台执行。但严格来说 Eclipse 不是跨平台的，因为它使用作业平台的原生图形来建置。因此要等 SWT(Standard Widget Toolkit)移植到该平台，Eclipse 才能在那个平台执行。但就现实而言到不是什么大问题，因为 SWT 已经被移植到数个常见平台上了，包括 Windows、Linux/Motif、Linux/GTK2、Solaris、QNX、AIX、HP-UX 与 Mac OS X。

3.2 GEF 简述

GEF（Graphical Editor Framework）是一个图形化编辑框架[22]，它允许开发人员以图形化的方式展示和编辑模型，从而提升用户体验。

GEF 最早是 Eclipse 的一个内部项目，后来逐渐转变为 Eclipse 的一个开源工具项目，Eclipse 的不少其他子项目都需要它的支持。Eclipse 3.0 版本花了很大功夫在从 Platform 中剥离出各种功能部件，包括 GEF 和 IDE 在内的很多曾经只能在 Eclipse 内部使用的工具成为可以独立使用的软件/插件包了。

GEF 的优势是提供了标准的 MVC[23]（Model-View-Control）结构，开发人员可以利用 GEF 来完成所有图形化建模工具，这些建模工具提供一个编辑器和一个工具条，用户能够在工具条里选择需要的工具，以拖动或单击的方式将节点放置在编辑区域中，同时能够查看和修改某个节点或连接的属性等等功能。与其他一些 MVC 编辑框架相比，GEF 的一个主要设计目标是尽量减少模型和视图之间的依赖，好处是可以根据需要选择任意模型和视图的组合，而不必受开发框架的局限（实际上还是很少有脱离 Draw2D 的实现）。GEF 的框架如图 3.1 所示，下面简要介绍一下 GEF 的 MVC 模型。

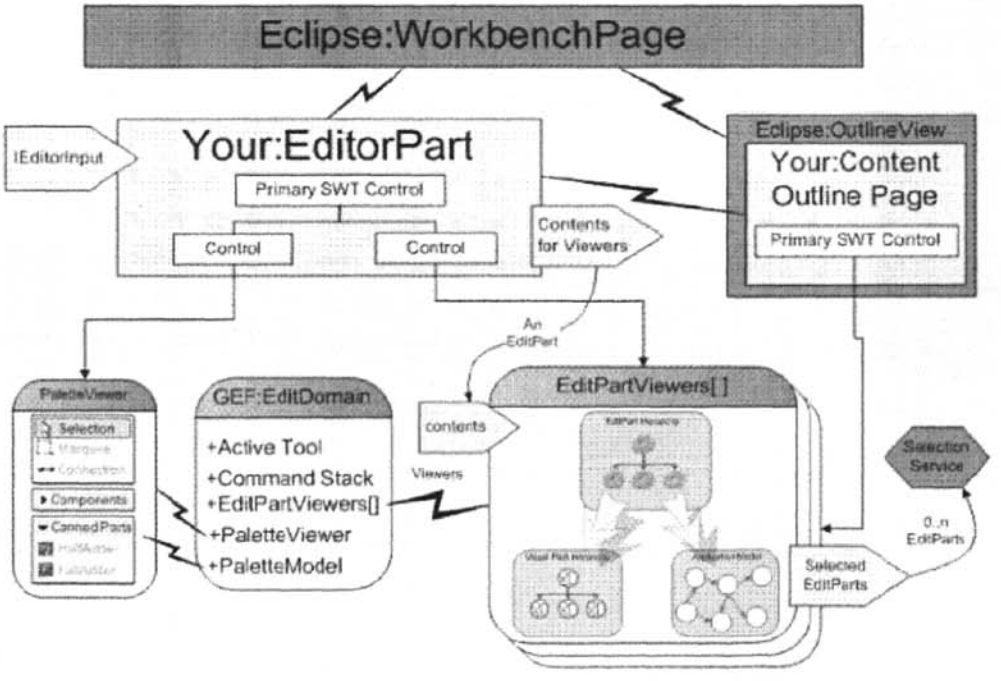

图 3.1 GEF 结构图[21]

Figure3.1 GEF Structure

3.2.1 模型

GEF 的模型（如图 3.2 所示）只与控制器打交道，不知道任何与视图有关的东西。为了能让控制器知道模型的变化，应该把控制器作为事件监听者注册在模型中，当模型发生

变化时，就触发相应的事件给控制器，控制器负责通知各个视图进行更新。

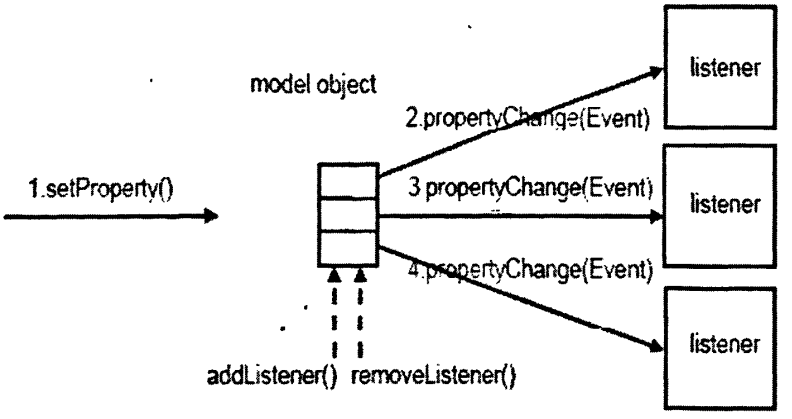

图 3.2 GEF 的模型图[21]

Figure3.2 GEF Model

典型的模型对象会包含 PropertyChangeSupport 类型的成员变量，用来维护监听器成员即控制器；同时对于与其他对象具有连接关系的模型，要维护连入/连出的连接列表；如果模型对应的节点具有大小和位置信息，还需要这样的成员变量来维护。相对来讲 GEF 中模型是 MVC 中最简单的一部分。

3.2.2 控制器

在 MVC 结构里控制器是模型与视图之间的桥梁，也是整个 GEF 的核心。它不仅要监听模型的变化，同时当用户编辑视图时，还要把编辑结果反映到模型上。举个例子来说，用户在数据库结构图上删除一个表时，控制器应该从模型中删除这个表对象、表中的字段对象、以及与这些对象有关的所有连接。

GEF 中的控制器是所谓的 EditPart 对象(如图 3.3 所示)，更确切的说应该是一组 EditPart 对象共同组成了 GEF 的控制器，每一个模型对象都对应一个 EditPart 对象。在应用程序中需要有一个 EditPartFactory 对象负责根据给定模型对象创建对应的 EditPart 对象，这个工厂类将被视图利用。 RootEditPart 是一种特殊的 EditPart，它和模型没有任何关系，它的作用是把 EditPartViewer 和 contents(应用程序的最上层 EditPart，一般代表一块画布)联系起来，可以把它想成是 contents 的容器。EditPartViewer 中的方法 setRootEditPart()专门用来指定视图对应的 RooEditPart。

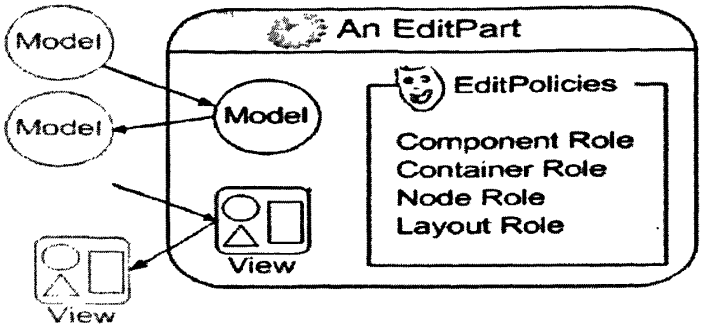

图 3.3 EditPart 对象[21]

Figure3.3 EditPart Object

用户的编辑操作被转换为一系列请求（Request），请求的类型在 GEF 里被称为角色（Role），角色这个概念是通过编辑策略（EditPolicy）来实现的，EditPolicy 的主要功能是根据请求创建相应的命令（Command），Command 会直接操作模型对象。对每一个 EditPart，可以“安装”一些 EditPolicy，用户对这个 EditPart 的特定操作会被交给已安装的对应 EditPolicy 处理。这样做的直接好处是可以在不同 EditPart 之间共享一些重复操作。

3.2.3 视图

GEF 目前提供了图形（GraphicalViewer）和树状（TreeViewer）两种视图，前者多用于编辑区域，后者则多用于实现大纲的展示[24]。在图形中 Draw2D 图形（IFigure）作为表现方式。Draw2D 提供了许多缺省图形，最常见的有三类：1、形状（Shape），如矩形、三角形、椭圆形等等；2、控件（Widget），如标签、按钮、滚动条等等；3、层（Layer），它们用来为放置于其中的图形提供缩放、滚动等功能。Draw2D 图形并不是上述的单一的图形，而是有一组基本图形构成的图形树（如图 3.4 所示）。

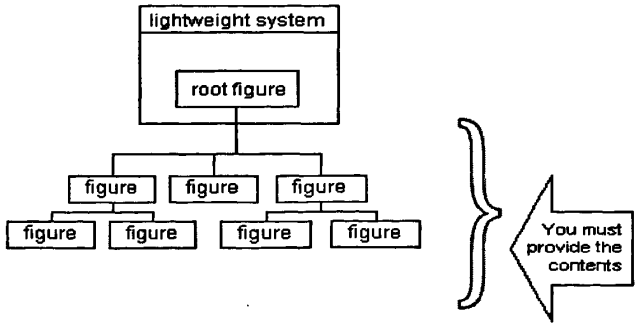

图 3.4 Draw2D 图形[22]

Figure3.4 Draw2D Figures

视图的任务同样繁重，除了模型的显示功能以外，还要提供编辑功能、回显（Feedback）、工具提示（ToolTip）等等功能。GEF 使用 EditPartViewer 作为视图（如图 3.5 所示），它的作用和 JFace 中的 Viewer 十分类似，EditPart 就相当于是它的 ContentProvider 和 LabelProvider，通过 setContents()方法来指定。在 GEF 中经常使用的 Editor 是一个 GraphicalEditorWithPalette，这个 Editor 是 EditorPart 的子类，具有图形化编辑区域和一个工具条，通常使用 GraphicalEditViewer 和 PaletteViewer 这两个视图类，PaletteViewer 也是 GraphicalEditViewer 的子类用来显示工具条。在 configureGraphicalViewer() 和 initializeGraphicalViewer()这两个方法里对 EditPartViewer 进行定制，包括指定它的 contents 和 EditPartFactory 等等。

EditPartViewer 同时也是 ISelectionProvider，这样当用户在编辑区域做选择操作时，注册的 SelectionChangeListener 就可以收到选择事件。EditPartViewer 会维护各个 EditPart 的选中状态，如果没有被选中的 EditPart，则缺省选中的是作为 contents 的 EditPart。

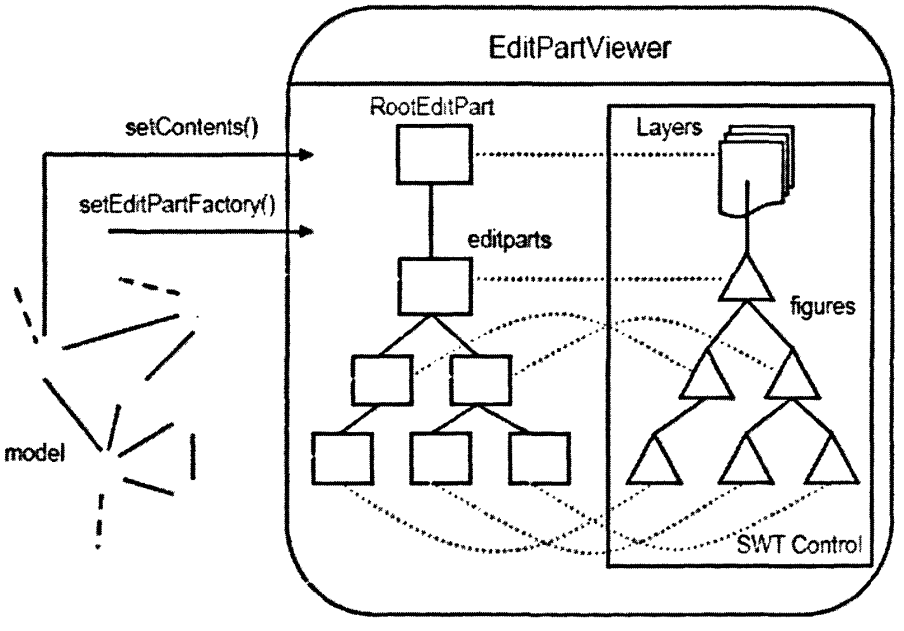

图 3.5 GEF 的视图[22]

Figure3.1 GEF View

3.2.4 GEF 的工作机制

当用户执行操作后，EditPartViewer 接受操作，同时发送事件到 PaltetteViewer 的 ActiveTool，ActiveTool 将用户的操作转换为 Request 分配给 EditPart 中适当的 EidtPolicy，

由 EditPolicy 创建适当的 Command 来修改模型，同时 Command 会保留在 EditDomain（专门用于维护 EditPartViewer、Command 等信息的对象，一般每一个 Editor 对应唯一一个该对象）的命令栈里，用于实现撤销、重做功能。当模型修改之后，其对应的 EditPart 侦听到模型的改变在视图上做出反馈。其工作的过程如图 3.6 所示。

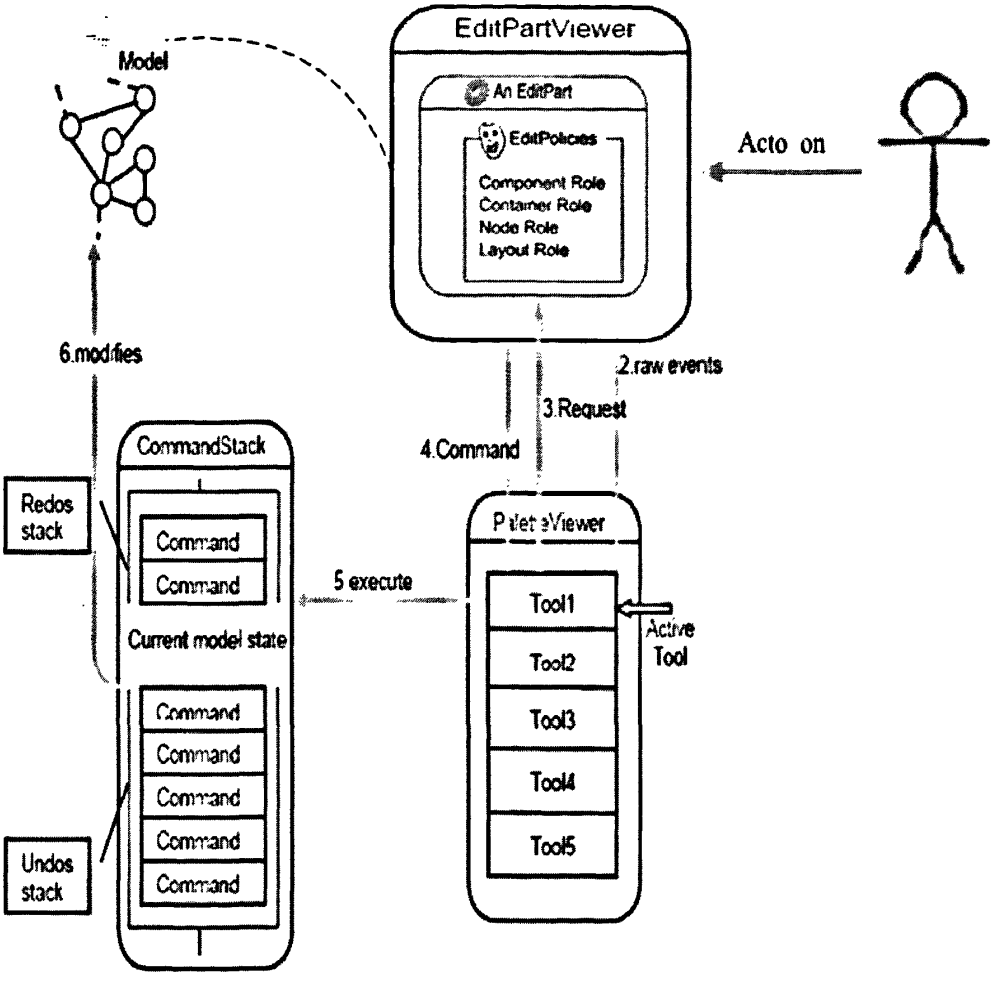

图 3.6 GEF 工作机制[22]

Figure3.6 GEF Working Mechanism

第四章 SysModeler的设计

4.1 软件的实现目标

在应用 SysML 对系统工程问题建模时，需要相应的 SysML 建模环境。目前单独支持 SysML 的建模环境尚不存在，同时 UML 的建模环境又无法满足 SysML 建模的实际需求。因此本文提出了基于 SysML1.0 规范的建模工具 SysModeler，来满足系统工程师建模的实际需求，推动系统工程理论和实践的发展。

SysModeler 按照系统工程项目为中心的文档组织方式，囊括了系统工程建模时所需的若干个完整的工具，有助于整合系统工程过程[26]。在系统工程的需求分析阶段可以使用需求模型建模工具，详细说明和设计阶段可以使用结构模型、行为模型、以及参数模型建模工具对实际领域建模。其中需求模型建模绘制需求图，用来捕获系统需求，强调需求之间的追溯关系；结构模型建模绘制模块定义图、内部模块图和包图，用来描述系统的层次和对象之间的相互连接关系；行为模型建模建立活动图、顺序图、用例图和状态机图，用来描述系统中对象的行为；参数模型建模建立参数图，用来描述系统或部件之间的参数约束关系。最后在系统工程的编码阶段将使用行为模型、结构模型以及参数模型建模工具生成的模型自动转化为整个应用系统的部分源代码，建模人员只需加入具体服务功能的实现代码即可。

SysModeler 让系统工程的过程成为一个整体，提供丰富的系统工程功能。系统工程师使用 SysModeler 能确保给出正确的系统规范，同时可以在开发过程中与所有的项目成员有效地进行沟通。同时系统工程师使用 SysModeler 可以在开发过程的早期消除模型中的错误，与在最终测试的时候或者系统已经开始运营的时候发现同样的错误相比较，纠正这些错误的花费会大大降低。SysModeler 可以使系统工程师在直观环境中，用建模语言管理复杂系统，从而提升系统设计开发质量。

4.2 软件设计原则

在系统的整个设计过程中，要遵循以下原则：

1) 实用性

系统提供友好的用户界面，实现“傻瓜式”的操作；系统的运行维护简单方便，用户只要用 UML 的基础知识即能掌握系统维护、管理和使用；系统能够对正确操作高效快速响应、

误操作警告提示、非法操作则被禁止。

2）稳定性

系统要有较强的容错能力，良好的存储和恢复机制。

3) 可扩展性。

系统要有良好的功能扩展性，当有新的需求时，在不影响系统原有功能的前提下，实现系统功能的扩展。

4) 面向用户

用户是系统最终的鉴定人，也是系统的使用人、管理人。系统首先从用户的特征入手，保证在开发过程中用户不断参与，从而使系统实用、高效、灵活。同时，系统界面友好，方便灵活、易于操作。

5) 响应快速性

根据一般的心理学统计，执行某一个操作的响应时间如果超过 2 秒，用户就会感到等待。我的系统设计目标是使用户的大多数操作的响应时间能在 2 秒钟内完成。

4.3 软件的体系框架

SysModeler 的体系结构分为三层（如图 2 所示）。最底层的模型层，是根据业务来设计的，它提供模型改变的通知机制，用来把模型的改变告诉中间层。中间层为控制器层，是本系统的核心层，同时也是模型层和视图层之间的桥梁。最上层是视图层，其除了显示模型之外，还要提供编辑、回显（Feedback）、工具提示(ToolTip)等功能。模型以图形化的形式显示在视图层的编辑器上。本体系结构的工作机制如图 4.1 所示。首先用户操作视图层中的控件，控制器层接收用户操作的请求（步骤①）；紧接着控制器层把用户操作反映到模型层（步骤②），模型改变之后，控制器层侦听到模型的改变（步骤③），最后控制器层将模型的改变以图形化的形式显示在视图层上（步骤④）。

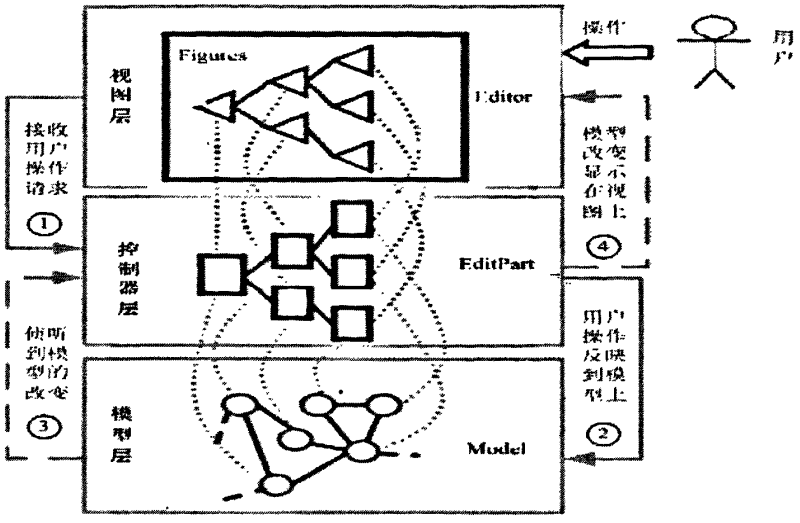

图 4.1 SysModeler 体系结构

Figure4.1 SysModeler Architecture

4.4 软件的功能设计

SysModeler 和普通的图形编辑工具一样具有以下功能：

1) 绘制图例: 提供轻易绘制图形的功能,同时能检查图形的语义和规则。绘制的图形有行为模型图（包括用例图、状态机图、顺序图和活动图）、结构模型图（包括模块定义图、内部模块图和包图），同时还可以绘制参数图和需求图。
2) 实现项目管理视图支持导航，在项目管理视图上能够能过右键菜单和键盘命令添加和删除项目和模型图，双击模型图名称打开模型图的内容，同时在编辑器上显示。
3) 实现绘制模型图的编辑区，编辑区提供工具条，可以在工具条里选择需要的工具，以拖动或单击的方式将节点或连接放置在编辑区域内，在编辑区域内可以选中已经创建的节点通过属性视图查看和修改其的部分可以修改的属性；可以通过上下文菜单和键盘命令执行删除和撤销操作和图形的缩放等功能。
4) 实现能够显示编辑区内内容的缩略图的大纲视图等。

4.5 软件界面框架设计

在设计阶段，除了软件功能和系统结构设计等内容外，一个很重要的部分就是系统界面的设计，系统界面是人机交互的接口，包括用户如何命令系统以及系统如何向用户返回信息。一个设计良好的用户界面使得用户更容易掌握系统，从而增加用户对系统的接受程

度。

一个成功的用户界面必然是以用户为中心的，集成的和互动的。本软件 SysModeler 是根据 Eclipse 的 RCP 插件的工作台来设计的界面。包括项目管理视图、编辑器、大纲视图、属性视图四个窗口，和菜单栏和工具栏。

如图 4.2 为软件的用户界面的示意图：

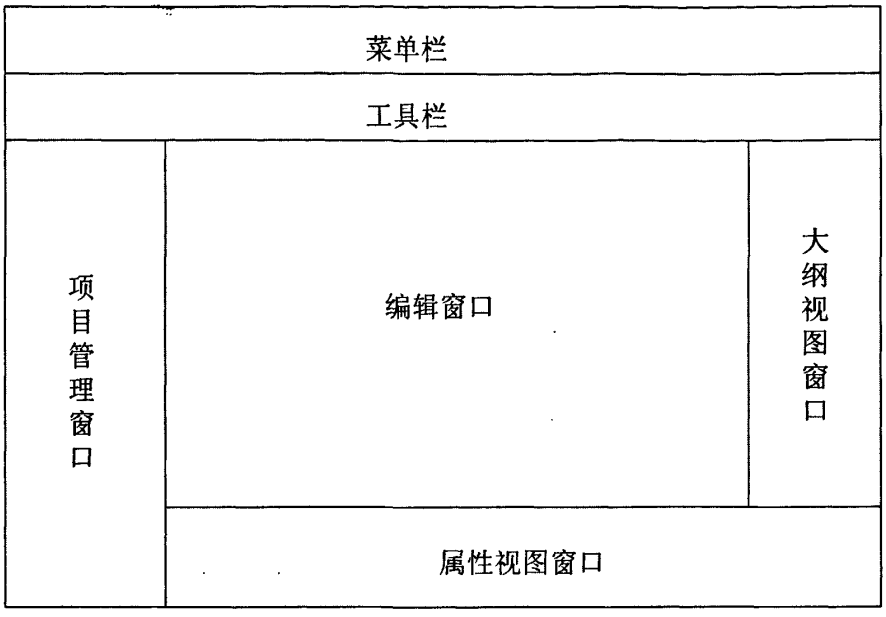

图 4.2 SysModeler 的用户界面

Figure4.2 SysModeler User Interface

- 项目管理视图：允许您创建、选择和删除项目，同时在每个项目中创建、选择和删除模型图。当选中模型图时双击模型图名称可在编辑区内打开此文件，文件以图形的形式显示在编辑区内。
- 编辑窗口： SysModeler 的编辑器是一个带有工具条 Palette Viewer 的编辑器。可以在工具条里选择需要的工具，以拖动或单击的方式将节点或连接放置在编辑区域内，在编辑区域内可以选中已经创建的节点通过属性视图查看和修改其的部分可以修改的属性，也可以通过上下文菜单和键盘命令执行删除和撤销操作和图形的缩放等功能。
- 大纲视图：在 SysModeler 的大纲视图显示编辑区域的缩略图，同时也显示树状模型结构。这个大纲的准确性不光显示模型图的名称子节点，同时还显示其具有的一些属性（如大小和位置等）。和编辑器一样大纲视图也有右键菜单，通过右键菜单也可以和键盘命令执行删除和撤销操作。

- 属性视图：使用属性视图修改的是图形模型的属性。图形模型是视图的数据源，当图形模型改变时（如图形尺寸或文本改变时），属性视图要反映相应改变；而在属性视图中修改属性时，编辑区中的图形也要发生相应改变。
- 菜单栏：在菜单栏中包括文件、编辑、帮助、窗口等菜单项，通过这些菜单项可以新建、删除项目、以及项目中的模型图。
- 工具栏：主要是操作模型图中的对象，实现模型图中对象的的放大、缩小、左对齐、右对齐等功能。

界面的设计是系统设计中相当重要的一部分，无论后台采用的是什么架构技术，用户能够体验到的就是系统的界面和交互性，所以本系统的设计尽量做到简洁、易用、美观，同时提高用户的交互体验。在系统界面的设计中，从用户的角度出发，采用常规的操作流程，使用户能够很快熟悉系统的使用。

4.6 软件的包结构设计

根据上述的体系结构以及界面设计得到的整个系统的包结构图（如图 4.3 所示）。SysModeler 软件系统主要包括九个包，其中 SysModel 包主要存放系统的模型，包括模块定义图（BlockDefModel）、模块（BlockModel）、参与者（ActorModel）等模型来完成体系结构中模型层所要完成的功能；EditPart 包存放系统的控制器类相关的包，包括模型定义图控制器（BlockDefEditPat），模型定义图工厂类（BlockDefEditFactory）、模块控制器（BlockEditPart）等类；EditPolicy 包主要包含一些 EditPart 类中应该安装的策略对象类；Command 包中包含直接对模型进行修改的类。EditPart 包、EditPolicy 包和 Command 包三者结合起来共同完成软件的体系结构中控制器所要完成的功能。Editor 包中放置各个模型图所需的编辑器类，在 UI 包中包含决定整个用户界面的布局的 Perspective 类和项目管理视图 NavigatorView 等类；Action 包中包括菜单栏和工具栏中要实现的功能菜单类；Wizard 包中存储新建项目时弹出的向导页相关的类；DirectEdit 包中放置的对编辑区内节点的直接修改删除等操作的相关类。

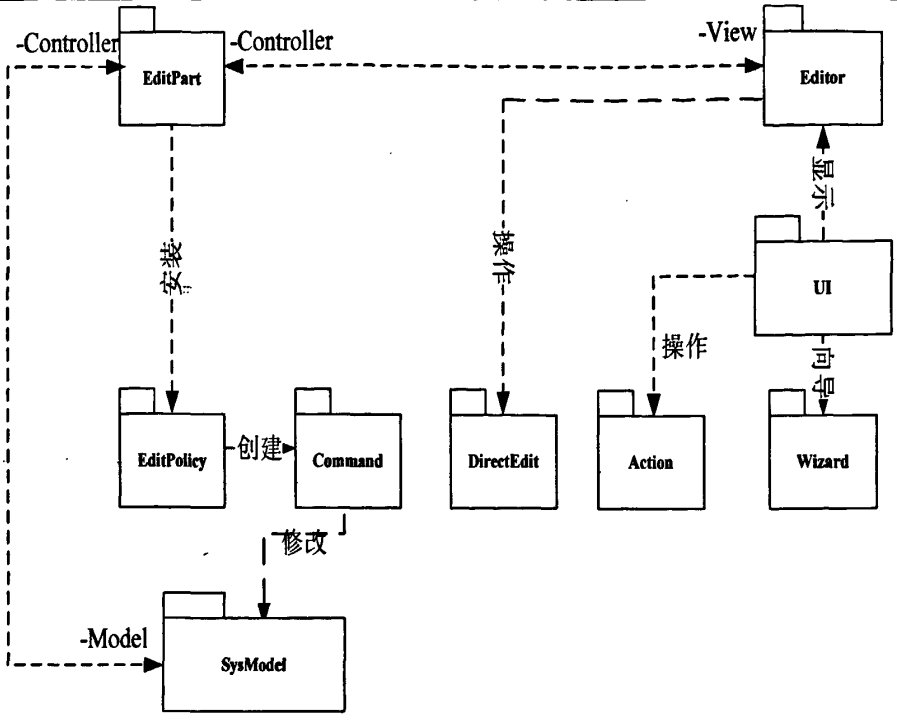

图 4.3 SysModeler 包结构图

Figure4.3 SysModeler Structure of Package

第 五 章 SysModeler 的实现和应用

5.1 SysModeler 工作平台的实现

根据第四章中介绍的软件用户界面设计，实现 SysModeler 的工作平台需要设计整个工作平台的结构以及实现项目管理视图、编辑器、大纲视图、菜单栏和工具栏等。SysModeler 在实现工作平台时使用了 Eclipse 的 RCP 插件。Eclipse RCP 界面专业、漂亮、性能良好，同时这一构架容易扩展，能够吸引更多的人在上面为其开发。Eclipse RCP 在实现用户界面方面替开发者考虑得很周全，用 Perspective 实现界面的布局，同时继承不同的 Viewer 和 Editor 即可实现项目管理、大纲等视图。SysModeler 的工作平台的实现结果如图 5.1 所示：

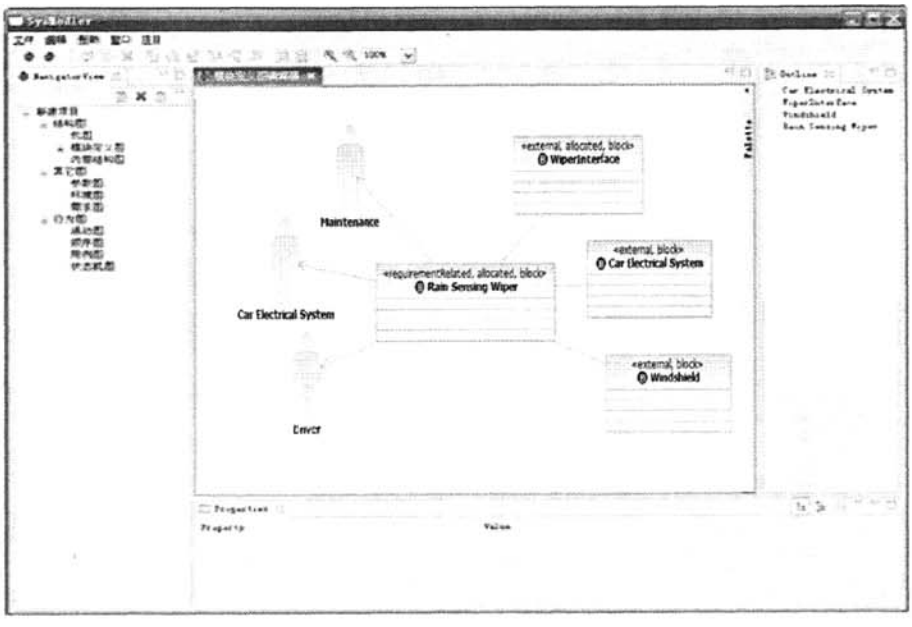

图 5.1 SysModeler 的工作平台

Figure5.1 SysModeler Working Platform

5.1.1 整体布局的实现

SysModeler 用 Perspective 类来实现整个工作平台的布局。perspective 的定义只是一个“屏幕页面的模板”，它记录一组缺省的 view 和他们的布局情况，作为方便用户的一种选择。Perspective 类实现的代码如下所示：

```
public class Perspective implements IPerspectiveFactory {
 public void createInitialLayout(IPageLayout layout) {
```

```
        final String properties = "org.eclipse.ui.views.PropertySheet";
        final String outline = "org.eclipse.ui.views.ContentOutline";
        String editorArea = layout.getEditorArea();
        IFolderLayout left = layout.createFolder("left", IPageLayout.LEFT,
                0.2f, editorArea);
        // 左面的大纲视图
        left.addView("test.gef.view.NavigatorView");
        // 添加菜单栏
        layout.addActionSet("SysModeler actionSet");
        IFolderLayout bottom = layout.createFolder("bottom",
                IPageLayout.BOTTOM, 0.8f, editorArea);
        // 属性视图
        bottom.addView(properties);
        IFolderLayout right = layout.createFolder("right", IPageLayout.RIGHT,
                0.8f, editorArea);
      // 大纲视图
        right.addView(outline);
        layout.setEditorAreaVisible(true);
    }
}
```

5.1.2 项目管理窗口的实现

项目管理窗口主要实现的功能在第四章中已经做过介绍，现主要介绍它的实现过程。SysModeler 的项目管理视图由 NavigatorView 类来完成，在 NavigatorView 类中由 createPartControl()方法来创建树形结构来显示其的内容。其中的另一比较重要的 hookDoubleClickAction()方法是实现双击树形节点中的一个节点在编辑区内显示其节点应该显示的内容。相面是 NavigatorView 的实现代码，由于篇幅原因只显示其中比较重要的方法。

```
public class NavigatorView extends ViewPart {
    // 定义变量
    .....
```

```
    public void createPartControl(Composite parent) {
        viewer = new TreeViewer(parent, SWT.MULTI | SWT.H_SCROLL | SWT.V_SCROLL);
        drillDownAdapter = new DrillDownAdapter(viewer);
        // 树形结构的内容提供者
        viewer.setContentProvider(new TreeViewerContentProvider());
        // 树形结构的标签提供者
        viewer.setLabelProvider(new TreeViewerLableProvider());
        viewer.setSorter(new NameSorter());
        makeActions();
        hookContextMenu();
        hookDoubleClickAction();
        // 菜单提供者
        contributeToActionBars();
    }
    private void hookDoubleClickAction() {
        viewer.addDoubleClickListener(new IDoubleClickListener() {
            public void doubleClick(DoubleClickEvent event) {
                /*
                 * 根据列表的不同项得到相应的editorInput和editorID，
                 * 其中editorID是该编辑器在plugin.xml中设置的id
                 * 该id存储在EditorInput中
                 */
                ISelection selection = (ISelection)event.getSelection();
                Object obj = ((IStructuredSelection) selection).getFirstElement();
                if((obj instanceof DiagramEntry)||(obj instanceof ViewEntry)||(obj instanceof
NavigatorEntry))
                    return;
                DiagramExampleEntry entry = (DiagramExampleEntry) obj;
                childDiagramName = entry.getName();
                BaseEditPart.graphicName = childDiagramName;
```

```
                IEditorInput editorInput = entry.getEditorInput();
                String editorID = entry.getEditorID();
                //如果editor和eidtorInput为空中断返回
                if(editorInput==null || editorID==null)
                    return;
                // 取得IWorkbenchPage,并搜索使用editorInput对象对应的编辑器
                IWorkbenchPage workbenchPage =
getSite().getWorkbenchWindow().getActivePage();
                IEditorPart editor = workbenchPage.findEditor(editorInput);
                /*
                * 如果该编辑器已经存在，则将它设为当前的编辑器，否则重新打开*一个 */
                if(editor!=null)
                {
                    workbenchPage.bringToTop(editor);
                }
                else
                {
                    try{
                        workbenchPage.openEditor(editorInput, editorID);
                    }
                    catch(PartInitException exception){
                        exception.printStackTrace();
                    }

                }
            }
        });
}
.....
}
```

5.1.3 其他窗口的实现

SysModeler的其他窗口是继承GEF插件中的类，同时在介绍本软件的具体内容的实现的过程中要涉及到，因此其他窗口以及菜单栏等的实现将在5.2节做具体的介绍。

5.2 SysModeler 的三层结构的实现

根据第四章中介绍的体系结构，把系统分为三大实现模块——模型层、控制器层以及视图层模块。限于篇幅，下面着重介绍一下 SysML 中模块定义图在这三层上的实现过程，从而了解整个 SysModeler 的实现过程。

模块定义图定义了模块的属性以及模块之间的相互关系。模块定义图（如图 5.2 所示）中除了包括 Block 外，还包括 Actor、VuleType、DataType 等模型元素，由于篇幅原因再次只默认模块定义图中只包括模块元素。模块有一个唯一的名称，封装了若干属性（properties）和操作（operations），操作的每个参数有名称和类型。

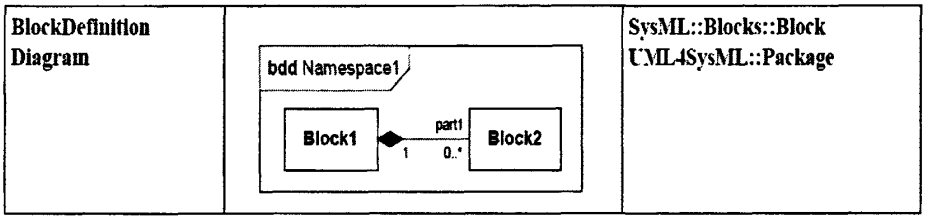

图 5.2 模块定义图的抽象语法和具体语法[10]

Figure 5.2 The concrete syntax example and abstract syntax reference of block define Figure

5.2.1 模型层的实现

在一个模型类中应该包含属性和操作两部分。

1) 属性。模型中属性定义不但要根据模型自身的一些属性，而且要根据其所处的环境。因此一个模型中的属性不光包括自身的属性还应该包括其和环境相联的一些属性。模块定义图有唯一的名字，同时包含 Block 等模型元素，因此在模块定义图的模型中（名为 BlockDefModel）中需定义 name 、Blocks 等变量,这些变量属于模型自身的属性；当模型改变之后要通知其自身的控制器，因此 BlockDefModel 中应定义 PropertyChangeSupport 类型的成员变量（在 SysModeler 中其在 BaseModel 中定义，BlockDefModel 继承 BaseModel）来完成此功能；同时模型要显示在视图层上应具有位置和大小信息，因此一般模型中还需定义 location 和 width 两个变量（由于 BlockDef 占满整个编辑区，因此没定义这两个属性，但在 Block 的模型中定义了这两个属性）。

2) 操作。操作的定义是根据属性的定义来确定的。在模型层的模型类中的操作主要是一些变量的 get 和 set 操作，还有包含的子模型元素的 remove 和 add 操作，在这两个操作中应该添加 firePropertyChange（）方法来通知控制器模型的改变。因此 BlockDefModel 中包括 getName()、setName()、addBlocks（）、removeBlocks（）和 getBlocks（）等操作。模块定义图的模型 BlockDefModel 的实现代码如下所示：

```
public class BlockDefModel extends BaseModel{
    private String name;
    private List blocks = new ArrayList();//子模型列表
    ……
    public void addBlocks(BlockModel block)
    {
        blocks.add(block);
       // 继承 BaseModel 的方法
        firePropertyChange(CHILD, null, block);
    }
    public List getBlocks() {
        return blocks;
    }
     public void removeBlocks(BlockModel block){
        blocks.remove(block);
        firePropertyChange(CHILD, block, null);
     }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    ……
}
```

PropertyChangeSupport 类型的成员变量定义类 BaseModel 的实现代码如下所示：

```
public class BaseModel {
    // 定义静态变量
     public static final String CHILD = "CHILD";
     .....
     private PropertyChangeSupport listeners
                                  = new PropertyChangeSupport(this);
      // An addition of listeners
      public void addPropertyChangeListener
                          (PropertyChangeListener listener) {
              listeners.addPropertyChangeListener(listener);
      }
     // Change of a model is notified.
     public void firePropertyChange(String propName,
                    Object oldValue,Object newValue)       {
                listeners.firePropertyChange(propName,
                                    oldValue, newValue);
      }
       // Deletion of listeners
      public void removePropertyChangeListener
                (PropertyChangeListener listener) {
          listeners.removePropertyChangeListener(listener);
      }
}
```

5.2.2 控制器层的实现

控制器层模块是连接模型层模块和视图层模块的桥梁。控制器层模块要实现的功能不仅是在监听到模型层的改变后把改变显示到视图层上，而且要捕获用户在视图层上的操作请求进而发出命令来改变模型层。一个控制器由 EditPart、EditPolicy 和 Command、EditPartFactory 四种类才能共同完成控制器的全部功能。

1) EditPart 类。一个 EditPart 类是实现控制器模块的核心，一个 Model 对应一个 EditPart，一个 EditPart 对应一个 model，模型和控制器之间是一一对应关系。EditPart 类中用 active()方法注册监听器，删除监听器用 deactive()方法（在模型定义图的控制器 BlockDefEditPart 继承 BaseEditPart 来完成此功能的。）；实现监听到模型的改变主要由 propertyChange()方法来实现；用 createFigure()方法来返回模型在视图中显示的图形；更新模型的改变通过 refreshVisuals()或 refreshChildren()方法，用 createEditPolicy（）方法来安装策略；如果控制器对应的模型包含子模型用 getModelChildren（）方法来得到其所包含的子模型。

模块定义图的控制器 BlockDefEditPart 的实现代码如下所示：

```
public class BlockDefEditPart extends BaseEditPart {
    // 安装 EditPolicy
    protected void createEditPolicies() {
        installEditPolicy(EditPolicy.LAYOUT_ROLE,
                new BasePicXYLayoutEditPolicy());
    }
   // 显示模型的图
    protected IFigure createFigure() {
        BlockDefModel blockDef = (BlockDefModel) getModel();
        baseName = "bdd" + "      " + graphicName;
        return new BaseFigure();
    }
   // 得到子模型列表
    protected List getModelChildren() {
        return ((BlockDefModel) getModel()).getContainers();
    }
  // 监听到模型的改变更新视图
    public void propertyChange(PropertyChangeEvent evt) {
        String pName = evt.getPropertyName();
        if (pName.equals(BaseModel.CHILD)) {
            refreshChildren();
```

```
        }
    }
    ……
}
```

注册监听器的类 BaseEditPart 的实现代码如下所示：

```
public class BaseEditPart extends AbstractGraphicalEditPart implements
        PropertyChangeListener {
    public static String baseName,graphicName;
    // 注册监听器
    public void activate() {
            if(getModel() != null && getModel() instanceof BaseModel)
        {
        ((BaseModel)getModel()).addPropertyChangeListener(this);
            }
            super.activate();
        }
    // 删除监听器
     public void deactivate() {
         if(getModel() != null && getModel() instanceof BaseModel)
((BaseModel)getModel()).removePropertyChangeListener(this);
            }
            super.deactivate();
        }
    protected IFigure createFigure() {
        return null;
    }
    ……
}
```

2) EditPartFactory 类。EditPartFactory 是把模型和控制器联系起来的桥梁。简单说连接模型和控制器只需两步，就是（1）首先根据模型创建其控制器（2）然后用 setModel

（）函数连接模型和器控制器。模块定义图对应的 EditPartFactory 是 BlockDefEditPartFactory（实现代码如下所示），其中要连接的模型与控制器分别是 BlockDefModel 和 BlockDefEditPart 以及其子节点 BlockModel 和 BlockEditPart。

```
public class BlockDefEditPartFactory implements EditPartFactory
 {
    public EditPart createEditPart(EditPart context, Object model) {
          // get EditPart for model element
          EditPart part = getPartForElement(model);
          // store model element in EditPart
          part.setModel(model);
          return part;
     }
    // Maps an object to an EditPart.
    private EditPart getPartForElement(Object modelElement) {

          if (modelElement instanceof BlockDefModel)
               return new BlockDefEditPart();
          else if (modelElement instanceof BlockModel)
               return new BlockEditPart();
          ..............
          throw new RuntimeException(
               "Can't create part for model element: "
              +((modelElement != null) ?
               modelElement.getClass().getName(): "null"));
       }
}
```

3) EditPolicy 类。用户在视图层对视图进行操作之后，操作转换成 Role 之后必须有相应的捕捉此 Role 的 EditPolicy。EditPolicy 用 CreateEditPolicy（）方法中的 installEditPolicy（）方法安装在 EditPart，installEditPolicy（）中的第一个参数是 EditPolicy 对应代表 Role 的字符串，第二个参数则是需安装的 EditPolicy。EditPolicy

中由 getXXXCommand（）或者 createXXXCommand（）来得到相应的 Command。在上面的模块定义图的控制器 BlockDefEditPart 的实现代码中我们看到安装了 BasePicXYLayoutEditPolicy 策略，BasePicXYLayoutEditPolicy 继承了 XYLayoutEditPolicy，主要调用到了 ChangeNodeBoundsComand 和 CreateNodeComand 两个 Command 对象，前者是创建一个 Node 对象的命令，后者是改变 Node 的边框的命令。

BasePicXYLayoutEditPolicy 的实现代码如下所示：

```
public class BasePicXYLayoutEditPolicy extends XYLayoutEditPolicy {
    protected EditPolicy createChildEditPolicy(EditPart child) {
        ResizableEditPolicy childPolicy = new ResizableEditPolicy();
        childPolicy.setResizeDirections(PositionConstants.EAST_WEST);
        return childPolicy;
    }
  ..............
    protected Command createChangeConstraintCommand(EditPart child,
            Object constraint) {
        ChageNodeBoundsCommand command = new ChageNodeBoundsCommand(
            child.getModel(), ((Rectangle) constraint).getLocation(),
                ((Rectangle) constraint).width);
        return command;
    }
    public Rectangle getCurrentConstraintFor(GraphicalEditPart child) {
        IFigure fig = child.getFigure();
        Rectangle rectangle = (Rectangle)
                fig.getParent().getLayoutManager()
                .getConstraint(fig);
        if (rectangle == null) {
            rectangle = fig.getBounds();
        }
        return rectangle;
```

```
    }
    protected Command getCreateCommand(CreateRequest request) {
        Object newObject = request.getNewObject();
        Rectangle constraint = (Rectangle) getConstraintFor(request);
        Point location = constraint.getLocation();
        CreateNodeCommand creatCommand =
                new CreateNodeCommand(getHost()
                .getModel(), newObject, location, constraint.width);
        return creatCommand;
    }
}
```

4) Command 类。Command 是用来直接修改模型的类。在 Command 对象里最重要的是 execute()方法，是执行命令的方法。除此之外，Comamnd 对象里有 Undo（）和 Redo()方法，同时在 Command 对象里有成员变量负责保留执行该命令时的相关状态。每一个被执行过的 Command 对象实例都是被保存在 EditDomain 的 CommandStack 中。在 BasePicXYLayoutEditPolicy 安装的 CreateNodeComand 是添加节点的命令(实现代码如下所示)。在此 CreateNodeComand 中 location 和 width 是保留执行该命令时的相关状态的变量，execute（）方法是添加 BlockModel 等对象，undo（）方法是删除刚添加的 BlockModel 对象。

```
public class CreateNodeCommand extends Command {
    // 负责保留执行该命令时的相关状态的变量
    private Object parent;
    private Object node;
    private Point location;
    private int width;
   // 构造方法
    public CreateNodeCommand(Object parent, Object node, Point loc, int width)    {
        this.parent = parent;
        this.node = node;
        this.location = loc;
```

```
        this.width = width;
    }
    public void execute() {
        if (parent instanceof BlockDefModel) {
            BlockDefModel blockDef = (BlockDefModel) parent;
            if (node instanceof BlockModel) {
                BlockModel block = (BlockModel) node;
                block.setName("Block"
                +(((BlockDefModel)parent).getSumBlocks().size()+ 1));
                block.setLocation(location);
                block.setWidth(width);
                blockDef.addBlocks(block);
            }
                    ....
        } else if (parent instanceof BlockModel) {
            ....
        }
            ....
    }
    public void undo() {
        if (parent instanceof BlockDefModel) {
            BlockDefModel blockDef = (BlockDefModel) parent;
            if (node instanceof BlockModel) {
                BlockModel block = (BlockModel) node;
                blockDef.removeBlocks(block);
            }
                        ....
        } else if (parent instanceof BlockModel) {
            ....
        }
```

```
        ....
    }
}
```

5.2.3 视图层的实现

视图层主要是用来显示模型的，Editor 是 GEF 的操作界面，也是 SysModeler 的操作界面，因此视图层的实现主要是集中在 Editor 对象。要实现视图层需要在 Editor 上配置 GraphicalViwer 来创建视图 Viewer。在 GraphicalViwer 需要配置两个函数分别是 initializeGraphicalViewer（）函数和 configureGraphicalViewer（）函数。initializeGraphicalViewer（）用来接收模型。configureGraphicalViewer（）配置 GraphicalViewer，在其中需要配置合适的 RootEditPart 和 EditPartFactory。RootEditPart 的不同 Editor 所具备的功能也有所不同。在一个 Editor 对象的构造函数中同 setEditDomain（new DefaultEditDomain（this））方法来添加 EditDomain，它主要用来管理命令栈 cmmand statck、工具条 palette viewer 等。一个 Editor 中必须建立一个 EditDomain。由于 SysModeler 与其他图形编辑软件一样，要实现模型体的绘制和拖拽等功能，所以 SysModeler 中各个 Editor 的都继承了 GEF 自身的 GraphicalEditorWithFlyoutPalette 类使得每个 Editor 都包含一个画板 Palette Viewer，同时通过 getPaletteRoot（）方法添加配置 Palette Viewer。通过 getAdapter（）方法添加模型图的大纲视图（模型图的缩略图）等等。模型定义图的编辑器 BlockDefEditor 的实现代码如下所示：

```
public class BlockDefEditor extends GraphicalEditorWithFlyoutPalette{
    private BlockDefModel blockDefModel;
        public BlockDefEditor() {
            super();
            setEditDomain(new DefaultEditDomain(this));
            editPartFactory = new BlockDefEditPartFactory();
            treeEditPartFatory = new BlockDefTreeEditPartFactory();
    }
    protected void initializeGraphicalViewer() {
        blockDefModel = new BlockDefModel();
        model = blockDefModel;
```

```
        viewer.setContents(blockDefModel);
    }
    protected void configureGraphicalViewer() {
        super.configureGraphicalViewer();
        viewer = getGraphicalViewer();
        viewer.setEditPartFactory(editPartFactory);
        // 得到一个具有放大缩小功能的 rootEditPart
        ScalableRootEditPart rootEditPart=
        new ScalableRootEditPart();
        viewer.setRootEditPart(rootEditPart);
          .....
     }
    protected FlyoutPreferences getPalettePreferences() {
        return new FlyoutPreferences() {
            ....}
    }
    protected PaletteRoot getPaletteRoot() {
        // 首先在 Plette 上添加 Route
        root = new PaletteRoot();
        //创建一个工具组用于放置常规的 Tools
        PaletteGroup toolGroup = new PaletteGroup("工具");
        //创建一个 GEF 提供的 Selection 并将其放置在 ToolGroup 中
        ToolEntry tool = new SelectionToolEntry();
        toolGroup.add(tool);
        root.setDefaultEntry(tool);//该选择工具为缺省的被选择工具
        //创建创建一个 GEF 提供的 Marquee 多选工具并将其放置在 ToolGroup 中
        tool = new MarqueeToolEntry();
        toolGroup.add(tool);
        toolGroup.add(new PaletteSeparator());
        PaletteDrawer connectionDrawer = new PaletteDrawer("连接");
```

```
        ImageDescriptor newConnectionDescriptor = EDiagramImages
                .getImageDescriptor(EDiagramImages.CONNECTION);
        .....
        connectionDrawer.add(connxCreationEntry);
        // 创建一个 Drawer 抽屉放置绘图工具，该抽屉名称为"画图"
        PaletteDrawer drawer = new PaletteDrawer("模块定义图");
        // 指定创建模型工具的图形
        ImageDescriptor descriptor = EDiagramImages
                .getImageDescriptor(EDiagramImages.CLASS);
        // 创建模型工具
        CreationToolEntry creationEntry = new CreationToolEntry(
                "New    Block",// 在 Platte 中显示的文字
                "Create a new block",// Tool 提示
                new SimpleFactory(BlockModel.class),// 模型创建工厂
                descriptor, descriptor);
        drawer.add(creationEntry);
        .....
        // 将两个工具组添加到 root 上
        root.add(connectionDrawer);
        root.add(drawer);
        return root;
        root.add(toolGroup);
        return root;
    }
    public void doSave(IProgressMonitor monitor) {
        .....
    }
    public void doSaveAs() {
        .....
    }
```

```
public boolean isSaveAsAllowed() {
    return false;
}
    //----------Content Outline page----------------
class CustomContentOutlinePage extends ContentOutlinePage {
    //使用 SachForm 把大纲视图分为两个部分：显示大纲和显示鹰眼
    private SashForm sash;
    private ScrollableThumbnail thumbnail;
    private DisposeListener disposeListener;
    public CustomContentOutlinePage(){
        super(new TreeViewer());
    }
    public void init(IPageSite pageSite){
        super.init(pageSite);
        ActionRegistry registry=getActionRegistry();
        IActionBars bars=pageSite.getActionBars();
        String id=ActionFactory.UNDO.getId();
        bars.
          setGlobalActionHandler(id,registry.getAction(id));
        id=ActionFactory.REDO.getId();
        ……
    }
    //重载
    public void createControl(Composite parent){
        //创建 SachForm
        sash = new SashForm(parent,SWT.VERTICAL);
        //添加分隔条
        getViewer().createControl(sash);

        //设置 editor domain
```

```
                getVjewer().setEditDomain(getEditDomain());
                //setup EditPartFactory
                getViewer().setEditPartFactory(treeEditPartFatory);
                getViewer().setContents(model);
                getSelectionSynchronizer().addViewer(getViewer());
                ..........
                };
                ..........
        }
        public Object getAdapter(Class type){
            ....
            if(type==IContentOutlinePage.class){
                return new CustomContentOutlinePage();
            }
            return super.getAdapter(type);
        }
          // 添加右键菜单
        protected void createActions() {
          super.createActions();
          .....
        }
            ......
    }
```

5.3 SysModeler 的应用场景

SysModeler 是支持 SysML 建模语言的建模工具，主要用户是系统工程界的建模人员，根据系统工程的通用建模过程以及 SysModeler 自身的特点，在本节中结合一实例（Rain Sensing Wiper （RSW） 系统）说明了 SysModeler 的应用场景。

RSW 系统的主要目标是无论任何时候挡风玻璃外表面检测到液滴时，都能够自动化的进行擦洗（无需用户干预）。此外，依据检测到的液滴量调整雨刷器的速度。这个系统拥有

三种主要部件：（1）控制雨刷器行为的软件，（2）执行软件的电子控制单元，（3）固定在挡风玻璃内表面的传感器，它的任务是通过挡风玻璃感应液滴。

下面结合 SysModeler 自身的特点和 RSW 系统要实现的功能，简要说明系统工程建模者如何使用 SysModeler 对各个系统进行建模，及其建模的过程。首先进行需求和用例建模，然后结构建模，最后进行行为建模。

5.3.1 需求和用例建模

1) 对不同的问题域进行知识抽取，确定系统的边界绘制系统环境图。

环境图不属于 SysML 中的图形，但是可以用 SysML 模块图表示，但此模块图中没有属性和方法。图 5.3 展示了 RSW 的环境图。环境图构建了系统的范围。在本例中，它为系统确定了三类角色：Maintenance （维修用途）， Car Electrical System （启动汽车中的系统）， Driver （例如手动关闭系统）。需考虑三种外部系统：雨刷器接口，挡风玻璃，和汽车电子系统。注意用户定义的关键字 "external" 是用来限制外部部件的。还要注意汽车电子系统也要为 RSW 提供电力。因此，它应同时属于角色和外部系统。

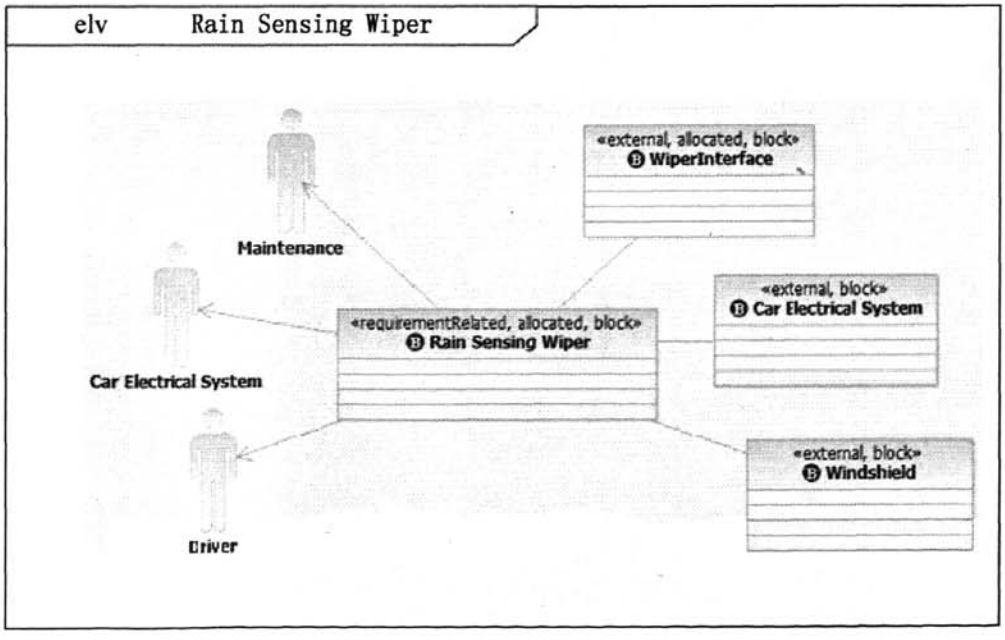

图 5.3 RSW 的环境图

Figure 5.3 RSW Environment picture

2) 捕获系统需求绘制需求图（包括功能性需求和非功能性需求）。

开发周期的概念阶段被分离到用户需求分析过程中，这就形成了产品需求。传统

的需求都被表示为文档的形式，它们经常与数据和图表联系在一起，存储在文件或数据库中。需求描述了所有产品功能，并且定义了实现这些功能的约束条件。SysML 允许需求描述作为模型元素。因此，需求变为产品架构的一部分。这种语言灵活的表达了任何种类、任何关系的基于文本的需求。在 RSW 中一组产品的核心功能源自于 Automatic Wiping 下的需求组。Automatic Wiping 内的部分子需求通过交叉关系与它联系在一起。父需求相当于封装了嵌入的需求。在某种意义上说，删除父需求将会自动删除所有嵌入其中的需求。另一个封装其他需求的例子是 Core Functions，它包含了两个子需求等。

3) 根据环境图和需求图，绘制用例图展现外部角色的之间交互。

SysML 提供了继承于 UML 2.0 的用例图。我们用主要的用例（由椭圆图显示）展示了外部角色的交互作用。RSW 系统的用例图（如图 5.4 所示）展示了三类角色并分别介绍了各自的用例。被称为 Automatic Wiping 的关键用例是由一系列子用例组成。这种等级关系通过包括（Include）依赖关系形成。

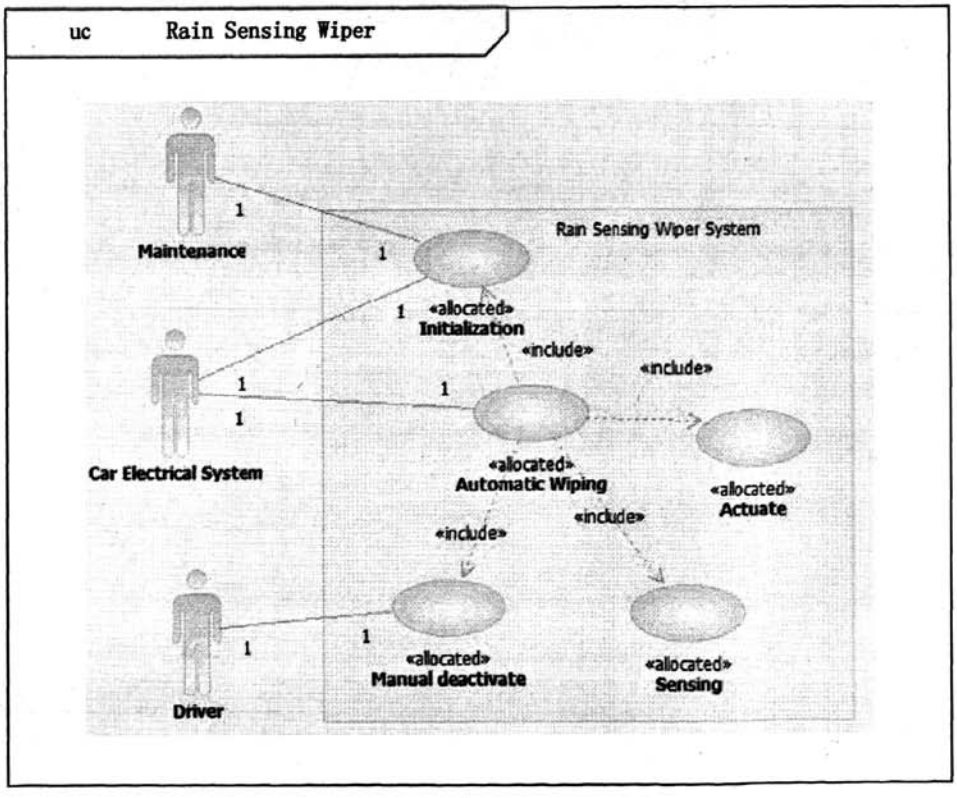

图 5.4 RSW 的用例图

Figure 5.4 RSW UseCase Picture

5.3.2 结构建模

4) 绘制系统的模块定义图描述系统结构，并将相关需求分配到模块定义图中，检查需求和系统结构之间的联系。

SysML 模块定义图（BDD）是描述系统结构的最简单的方法。它等同于 UML 中的类图。它用来描述系统分解，例如，关联与合成关系。BDD 最适合用来表现模块的特性，例如模块的属性和操作。图 5.5 显示了 RSW 的一种 BDD。虽然模块中的子系统和 Rain Sensing Wiper 元素之间有关联关系，但是出于图表的可读性考虑，我们并没有将它们表现出来。而是使用一个指示框将每一个组件（合成的和外部的）包含在其中，同时在合成的组件上使用一个黑色菱形箭头。RSW 的主要组件是：一个开动雨刷的接口、一个电子控制单元、一个传感器和一风挡部件。不管是否存在 RSW，汽车中都可以有接口和风挡。同时在图 5.5 中显示了每一个模块的属性与操作。

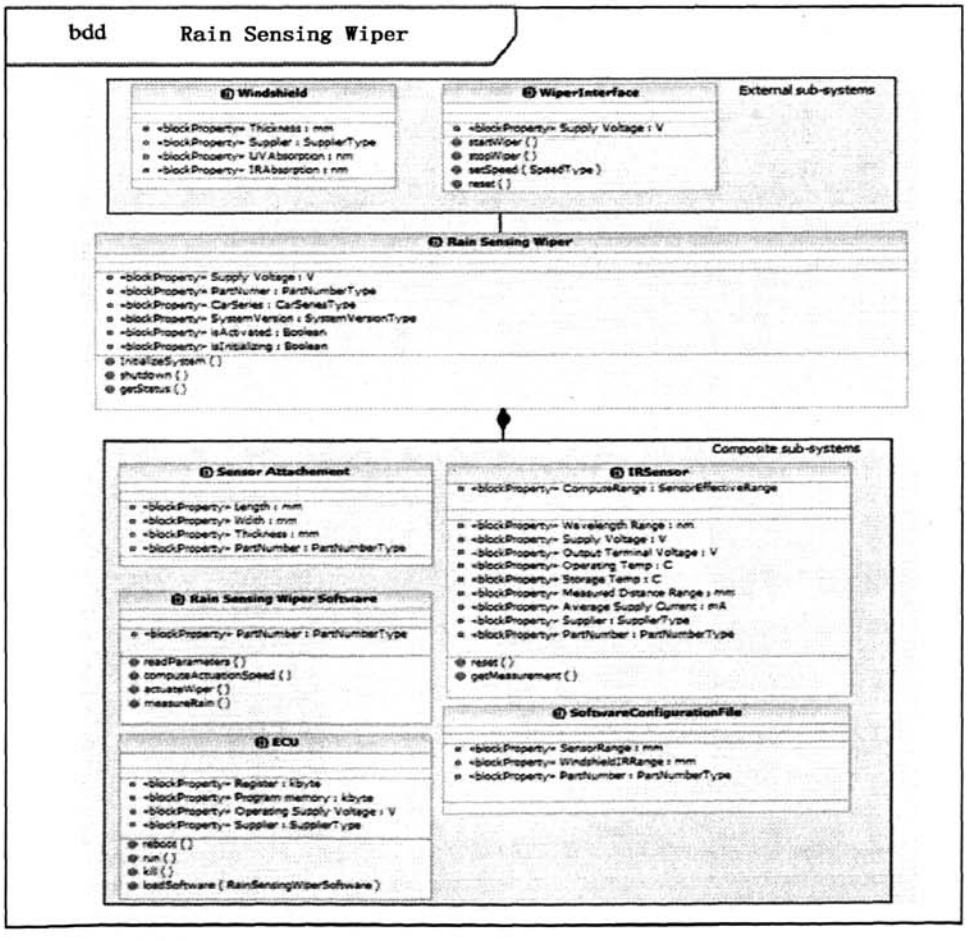

图 5.5 RSW 的模块定义图
Figure 5.5 RSW Block Define Picture

5) 绘制内部模块图描述模块内部结构，并将相关需求分配到内部模块图中，显示内部模块图中满足的需求。

SysML 内部模块图（IBD）允许设计人员去改进模块的结构。IBD 等同于 UML 中的复合结构。IBD 最重要的方面是它允许设计人员通过定义端口（port），来改进模块用法之间的定义交互作用。在 RSW 系统中需要定义三部分模块内部结构，主要包括嵌入硬件部分、用来在汽车中装配系统的部分以及软件部分。同时需要定义一系列标准化的端口和接口，用来表示各部分之间通讯的功能性方面。例如，处理单元通过 WiperECUCommunication 接口访问雨刷活动接口。

6) 绘制系统的参数图，来描述系统约束模块之间的定义的参数直接的连接关系。

通过上面的几步已经描述了如何为模块定义属性，让其表现他们的物理特性。通常情况下一系列系统的属性是相互关联的。假设两个子系统 A 和 B 分别拥有属性 a 和 b，它们的约束{A.a 大于 B.b}关系必须要保持正确。因此在 SysML 中定义了参数图来描述系统约束模块之间的定义的参数直接的连接关系。。

RSW 使用一系列分析约束来确认系统是否被恰当的校准。在其的参数图中包含三个约束关系：

约束关系 SensorEffectiveRange 根据红外线传感器的参数，为其计算了一个可运行的范围。

同样的，约束关系 WindshieldIREffectiveRange 为风挡计算了一个可运行的范围，它可以和红外线传感器的范围做比较。

约束关系 SensorWindshieldRangeCompare 用来比较上述两个范围的值。

5.3.3 行为建模

7) 描述模块之间的交互关系，绘制系统行为模型图（包括顺序图、状态机图和活动图）来实现用例，同时可以把相关需求与行为模型图联系起来，来测试用例。

顺序图

RSW 的交互图（如图 5.6 所示）中包括组件间的同步协议。首先，汽车电子系统启动 RSW，由 RSW 依次确认传感器和雨刷器接口。之后，软件被加载到引擎控制单元的内存中并开始执行。一旦软件启动，它将会读取被存储在校准文件中的参数。

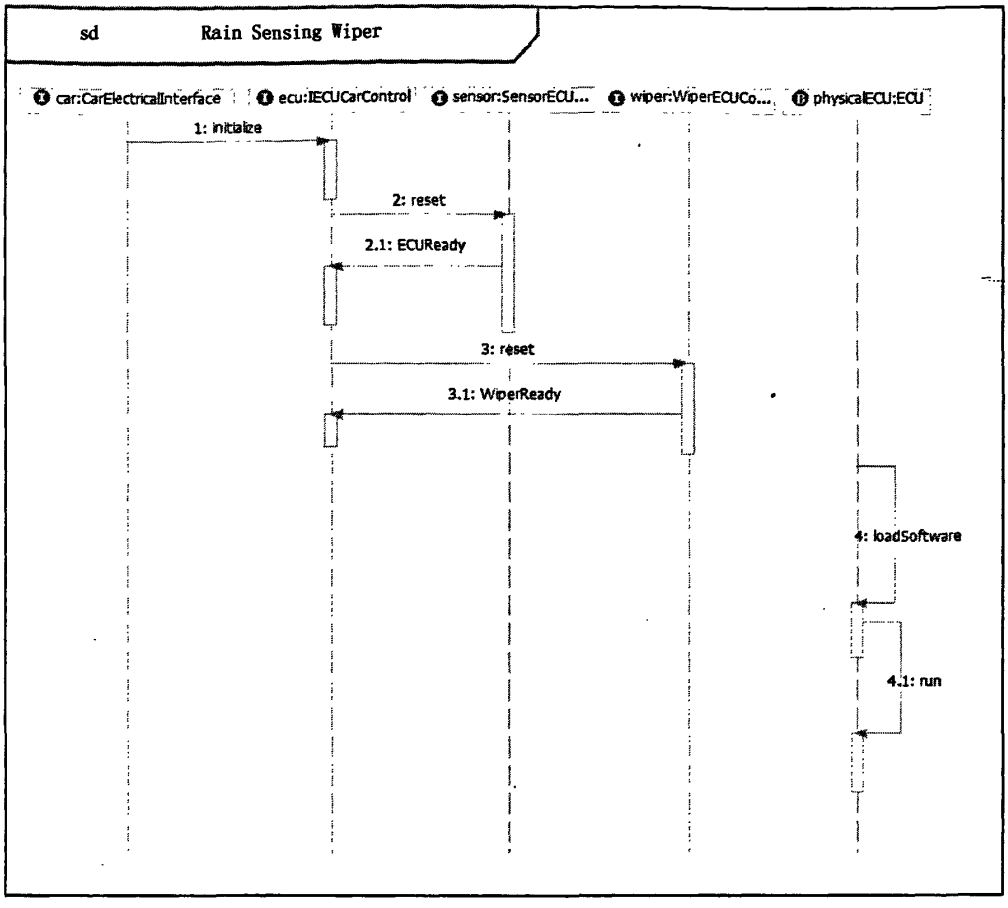

图 5.6 RSW 的顺序图

Figure 5.6 RSW Seque Picture

状态机图

SysML 状态机图用来表示产品的不同状态。这种行为模型来自于 UML。RSW 的状态机图（如图 5.7 所示）中包括三种状态：Deactivated、 Initializing 和 Activated。在 Deactivated 模式中（例如，用户手动停止），系统等待激活命令。当接收到信号时，它将过渡到 Initializing 状态。此时，执行图 5.6 中的互动序列。成功完成后，系统过渡到 Activated 状态。

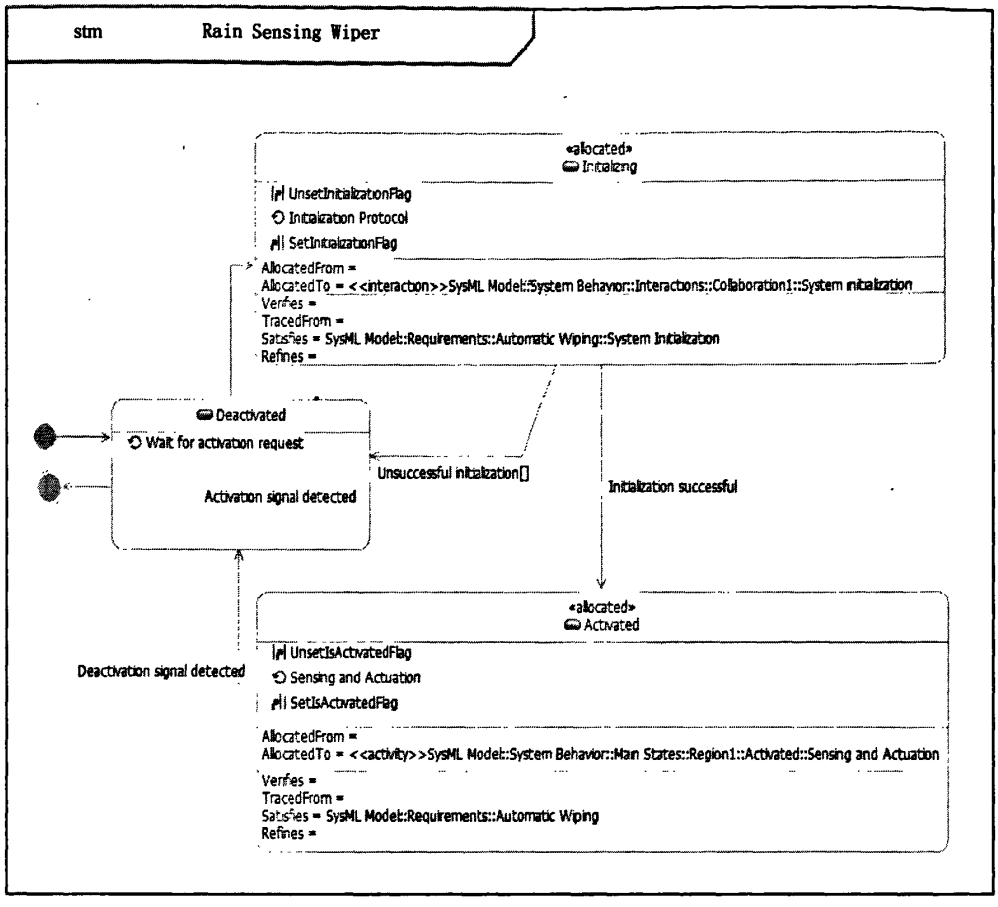

图 5.7 RSW 的状态机图

Figure 5.7 RSW State Machine Picture

活动图

SysML 活动图利用并扩展了 UML 中的活动模型以支持连续系统。SysML 活动图具有多种创新。在 RSW 系统中的活动图（如图 5.8 所示）主要实现了雨水感知（rain-sensing）功能。ManualControl 控制操作符用来启动和停止雨水感应功能。当接收到 "deactivate" 信号（例如，由仪表盘上的按钮触发）时，它将使这项功能无效。 当接收到适当的信号时，它将重新启动这项功能。活动 Process 和活动 ManualControl 间交换的数据类型是 ControlValue 类型。Process 动作具有 Overwrite 类型的插脚以接收来自传感器的数据。Overwrite 类型的对象结点通过擦除对象结点中的旧信息可以确保来自传感器的最新信息对于 Process 活动是可得到的。

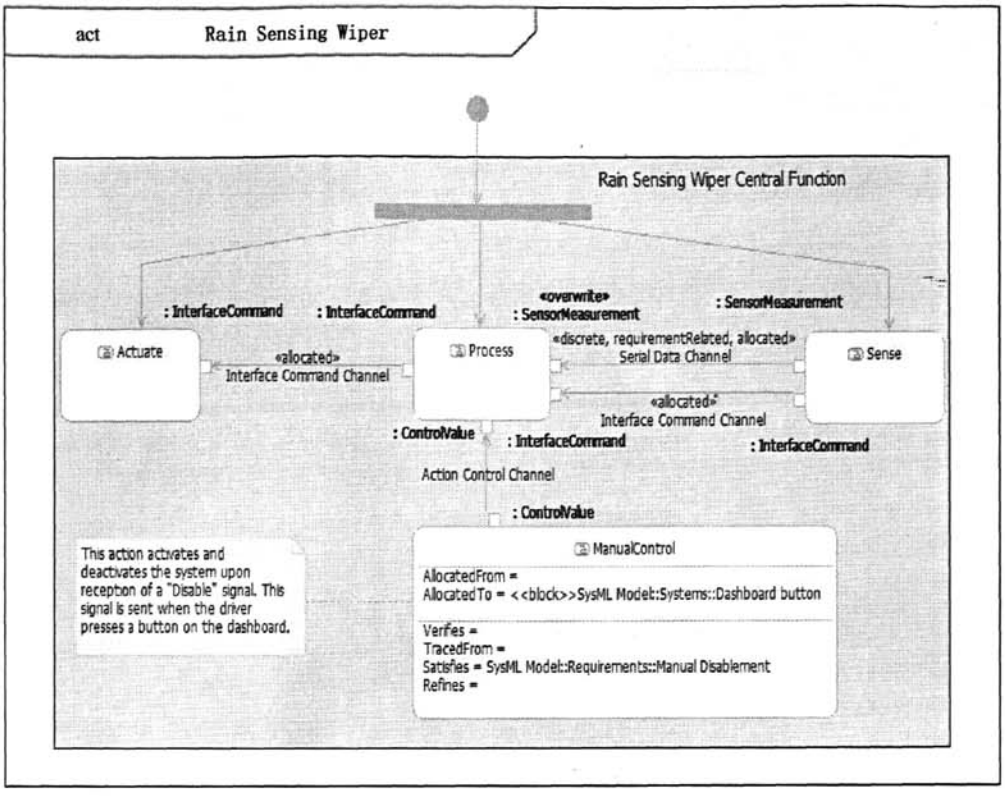

图 5.8 RSW 的活动图

Figure 5.8 RSW Active Picture

第六章 总结与展望

本系统的设计出发点是设计和实现一个按照系统工程项目为中心的文档组织方式，囊括了系统工程建模时所需的若干个完整的工具，有助于整合系统工程过程的 SysML 建模工具。本论文的主要研究工作如下：

1) 分析了 SysML 的语义和表示法，帮助进一步了解和掌握 SysML，同时为设计软件中所需的模型提供理论基础。
2) 介绍了 SysModeler 所使用的开发平台 Eclipse 以及其的插件 GEF 的工作机制，从而为提出软件的设计方案打下坚实的基础。
3) 提出了 SysModeler 的软件设计方案，自主完成了软件结构设计。
4) 设计并实现了 SysModeler 软件，从而为系统工程师提供了一个简单便捷的绘制模型图的建模工具。

本软件 SysModeler 的主要特点是。

1) 简单清晰：专门为系统工程师设计。
2) 易学易用：具有 UML 建模经验的用户很容易使用该工具。
3) 完美的 Windows 特性支持汉化图形界面：界面友好，操作简单，采用 Windows 应用程序风格的中文界面。

由于时间与工作量的限制，不可避免的存在各种不足，还有很多方面值得进一步深入研究和探索，因此还有很多未尽的工作需要完成。对其功能的改善可以从如下几个方面进行：

第一，系统功能进一步完善。如今，系统已经有了一个雏形，但还没有成熟。目前的 SysModeler 尚不能支持其他工具模型的导入，以及校验模型的正确性等。这些功能需要相关研究工作的进展和工具完善。

第二，各个功能模块代码结构的优化。由于各个模块之间的相互关系完全继承 GEF 的 MVC 模式，其目的是减少代码编写的工作量，但是由于对 MVC 了解的程度不够，所以在系统的实现过程中还存在一些冗余代码，这部分工作还需要进一步的完善。

第三，用户界面的优化，由于时间和能力的原因，界面比较粗糙需要进一步优化。

参 考 文 献

[1] What is Systerm Engineering,INCOSE,URL http://www.incose.org/whatis.html,2005.

[2]Ma HH, Shao WZ and Ma ZY. The Overview of UML 2.0. Computer Seience, 2004, 31(7):1-8. (马浩海, 邵维忠, 麻志毅. UML 2.0 述评. 计算机科学,31(7):1-8, 2004.).

[3] UML 2.0 Infrastructure Specification, OMG Document: ptc/03-09-15..

[4]Ingmar Ogren.Possible Tailoring of the UML for Systems Engineering Purposes[J].Systems Engineering(S1098-1241).2000,3(4):212-224.

[5]Jakob Axelsson.Model Based Systems Engineering Using a Continuous-time Extension of the United Modeling Language(UML) [J].Systems Engineering(S1098-1241).2002,5(3):165-179.

[6]Terry Bahill,Jesse Daniels.Using Objected-Oriented and UML Tools for Hardware Design:A Case Study[J].Systems Engineering (S1098-1241).2003,6(1):28-48.

[7]Conrad Bock.UML 2 Activity Model Support for Systems Engineering Functional Flow Diagrams[J].Systems Engineering (S1098-1241).2003,6(4):249-265.

[8]吴娟，基于 SysML 的 DoDAF 产品设计研究, 华中科技大学学报，2006 [5].

[9]SysML Partners.System Modeling Languages(SysML)Specification Version 0.9 DRAFT.URL http://www.sysml.org/artifacts.htm,2005.

[10]SysML Partners.SysML v.1.0a Specification Overview. http://www.sysml.org/artifacts.htm, 2005.

[11] Systems Modeling Language (SysML) Specification, OMG document: ad/2006-03-01.

[12]Jiang CY，Wang WP and Li Q. SysML:A New Systems Modeling Language.Journal of System Simulation，2006，18(6):1483-1487.

（蒋彩云，王维平，李群.SysML：一种新的建模语言.系统仿真学报，18(6):1483-1487，2006）

[13] OMG SysML specification，OMG Document: ptc/06-05-04.

[14] Laurent Balmelli .An Overview of the Systems Modeling Language for Products and Systems Development ， IBM Technical Report 2006.Object Technology (JOT) at www.jot.fm,2006-06-03 .

[15] Matthew Hause and Alan Moore.The Systems Modeling Language.ARTISAN software Tools,2006-07.

[16] Jeremy Dick, Jonathon Chard. UseCase-driven and Model-driven：Be Good to Systems Engineers.Telelogic.2003-9.

[17]Wu J et al. Product design of systems architecture using SysML.Systems Engineering and Electronics，2006，28(4):593-598.

(吴娟等.基于 SysML 的系统体系结构产品设计.系统工程与电子技术，28(4):593-598，2006).

[18] 吴娟等， DoDAF 产品集的 SySML 模型支持，兵工自动化，25（2）13～15，2006.

[19] 陈刚.Eclipse 从入门到精通.清华大学出版社.

[20] zhlmmc，Eclipse RCP 入门，http://www.eclipseworld.org/bbs/index.php.

[21] 张浩，GEF 入门系列， http://bjzhanghao.cnblogs.com.

[22] Chinan,GEF Description， http://eclipsewiki.editme.com.

[23] carolwoo,GEF-whole-upload， http://www.eclipseworld.org/bbs/index.php.

[24] Ma ZY et al. Research and Implementation of Jade Bird Object-Oriented Software Modeling Tool. Journal of Software, 2003, 14(1):97-102.

(麻志毅等. 青鸟面向对象软件建模工具的研究与实现. 软件学报, 14(1):97-102, 2003.).

[25] 鲍鹏丽，马浩海.SysModeler：一个系统建模语言的建模工具.计算机应用与研究已录用.

[26] MA HH, XIE B, MA ZY et al. PKUMoDEL: A Model-driven Development Environment for Languages family. Accepted to appear in Journal of Computer Reseach and Development.

(马浩海，谢兵，麻志毅等. PKUMoDEL:模型驱动的开发和语言家族支持环境. 计算机研究与发展已录用).

[27] 石福丽、朱一凡等，基于 SysML 的无人侦察机需求描述方法研究,，计算机仿真，24（6）53~57，2007.

[28] Wu C W, Chua L O. Synchronization in an array of linearly couple dynamical systems [J] . I EEE Trans. Ci rcuits Syst . , 1995 , 42(8) : 430 - 447.

[29] OMG Systems Engineering Domain Special Interest Group (SEDSIG).UML for systems engineering RFP [EB/OL]. (2003-03-01).

[30] OMG.SysML-v0.9-PDF-050110.pdf [EB/OL]. (2005-01-10) [2005-04-18]. http://www.sysml.org, 2005.

致 谢

时间飞快，转眼三年研究生生活即将结束，在本论文完成之际，衷心感谢马浩海老师、高光来老师，感谢他们在学习、生活方面的给予我的帮助，他们渊博的知识、宽广的胸襟、严谨的治学态度、一丝不苟的工作作风都是我今后人生道路上学习的榜样，他们的谆谆教导和言传身教将使我终生受益，在这里再次衷心地向他们说声：谢谢！

感谢北京红旗中文2000给我提供了这么好的锻炼机会，感谢杨彦林、金哲求同事在系统开发过程中给与我的指导和帮助。

感谢马老师给予我专业方面的指导，感谢张冠女、赵永安和赵凤荣等同学对我的无私帮助，他们的拼劲、工作作风和渊博的知识永远值得我学习。

感谢所有曾经给给予鼓励、帮助和支持的老师和同学。

衷心感谢我的父母及家人，是你们的无私奉献才使我有了这么好的学习和深造的机会。

攻读硕士期间发表的学术论文

第一作者，1 篇

[1] .SysModeler：一个系统建模语言的建模工具，计算机应用与研究，核心期刊，2008 年 7 月