摘　要

Web 应用的出现与盛行已将软件产业悄然带入规模化、复杂化的时代，提高其检测方法的自动化程度是当务之急。因此，我们采用基于模型的测试方法，以 Web 应用模型作为测试用例的来源。另外，为了使该模型与用户需求保持一致，必须对其应满足的各种性质进行验证。由此可知，模型构建是重要问题。

UML 是理想的需求与设计模型语言，而模型检验与测试自动化要求模型用统一的形式化语言描述。鉴于大多数模型检验工具以 FSM 模型作为输入，本文提出了一种将 UML 模型形式化为 FSM 模型的方法，使之满足检测过程的需要。由于状态图代表系统行为的最小单位，本文从它们的形式化着手。另外，顺序图也是描述系统行为的常用模型，本文亦对其进行了研究并给出相应的形式化方法，不但服务于检测过程，也可用于后续归约过程。

高复杂度的 Web 应用软件将导致大规模的 FSM 模型，并可能引发状态爆炸。为了解决这个问题，我们利用归约指示及本文提出的一系列相关算法，对前期模型形式化后获得的系统 FSM 模型进行归约优化，产生只描述用户所要求测试的功能的 FSM 模型。该方法主要可分为两步，首先生成形式化的归约指示，将顺序图模型转换为 FSM 模型，然后以该形式化的归约指示为指导，对系统 FSM 模型执行归约操作，达到优化的目的。

FSM 是我们研究工作中系统行为的重要描述形式，它为各环节传递信息，但计算机处理能力的局限性迫使模型只能以文本为载体。为了直观地了解其结构，本文给出了一种层次 FSM 模型的可视化方法。它首先读取模型文本，并以层次化的数据结构进行存储，然后根据用户要求查看的复合结点的对应关系，对相应 FSM 模型进行可视化。

为了支持本文提出的上述一系列方法，我们开发了相应工具原型，它利用 XMI 文件作为 UML 图形式模型的对应文本，并使用自定义的 XML 文件记录 FSM 模型。通过本文给出的读取算法，该工具先对文本中的模型信息进行分析，并以某种数据结构存储，然后根据用户需求，结合论文中提出的方法对 UML 模型进行形式化并对 FSM 模型进行优化与可视化。

关键词：UML，FSM，模型形式化，模型归约，模型可视化

ABSTRACT

With the emergence and popularity of Web applications, testing requires for a higher automation level because of their large scale and high complexity. To this end, we use Model-based testing which takes the models of Web applications as the origins of test cases. Furthermore, in order to verify that these models are consistent with users' requirements, we will check if they satisfy the specified properties. Therefore, construction of models is a critical part of the problem.

UML is an ideal modeling language for requirements presentation and design process, however, model checking and automatic testing can only be implemented with models represented in unified formal notations. Considering that model checking tools usually take FSM-based model as their specified input, we propose a transformation method from UML models to FSM models, meeting the demands for detection works. Since UML state diagrams are considered to be the minimal unit representing behaviors, our work started with formalizations of these diagrams. Besides, sequence diagrams are also often used when describing the behavior of the system, an approach to transforming them into FSM models is presented, not only facilitating automatic test case generation and model checking, but also the later process of reduction.

FSM models of Web applications would become larger and larger as the complexity of software increases, leading to the problem of state explosion. To solve this, we use reduction directives and several relative algorithms proposed by this paper to reduce the original FSM obtained in the earlier process of model transformation, resulting in much smaller ones with functions we would like to test specifically. It can be divided into two parts: generation of formalized reduction directives in FSM models transformed from sequence diagrams and reduction of original FSM models in the guidance of the obtained reduction directives, reaching the goal of model optimization.

FSM is an important representation for describing behavior of the system in our research works; it transfers information between different phases. But due to computers' limited capability on information processing, we could only use textual presentation. To directly see the architectures of FSM models, we've figured out a method for the visualization of them. We firstly analyze the documents of the hierarchical FSM model, and then store the information extracted from these documents in a hierarchical data structure. Next, visualizing specified FSM model according to the composite state selected by users and its corresponding FSM model.

For the purpose of supporting the methods proposed by this article, we develop a tool prototype which takes XMI files as textual presentation for UML models and self-defined XML files for FSM models. It firstly analyzes the input file, extracting information of its corresponding model which will be stored in a certain data structure. And then, it implements formalization of UML models, reduction and visualization of FSM models with algorithms given in this thesis according to users' requirements.

Keywords: UML, FSM, Model formalization, Model reduction, Model visualization

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。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已发表或撰写过的研究成果。参与同一工作的其他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。

签 名：王哲 日 期：2009.3.7

本论文使用授权说明

本人完全了解上海大学有关保留、使用学位论文的规定，即：学校有权保留论文及送交论文复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容。

（保密的论文在解密后应遵守此规定）

签 名：王哲 导师签名：侯胜利 日期：2009.3.7

第一章 绪论

1.1 课题来源

本课题来源于国家高技术研究发展计划（863 计划）（项目编号：2007AA01Z144）和国家自然科学基金项目（批准号：60673115）。

1.2 课题研究的目的和意义

20 世纪上演了两场举世瞩目的社会变革，堪称人类文明史上第三次及第四次科技革命的主要标志，即电子计算机与互联网的发明与广泛应用。生产自动化、办公自动化与家庭生活自动化都预示着我们将进入一个全新的自动化时代，而随着个人计算机的普及，每个普通公民都能享受到科技带给他们的便利与改变，无论是工作形式的改变、工作质量的提高、知识与信息的有效管理及获取还是新鲜的娱乐模式，都深刻地影响着每个人的生活方式。网络的出现与流行更是将这一发明的空前与伟大性推向极致。一台计算机与一根网线便能将自己与世界连接，你可以在这个取之不尽用之不竭的宝库中找到任何想要的资源与服务，让你无时无刻感受到它的存在与万能。

计算机硬件只为上述种种服务提供运行平台，真正实现其功能的是建立于平台之上的软件系统，这些软件系统才是计算机的灵魂所在。所以，与其说计算机无孔不入地渗透于社会各领域中，倒不如说软件产品已成为与我们息息相关的生活必备，其质量好坏已被越来越多的人所关注，尤其是关键领域内的软件产品，稍有纰漏都有可能造成极其严重的后果。据美国国家标准和技术研究院的报告显示，美国大型专用软件开发的失败率高达 70%，美国经济因软件错误，每年损失高达 595 亿美元，中国软件的市场应用状况也如出一辙。如果企业在使用软件系统之前采用相关的质量保证手段，则可以有效避免各种风险。模型检验与测试则分别是最重要的检测方法之一[1]，测试更是成为软件开发过程中必不可少的环节。有数据表明，国外优秀的软件开发机构把 40%的工作量

花在软件测试上，软件测试费用则占软件开发总费用的 30%—50%，足见软件测试举足轻重的地位，在如此大量投入的形势面前，其效率自然而然地被业界视为焦点问题。

手工测试是软件测试的最初形态，至今仍占据着主导地位。由于自动化测试技术尚不成熟，大部分公司或企业仍旧依靠大量人力完成软件产品的测试任务。但是，用户不断增长的需求使软件朝着复杂化方向发展，规模日益扩大，手工技术已很难满足测试要求。另外，网络化时代的到来推动着一种新型软件——Web 应用软件以惊人的速度向前发展，并开始涉及与其相关的各个层面。较之传统软件，Web 应用软件呈现出一些新的特点：

异构性，各实体运行于不同的软/硬件平台，使用不同的编程语言，并与其它跨组织、跨平台的构件、中间件及服务器等进行交互；

动态性，一方面技术、用户需求与构件版本都在不断更新，另一方面其状态转换路径由用户输入、用户状态数据及服务器状态数据共同决定。

传统的取决于测试者的测试手段并未考虑针对以上特点的应对措施，若盲目用于 Web 应用软件的测试无疑将是徒劳。另外，迫于竞争压力，大部分公司或企业更加青睐于快速开发，希望最大限度地缩短生产周期，使产品尽早投入市场。而自动化程度更高的方法可将单调繁冗的工作交给计算机，其高效率与高正确率优势，实为人工所不及。综上所述，工业界期望的是一种自动化程度高、周期短，且对有如 Web 应用的复杂大型软件亦有效的测试方法。

基于模型的测试方法让我们看到了曙光。测试的主要工作是测试用例的产生，根据这些测试用例对被测对象进行输入操作，并在其实际相应输出与预定输出的比较过程中发现可能存在的错误，达到提高软件质量的目的。而基于模型的测试方法使自动产生测试用例成为了可能，它的信息来源是需求分析与设计阶段给出的软件结构与行为模型。由于需求与设计模型在软件开发的初级阶段就已按照用户需求创建[2]，测试用例的生成可与系统实现并行完成，不必等到编程结束再由代码产生测试用例，从而节省了大量串行模式白白浪费的时间。另外，模型及由其产生的测试用例具有平台无关性与复用性，当规格说明发生变化时，改变模型相对简单很多[3]，当平台发生变化时，只需对测试执行环境

重新进行动态配置，自动产生实例化的可执行测试包。这一特性大大降低了 Web 应用的动态性和异构性带来的测试复杂度。因此，基于模型的测试方法对于大型软件包括 Web 应用软件的测试极具优势，是本课题组选定的研究方向，也是论文的研究背景。

测试模型的构造是基于模型的测试方法的关键。在各种建模语言中，UML 凭借其简单性与易用性被业界广泛接受与应用，它可以帮助用户从各种角度分析待开发的系统。作为一种可视化的语言，UML 能清楚地显示系统的结构与功能，促进了设计人员、建模人员、开发人员和用户之间的交流。另外，在 UML 的发展历程中，业界设计并实现了许多与之相关的支持工具，以帮助用户更方便地利用 UML 进行建模。但是，UML 图中某些标记及某些图本身都无法提供精确的语义，尽管它拥有上述众多优点，其半形式化特性却使之难以承受严格推理，缺乏可执行性，被阻隔于自动测试之外[4]。而且，UML 语言可从多个方面为描述对象进行建模，所涉及的图形种类繁多，在设计阶段将得到一个反映目标 Web 应用各种特性的图形集。这样虽然为用户提供了多重视角，使得模型更为丰富与全面，但也给测试工作带来了麻烦，需要为各种 UML 图形确定其对应生成测试用例的方法，既牺牲了效率也不利于测试过程中各种图形之间信息的互补与组合。

鉴于此，我们可以使 UML 模型适应自动产生测试用例的要求，这便是要确定其中哪些图形对产生测试用例有效，并将它们统一形式化并转化成为最有利于测试用例自动生成的模型。这样，只需分别为各图形给定一个形式化为目标模型的方法，便可共享针对目标模型的测试用例自动生成方法、测试用例集的优化方法、测试用例覆盖度与结果评估方法等。

FSM 模型即为本课题组选用的目标模型。首先，FSM 是一种形式化的建模语言，它能精确且无二义地描述软件应提供的服务及其功能。其次，FSM 模型的结构与描述方法决定了其高度自动化的操作特性[5]，深厚的相关理论基础大大有助于测试并使之更为有效。并且，除了传统软件，FSM 亦可用以为 Web 应用的行为进行建模，而测试用例则通过遍历其相应 FSM 模型的路径自动产生，每一条相异于其它的路径便可视作一个测试用例[6]。另外，FSM 模型的可

视化图形可直观地展现每个测试用例的路径走向，因为基于状态的规格说明语言本身具有图形形式，转化成为图形是可行的，也并不困难[5]。最后，作为保证软件质量的手段之一，模型检验用以验证设计阶段的模型是否满足用户所要求的若干性质，尤其对于基于模型的测试是十分有必要的前期工作，因为未经验证的模型无法保证由其产生的测试用例的正确性，且模型检验可以和已有的各种测试方法结合以实现测试用例生成的自动化，它产生的反例可以被解释为测试用例[7]。模型检验工具的输入模型通常为 FSM 模型，此种转换也可实现模型检测的模型源获取及自动化。但是，与许多形式化语言一样，正是 FSM 模型与数学的紧密联系，使之需要用户掌握丰富而扎实的相关知识，于是，造成了学术界广泛使用但工业界少人问津的局面。因此，先用工业界统一标准的建模语言 UML 描述系统框架与功能，再将它形式化成为 FSM 模型便能够同时享有两种模型特性带给我们的便利：UML 建模语言的简单易懂与 FSM 建模语言的精确可推导，这是本文将研究的主要问题之一。

尽管我们可以通过模型形式化获得的统一的 FSM 模型进行测试用例的自动生成与模型检验工作，感受到它许多方面的优势，但是，正如文章开头说到的那样，包括 Web 应用在内的软件产品呈现出扩大化复杂化的趋势，待检测系统的规模之大常常使其对应的 FSM 模型包含过多信息，面临状态爆炸的窘境，可能会产生难以完成的测试量[8]和模型检测工作。所以，对于大型软件产品，尤其是 Web 应用软件的 FSM 模型的归约优化将大大改善这一状况，删去其中无关于指定范围内的各项元素，减少人力物力，使测试与模型检验工作只针对用户要求检测的系统功能而展开。这也是本文将要研究的主要问题之一。

无论是模型形式化、模型归约、测试用例的自动生成还是模型检验，它们涉及到的 FSM 模型信息均以文本文件作为载体进行传递，因为以 FSM 模型为输入的功能模块只能从文本中抽取必要的信息。测试者或用户若想了解检测空间内的状态或路径，需要花费相当长的时间熟悉语法规则、阅读文档，且由于信息量过大，只能在脑中获得一个模糊的印象。上文提及，FSM 模型具有图形式的表示形式，且测试用例是 FSM 模型上的路径，为了方便用户更快、更直观地掌握测试信息，必须充分利用人类对图形的敏感与理解能力，实现 FSM 文本

的可视化。由于在模型形式化过程中，需要借用层次有限状态机模型以存储中间结果，我们对可视化对象的范围进行进一步延展，使其既能作用于无层次信息的 FSM 模型的文档，也可作用于由有限个附有层次信息的 FSM 模型组成的层次有限状态机模型的文档。

1.3 国内外研究概况

1.3.1 国外研究概况

测试自动化已成为软件工程领域内的一个研究热点，它简化了软件开发过程中占据时间最长的测试环节。但现在大多数技术都采用"捕捉 / 回放"机制，这些技术需要耗费大量的时间与人力以记录测试场景，且当系统用户界面或功能设计稍有变动时，所有的测试工作又将重新进行。基于此种机制开发的工具必须依靠人的参与才能设计并产生测试用例，且无法提供关于测试用例覆盖度的任何信息。而由于 Web 应用的大规模与复杂性要求针对其设计的测试对自动化程度要求更高，这类工具则更少。现有的大部分工具[9]不支持 Web 应用的功能测试，例如用于验证 Web 应用软件链接信息的 Link Checks；HTML Validators，依照用户指定的某种标准 HTML 格式的语法对 Web 应用软件代码进行检查；Web Site Security Test Tools，对于 Web 应用软件的安全性进行验证；还有用于检测 Web 应用软件负载限度与性能参数的测试工具 Load and Performance Test Tools 等等。以上绝大部分工具只局限于静态方面的检测，而信息来源均为 Web 应用软件的代码，意味着待系统开发完毕才能开始测试用例的生成工作，所以工业界需要一种能在最小代价下，帮助用户完成 Web 应用行为检测的自动化工具。

随着面向对象和模型驱动概念的出现与盛行，软件产品基于模型的测试方法在业界引起了广泛关注。虽然许多专家学者致力于该领域的研究，且获得了一些进展，但其理论与实际应用尚存在差距，由于系统性的缺乏与低自动化程度，这些理论指导下开发的工具均无法在工业界代替传统测试手段[10-20]。

创建模型是基于模型测试方法的首要步骤。对于 Web 应用，常用的建模方

法是利用实体关系图或 UML 类图描述 Web 页面结构及各页面间的关系。Isakowitz 等人提出一种称为关系管理方法论的方法对 Web 应用进行建模[21]；Coda 等人设计了一个在更高抽象层次对 Web 应用进行建模的模式，称为 WOOM[22]；Gellersen 等人引入一种 WebComposition 标记语言，用以运行 Web 应用开发时期创建的称为 Web Composition 的模型[23]；Coallen 等人扩展了 UML 建模语言，使之能够根据 Web 应用的固有特征对它们进行更方便更准确的描述[24]。文献[25]提出了一种用 GFSM（Guarded FSM）对 Web 浏览器的交互行为进行描述的方法，它针对 Web 浏览器的独特性质对 FSM 进行扩展，使之能够更为精准地对 Web 浏览器建模。然而这些方法大部分不涉及 Web 应用的行为及功能，而且与它们配套的测试方法也没有给出。UML 模型确实可以为 Web 应用的动态方面进行建模，也有不少通过已有 UML 模型进行测试用例自动生成的成果，但往往因为 UML 语言的二义性需要人工干预才能完成。由于 UML 的半形式化特性阻碍了测试自动化，目前为止，还没有完整支持工具体系的报道。另外，仅仅使用 UML 模型描述 Web 应用的初衷只是研究测试方法，却忽略了模型检验对模型的要求。另一方面，学术界提出了许多从形式化模型产生测试用例的方法。文献[26]设计了一种用于产生测试用例的面向对象的 Web 应用测试模型，它由对象关系图、页面导航图、对象状态图、段落分支图及功能簇图组成，只是一旦 Web 应用的对象过多，模型也会过于复杂；Ricca 等人通过对每个 Web 页面进行建模来达到描述整个 Web 应用的目的，并提出相应准则以产生测试用例，但只有处理简单 Web 应用时才适用[27]；[6]通过 FSM 模型对 Web 应用进行建模，然后依照不同的测试准则，搜索模型中的不同路径以得到测试准则要求的测试用例集。这一方法充分利用了 FSM 模型的优势，简化、精确化并极大程度地自动化了测试过程，成为研究的新方向。

形式化模型固然有种种非形式化模型所不能替代的优点，但直接利用它们进行 Web 应用建模仍有一定困难。为了增强相关工具的通用性，我们采用的最初模型依然是 UML 模型，因为它已成为工业界广泛认可的标准语言，能够强有力地辅助用户对复杂的 Web 应用软件进行建模。而要避免陷入计算机无法自动识别模型的危机，就必须对已创建的 UML 模型进行形式化，学术界在这方

面已颇有些研究成果。Precise UML group[28]是一个由领域内各国学者组成的团体，是 UML 形式化的主力部队之一，他们致力于使用如 Z 或 VDM 等面向模型的标记为 UML 建模语言提供精确的语义。另外，还有其它研究成果，Boreges 等人[29]将 UML 类图与形式化规格说明语言 OhCircus 结合，用 OhCircus 来描述各种 UML 元素；Latella 等人[30]将 UML 状态图形式化成为语言 Promela；Traore 等人[31]提出一种将 UML 状态图形式化成为 PVS 的方法，方便模型检测的自动化过程；文献[32,33]提供了一种 UML 顺序图的基于形式化轨迹的语义；文献[34,35]根据 Pomset 框架方法给出了 UML 顺序图的形式化语义；文献[36]利用高阶逻辑表示一些UML标记的形式化语义，并将UML顺序图语义编码为PVS；[37]提出 UML 顺序图的一种形式化语义用于模型一致性检测功能的开发。但是，将 UML 模型形式化为 FSM 模型的研究却很少。Erich 等人[38]给出了一种层次有限状态机模型，可用于存储 UML 状态图的信息，也包括层次信息，而这种层次性却没有进一步的方法加以去除，无法得到真正的 FSM 模型。鉴于 UML 状态图实际上是建立在 FSM 相关理论基础上的，除此之外的 UML 图到状态图的转换的已有方法也为其它 UML 图的形式化研究提供了很好的参考。Jon Whittle 等人[39]提出一种通过添加语义信息，自动将一组 UML 顺序图转换成为 UML 状态图的方法；文献[40]引入一种自动转换 UML 模型的框架，它借助于重写逻辑的形式化描述方法。

在对 FSM 模型进行归约优化的领域中，由于 FSM 自身发展历史悠久的理论基础，该模型的归约与约简也早已成为研究的重点[41-44]。提出的方法大部分集中于去除 FSM 模型中的冗余信息，避免重复记录的现象发生，而我们则希望根据用户需求，仅提取原系统 FSM 模型中描述指定功能的那一部分。这一机制与在测试意图或测试指示指导下产生测试用例比较类似，可将归约指示（Reduction Directive）对应至测试意图或测试指示，将归约 Web 应用模型对应至归约测试用例。对前者给予关注与了解将大大推进后者的研究进度，使我们站在已有成果的基础上继续向上攀登。文献[45]扩展来源于 LOTOS 规格说明的迁移标记系统，并阐述了一种根据其相关覆盖信息达到测试套件最小化的方法；Hsiou-Wen Hsueh 等人[46]提出了一种新型的基于启发式逻辑编程的测试指示生

成方法，并利用生成的测试指示产生相应的测试用例集；在文献[47,48]中，测试指示中的元素首先以对象图和状态图表示，当它用于约束目的测试用例集的大小及其它方面时，它将通过为每个 construct 单元添加相应 IF 片断被赋予有效语义。以上所述大部分研究均假设测试意图或测试指示已构建，并以某种模型的形式存在，且旨在说明最终生成的测试用例所需满足的条件，从而在原测试模型的基础上获得更小的测试用例集。我们要解决的问题，却是如何有效构建代表指定功能的归约指示模型，最终获得更小的 Web 应用行为模型，测试用例的产生与模型检验则被划作后续工作。

对于 FSM 模型的可视化，已有一些图形化工具，它们以自己定义的无语义的图形描述语言为输入，产生相应的图形，如 Graphviz 等。另外，德国的 Axel Schneider 和 Stephan Walter 提出一种将描述有限状态机的表格式说明语言 ADeVA 转化成图形表示形式的方法[49]，而微软公司则自定义了一种抽象状态机语言 Asml，并开发了名为 Spec Explorer 的测试工具，可将该语言的文本转化成为相应的有限状态机的图形表示[50-53]。但也由于其特定输入语言并未广泛被工业界接受与采纳，直接用于实际 FSM 模型文本的可视化尚有困难，我们选择的则是已成为国内外行业标准的 XML 标记语言，它的发展前景将因此变得更好。当然，虽然都与项目采用的 FSM 模型语言不同，但大体方向一致，仍有许多可借鉴之处。另外，上述工具均只能作用于无层次信息的 FSM 模型文档，但本文提出的方法对于层次有限状态机模型的文档亦有效。

1.3.2 国内研究概况

国内数所研究机构也有学者对形式建模或基于模型的测试进行了研究，并取得了一些进展，但都是比较初步的，也未见系统级研究的报道。

在建模相关方面，UML 模型之间的相互转换及 UML 模型形式化为 FSM 模型已有部分专家进行了研究，并取得初步进展。文献[54]提出了一种将 UML 顺序图转换为多个 UML 状态图的方法，它针对每个描述对象的具体运行机制自定义一个向量，通过向量中各项的不同赋值区分最终得到的状态图的各状态。向量中各项均为系统变量，它们分别取值后的组合可以唯一地表示系统所处状

态。但是，对于每个具体的 Web 应用，这种方法都必须人为设定该向量的组成项及最终得到的状态图中各状态所对应的赋值向量，使得测试自动化无法实现，而且归约指示与测试用例只需要 FSM 模型中的迁移信息，状态只是为了迁移的存在而存在，故其详细含义可忽略。文献[55]提出了一种通过时间自动机来形式化带有时间扩展的 UML 状态图的方法，该过程分为两部分完成：层次状态图的平面化以及时间化自动机的构造。但是，它的应用对象仅限于由最基本元素组成的 UML 状态图，我们则将范围扩大化，使常用的一些特殊元素也能够有对应的算法进行相应转换，如 **completion 迁移、fork、join 和历史状态结点**等，提高其通用性。

关于在归约指示的指导下进行模型归约优化的研究很少，但有一些基于场景归约的构件式系统分析方面的探索对本课题的研究较有帮助。文献[56,57]使用接口自动机及接口自动机网络来描述构件式系统的行为设计模型，使用 UML 顺序图表示基于场景的需求归约，对系统设计阶段的构件交互行为的动态兼容性进行形式化分析和检验，通过对接口自动机网络状态空间的分析，给出了一系列算法以检验系统行为的存在一致性以及几种不同形式的强制一致性性质。而我们归约后获得的 Web 应用模型亦为自动机，选用的归约指示初始描述语言亦为 UML 顺序图，这些相似之处使得汲取该成果中部分思想为我们自身所用成为可能。

1.4 论文的主要研究内容

本论文是以作者攻读硕士学位期间承担课题的工作为基础，在第一章中阐述了课题研究的来源、目的、意义以及国内外研究的现状。

第二章介绍相关概念，其中一部分是理解本文研究内容的目的与背景时必须了解的知识，例如基于模型的测试等；另一部分是研究内容中将要运用到的理论，是后续理论的基础，例如 UML 建模语言、UML 状态图、UML 顺序图、FSM 模型等。

第三章提出并详细说明了将 UML 模型形式化为 FSM 模型的方法。UML 建模语言包含数种图形，以描述系统的不同方面，由于本课题的目的在于测试

Web 应用的行为，故集中于对其行为模型的研究。该章的重点在 UML 状态图的形式化上，因为它是最常用以建模系统中单个对象的行为，记录系统最小单位的生命周期。顺序图的形式化将在第四章阐述，它将作为归约指示而存在。

第四章提出了一种优化系统 FSM 模型的方法，即在归约指示的指导下归约系统 FSM 模型，得到的归约模型出原模型中足够描述归约指示指定的功能的所有元素组成，并将原模型中的不影响指定功能描述的元素全部去除，以达到优化的目的。该方法大致可分为两步：首先，对非形式化的测试指示进行形式化，因为语义的不精确性将影响计算机自动读取分析测试指示及其对原系统 FSM 模型的约束；然后，便可在上一步中得到的形式化归约指示的指导下，根据该章提出的归约算法将原系统模型的规模减至最小。

第五章以层次有限状态机模型文档可视化的实现方法为主要内容，也介绍了前两章中所述模型形式化与模型优化方法的实现方案，并给出部分模拟运行结果，以说明本文所提出理论及方法的可行性。层次有限状态机模型的可视化是以某种格式的文本为输入，以其对应模型的图形形式为输出，大致分为文本的获取、信息抽取、信息处理与图像输出四个子过程。我们分别针对这四个子过程提出算法，最终实现层次有限状态机模型文本向图形的转换。而第三和第四章中所阐述方法的实现，要求所涉及模型拥有计算机可识别的载体与用于相关信息存储的数据结构，这些都将在该章提及。

最后，第六章总结全文，对本文所做的全部研究工作进行归纳与分析，指出有待完善之处，并对该领域未来研究方向阐述了本人的观点。

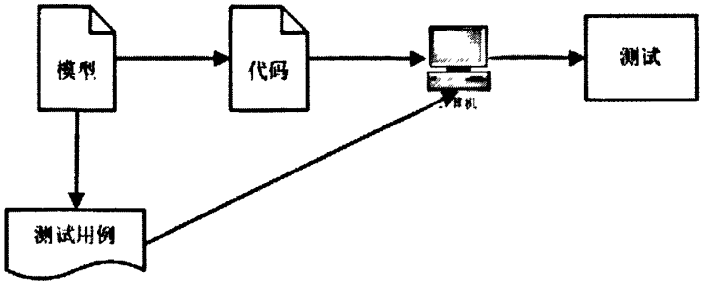
第二章　理论基础与基本概念

2.1 基于模型的测试

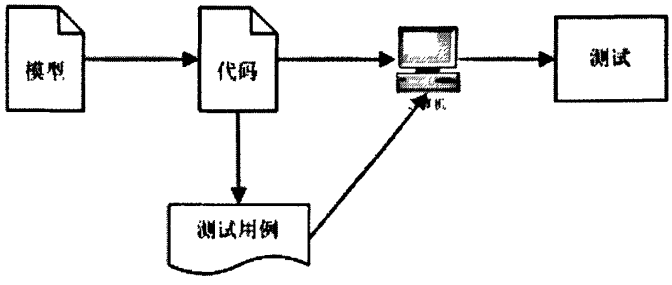
软件测试方法主要可分为白盒测试与黑盒测试，前者利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试；后者则不必考虑程序内部的逻辑结构和内部特性，依据需求规格检测系统行为是否符合预期。本课题旨在 Web 应用的功能测试，故采用黑盒测试，减少其代码复杂性带来的测试困难。但传统方法的自动化程度并不尽如人意。基于模型的测试是一种突破性的黑盒测试技术，它以明确描述系统预期行为的抽象模型为依据，自动生成可执行的测试用例，自动地产生测试脚本，执行测试并自动评价测试结果，实现测试过程的完全自动化。

模型是其建模对象的一种抽象蓝本，它可以从结构或行为角度分别对系统的组织模型和动态行为方面进行描述。由于模型可以帮助我们更好地理解系统，更重要的是其在用户和开发人员之间的桥梁作用，建模工作应该在编程前完成，以使产品最大程度地满足用户的需求。而基于模型的测试方法正是在此阶段创建的模型的基础上，自动产生测试用例。因此，测试用例的生成被提前到设计建模之后，无需等待产品代码全部完成，从而将整个测试周期缩小，这样也可以有更多的时间制定更为有效的测试计划并更充分利用资源[5]。该方法的大致流程如图 2.1a 所示，图 2.1b 则是传统测试方法的流程图，两者的差别显而易见。

本课题组采用基于模型的测试方法思想对 Web 应用进行测试，虽然它让我们离自动化测试更近了一步，但测试模型的选取依然直接影响到自动化程度的高低，因为此模型将是自动用例产生的唯一信息来源，如果缺乏语义精确性，则不可避免地在测试过程中加入人工分析。所以本课题组的研究方向是先将初始需求与设计模型转换成为形式化的测试模型的测试方法，其具体细节介绍如下。首先，由开发人员根据用户需求手工创建 UML 模型，得到用以全面描述待实现软件产品的图形集。然后，根据各图形相应的映射规则将其元素转换成



图(a) 基于模型的测试方法



图(b) 基于代码的测试方法

图 2.1 两种测试方法的流程图比较

为目标模型中的元素，从而转换成为目标模型。至此，开发人员将得到多个目标模型，接下来，可以选取单个目标模型本身或对其进行归约后得到的模型作为测试模型，也可以将几个目标模型以某种机制进行组合来获得测试模型。最后，在自定义的测试准则的约束下，产生抽象测试用例并用于测试。

2.2 UML

统一建模语言（UML）是用来对软件密集系统进行可视化建模的一种语言，它为面向对象开发系统的产品进行说明、可视化和编制文档。由于其简单易用、便于理解且能全面描述系统的各个方面，已被广泛使用并被 OMG 组织采纳作为业界的标准。作为一种建模语言，UML 专注于从不同角度建立软件产品的模

型和结构，对于具体的语言及算法并不涉及，这使开发人员可以在较高的抽象层次上整体把握待开发系统的组织结构及行为等；作为一种可视化模型，UML通过图形直观地将信息传达给开发人员或用户，帮助他们更好地理解系统结构与功能，促进沟通，最大程度地保证与用户需求的一致性。

UML中有多达十几种图形供建模人员选择，分别满足不同角度、不同建模方式的不同需求，其中最常用的有类图、对象图、用例图、顺序图、通信图、活动图和状态图。类图和对象图一般描述系统中对象间的关系或系统层次关系，为系统的静态特性建模；用例图、顺序图等其余图形大多描述对象间的交互机制或单个对象的生命活动周期，为系统的动态特性建模。由于本文针对的测试系功能测试，即在预定输入或事件触发下，捕捉与预期不符的动作或输出，测试信息的大部分来源是行为模型。

作为行为模型之一的状态图用以描述某个对象的生命周期，从它被创建或初始化开始，直到被撤销或回到初始状态。几乎所有系统都包含一个以上的对象，因而状态图被认为是描述系统行为的最小单位，所以，我们的模型形式化研究工作也从状态图开始入手。另外，在第四章的模型优化过程中，考虑到前期模型形式化的源模型是UML，归约指示亦采用UML语言描述。归约指示要求表明待检测功能，而UML顺序图正是以场景的形式呈现系统执行某种功能时的行为，用它建立归约指示十分合适。下面便是上述两种图的介绍。

2.2.1 状态图

状态图的基本元素有状态和迁移，对象当前所处状态会在某种事件的作用下发生改变，并做出相应的动作。

每个状态都有一个区别于其余状态的名字，代表着对象在特定条件下所处的状态。在进入或离开某个状态时，一般都会有系统的某个反应动作被执行。没有内部结构的状态被称为简单状态，不是简单状态的状态均被称为复合状态。一个复合状态的子状态有两种组织形式：以顺序关系形成子状态机、以并发形式组成子状态机，前者称为非正交子状态，后者则称为正交子状态。一个只包含非正交子状态的复合状态称为或－状态，当对象处于该状态时，任何时候都

有且仅有一个子状态被激活；而一个包含正交子状态的复合状态称为与一状态，在此引入域的概念，代表其中各个子状态机，这些子状态机以并行关系模式运行，当对象处于与一状态时，该状态中同一层次的每个域中有且只有一个子状态被激活[58]。另外，初始状态与最终状态也是比较特殊的状态。前者代表整个状态图或某个域的起点，后者则表示整个状态图或某个域的运行结束。由于每个域中均有一个最终状态，只有当同一层次的所有域都到达其相应最终状态时，对象才能离开它们上一层的与一状态。还有一种特殊状态也经常会被用到，这就是历史状态，它一般位于一个或一状态中，可记录非正常离开该或一状态时处于激活状态的子状态。当意外事件完成，回到该或一状态时，根据历史状态的记录直接进入断点，继续子状态机的运行。

迁移代表两个状态间的转换关系，由单向箭头表示，箭头指向的状态为目标状态，另一个则为源状态。当两个状态间存在迁移时，意味着当对象处于源状态时，在某个事件的作用下将转入目标状态。因此，一个迁移 ***t*** 一般由五部分确定：源状态，受到迁移转换作用的状态，本文中用 ***src(t)*** 表示；目标状态，迁移完成后对象进入的状态，本文中用 ***dst(t)*** 表示；触发事件，触发迁移启动的事件；保护条件，一个衡量迁移是否可被触发的布尔表达式；动作，当迁移被触发后对应对象对其它对象的操作或作用。后三个部分的组合称为标签，本文中用 ***EGA(t)*** 表示。在特殊情况下，触发事件可以为空，这类迁移被称为 Completion 迁移，只要其源状态的动作完成，被可自动触发该迁移，使对象进入对应的目标状态。

当源状态是复合状态，离开它的迁移被触发时，意味着对象将先从当前活跃的子状态离开，再离开该复合状态。如果不是 Completion 迁移，则该迁移的触发事件将打断子状态机的正常操作流，强行从当前活跃的所有子状态退出，统一进入目标状态。如果是 Completion 迁移，则待子状态机均到达最终结点后才统一离开源状态。当目标状态是复合状态，到达它的迁移被触发时，意味着对象将在进入该状态后，自动同时进入其各子状态机的初始状态。UML 还提供了两种特殊的迁移用以描述源状态或目标状态大于 1 的情况。名为 ***fork*** 的迁移可指定当其经触发，并到达某与一状态后，将直接进入各域中的哪个子状态。

对于未被指定的域，即不包含 ***fork*** 目标状态的域，仍然在 ***fork*** 被触发后自动进入这些域的初始状态。与此类似，名为 ***join*** 的迁移可指定，当其经触发，并离开某与一状态时，将直接从各域中的哪个子状态离开。若触发事件发生时各域并未处于指定子状态，则迁移将不能被触发，若符合触发条件，则将强行使对象离开各指定子状态。对于未被指定的域，即不包含 ***join*** 源状态的域，仍然在 ***join*** 被触发后强行离开这些域当前激活的子状态。

2.2.2 顺序图

顺序图是交互图的一种，但不同于通信图，它更注重于体现对象间消息传递的时间顺序，但并没有严格的时间限制。顺序图的建模对象是系统中指定对象的交互行为所构成的场景，从而反映与此场景相对应的系统功能。

图形的顶端的水平线上有几个带有文字的矩形框，它们分别表示参与该顺序图所描述场景的各个对象。每个对象下方有一条与其对应的矩形框垂直的直线，这是该对象的生命线，表明其在某个时间段存在或不存在。对象间的交互以传递消息的形式展现，每个传递消息的动作由一根带箭头的线段表示，它的起点是消息发出对象的生命线的某一点，箭头指向的则是消息接收对象的生命线的某一点，消息的内容写于线段之上。传递消息的动作的先后顺序由该动作对应的带箭头线段在图中所处的位置决定。可把所有动作在垂直方向上作比较，位置越靠上的动作则发生的时间越早。

2.3 FSM 模型

有限状态机（FSM）模型是描述系统动态特性的模型，它由一组状态及状态间转换的规则所组成。这些规则限定，当系统处于某些状态下，经由内部参数变化或外界刺激可使系统运行至另一个状态。而有限状态机模型所描述的任何一个运行均需从一个称为开始状态的状态出发，并在条件改变及规则的驱动下一直变换状态，直至到达称为接受状态的状态为止。

第三章 UML 模型的形式化

在上一章相关概念与理念的基础上，便可进行本文主体研究内容的展开。为了获得更适合测试与验证的模型，我们需要对需求分析与设计阶段创建的初始模型制定一系列变换与约简方案，以最大程度地支持后续工作。本章是整体处理过程的第一步，旨在直接对原始模型进行形式化，即由 UML 模型到 FSM 模型的转换，从而提高测试的自动化程度并为模型检验提供模型来源。

3.1 模型的表示形式

在进行模型形式化之前，首先要确定源模型与目标模型的表示形式，因为模型形式化机制本质上是从源模型元素到目标模型元素的映射规则，只有在此基础上才能设计相应算法。而如前文所说，本章模型转换的意义主要是模型的形式化及统一化，所以，目标模型应达到精确性与可自动操作的要求。在此给出 FSM 模型的形式化表示[59]：

定义 3.1 一个 FSM 模型 A 是一个五元组（Q，L，δ，q_0，q），其中 Q 是 A 上所有有限个状态的集合；L 是 A 上所有有限个迁移标签的集合；$\delta: Q \times L \rightarrow Q$ 是 A 上所有有限个迁移的集合，它为 A 上所有具有转移关系的两个状态建立联系，表示前状态通过标签中给出的事件触发将使系统进入后状态；$q_0 \in Q$ 是 A 的开始状态，$q \in Q$ 是 A 的接受状态，即终态，系统进入该状态时将停止接受任意事件触发。

假设存在可表示成为（s，l，s′）的迁移 t，且 t 满足 $t \in \delta$，则有如下表达式 source(t) = s，target(t) = s，label(t) = l 成立。

从 FSM 模型的数学定义可以看出，其中的状态均为基本状态，即无嵌套状态，不存在子状态与子 FSM，而 UML 状态图中会出现复合状态，这便说明在模型转换过程中，去层次化是必需的。而为了不丢失源模型的语义，必须先获取状态图中各状态之间的层次关系，再利用适当的模型转换规则最终生成无层次的 FSM 模型。所以，我们需要一个中间模型，以记录目标模型中无法保留的

层次信息，这便是层次有限状态机 HFSM。层次有限状态机不同于有限状态机，它的状态可以有内部结构，我们可以将它看作许多个 FSM 模型以串行和（或）并行方式组合而成的复合模型，非最底层的状态可由其直接下层的子状态机表示，其定义需要在函数ϕ的基础上给出[32]。

定义 3.2 给定一个有限状态机的集合 $F = \{A_1, A_2, \ldots, A_n\}$，$Q(A_i)$表示集合中任意有限状态机 A_i 的状态集合，则

$\phi: \bigcup_{A \in F} Q(A) \rightarrow P(F)$是集合 F 上的一个组合函数当且仅当

$-\exists_1 A \in F \wedge A \notin \bigcup \mathrm{ran}(\phi)$

（其中 A 是最高层次的有限状态机，可表示为ϕ_{root}，root 是默认设定的层次有限状态机的最高级状态，它对应着 A 满足此表达式的有限状态机是顶层 FSM）

$-\forall A \in \bigcup \mathrm{ran}(\phi) \bullet \exists_1 s \in \bigcup_{A' \in F \setminus \{A\}} Q(A') \bullet A \in \phi(s)$

$-\forall S \subseteq \bigcup_{A \in F} Q(A) \bullet \exists s \in S \bullet S \cap \bigcup_{A \in \phi(s)} Q(A) = \varnothing$

定义 3.3 层次有限状态机模型（HFSM）是一个二元组（F，ϕ），其中 F 是一个有限的有限状态机集合，对于每两个顺序有限状态机 A_i, A_j（$0<i<n, 0<j<n, i \neq j$）都有 $Q(A_i) \cap Q(A_j) = \varnothing$，$\phi$是集合 F 上的一个组合函数。设 S 为该 HFSM 中的任意状态，若$\phi(S) \neq \varnothing$，则 S 为复合状态。

在这些模型定义的基础上，我们便可开始模型形式化方法的描述。

3.2 模型形式化机制

3.2.1 层次结构信息的获取与存储

正如前文中所提到的，由于本课题的目的在于测试 Web 应用的行为，故集中于对其行为模型的研究。而本章的重点在于 UML 状态图的形式化机制。

UML 状态图的复合状态与 FSM 模型的单一结构表明两者在层次结构上有着很大的区别。虽然最终得到的模型为了更有利于模型检测与测试用例的自动产生过程，将层次结构全部去除，但在模型转换之前仍需获取并存储源模型关

于层次结构的信息，以最大程度地求得前后模型的语义一致性。因此，我们利用层次有限状态机模型记录下 UML 状态图的拓扑结构，因为 HFSM 提供了一种简便而精确的描述的方法。

根据 HFSM 的定义，由 UML 状态图的拓扑结构信息得到一个有限状态机集合上的组合函数后，便可以利用 HFSM 数学形式表示该拓扑结构。而组合函数的建立过程自最顶层的有限状态机开始，逐层向下，将每层的复合状态映射到其对应的子 FSM，然后将其作为一个元素添加到组合函数中。若复合状态 s 是一个或一状态，它对应的子 FSM 为 A_i，那么就有$\phi(s) = A_i$且 $F = F \cup \{A_i\}$；若复合状态 s 是一个与一状态，它的各个域对应的子 FSM 分别是 A_1，A_2，...，A_n，那么就有$\phi(s) = \{A_1, A_2, \ldots, A_n\}$且 $F = F \cup \{A_1\} \cup \{A_2\} \cup \ldots \cup \{A_n\}$。状态图中每一个被初始状态指向的状态是其所在层次的对应 FSM 的开始状态，而指向最终状态的状态则转换成为其所在层次的对应 FSM 的接受状态。

这样，原 UML 状态图的拓扑结构便完全由其对应的 HFSM 模型存储起来，在这个 HFSM 的基础上，便可获取原 UML 状态图中任意两个状态间的层次关系，这个层次关系可由函数χ表示。

定义 3.4 给定一个层次有限状态机 HFSM（F，ϕ），函数χ：

$$\bigcup_{A \in F} Q(A) \rightarrow P\left(\bigcup_{A \in F} Q(A)\right)$$

$$\chi(s) = \{s' \mid \exists A \in F \bullet A \in \phi(s) \wedge s' \in Q(A)\}$$

于是，UML 状态图的图形信息不仅形式化为数学表达形式，而且可以利用自定义函数判断状态间的关系并得到任意状态的所有子状态及其父状态。

3.2.2 模型的形式化

拓扑结构的数学模型的形式化描述为模型形式化提供了重要的层次信息，在它的帮助下，我们便可以从原 UML 状态图的迁移元素开始，进行目标 FSM 模型的生成工作，先引入一些后面要用到的基本概念。

定义 3.5 给定一个层次有限状态机 HFSM(F, ϕ)，满足条件 $C \subseteq \bigcup_{A \in F} Q(A)$ 的状态集合 C 是它的一个格局（Configuration）当且仅当

$-\exists_1 s\in Q(\phi_{root})\bullet s\in C$

$-s\in C\wedge A\in\phi(s)\Rightarrow\exists_1 s'\in Q(A)\bullet s'\in C$

$-s\in C\wedge\exists\, s'\bullet s\in\chi(s')\Rightarrow s'\in C$

定义 3.6 给定一个层次有限状态机 HFSM（F，ϕ）和它所有格局的集合 C，对于其任意一个状态 s，函数 config：

$$\bigcup_{A\in F} Q(A)\rightarrow P\left(\bigcup_{A\in F} Q(A)\right)$$

$$config\,(s)=\{\,c_i\mid c_i\ \subseteq C\wedge s\in c_i\,\}$$

定义 3.7 给定一个层次有限状态机 HFSM（F，ϕ），其任意一个状态 sd 的缺省格局（Default Configuration）可表示为函数 deconfig：

$$\bigcup_{A\in F} Q(A)\rightarrow P\left(\bigcup_{A\in F} Q(A)\right)$$

$$deconfig\,(sd)=X\Leftrightarrow\exists_1 X: config\,(sd)\bullet$$

$$\forall s\bullet(\,s\in X\wedge sd\notin\chi^*(s)\Rightarrow\bigcap q_0\,(\phi_i(s))\subseteq X\,)$$

定义 3.8 给定一个 UML 状态图及其任意一条迁移 t，Uexit 是指状态集合 $exit=\{exit_i\mid\forall j: N\bullet src_j\,(t)\in\chi^*(exit_i)\wedge dst_j\,(t)\notin\chi^*(exit_i)\}$中层次最高的状态，Uenter 是指状态集合 $enter=\{enter_i\mid\forall j: N\bullet src_j\,(t)\notin\chi^*(enter_i)\wedge target_j\,(t)\in\chi^*(enter_i)\}$中层次最高的状态。

由以上定义可知，一个格局实际上是由层次有限状态机中的 N（$N\geq 1$）个状态组成，它代表着某一时刻系统中所有被激活的状态。所有格局都含有层次有限状态机顶层 FSM 的一个状态，并且，若某复合状态存在于某格局中，则其对应的各 FSM 中均有一个状态存在于该格局中。函数 config 是从任意状态 s 到所有包含 s 的格局集合的映射，函数 deconfig 是从任意状态 sd 到包含 sd 的缺省格局的映射。Uexit 表示任意迁移 t 离开的所有状态中层次最高的状态，则表示任意迁移 t 进入的所有状态中层次最高的状态。

目标 FSM 模型的每个状态均是原 UML 状态图对应 HFSM 模型的一个格局，那么迁移便是触发一个格局到另一个格局的变化，由于格局是 HFSM 模型中数个状态的集合，所以原 UML 状态图中的每个迁移将可能被映射为目标 FSM 模型的数个迁移，而此数量由原 UML 状态图中的该迁移的源状态所属的格局

数量决定。我们设计了一个算法，假设 confTranSet 为目标 FSM 模型的迁移集合，该算法可在前期工作得到的原 UML 状态图拓扑结构数学模型的基础上计算出集合 confTranSet。

算法 3.1

对于原 UML 状态图中的每一个迁移 t

if EGA(t) = ∅

TempSet = $\bigcap$ q (ϕ_i (src (t)))

for each $q_i \in$ TempSet

$config_i$ = config (q_i)

ConfSet = $\bigcap$ $config_i$

DefConf = deconfig (dst (t))

if t is a ***join***

for each $s_i \in$ src (t)

$config_i$ = config (s_i)

ConfSet = $\bigcap$ $config_i$

DefConf = deconfig (dst (t))

if t is a ***fork*** $\wedge$ | dst (t)| > 1

ConfSet = config (src (t))

defDst = $\bigcup$ (deconfig (dst_i (t)) $\cap$ χ^* (dst_i (t)))

NdefDst = $\bigcap$ (deconfig (dst_i (t)) \ χ^* (dst_i (t)))

DefConf = defDst $\cup$ NdefDst

else

ConfSet = config (src (t))

DefConf = deconfig (dst (t))

while (ConfSet is not empty)

get a souconf $\in$ ConfSet

tarconf = (souconf \ χ^* (Uexit (t)) $\cup$

$(\chi^{*}(\text{Uenter}(t) \cap \text{DefConf})$

source (t') = souconf

target (t') = tarconf

label (t') = EGA (t)

confTranSet = confTranSet $\cup \{t'\}$

confSet = confSet\{souconf}

在根据算法 3.1 构建出目标 FSM 模型的迁移集合 confTranSet 后，便可将所有与 confTranSet 中迁移相关的状态置于一个集合内，这便是目标 FSM 模型的状态集合。而其初始状态 InitState 和接受状态 AccState 则可由以下表达式确定：

InitState = deconfig (q_0 (ϕ_{root})) AccState = config (q (ϕ_{root}))

上述方法基本上将原 UML 状态图的迁移与状态映射为目标 FSM 模型的基本元素，但还有一种状态没有考虑到，这就是历史状态。由于历史状态与普通状态的语义差异很大，故不在算法中提及，而是为其专门设计一种转换方法，并于模型形式化的最后阶段执行。

对于每一个历史状态 h，若它属于或一状态 Ors，HS 是 Ors 的所有非正交子状态的集合。通过历史结点的语义可以知道，它旨在记录对象离开 Ors 状态时正处于激活状态的子状态，而进行返回操作后不必再重新由 Ors 的开始状态开始执行，而是直接进入上次离开时记录的子状态。所以，应针对所有以 Ors 为源状态的迁移，建立 Ors 的子状态与这些迁移目标状态的新转换关系。假设一个以 Ors 为源状态的迁移的目标状态为 Htar、标签为 l，对于每一个 $hs_i \in HS$，新建一个标签为“back（hs_i）”的迁移，它的源状态为 Htar，目标状态为 hs_i。一旦将这些新迁移都给出，问题就转化成为运用算法 3.1 对它们组成的迁移集进行处理，得到一个对应于目标 FSM 模型的迁移集，然后与前期得到的原目标 FSM 模型的迁移集合进行合并，用以建立更为合理的目标模型迁移集合。但这并不是最终的迁移集合，因为以 Ors 为源状态的迁移需记录它被触发时活跃的子状态。所以，对于每一个以 Ors 为源状态的迁移 t，均进行如下操作：label'(t) = label (t) + s (s∈source(t))。

至此，与原 UML 模型语义等价的 FSM 模型就创建完成了，下面我们利用一个实例说明这个形式化机制。

3.3 实例研究

3.3.1 实例介绍

本节引入的是一个 Web 应用软件的实例，它提供软件产品的下载服务，其对应的 UML 状态图如图 3.1 所示。

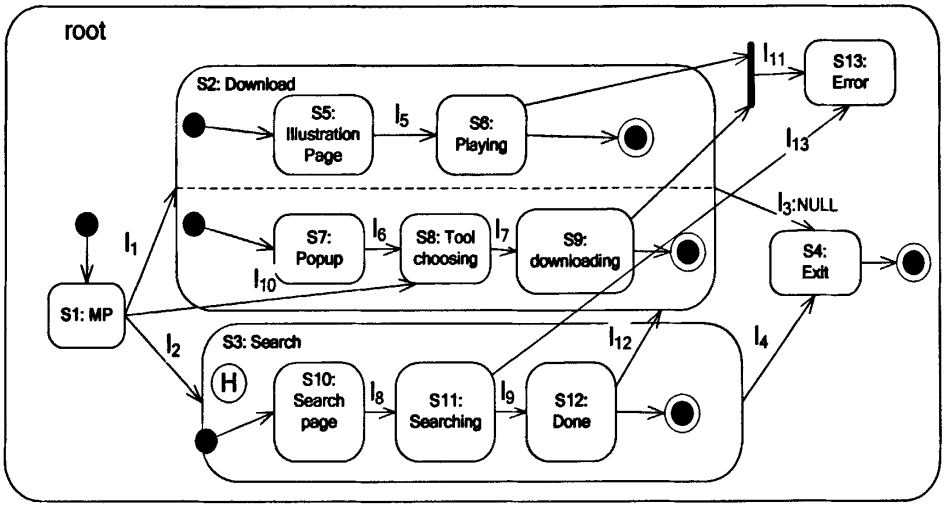


图 3.1 用于软件产品下载的 Web 应用的 UML 状态图

该 Web 应用的运行从进入其主页面 MP 开始，可点击相应链接分别进入下载或搜索页面。当选择主页上某软件的下载页面链接时，将转入对该软件的介绍页面并同时自动弹出下载框，介绍页面将显示软件使用相关信息的文字，并自动开始播放使用某些典型功能的示范视频，而下载框则要求用户选择下载工具，在用户做出响应后开始下载，该下载框在主页面上有直接链接，可在进入主页面后选择软件产品的下载方式以直接对其进行下载；当选择主页上的搜索链接时，将转入依照不同方法搜索软件产品的搜索引擎页面，待用户选择搜索方法、输入搜索关键字并点击搜索按钮后页面进入正在搜索的状态，搜索任务成功完成后返回搜索结果，用户可根据该搜索结果选择需要的软件产品的下载链接，也可退出该页面。另外，在介绍页面进行视频播放、下载页面处于下载

状态及搜索引擎页面处于搜索状态时，都有可能因为网速过慢、无法找到资源或内部错误等原因转入错误页面。

特别地，由状态图也可以看出，S3 状态有一个历史结点 H，这意味着当 Web 应用处于搜索功能范围内时，若有外部事件使其强行退出搜索状态，系统将记录当前处理数据及进程后再执行退出任务。一旦返回至搜索功能页面，Web 应用将直接根据该记录继续上次没有完成的搜索。图中 l_4 表明系统处于 S3 状态的任何一个子状态时都可在触发 l_4 的前提下离开 S3 状态及其当前活跃的子状态，即该 Web 应用正执行搜索功能且满足触发 l_4 的外部事件发生时，无论处于哪种搜索状态都可强行退出，结束 Web 应用的执行。相比之下，图中 l_3 的 NULL 属性则说明必须等到 S2 中两个域均到达结束状态才能触发 l_3，即该 Web 应用正执行用户选定软件的下载功能时，只要系统尚未完成产品介绍或下载流程就无法强行退出。

3.3.2 实例模型的形式化

对于该实例，本节将运用前文所述方法对其进行模型形式化操作，使其原 UML 状态图形式化为 FSM 模型。根据我们提出的形式化机制，应该先利用 HFSM 模型捕获 UML 状态图的拓扑结构，因此，通过层次结构的信息与获取办法，可将实例的层次信息表示为如下 HFSM 模型：

A_1: ({ S1, S2, S3, S4 }, { l_1, l_2, l_3, l_4 }, { (S1, l_1) → S2, (S1, l_2) → S3, (S2, l_3) → S4, (S3, l_4) → S4 }, S1, S4)

A_2: ({ S5, S6 }, { l_5 }, { (S5, l_5) → S6 }, S5, S6)

A_3: ({ S7, S8, S9 }, { l_6, l_7 }, { (S7, l_6) → S8, (S8, l_7) → S9 }, S7, S9)

A_4: ({ S10, S11, S12 }, { l_8, l_9 }, { (S10, l_8) → S11, (S11, l_9) → S12 }, S10, S12)

ϕ: $\phi_{root} = \{ A_1 \}$, $\phi(S2) = \{ A_2, A_3 \}$, $\phi(S3) = \{ A_4 \}$, $\phi(S1) = \phi(S4) = \ldots = \phi(S13) = \varnothing$

$F = (\{A_1, \ldots, A_4\}, \phi)$

在创建原 UML 模型的相应 HFSM 模型后，源模型的层次信息也随之被获

取并存储下来，并在转换算法运行时提供必要的状态间的层次关系判断操作。形式化机制在此基础上可进入下一环节，即转换方法的执行。方法的第一步是针对原 UML 状态图中的迁移，将这些迁移分别映射成为目标 FSM 模型的迁移，从而建立目标模型的最初迁移集合，下列表达式便描述了通过算法 3.1 得到的所有最初迁移（其中 Li 表示 UML 状态图中标签为 l_i 的迁移，Ci 表示 HFSM 模型的一个格局）：

L_1：C1 = { root, S1}, C2 = { root, S2, S5, S7 }, (C1, l_1) → C2

L_2：C3 = { root, S3, S10 }, (C1, l_2) → C3

L_3：C4 = { root, S2, S6, S9 }, C5 = { root, S4 }, (C4, l_3) → C5

L_4: C6 = { root, S3, S11 }, C7 = { root, S3, S12 }, (C3, l_4) → C5, (C6, l_4) → C5, (C7, l_4) → C5

L_5：C8 = { root, S2, S5, S8 }, C9 = {root, S2, S5, S9 }, C10 = { root, S2, S6, S7 }, C11 = { root, S2, S6, S8,}, C12 = {root, S2, S6, S9 }, (C2, l_5) → C10, (C8, l_5) → C11, (C9, l_5) → C12

L_6：(C2, l_6) → C8, (C10, l_6) → C11

L_7：(C8, l_7) → C9, (C11, l_7) → C12

L_8：(C3, l_8) → C6

L_9：(C6, l_9) → C7

L_{10}：(C1, l_{10}) → C8, (C1, l_{10}) → C11

L_{11}：C13 = { root, S13 }, (C12, l_{11}) → C13

L_{12}：(C7, l_{12}) → C2

L_{13}：(C6, l_{13}) → C13

以上所有迁移便构成了目标 FSM 模型的最初迁移集合，接下来便可根据这个集合中的每个元素得出目标 FSM 模型的状态集合，由与最初迁移集合中元素相关的状态组成，在本文实例中便是：Q = { C1,..., C13 }。而其中的初始状态与接受状态也可由相应算法得出，分别为 C1 和 C5。

最后，我们来处理原 UML 状态图中状态 S3 包含的历史状态 H。根据本文提出的算法，在对历史状态进行映射操作时，需要在前期创建的目标模型的最

初迁移集合中添加新迁移，并对以 S3 为源状态的迁移在目标模型中的对应迁移稍作修改。所以，经过算法运行，将以下所列迁移添加至最终 FSM 模型的迁移集合中：

(C5, “back (S10)”) → C3

(C5, “back (S11)”) → C6

(C5, “back (S12)”) → C7

同时，将所有标签为 l_4 的迁移进行如下修改：

(C3, l_4 (S10)) → C5

(C6, l_4 (S11)) → C5

(C7, l_4 (S12)) → C5

这样，目标 FSM 模型的迁移集合便最终确定，加上前面得出的状态集合、初始状态和接受状态，一个完整的 FSM 模型已创建成功，而原 UML 模型的信息也被很好地保留于该 FSM 模型中。

第四章 FSM模型的优化

4.1 归约指示

通过上一章的模型形式化机制，可以使需求分析与设计阶段的 UML 模型得到它们各自对应的FSM模型。虽然，目前本文的研究只给出UML状态图的形式化方法，但其余需转换的UML图形形式化后得到的FSM模型结构与本质是相同的，只是具体的状态名及状态与状态间的转移关系不同而已，并不影响任何对FSM模型进行优化的方法对它们的通用性。因此，本章的内容适用于所有从各种UML图形式化后得到的FSM模型，也包括根据任何建模理念人工创建的，或者其它途径获得的FSM模型，均无异处。

计算机完全可以在自定义的测试覆盖准则下，通过这些模型形式化后得到的 FSM 模型自动生成测试用例，或直接成为模型检验的输入。但过于庞大的FSM模型将带来灾难性的结果，即状态组合爆炸，产生的测试用例集元素个数太多，测试效率极低。为了避免这一现象的发生，便要对用作测试与待检验模型的FSM模型进行归约优化，使其按照检测要求减小为尽可能小的模型。由于我们关注的是系统功能测试，测试用例旨在检验系统实际行为或输出与预期是否相符，所以在进行模型归约的时候应以功能为依据，本文便是通过归约指示来表示检测需求，它描述了开发人员希望检测的那部分系统功能，在它的指导下，归约后的模型将只保留该功能的最小元素集并将与之无关的所有元素删除，使测试人员只在这个更小的模型基础上获得计算机围绕指定功能自动生成的测试用例，大大简化了测试过程，尽量不做无用功。

鉴于前期工作是集中于UML模型的形式化，而UML模型又具有强大的描述能力，故我们选用它作为归约指示的最初承载模型。从本质上说，归约指示实际上将原FSM模型按功能划分为两大部分，一部分是描述归约指示中所包含的功能的所有元素，另一部分则是不参与描述任何指定功能的所有元素。那么，归约指示就需要给出界定这两个部分的功能，而在 UML 模型的各类图中，顺

序图通过系统内部对象或系统与外部对象间的交互场景说明系统具有的某些功能，并将这些功能与其它功能区别开来，正是用以建立归约指示的最佳选择。因此，本文利用 UML 顺序图进行归约指示的设计。

4.2 归约指示的形式化

虽然归约指示已有模型加以描述，但顺序图中的非形式化部分依然有碍于严格的模型分析与 FSM 模型归约的自动化，这便需要对初始的归约指示模型进行形式化。本文采用 FSM 模型作为其形式化的目标模型，除了前文提及的所有关于 FSM 模型的优点以外，也是出于同种模型之间的合成与作用将会更便利，毕竟它们拥有相同的基本结构及元素。另外，UML 顺序图形式化为 FSM 模型正好是 UML 模型形式化为 FSM 模型转换的一部分，对上一章的模型形式化机制进行了补充，该过程生成的目标模型亦可用于自动产生测试用例与模型检验。

因而，在对原 FSM 模型进行归约前，必须先对人工建立的 UML 顺序图形式的归约指示进行形式化。与模型形式化机制一样，我们首先要考虑源模型与目标模型的表示形式，然后才能在这两种形式的基础上进行各自元素间映射关系的建立。上一章中已经给出 FSM 模型的数学表示形式，这里就只引入用于存储 UML 顺序图信息的描述结构。

定义 4.1 一个顺序图是一个三元组（Obj，Msg，Act），其中 $Obj = \{obj_1, obj_2, \ldots, obj_n\}$是在该顺序图描述场景中出现的对象集合；$Msg = \{msg_1, msg_2, \ldots, msg_n\}$是在该顺序图描述场景中出现的消息集合；*Act* 是该顺序图描述场景中出现的发送并接收消息的动作集合，其中每一个元素可表示成为一个四元组（obj_{origin}：Obj，obj_{dest}：Obj，msg：Msg，order），其中 obj_{origin} 是发送消息的对象，obj_{origin} 是接收消息的对象，***msg*** 是发送的消息内容，***order*** 是一个数字，它用来表示这个动作在顺序图中出现的所有动作中的排序。

对于任意一个 $act_i \in Act$，假设它可写作四元组（obj_1，obj_2，msg_i，num），那么 $(act_i) = obj_1$，$dest(act_i) = obj_2$，$message(act_i) = msg_i$，$order(act_i) = num$，其中 ***num*** 可由该动作所在的视觉位置确定：$\forall act_i, act_j \in Act \bullet$ (act_j 的视觉位置在 act_i 的直接下方) $\bullet$ $order(act_j) = order(act_i) + 1$。

当用户或测试人员按照他们的要求，用 UML 顺序图建立出归约指示之后，便可将图中所含信息全部写入上述表示形式中，然后为进一步地用于原模型归约而形式化。虽然所有的 UML 顺序图均用以描述对象间的交互活动，但展现的场景不同，情况也不尽相同。如果某顺序图是表示待检测系统内部的对象之间的交互，那么在形式化过程中，它的所有元素都需考虑在内；如果它表示的是待检测系统与系统以外的对象间的交互，只需提取那些与代表系统生命线的直线相交的动作，并以发出或接收将这些动作分类（任意 $msg_i \in Msg$ 前的"？"符号表示该消息对应的接收动作，任意 $msg_i \in Msg$ 前的"！"符号表示该消息对应的发出动作）。

由于顺序图中对于归约原模型的有用信息只有消息，因此目标 FSM 模型中的状态无需具有任何含义，本文用数字来表示，以简化算法。几乎每一个顺序图的场景描述都是从一个外部消息刺激开始的，而我们的形式化方法又是消息驱动的，所以形式化工作从序号为 1 的消息开始。下面是第一种情况下 UML 顺序图的形式化机制：创建一个新的 FSM 模型，写作（Q，L，δ，q_0，q），并进行如下初始化：Q = {0}，L = ∅，δ = ∅，q_0 = q = 0。遍历 UML 顺序图的动作集合 *Act* 并依据其中每个元素 act（act ∈ Act）对新建的 FSM 模型进行如下操作：Q = Q∪{order(act)}，L = L ∪ {?message(act)}，δ = δ ∪ {(order(act) − 1, ?message(act)) → order(act)}, q = order(act)，这样得到的 FSM 模型便是转换后的形式化的归约指示模型。

在第二种情况下，有价值的信息仅限于待检测系统发出和收到的消息，这些消息实际上代表着系统的行为。正因为我们需要将原 UML 顺序图中的每个动作拆解成为两个有向动作，即一个发出动作与一个接收动作，消息的顺序已不再能单从其对应动作的视觉位置来判断了，这里以一个小例子来说明这一点。假设在某个 UML 顺序图中描述了如下场景：对象 A 在给待检测系统发送了消息 a 后，给对象 B 发送了消息 b，当对象 B 接收到消息 b 后便立即给系统发送消息 c。这时，为了表示对象 A 是在发送消息 a 后才发送消息 b，必须将 a 对应的动作画在 b 所对应的动作上方，而为了表示对象 B 在接收到消息 b 后才发送消息 c，必须将 b 对应的动作画在 c 所对应的动作上方，所以，在视觉上，消

息 a 所对应的动作在消息 c 所对应动作的上方。如果按照前面的规则，越靠上部的动作，其序号越小，那么意味着系统先接收 a 再接收 c，这就有可能与事实不符。实际情况是取决于传递消息 b、c 的时间总和与传递消息 a 的时间的关系，若前者大于后者，则系统先接收消息 a；若后者大于前者，则系统先接收消息 c。所以需要另一种方法来判断消息对应动作的先后顺序，而不再仅仅是只依据视觉位置，以下便是判断规则（这里扩展了函数 ***order***，使它也能表示所有消息对应的发送与接收动作的序号）：

— $\forall msg_i \in Msg \bullet order(?msg_i) < order(!msg_i)$

— $\forall msg_i, msg_j \in Msg \bullet (order(msg_i) < order(msg_j) \wedge origin(message^{-1}(msg_i)) = dest(message^{-1}(msg_j))) \bullet order(!msg_i) < order(?msg_j)$

— $\forall msg_i, msg_j \in Msg \bullet (order(msg_i) < order(msg_j) \wedge origin(message^{-1}(msg_i)) = dest(message^{-1}(msg_j))) \bullet order(?msg_i) < order(!msg_j)$

虽然以上规则可以判定大部分动作的先后顺序，但刚才提到的情景只有相关消息的传递时间已知的情况下才能决定它们相应发出和接收动作的排序。因此，除非获得时间信息，否则需要将所有可能的动作序列均写出，再分别建立对应的 FSM 模型。对于根据动作顺序判定规则得出的任意动作序列 s_i，建立一个新的 FSM 模型并初始化为：$Q = \{0\}, L = \varnothing, \delta = \varnothing, q_0 = 0, q = 0$，然后用下面的算法遍历 s_i 中的所有元素。

算法 4.1

```
m = 1
For each item∈s_i
if(("?"∈item ∧ dest(item\"?") = system) ∨
                ("!"∈item ∧ origin(item\"!") = system))
  Q = Q∪{m}
  L = L∪{item}
  δ = δ∪{(m – 1, item) → m}
```

$q = m$

$m = m+1$

这样，便把第二种情况的 UML 顺序图形式化为 FSM 模型。但由于顺序图表示的动作序列的不唯一性，可能对于一个归约指示会产生多个 FSM 模型，这些模型都将分别被当作一个形式化后的归约指示对原 FSM 模型进行归约操作。因此，下一节给出的归约机制是单个归约指示的 FSM 模型对原模型的归约操作，若需多次归约，叠加使用即可。

4.3 归约机制

得到了形式化的归约指示后，便可以开始运行其对原模型的归约机制了，但首先必须引入一些后面要用到的基本概念[60]。

定义 4.2 给定一个有限状态机 A（Q, L, δ, q_0, q），则当 A 的状态与迁移交替组成的有限序列 $q_i l_i q_{i+1} l_{i+1} \ldots q_n$（$0 \le i \le n$）满足条件$((q_{i+a}, l_{i+a}) \rightarrow q_{i+a+1}) \in \delta$ （$0 \le a \le n-i-1$), $q_i = q_0$）时，它便称为 A 的一个执行片段，记作 ***frag***。特别地，如果执行片段满足 $q_n = q_0$ 或 $q_n = q$，那么称 ***frag*** 是 A 的一个运行。

定义 4.3 给定两个序列 S_1 与 S_2，函数_∝_: sequence → sequence

$$S1 \propto S2 \Leftrightarrow \forall item_i, item_j \in ran(S_1) \wedge S_1^{-1}(item_i) < S_1^{-1}(item_j) \bullet$$
$$item_i, item_j \in ran(S_2) \wedge S_2^{-1}(item_i) < S_2^{-1}(item_j)$$

定义 4.4 给定一个有限状态机 A（Q, L, δ, q_0, q）及其所有执行片段组成的集合 $Frag_A$，函数 ***reach***: Q → Q

$$reach(q_i) = \{q_j \in Q \mid \exists frag \in Frag \bullet q_i, q_j \in frag \wedge i < j\}\}$$

定义 4.5 给定两个有限状态机 A 与 A′，分别表示为（Q, L, δ, q_0, q）与（Q′, L′, δ′, q_0′, q′），A 与 A′的积 **A×A′**也是一个有限状态机模型：（$Q_{A\times A'}$, $L_{A\times A'}$, $\delta_{A\times A'}$, $q_{0(A\times A')}$, $q_{A\times A'}$），

其中 $Q_{A\times A'} = Q \times Q'$;

$L_{A\times A'} = L \cup L'$;

$\delta_{A\times A'} = \{((q_i, q_i'), \Delta l) \rightarrow (q_j, q_j') \mid q_i, q_j \in Q \wedge q_i', q_j' \in Q' \wedge ((((q_i, (\text{“?”} \cup \Delta l)) \rightarrow q_j) \in \delta \wedge ((q_i', (\text{“!”} \cup \Delta l)) \rightarrow q_j') \in \delta') \vee (((q_i, (\text{“!”} \cup \Delta l)) \rightarrow q_j) \in \delta \wedge ((q_i', (\text{“?”}$

$\cup \Delta l)) \rightarrow q_j')\in\delta'))\} \cup \{((q_i, q_i'), \Delta l) \rightarrow (q_j, q_i') \mid q_i, q_j\in Q \wedge q_i'\in Q' \wedge ((q_i, \Delta l) \rightarrow q_j)\in\delta\} \cup \{((q_i, q_i'), \Delta l) \rightarrow (q_i, q_j') \mid q_i\in Q \wedge q_i', q_j'\in Q' \wedge ((q_i', \Delta l) \rightarrow q_j')\in\delta\}$；

$q_{0(A\times A')} = q_0$；

$q_{A\times A'} = q$。

定义 4.6 给定一个有限状态机 A（Q, L, δ, q_0, q）及其所有执行片段组成的集合 $\boldsymbol{Frag_A}$，对于每一个 frag∈$Frag_A$，可将其写作 $q_il_iq_{i+1}l_{i+1}\ldots q_n$（$0 \le i \le n$），它的所有标签组成的序列称为这个执行片段 frag 的轨迹，记为 trace(frag) = $l_il_{i+1}\ldots l_{n-1}$。

定义 4.7 给定两个有限状态机 A 与 A^*，分别表示为（Q, L, δ, q_0, q）与（Q^*, L^*, δ^*, q_0^*, q'），设 $\boldsymbol{Frag_A}$ 与 $\boldsymbol{Frag_{A\times A^*}}$分别是 $\boldsymbol{A}$ 与 $\boldsymbol{A\times A^*}$的所有执行片段组成的集合，***Cir*** 是 $\boldsymbol{A\times A^*}$的所有运行组成的集合。对于每一个执行片段 frag∈$Frag_{A\times A^*}$，它在 A 上的**投影**是其出现于 A 上的那部分标签序列，数学表达式可写为 $\pi_A(trace(frag)) = (frag)\backslash\{l \mid l\in A \wedge l\notin A^*\}$。

函数 ***cover***: (Cir, $Frag_A$) → 0/1

$$cover(cir, frag) = \{1 \mid cir\in Cir \wedge frag\in Frag_A \wedge \exists frag'\in \boldsymbol{Frag_{A\times A^*}} \bullet frag' \propto cir \wedge \pi_A(trace(frag')) = trace(frag)\}$$

FSM 模型的执行片段可以代表其描述的对应系统的行为，因此，归约原 FSM 模型意味着，经归约后的模型应该包括描述归约指示中要求检测的功能的所有元素，并且将其余无关元素全部清除。例如，假设给定一个待归约的 FSM 模型 ***A*** 和一个归约指示的 FSM 模型 ***RD***，其中 ***A*** 的所有运行组成的集合记为 $\boldsymbol{Cir_A}$，***RD*** 的所有执行片段记为 $\boldsymbol{Frag_{RD}}$。为了使归约后的模型包含 ***RD*** 要求的所有动作序列，我们可以构造一个 FSM 模型 ***A'***（这里用 $\boldsymbol{Cir_A}$表示其所有运行组成的集合），使得 ***A'***满足如下条件：$\forall frag\in Frag_{RD} \bullet (\exists cir_A\in Cir_A \wedge frag \propto cir_A) \bullet (\exists cir\in Cir_{A'} \bullet frag \propto cir_{A'})$；为了保证原模型中与描述指定功能无关的元素都不在归约后的模型中出现，***A'***还应满足如下条件(这里用 $\boldsymbol{Cir_{A''}}$ 表示模型 ***A''***的所有运行组成的集合)：$\neg\exists A'' \bullet ((\forall frag\in Frag_{RD} \bullet (\exists cir_A\in Cir_A \wedge frag \propto cir_A) \bullet (\exists cir_{A''}\in Cir_{A''} \bullet frag \propto cir_{A''})) \bullet (\exists frag'\in Frag_{RD} \bullet (\exists cir_{A'}\in Cir_{A'} \bullet frag' \propto cir_{A'} \bullet (\neg\exists cir_{A''}\in Cir_{A''} \bullet frag' \propto cir_{A''}))))$。

给定一个执行片段 ***frag*** 和一个运行 ***cir***，表达式 frag∝cir 与 cover(frag, cir) 的唯一不同点在于，前者同时考虑 ***frag*** 与 ***cir*** 的状态与迁移，以及它们之间的关系，而后者仅关注执行片段与运行中迁移的标签序列间的关系，并不包含相关的状态信息。然而，我们在确定归约后模型中需要保留的标签序列后，将会在创建新模型时加入与这些标签相关的状态信息，这便意味着，要得到用户要求的简单模型，可以首先计算出一个模型 A^*使得操作 $A\times A^*$满足如下条件（这里用 $Cir_{A\times A^*}$表示模型 $A\times A^*$的所有运行组成的集合，用 $Cir_{A\times A^{*'}}$表示模型 $A\times A^{*'}$ 的所有运行组成的集合）：$\forall frag\in Frag_{RD} \bullet (\exists cir_A\in Cir_A \wedge frag\propto cir_A) \bullet (\exists cir\in Cir_{A\times A'} \bullet cover(cir, frag)$以及$\neg\exists A^{*'} \bullet ((\forall frag\in Frag_{RD} \bullet (\exists cir_A\in Cir_A \wedge frag\propto cir_A) \bullet (\exists cir_{A''}\in Cir_{A\times A^{*'}} \bullet cover(Cir_{A\times A^{*'}}, frag)) \bullet (\exists frag'\in Frag_{RD} \bullet (\exists cir_{A'}\in Cir_{A'} \bullet frag'\propto cir_{A'} \bullet (\neg\exists cir\in Cir_{A\times A^{*'}} \bullet cover(cir, frag'))))$。

因此，当归约指示指定的执行片段确定后，便可构造出模型 A^*，它是由所有包含这些片段的运行轨迹所组成的，其所需满足的性质可表示为如下形式化的数学表达式：$(\forall l\in L_A \bullet l\in cir\in\{cir\in Cir_A \mid \exists frag\in Frag_{RD} \bullet trace(frag)\propto trace(cir)\} \bullet l\in L_{A^*}) \wedge (\forall l\in L_A \bullet l\in cir\in\{cir\in Cir_A \mid \neg\exists frag\in Frag_{RD} \bullet trace(frag)\propto trace(cir)\} \bullet l\notin L_{A^*})$。由此可以得出，整个归约工作应该从以下两个集合的构建开始：集合 ***Cir****，它是一组满足特定条件的运行所组成的集合$\{cir\in Cir_A \mid \exists frag\in Frag_{RD} \bullet trace(frag)\propto trace(cir)\}$；集合$\overline{Cir^*}$，它是集合 ***Cir****的补集，可表示为$\{cir\in Cir_A \mid \neg\exists frag\in Frag_{RD} \bullet trace(frag)\propto trace(cir)\}$。

但是，在求解上述两个集合之前，我们必须注意到上一章模型形式化后得到的系统原 FSM 模型中，各迁移的标签是不包含其类属信息的，即无法判断该迁移对应的触发事件是输入、输出亦或是内部事件。另一方面，计算 $A\times A^*$时是将两者迁移中的输入输出事件进行比较与匹配，并合并相同的迁移项。另外，对归约指示进行形式化时，由于顺序图的固有属性就是描述了消息发出与接收，其转化而来的 FSM 模型也是包含了输入输出信息的，而本文算法需要将它与原 FSM 模型进行比较与匹配方能获取上述两个集合。因此，在原 FSM 模型中添加迁移上标签的输入输出类别十分必要。根据原 UML 状态图中迁移标签各部分的含义，事件代表对系统的外部刺激，即输入；动作则代表系统对该外部刺

激的反应，即输出。在模型形式化的过程中，原 UML 模型迁移与转换后 FSM 模型中对应迁移的标签是一致的，故原系统 FSM 模型的迁移标签也可按 UML 状态图语义加以分解。我们在此把事件视为输入消息，而把动作视为输出消息，并标记于原系统 FSM 模型中以为后续算法所用。

算法 4.2 便是我们为获得集合 Cir^* 与 $\overline{Cir^*}$ 所设计，其输入是附有输入输出信息的系统原 FSM 模型与形式化为 FSM 模型的归约指示，它的输出结果是集合 Cir^* 与 $\overline{Cir^*}$（其中 last(*s*)表示序列 *s* 的最后一个元素，变量 ***flag*** 用以记录其对应状态是否被访问过）。

算法 4.2

current = $<q_0>$, $Cir^* = \{\}$

$\forall q_i \in Q_A \bullet q_i.flag = 0$

while(current != ∅)

currentNode = last(current)

if(currentNode) = q | q_0)

if($\exists frag \in Frag_{RD} \bullet frag \propto current \vee$

$(\forall qi,qj \in Ran(current) \bullet q_i = q_j \bullet$

$\exists n:N \bullet frag \propto (<q_0 l_0 \ldots q_{i-1} l_{i-1}> \cup n \times <q_i l_i \ldots q_j l_j> \cup$

$<q_{j+1} l_{j+1} \ldots q_n>)))$

$Cir^* = Cir^* \cup \{current\}$

else

$\overline{Cir^*} = \overline{Cir^*} \cup \{current\}$

current = current \ <current(n–1), current(n)>

else

nextNode = $\{Nq \in Q_A \mid \exists l \in L \bullet (current,l) \rightarrow Nq \in \delta_A\}$

if($\neg \exists Nq \in nextNode \bullet Nq.flag = 1 \wedge$

$\neg \exists q \in Ran(current) \bullet currentNode = q$)

current = current \ <current(n–1), current(n)>

else

$q' = \text{random}\{\forall Nq \in \text{nextNode} \mid Nq.\text{flag} = 1\}$

$l' = \text{label}(\text{source}^{-1}(\text{currentNode}) \cap \text{target}^{-1}(q'))$

$\text{current} = \text{current} \cup \langle l'q' \rangle$

在给定了输入模型与归约指示后，集合 Cir^* 与 $\overline{Cir^*}$ 也由算法 4.2 得出，此时便可根据这两个集合创建模型 A^*，使操作 $A \times A^*$ 执行后的结果模型只描述归约指示中要求的功能。为此，我们设计了一个计算模型 A^* 中应存在的迁移序列的算法，大致原理是：每一个在集合 Cir^* 的任意运行中出现的标签都在模型 A^* 中，而那些只在集合 $\overline{Cir^*}$ 的运行中出现却不在集合 Cir^* 的任意运行中出现的标签则被排除在模型 A^* 之外。

算法 4.3

for each $\overline{Cir}: q_0 l_0 q_1 l_1 \ldots q_n \in \overline{Cir^*}$

flag = 1

i = 0

while($(\exists cir \in Cir^* \bullet l_i \in cir) \wedge$ flag)

if(i<n) $l_i = l_{i+1}$

else found = 0

if(flag)

$L = L_A \backslash \{l_i\}$

$\delta = \delta_A \backslash \{\text{label}^{-1}(L_A)\}$

for each $q' \in \{q \in Q_A \mid q \notin \text{reach}(q_{0A})\}$

$\delta = \delta_A \backslash \{t \in \delta_A \mid t \in \text{source}^{-1}(q') \cup \text{target}^{-1}(q')\}$

对集合 $\overline{Cir^*}$ 中每一个元素执行算法 4.3 的操作后，即可输出所有原 FSM 模型中应保留在归约后模型中的迁移的集合。然后，识别出该集合中所有附带输入输出标记的迁移，并将每个标记取反，即输入符号改为输出符号，输出符号则改为输入符号，经此取反处理后的迁移集合就构成了模型 A^*。最后，对模型 A 与模型 A^* 执行操作 $A \times A^*$，计算结果即为按归约指示归约后的 FSM 模型 A′。

但此时的 A'也将输入输出标记保留，标签处于分离状态，而且，由于它是两个模型执行"×"而来，故状态是复合的，其名字为两个原状态名的叠加。如果希望最终得到的模型的形态与最原始模型保持一致，可以采取如下措施：首先，将所有以动作为标签的迁移与相关状态删除，并删除所有输入输出标记，或者尽可能地合并能够合并的迁移序列，例如，某迁移以某事件 E 为标签，其后续迁移的标签是一个动作，且正好与 E 构成原 UML 模型中的某个完整标签，则可将两者合并为一个迁移，并将其标签还原为原 UML 模型的对应标签。其次，对每一个复合状态的状态名，只截取代表原 UML 模型中状态名的那一部分，而将剩余的部分全部去除。当然，如果归约后的模型只是为产生测试用例所用，则复合状态的名字无需做任何改变，因为测试用例的实质是标签序列，状态的意义与此无关，亦可不作考虑。

与上一章类似，下面我们也用一个实例来说明本章提出的方法。

4.4 实例研究

4.4.1 实例介绍

为了保持方法的连贯性，本节仍然使用上一章中提出的实例，但我们对原例稍做了些修改以更清楚地体现本章中提出的方法，修改后的原 UML 状态图如图 4.1 所示。

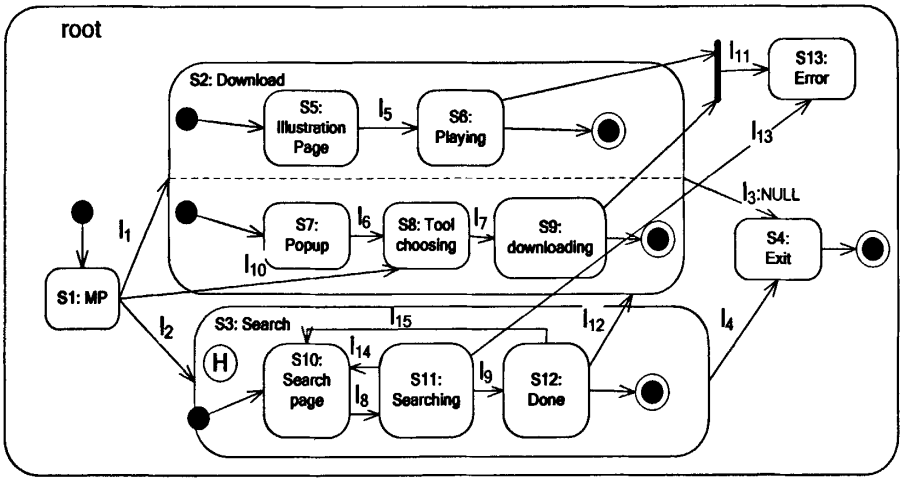


图 4.1 修改后的实例 UML 状态图

从图中可以看到，新 UML 状态图在原图基础上增加了 l_{14} 与 l_{15} 这两条迁移，分别代表从 Web 应用的正在搜索状态返回搜索主页面和从返回搜索结果页面返回搜索主页面。另外，我们对某些迁移边上的标签结构进行假设：$l_8 = l_8'/\text{Request}()$, $l_9 = l_9'/\text{Result}()$, $l_{13} = l_{13}'/\text{Error}()$。

在上一章中，该实例中的状态图已经根据本文提出的算法 3.1 形式化为相应的 FSM 模型，故此处可直接用本章叙述的方法对其进行模型归约，而针对原 UML 状态图的改变需做出如下变动：$L = L \cup \{l_{14}, l_{15}\}$, $\delta = \delta \cup \{(C6, l_{14}) \rightarrow C3, (C7, l_{15}) \rightarrow C6\}$，且应将 l_8 拆分成 $?l_8'$ 与 !Request()，l_9 拆分成为 $?l_9'$ 与 !Result()，l_{13} 拆分成为 $?l_{13}'$ 与 !Error()。经过以上的这些添加与拆分操作，我们便得到了一个可用于模型归约模拟的原系统 FSM 模型。

接下来便需要设计归约指示以指导原 FSM 模型的归约。根据上文中的分析，顺序图具有描述两种场景的功能，所以对每一种分别给出一个对应的归约指示模型，如图 4.2 与 4.3 所示。

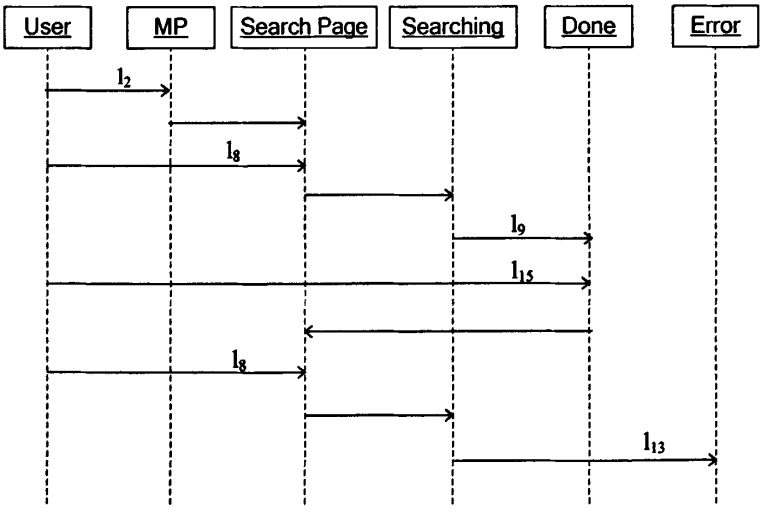


图 4.2　归约指示 sd_1 的顺序图模型

图 4.2 描述的是 Web 应用中各组件间的交互场景，其对象集合由 Web 应用的逻辑页面组成，它们均可在原 UML 状态图中找到匹配的状态，即可在原系统 FSM 模型中找到对应的状态。由于对象间传递的消息不用分出输入与输出两种不同类别，故系统原 FSM 模型亦无需将标签拆分为不同的部分。而图 4.3 描

述的是 Web 应用与其外部的其它对象间的交互场景，它的对象集合是由包括 Web 应用自身在内的若干个独立功能组件。在 User 点击该 Web 应用的某种命令操作后，Web 应用会向 Service Manager 发出与此命令对应的服务请求消息，而 Service Manager 负责对这些请求进行处理并转发给提供该服务的组件，在本实例中，是搜索引擎以完成用户提交的搜索请求。然后，Service Manager 会收到提供服务的外部组件返回的服务信息，并转发给 Web 应用，最终呈现在用户界面中以供用户查看。我们只从此种顺序图中提取所有与 Web 应用对应的生命线相交的消息信息，且需要标记它们的输入或输出属性。

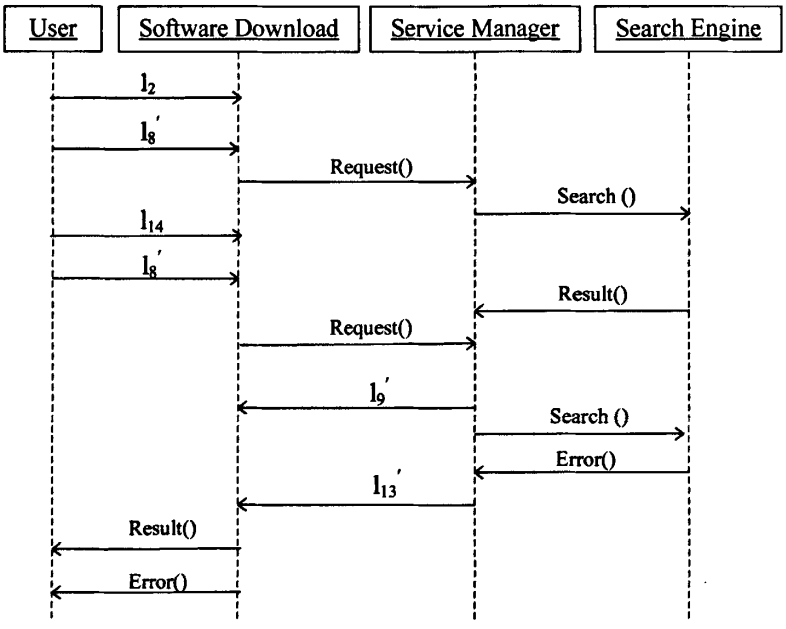


图 4.3　归约指示 sd_2 的顺序图模型

4.4.2 实例模型的优化

根据前文中给出的归约办法，归约过程的第一步应是先将归约指示进行形式化。通过本文提出的形式化方法，可将第一个归约指示 sd_1 的顺序图模型形式化为如下的 FSM 模型：

$$A_{sd1}: (Q_{sd1}, L_{sd1}, \delta_{sd1}, q_{0sd1}, q_{sd1})$$

$Q_{sd1} = \{0, 1, 2, 3, 4, 5, 6\}$, $L_{sd1} = \{?l_2, ?l_8', ?l_9', ?l_{15}, ?l_{13}'\}$

$\delta_{sd1} = \{ (0, ?l_2) \rightarrow 1, (1, ?l_8') \rightarrow 2, (2, ?l_9') \rightarrow 3, (3, ?l_{15}) \rightarrow 4,$

$(4, ?l_8') \rightarrow 5, (5, ?l_{13}') \rightarrow 6\}$

$q_{0sd1} = 0$, $q_{sd1} = 6$

对于第二个归约指示 ***sd₂*** 的顺序图模型，根据文中给出的关于发送与接收事件顺序的判断规则可知，有两种可能的动作序列。然后，运用提出的转换算法，分别对这两种动作序列进行各自对应 FSM 模型的构造，完成后，可得出以下两个 FSM 模型：

第一种情况：$: A_{sd2}: (Q_{sd2}, L_{sd2}, \delta_{sd2}, q_{0sd2}, q_{sd2})$

$Q_{sd2} = \{0, 1, \ldots, 10\}$

$L_{sd1} = \{?l_2, ?l_8', !Request(), ?l_{14}, ?l_9', ?\ l_{13}', !Result(), !Error()\}$

$\delta_{sd2}: \{ (0, ?l_2) \rightarrow 1, (1, ?l_8') \rightarrow 2, (2, !Request()) \rightarrow 3, (3, ?l_{14})$

$\rightarrow 4, (4, ?\ l_8') \rightarrow 5, (5, !Request()) \rightarrow 6, (6, ?l_9') \rightarrow 7,$

$(7, ?l_{13}') \rightarrow 8, (8, !Result()) \rightarrow 9, (9, !Error()) \rightarrow 10\},$

$q_{0sd2} = 0$, $q_{sd2} = 10$

第二种情况：$A_{sd2}: (Q_{sd2}, L_{sd2}, \delta_{sd2}, q_{0sd2}, q_{sd2})$

$Q_{sd2} = \{0, 1, \ldots, 10\}$

$L_{sd1} = \{?l_2, ?l_8', !Request(), ?l_{14}, ?l_9', ?l_{13}', !Result(), !Error()\}$

$\delta_{sd2}: \{ (0, ?l_2) \rightarrow 1, (1, ?l_8') \rightarrow 2, (2, !Request()) \rightarrow 3, (3, ?l_{14})$

$\rightarrow 4, (4, ?l_8') \rightarrow 5, (5, !Request()) \rightarrow 6, (6, ?l_{13}') \rightarrow 7,$

$(7, ?l_9') \rightarrow 8, (8, !Result()) \rightarrow 9, (9, !Error()) \rightarrow 10\}$

$q_{0sd2} = 0$, $q_{sd2} = 10$

在附有输入输出信息的系统原 FSM 模型与形式化后的归约指示的基础上，归约过程便开始了。根据前文给出的归约算法，首先针对归约指示 ***sd₁*** 给出的约束条件，以其对应的 FSM 模型与系统原 FSM 模型为输入，可得到如下所示的投影模型：

$A'_{sd1}: (Q'_{sd1}, L'_{sd1}, \delta'_{sd1}, q_0'_{sd1}, q'_{sd1})$

$Q'_{sd1} = \{C1, C3, C5, C6, C7\}$

$L'_{sd1} = \{l_2, l_8, l_9, l_4(S10), l_4(S11), l_4(S12), l_{15}\}$

δ'_{sd1}: {(C1, l_2) → C3, (C3, l_8) → C6, (C6, l_9) → C7, (C3, l_4 (S10)) → C5, (C6, l_4(S11)) → C5, (C7, l_4(S12)) → C5, (C7, l_{15}) → C6}

$q_0'_{sd1}$ = C1, q'_{sd1} = C5

然后再以归约指示 sd_2 为给约束条件，对系统原 FSM 模型进行归约。由于归约指示 sd_2 对应于两个可能的 FSM 模型，可分别代入归约算法以指导归约过程。对于第一种可能的 FSM 模型，代入归约算法并运行完成后，可得到如下所示的投影模型：

A'_{sd2}: (Q'_{sd2}, L'_{sd2}, δ'_{sd2}, $q_0'_{sd2}$, q'_{sd2})

$Q'_{sd2} = \{C1, C3, C5, C6, C7\}$

$L'_{sd2} = \{l_2, l_8, l_9, l_4(S10), l_4(S11), l_4(S12), l_{14}\}$

δ'_{sd2}: {(C1, l_2) → C3, (C3, l_8) → C6, (C6, l_9) → C7, (C3, l_4 (S10)) → C5, (C6, l_4(S11)) → C5, (C7, l_4(S12)) → C5, (C6, l_{14}) → C3}

$q_0'_{sd2}$ = C1, q'_{sd2} = C5

而对于归约指示 sd_2 的第二种可能的 FSM 模型，代入归约算法并运行完成后所得的结果与第一种的情况相同，故在此不再列出。

但是，应该注意到，尽管在本文实例中，由某归约指示对应的所有可能的 FSM 模型得出的投影模型相同，在更为复杂的实例中却会产生不同的投影模型。所以，考虑归约指示顺序图模型可能包含的不同消息序列仍是必要的。

第五章　系统设计与实现

为了支持前面两章中提出的理论和方法，需要相应的实现方法以证明其可行性并使之真正服务于开发过程。本章正是在这些理论与方法的基础上，选择合适的开发平台后，设计出工具的实现方案。但在给出实现方法前，我们要考虑中间模型与结果模型以何种形式展现给用户，如何能让用户对工具的输出结果一目了然，这便涉及到模型的可视化。

5.1 层次有限状态机模型的可视化

利用文本描述模型的数据结构及其具体参数才能使计算机自动读取并分析这些模型，这也是使模型贯穿于各环节的途径。它承载模型信息，连接各阶段的数据流，将其一直传递下去，并可留作日后备用。若只是让计算机完成使命，我们大可输入文本，输出也文本，但任何方法及其相关工具都是以用户为根本出发点的，是为了最大程度地满足用户的需求。在使用该工具前要求用户精通相关数学知识、熟悉语法规则显然是不合理的，而且，当信息量过大时，阅读文档不但费时费力，也效率极低，往往还未掌握模型的整体框架便遗忘了初期阅读成果。因此，利用人类对图形的高度认知与图形天然的直观性，可将文本格式的模型进行可视化转换，方便用户使用，减少用户的负担，通过界面中呈现的相应图形传达输入及输出模型的架构与细节。

纵观全文理论与方法的主体，除去初始阶段的各类 UML 模型，所有方法与算法的作用对象或输出结果均为 FSM 模型，它担负着对 UML 各模型的形式化、统一化以及使其适用于常用的模型检验工具等使命，又是归约过程输入信息与输出结果的载体。另外，在项目的其它环节中，FSM 模型也无处不在，它是模型检验的基础，是产生测试用例的依据。可见 FSM 模型在本课题中扮演着关键性的角色，它所携带的信息直接影响检验与测试结果。而那些作为源模型的 UML 模型本身就是以图形模式人工创建的，无需再图形化，因此，我们把重点放在 FSM 模型的可视化上。

FSM 模型的基本元素状态与迁移均在同一层次上，并无父子之分，但层次有限状态机模型也在中间环节出现，它可以帮助我们理解 Web 应用各组件或页面的逻辑关系，尤其是层次关系。而且，对于单用有限状态机描述系统行为的用户来说，层次有限状态机的可视化功能无疑使他们的建模途径更为丰富、更为有效、也更为合理。因此，我们旨在设计一种更为通用的可视化工具，适用于所有层次有限状态机模型文档，而 FSM 模型则可看作层次为 1 的层次有限状态机模型，实现模型描述方式的统一化。

5.1.1 模型的文本表示

可视化过程意味着从文本到图形的转换，而且计算机根据算法对模型进行处理时实质上是在处理模型所对应的文档，所以层次有限状态机模型的文本表示方法的选取显得尤为重要。

由于 XML 的简单性、可扩展性与平台无关性，我们决定采用此种语言作为项目中表述有限状态机的通用语言。作为一种被工业界广泛采用的标准，XML 提供了描述结构化数据的方法，它可以将数据保存为一种格式，从而提供给没有创建此数据的其他应用程序所使用。XML 通过一系列简单的标记描述数据及这些数据之间的关系，而这些标记可方便建立，正是简单性，使其迅速成为数据交换的常用公共语言之一，许多应用软件也都陆续支持 XML 的数据交换格式，这将赋予工具很好的通用性及延展性，也缩短了项目组成员掌握语言所需花费的时间。

在使用 XML 文件进行数据的存储与交换时，所有与该文件有关的服务接收方与服务提供方必须约定一个统一的格式与标准，这样各方才能正确识别与理解 XML 文件中包含的信息。XML 的标志信息在于其自定义的标签，按照某种标签机制便能读取或写入该类别的 XML 文件。鉴于此，本文定义了一套用于描述层次有限状态机模型的 XML 文件格式，以使包括 FSM 模型在内的层次有限状态机模型成为各环节通用的描述语言，实现数据的传递。

层次有限状态机模型的基本元素为状态与迁移，因此我们围绕它们创建自定义 XML 文件的规范。首先针对状态有如下 XML 元素：

<state>：用于表示层次有限状态机模型中的各层次状态，可出现任意次，有如下属性：

id：状态名，是其对应标签<state>代表的状态的标识符，在文档中必须唯一，不可为空

src：代表插入该状态的资料文件的 URL，可为空

另外，<state>有如下子节点：

<onentry>：为可选元素，用于保存进入该<state>代表的状态时将运行的可执行内容，可出现 0 次或 1 次，无属性列表，有子节点<handler>，由可执行内容构成

<onexit>：为可选元素，用于存储退出该<state>代表的状态时将运行的可执行内容，可出现 0 次或多次，无属性列表，有子节点<handler>，由可执行内容构成

<transition>：用于表示以此<state>代表的状态为源状态的迁移，可出现任意次

<state>：为可选元素，用于记录子状态信息，其内部标签结构与外层标签<state>一样并允许一直嵌套下去，可出现任意次。当该元素出现时，表示其直接上层<state>标签对应的状态为复合状态

<final>：用于表示层次有限状态机模型的接受状态，不含子节点，其属性“id”代表状态名，是状态的标识符，在文档中必须唯一

既然有表示接受状态的 XML 元素，那么也应该有表示初始状态的 XML 元素，这里，我们用 XML 顶层元素的属性来记录初始状态。该属性名为“initialstate”，与版本信息等属性并列，它的值是初始状态的状态名。

接下来，便是迁移的描述格式。由状态的描述格式可知，迁移信息标签<transition>是作为状态对应的 XML 元素的子节点而存在的，它直接上层的<state>标签代表的状态是该<transition>标签代表的迁移的源状态，其属性有如下几种：

event：事件名，代表触发迁移发生的事件，只有该事件发生时才能触发迁

移，使系统状态发生改变，可为空。当它为空时，说明只要进入其对应<transition>标签代表的迁移的源状态并完成其动作，该迁移就能被触发

target：状态名，表示其对应<transition>标签代表的迁移的目标状态，可为空。当它为空时，表示迁移发生在同一个状态中，即该迁移的源状态到自身的迁移。

综上所述，本文自定义的描述层次有限状态机的 XML 语言，主要通过其顶层元素、顶层元素之间的子节点<state>与<final>、这些子节点的子节点<state>、<transition>等，以及所有元素节点的属性来记录各种层次有限状态机模型的所有信息。FSM 模型与一般层次有限状态机模型的不同就在于，其 XML 文件的标签<state>不再含有子标签<state>，而在描述层次有限状态机模型时，可根据各状态相对于顶层状态机的层次关系，将其相关信息嵌入与此对应的层次标签中。以下便是一个利用本文自定义标签书写的 XML 文件片段，它对应于一个简单的层次有限状态机模型：

```
<?xml version="1.0" ?>
<scxml xmlns="http://www.w3.org/2005/07/scxml" version="1.0"
initialstate="Login">
  <state id="Login">
      <transition event="sumbit0" target="UserMain" />
  </state>
  <state id="UserMain">
      <transition event="link0" target="Login" />
      <transition event="link1" target="PwdManage" />
      <transition event="link2" target="ManageCenter" />
  </state>
  <state id="PwdManage">
      <transition event="return8" target="UserMain" />
```

```
</state>
<state id="ManageCenter">
    <transition event="link3" target="UserManage" />
    <transition event="link4" target="SettleMain" />
</state>
            ... ...
```

5.1.2 可视化工具的设计与实现

在确定了层次有限状态机的文本格式之后，便可以开始考虑可视化工具的整体架构与具体技术细节。整个可视化过程可分解为若干子过程：首先获取层次有限状态机文档中的文本，然后从中抽取用于图形生成的相关信息，并将这些文本信息转化成为某种计算机可识别的数据结构存储于计算机内，随即根据数据结构描述的层次有限状态机模型对各节点的位置进行优化布局，再将数据结构中的元素映射为相应的图形元素，如圆、双圆、弧线等。最后，根据布局及映射得出图形。综上，便可大致构建出系统的基本框架（见图 5.1），由各子过程划分出各功能模块。

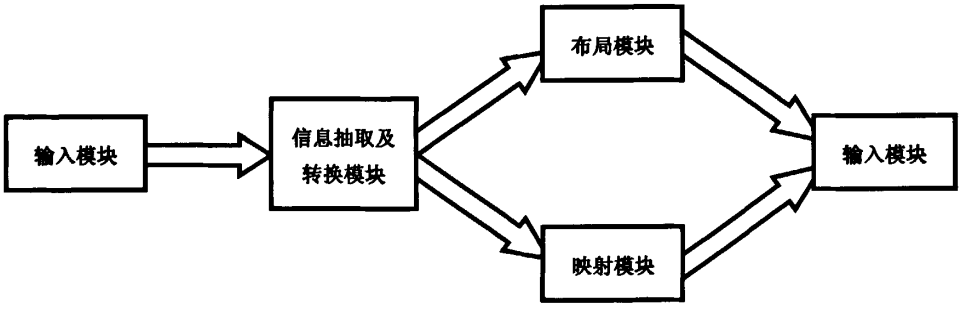


图 5.1　层次有限状态机模型可视化工具架构图

输入、输出模块：即用户交互界面，指引用户导入既定格式的层次有限状态机模型的文本文件，并读取其中的文本信息，暂存于某缓冲区内，用以后续的相关信息抽取及转换工作。另外，负责在界面的某一区域范围内向用户呈现最终生成的图形。

信息抽取及转换模块：分析获得的文本信息，提取其中有助于描述层次有限状态机的部分，并将这部分文本转化成为某种数据结构存储于计算机中。

这一模块的实现首先要确定用于存储层次有限状态机的数据结构。通过分析层次有限状态机的数学模型，我们给出了相应的层次有限状态机的存储与分析方法。在实际代码实现中，我们将顺序有限状态机写为一个类，该类成员包括了顺序状态机的所有基本元素：状态及迁移，而状态与迁移也各被写为一个类，各自将其相关属性定义为成员，如状态类中有状态名、状态类型、判断状态是否为复合状态的变量等，迁移类中有迁移对应的源状态、目标状态、标签等。顺序有限状态机对应的类写有对其任意实例进行修改的操作，如添加、修改或删除状态与迁移信息，并调用状态与迁移类中的操作以辅助实现。在层次有限状态机的类中，root 为一个固有变量，其意义与上述层次有限状态机的定义一致。其它成员包括一个顺序有限状态机的集合与复合状态到顺序有限状态机的映射集合。该类中的操作主要是对其实例信息的添加与读取，如为某状态添加一个到顺序有限状态机的映射，读取某复合状态对应的顺序有限状态机信息等，这些操作事实上实现了一个层次有限状态机实例的构建与部分信息提取，为可视化工作铺平道路。

然后，在描述层次有限状态机的自定义 XML 格式的基础上，利用对标签信息的定位，识别不同层次下的状态与迁移，并依此构建层次状态机模型的数据存储结构。由于其层次不可知，且标签呈现嵌套形式结构，故采用递归算法，对于每一个状态标签，寻找其下的子标签，并将读取子标签中的相关信息，添加至数据结构中，直到当前标签无子标签，算法终止，以下便是该方法的两个核心算法片段：

```
public   Logical load(Node node){
         Logical MM = new Logical();
         String initial = "";
         NodeList States=node.getChildNodes();
         for(int i=0;i<States.getLength();i++){
```

```
            Node sub =States.item(i);
            if (sub.getNodeName().equals("state")||
                                sub.getNodeName().equals("final")){
            String state =
              sub.getAttributes().getNamedItem("id").getNodeValue();
               State s = new State(state);
              MM.addState(s); }
            if(sub.getNodeType()==Node.ELEMENT_NODE&&
                        sub.getNodeName().equals("initial")){
              NodeList init = sub.getChildNodes();
              for(int l = 0; l<init.getLength();l++){
                  Node initNode = init.item(l);
                  if(initNode.getNodeName().equals("transition"))
                    initial =
    initNode.getAttributes().getNamedItem("target").getNodeValue();}}}
      MM.makeStart(MM.getState(initial));
       for(int i=0;i<States.getLength();i++){
          Node sub=States.item(i);
          if
(sub.getNodeName().equals("state")||sub.getNodeName().equals("final")){
              ... ...
```

```
public   void loadAll(Node root){
            Logical m = new Logical();
            m = load(root);
            h.addM(m);
        if (root.getNodeName().equals("scxml")){
            h.getRoot().setComposite();
```

```
            h.addStoL(h.getRoot(), m);
            h.addLtoS(m, h.getRoot());}
        else{
                String x
                 = root.getAttributes().getNamedItem("id").getNodeValue();
            State s = h.belongTo(x).getState(x);
            s.setComposite();
            h.addStoL(s, m);}
        NodeList n = root.getChildNodes();
        for(int i = 0; i<n.getLength(); i++){
            Node current = n.item(i);
            if (current.getNodeType()==Node.ELEMENT_NODE&&
           current.getNodeName().equals("state")&&has(current,"state"))
                loadAll(current);}}
```

布局、映射模块：安排各节点在图形显示区域内的位置，尽量避免边的交叉，注意对称等审美标准，使得呈现出的整体效果达到最大程度的美观。然后，将图形描述文档中的文本元素转化成为相应的图形元素，结合布局模块得出的结果，确定最终将产生的整体图形。

本文借助一个成熟的画图工具 Graphviz 对各顺序有限状态机进行可视化，从而实现层次状态机的可视化。Graphviz 的输入文本为 dot 文件，在调用前需要将数据结构中的指定顺序状态机写为 dot 文件格式并作为工具输入启动 Graphviz，最后捕获返回结果，呈现在视图面板中。在显示图形之前，本工具以树的形式将有限状态机模型的层次展现出来，只有复合状态对应的结点才有子结点，并给所有复合结点添加一个鼠标点击事件，在该事件被触发后，相应视图区域内将显示该复合结点对应的顺序有限状态机。

由于要借助 Graphviz 的画图功能，我们首先给出了围绕它的一个 API 类，方便软件调用以及生成按照它规定格式书写的文件，以下是该 API 类中的主要

方法：

public String getDot ()：返回对象的 dot 源码

public void add (String line)：向对象的 dot 源码中加入字符串 line，并不附回车字符

public void addln (String line)：向对象的 dot 源码中加入字符串 line，并附回车字符

public void addln ()：向图形对应的 dot 源码中加入回车字符

public GraphFile (String source , String file)：产生 dot 源码为 source 的图形，并将其保存为名为 file 的图形文件

然后，便可利用这些 API 方法将数据结构中的模型信息写作 Graphviz 能够识别的 dot 文件，再由程序代码自动开启 Graphviz 的执行库，以新生成的 dot 文件为输入，得到其相应的图形类数据流，从而获得一个与原文本模型一致的图片文件，并将其交给输入输出模块，最终呈现给用户。每一个 FSM 模型的图形表示都是对应着一个复合状态的，在捕获树形结构中的哪一个复合状态被点击后，即可按照数据结构的记录找到该复合状态对应的 FSM 模型，按上述方法进行可视化，以下便是实现单个 FSM 模型可视化的主要算法的片段：

```
Public void loadMC(Logical m){
    if (m != null){
        GraphViz gv = new GraphViz();
        gv.addln(gv.start_graph());
        for (int i=0; i<m.numConnections(); i++) {
                Connection c = m.connectionAt(i);
                State origin = c.getFrom();
                State destination = c.getTo();
                String label = c.getTrans();
                String originName = origin.getName();
                String destinationName = destination.getName();
```

```
gv.add(originName+" -> "+destinationName);
gv.add("[ label = \"");
gv.add(label);
gv.addln("\" ]");}
gv.addln(gv.end_graph());
System.out.println(gv.getDotSource());
String fileName = m.getStart().getName();
File out = new File(fileName+".gif");
gv.writeGraphToFile(gv.getGraph(gv.getDotSource()), out);
... ...
```

以上各大模块协同合作，在用户点击可视化相应按键时，获取当前激活文本框中文本的功能；语义抽取模块对所选取的文本信息进行分析，并提取与可视化相关的部分，然后将这些信息存储于相应的数据结构中；布局、映射模块安排各图形元素在整体图形中的相对位置并将数据结构中的文本元素映射为相应的图形元素；输出模块将数据结构对应的树形结构及最终得到的图形显示于用户界面的指定区域内。

我们的可视化支持工具正是在这样的方法指导下实现的，其交互界面与模拟运行结果如图 5.2 与 5.3 所示。在图 5.2 中，工具界面的文本框内是点击“载入模型”功能后显示的模型文本表示，而图 5.3 则是选择“可视化”功能后得到的相应图形。在点击菜单中的“可视化”后，程序自动感知当前处于激活状态的窗口，此例中为“main”选项卡，并在读入及分析窗口中的文本后写入自定义的数据结构，再根据这些数据结构将模型展示为一个树形结构，此例中为图 5.3 中主窗口左侧的树状图形。依照此树形结构，模型的层次及状态间的关系清晰可见，此时便可根据具体需求，点击该树形结构中代表特定复合状态的树结点，并在主窗口中查看其对应 FSM 模型的图形表示，此例中为顶层状态 root 对应的 FSM 模型的图形，是点击名为“root”的树结点后的输出结果。

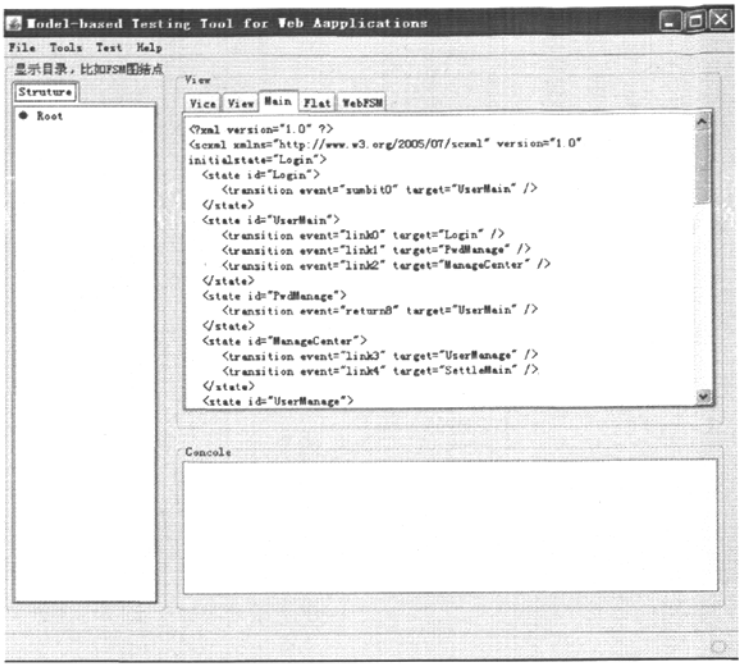


图 5.2　层次有限状态机模型可视化工具文本显示界面

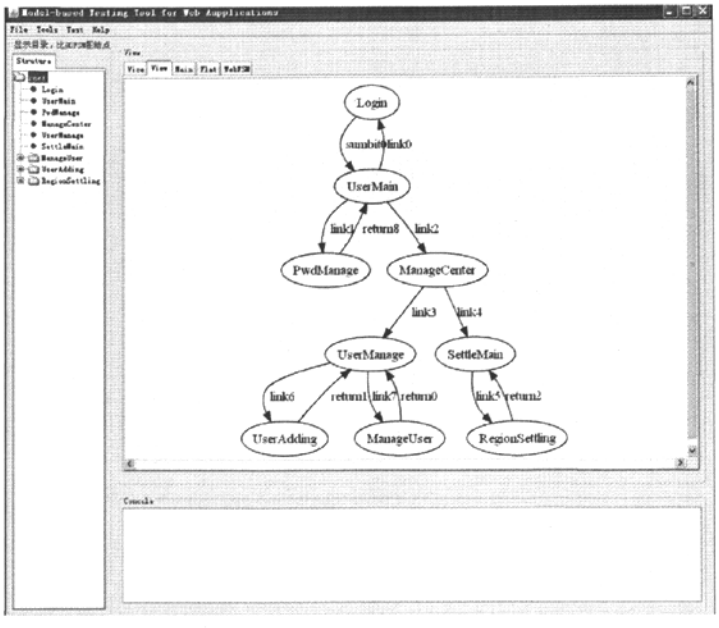


图 5.3　层次有限状态机模型可视化工具图形显示界面

5.2 模型形式化与优化的实现

本文的研究目标是尽量提高 Web 应用检测技术的自动化程度，因此，如何使提出的理论和方法在计算机上自动运行是它们重要的附加价值，运行的效率与结果直接影响着理论与方法本身的优劣，并可证明其可行性。本节针对第三、四章中 UML 模型到 FSM 模型的形式化方法与 FSM 模型的优化方法理论，提出了一种实现方案。由于计算机只能读取并分析文本，而对图形无能为力，所以最初创建的所有图形式 UML 模型必须先转化为与之语义等同的文本格式。本文用的是 XMI 语言。

5.2.1 XMI

XML 元数据交换（XMI）是一种标准，这种标准能够使用 XML（可扩展标记语言）来表达对象[42]。

由于 XML 不是面向对象的，因此需要将对象映射到 XML 以便满足面向对象建模语言与编程语言的数据信息记录要求，也包括 UML。但又因为 XML 的灵活性，可以有多种映射方式，问题就此产生。如果各方利用不同的映射方法将对象映射到 XML，那么他们均无法正确解释其它方的 XML 文档，更无数据共享与交换可言。因此，XMI 提供的对象表示标准将在共同使用该标准的各用户中间有效地实现信息传递。虽然 XMI 能表示对象这样的较复杂元素，但它并非专家才能使用，因为其本质上还是 XML 文档，延续了 XML 的简单易用性，且应对文档改变的能力较强。除以上优势外，XMI 还受到许多成熟工具的支持，它们能够把各种 UML 图转换成为相应的 XMI 文件，例如本文中使用的 UML 建模工具 argoUML，这将大大减少项目的工作量，使得源模型文本的获得更为便捷及可信。

综上，XMI 实为本文中涉及的 UML 模型之理想文本格式。

5.2.2 模型形式化的实现

在 UML 模型的文本语言基础上，先给出第三章中模型形式化机制的实现

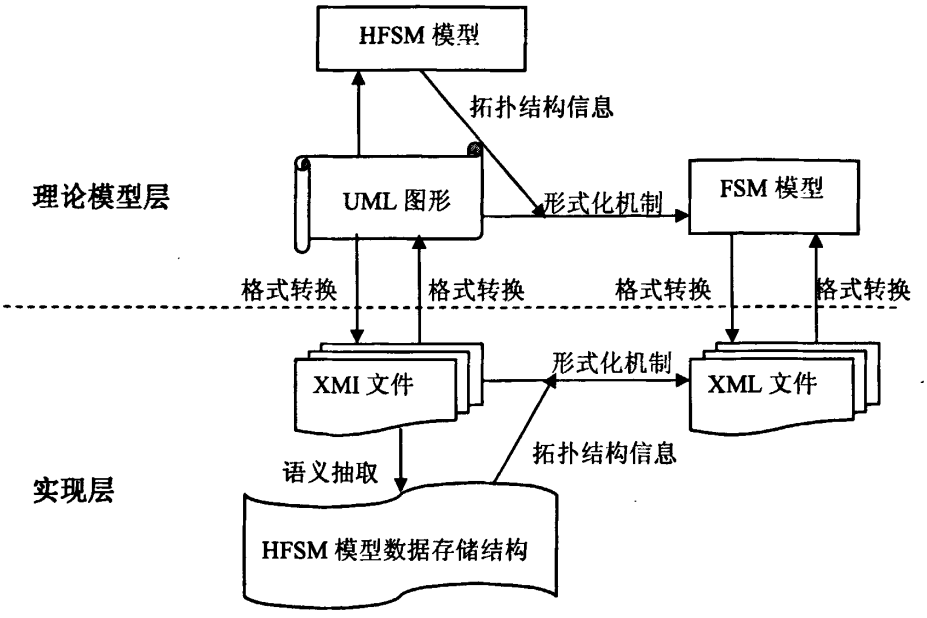


图 5.4 模型形式化实现方案架构图

方案，其大致架构如图 5.4 所示。首先，利用第三方工具 argoUML 将用户创建的 UML 模型转换成为相应的 XMI 文件。由于第三章中的模型形式化内容是具体围绕 UML 状态图展开的，所以在模拟运行中也只是由 UML 状态图出发，获取其对应的 XMI 文件。然后通过编程对生成的 XMI 文件进行自动分析，根据 XMI 文件格式标准，查找模型形式化过程中必要辅助信息对应的标签，并在指定区域范围内捕获原 UML 模型元素值、属性等。在此基础上，可以建立与原模型拓扑结构相一致的 HFSM 模型，用以在模型形式化过程中，提供各状态的层次信息及它们之间的层次关系。借助这些层次相关信息，原模型中各元素的映射规则便可通过代码实现，得到相应的 FSM 模型。最后，将该 FSM 模型写入 XML 文件，以文本格式存储。当然，如果希望以更为直观的方式查看形式化结果，可以激活该 FSM 模型文本所在的文本框，并选择可视化功能。

交互界面与模拟运行结果如图 5.5、5.6 与 5.7 所示。图 5.5 界面中的文本框内是一个 UML 状态图对应的 XMI 文件，由 argoUML 工具的转换功能实现，点击了“transform UML2FSM”选项后，便出现如图 5.6 中的界面，文本框中则是形式化后得到的 FSM 模型对应的 XML 文件，它是按照我们前文中给出的自

定义格式书写的。

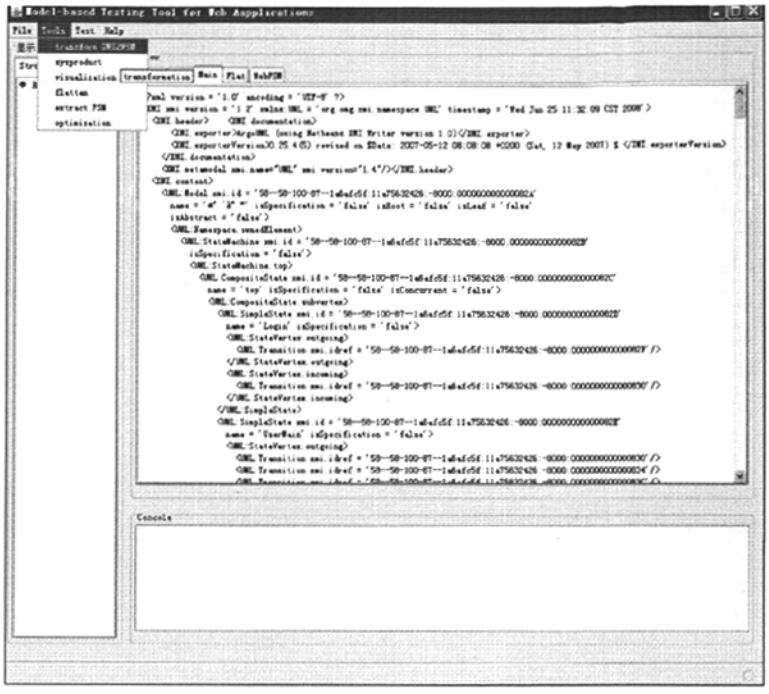


图 5.5　支持工具的模型形式化功能界面

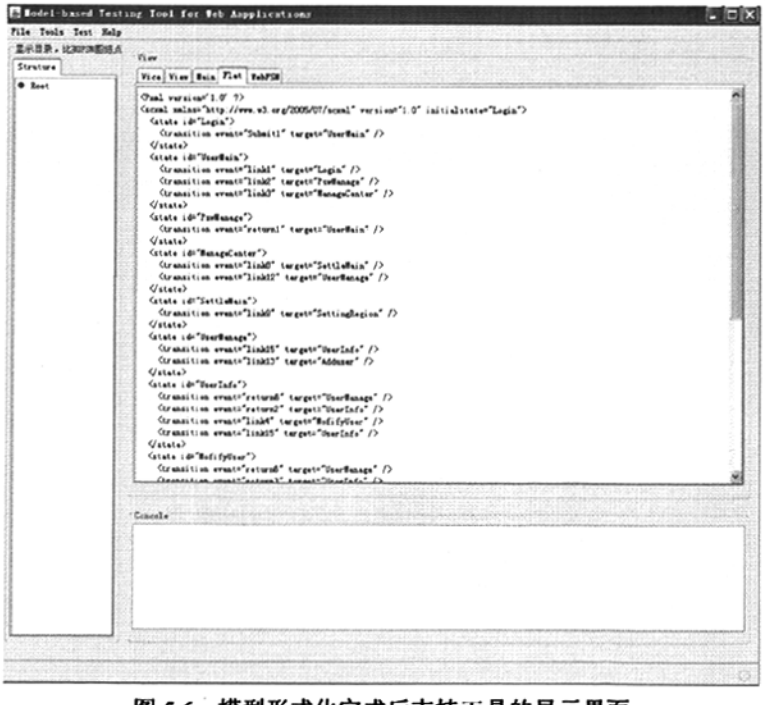


图 5.6　模型形式化完成后支持工具的显示界面

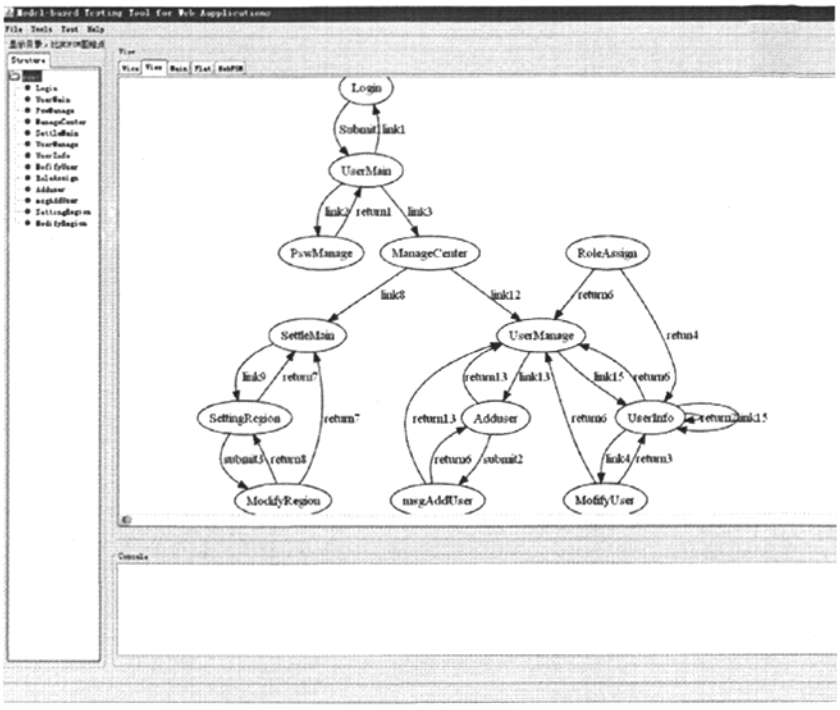


图 5.7　形式化后得到的 FSM 模型进行可视化的显示界面

为了更为清楚地感知形式化的结果，我们对结果 FSM 模型的文本进行可视化，则得到图 5.7 所示的模型树形结构表示及模型图形显示界面，从两者的形态均可以看出，最终得到的 FSM 模型是无层次的，这十分有利于后续的测试用例自动生成工作。

虽然支持工具目前只是针对 UML 顺序图，但该实现方案适用于 UML 的所有图，只是根据不同的图形采用不同的形式化机制，得出各自对应的 FSM 模型，而这些模型的文本与图形表示均相同。

5.2.3 模型优化的实现框架

在获得系统 FSM 模型后，便可执行第四章中提出的 FSM 模型优化方法，其实现的大致架构图如图 5.8 所示。首先，依然借助 argoUML 工具将归约指示的 UML 顺序图模型由图形格式转化成为 XMI 文件，再对 XMI 文件进行归约指示信息的提取，以选定的数据结构存储。然后，按照前文中归约指示的形式化方法，根据顺序图表示的场景类别，运用不同的算法，得到对应的 FSM 模型，

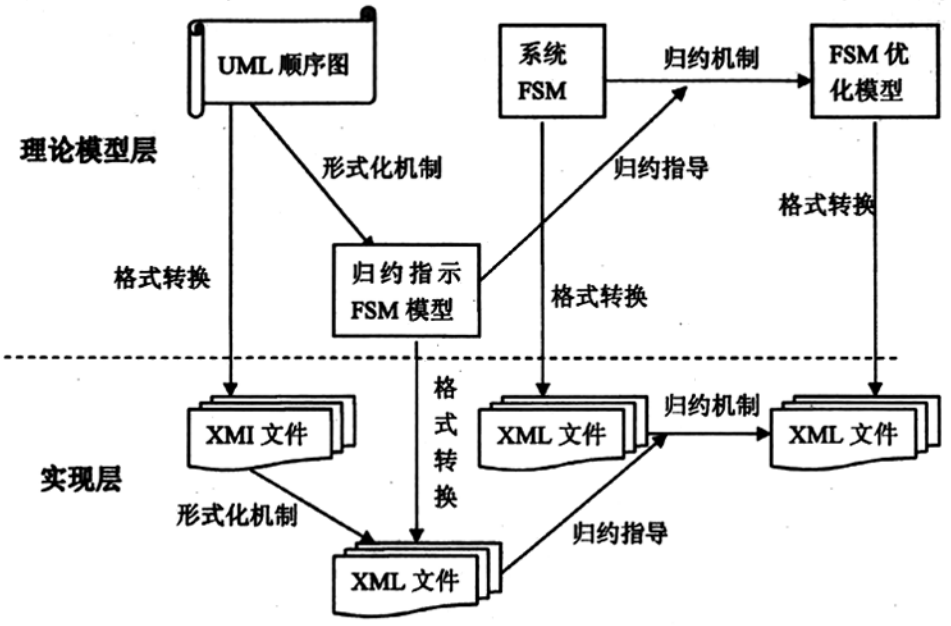


图 5.8　模型优化实现方案架构图

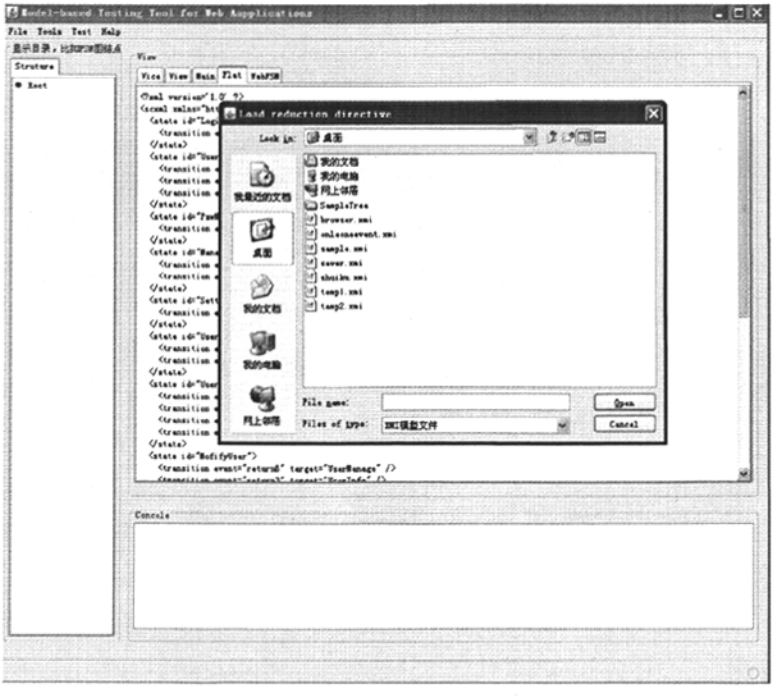


图 5.9　支持工具的模型优化功能界面

该模型以自定义的XML文件格式进行存储。最后，以该FSM模型与系统原FSM模型为输入，执行本文给出的归约算法，对系统原FSM模型进行归约优化，得到描述指定系统功能的更小的FSM模型。在此，我们沿用模型形式化后得到的系统原FSM模型作为原模型来源，结果FSM模型的个数由顺序图代表的可能的消息序列个数决定。

交互界面如图5.9所示，文本框内是选定的系统FSM模型的XML文件，弹出的对话框要求用户选择归约指示对应的XMI文件，以指导优化过程。

这里应该注意到，在形式化归约指示时，实质上是完成了UML顺序图到FSM模型的转换，可以作为一种模型形式化的方法加入，但由于该形式化方法是充分考虑了归约过程的结果，所以它是否能完全满足模型形式化及测试用例自动生成的要求仍需要进一步研究。

第六章　结论与展望

6.1 结论

软件开发过程中的人为因素势必造成此类产品的质量无从保证，虽然我们不可能将其中存在的问题全部找出，但必要的检测手段仍然能最大程度地减少代码错误带来的巨大损失。随着网络时代的到来，软件形态正发生着重大转变，Web 应用软件的出现与普及使得软件产品由原来的单一化本地化发展成为复杂化多元化，传统的检测手段已无法应对，尤其对于作为重要检测手段之一的测试工作。因此，我们采用了基于模型的测试方法，不仅将测试工作起始时间提前，使测试周期大大缩短，而且使测试的自动化成为可能，将大量单调机械及重复性工作交给计算机完成。另外，对于 Web 应用的多变性十分有效，用户需求的变化反应为规格说明的变化，而与规格说明对应的模型的修改要比实际代码容易得多。

本文便是在此研究背景下，以生成测试用例和模型验证为目的，希望能通过一系列变换与处理，最终获得一种既满足验证要求又方便测试用例生成的模型。取得的主要成果如下：

1) 在已创建的 UML 需求与设计模型的基础上，通过其对应文本，分析其传递的信息，如基本元素信息、层次信息等，将其形式化为 FSM 模型。由于本文是对该模型形式化过程的初探，所以选定描述系统行为的最小单位—UML 状态图作为研究对象，给出了其到 FSM 模型的转换方法及实现。
2) 通过模型形式化过程，可获得系统的 FSM 模型，但由于系统的庞大，FSM 模型往往过于复杂，元素过多，若直接产生测试用例有发生状态爆炸的危险。因此，提出了一种优化原系统 FSM 模型的方法，该方法的信息来源为表示系统行为的 FSM 模型以及表示用户测试意图的归约指示，后者指定需要进行测试的系统功能。为了方便用户的模型创建，

我们使用 UML 顺序图进行归约指示的建模。由于 UML 的半形式化特性，给出了一种 UML 顺序图到 FSM 模型的转换方法，实现归约指示的形式化。而这种方法亦可单独用于 UML 顺序图的模型形式化过程，与 UML 状态图的模型形式化方法一并服务于目标模型的生成。在形式化的归约指示的帮助下，本文提出了一个归约方法，使之作用于系统的原 FSM 模型，最终得到若干投影模型，只包含描述指定的功能的模型元素，使后续检测更为轻松也更具目的性。最后，给出了以上模型优化方法的实现框架。

3) 给出了一种以层次有限状态机模型的文本为输入，其相应图形为输出的转换方法，并实现了相应的工具。它利用有限状态机的图形属性使人们从头疼的符号代码中解脱出来，更为直观地掌握模型架构。该方法利用树形结构对层次有限状态机的层次结构进行描述与显示，使用户了解各状态的层次信息与它们之间的层次关系，并对各层次上的 FSM 分别进行可视化，使用户掌握每个 FSM 的组织结构。另外，由于测试用例是 FSM 模型中的路径，亦可利用该方法实现测试用例的可视化，可更为清晰地看到它们的走向，及其与系统 FSM 模型间的关系。

6.2 展望

通过本文对 Web 应用模型优化与处理的归纳总结，我们认为在该研究领域还存在以下几方面值得进一步扩充和深入：

1) 在对 UML 状态图进行模型形式化时，应考虑其它 UML 图对该形式化过程的影响。要使最终产生的 FSM 模型更为全面的描述系统行为，必须结合各种 UML 模型，让描述对象交互的 UML 图参与 UML 状态图的模型形式化过程。

2) 正像前文中提到的一样，本文只是针对 UML 状态图提出了较为完整的模型形式化方法，UML 顺序图的形式化方法亦是以有利于归约过程为前提条件。而 UML 建模语言提供了多达十几种图形帮助用户进行系统建模，因此，除 UML 状态图以外的其它 UML 图的形式化机制也是未

来的研究工作之一，至于是先统一转换成为 UML 状态图再形式化为 FSM 模型，还是各自直接形式化为 FSM 模型，则需要通过实验判断哪种方法更为高效。另外，多个 UML 状态图产生多个 FSM 模型，而不同 UML 图形也将各自产生 FSM 模型，这些 FSM 模型的合成方法有待考虑。

3) 在进行 FSM 模型的归约时，我们假设代表归约指示的顺序图与系统 FSM 模型无冲突，否则，它们不仅将失去指导意义，而且无法完成用户的测试意图。因此，应该给出一种在已有信息的基础上，检测此种冲突并能修复检测到的冲突的方法，保证归约过程的顺利进行，提高提出的归约方法的通用性。

4) UML 顺序图中的消息的结构与参数本文并未考虑，它们对后续 FSM 模型的归约的影响可能很大，需要进一步研究。

参考文献

[1] Miao, H. and Zeng, H, Model checking-based verification of Web application [C]. Proceedings of the 12th IEEE international Conference on Engineering Complex Computer Systems (ICECCS 2007), July 11-14, 2007., Auckland, pp.47-55

[2] S.R. Dalal, A. Jain, N. Karunanithi, and B.M.Horowitz, Model-based testing in practice [C]. Proceeding of the 21st International Conference on Software Engineering, May 16-22, 1999, Los Angeles, pp. 285-294

[3] Harry Robinson, Graph theory techniques in model-based testing [C]. International Conference on Testing Computer Software, 1999

[4] William E. McUmber, Betty H.C. Cheng, A general framework for formalizing UML with formal languages [C]. Proceeding of the 23rd International Conference on Software Engineering, May 12-19, 2001, Toronto, pp. 433-442

[5] Jeff Offutt, Shaoying Liu, Aynur Abdurazik, and Paul Ammann, Generating test data from state-based specifications [J]. The Journal of Software Testing, Verification, and Reliability, Vol. 13, No. 1, 2003, pp. 25-53

[6] Christopher Jerry Mallery, On the feasibility of using FSM approaches to test large web applications [D]. Doctoral Dissertation, Washington State University, 2005

[7] 曾红卫, Web 应用的验证与测试 [D].博士学位论文, 上海大学, 2008

[8] Alessandra Cavarra, Jim Davies, Language specification [EB]. http://www.agedis.de/

[9] Rick Hower, Web site test tools and site management tools [EB]. Software QA and Testing Resource Center. http://www.softwareqatest.com/qatweb1.html

[10] Belinfante, L. Frantzen, and C. Schallhart, Model-based testing of reactive systems [M]. Heidelberg: Springer Berlin, 2005, 7: 391–438

[11] Utting, M., Pretschner, A., Legeard, B., A taxonomy of model-based testing [R]. New Zealand: Department of Computer Science, The University of Waikato, April 2006

[12] I.K. El-Far and J.A. Whittaker, Encyclopedia of software engineering [M].

Wiley-InterScience, 2002: 825–837

[13] M. Blackburn, R. Busser, and A. Nauman, Why model-based test automation is different and what you should know to get started [C]. Proceedings of the International Conference on Practical Software Quality and Testing, March 22-26, 2004, Washington, D.C

[14] Huaikou Miao, Ling Liu, A test class framework for generating test cases from Z Specifications [C]. Proceeding. of 6th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS2000), September 11-15, 2000, Tokyo

[15] Ling Liu, Huaikou Miao, A framework for specification-based class testing [C]. The 8th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2002), December 2-4, 2002, Greenbelt

[16] Ling Liu, Huaikou Miao, A specification-based approach to testing polymorphic attributes. The 6th International Conference on Formal Engineering Methods (ICFEM 2004), November 8-12, 2004, Seattle

[17] Huaikou Miao, Zhongsheng Qian, Bo Song, Towards automatically generating test paths for Web application testing [C]. 2nd IEEE International Symposium on Theoretical Aspects of Software Engineering(TASE 2008), June 17-19, 2008, Nanjing, pp. 211-218

[18] B. Legeard, F. Peureux, and M. Utting, Controlling test case explosion in test generation from B formal models [J]. The Journal of Software Testing, Verification and Reliability, Vol. 14, No. 2, 2004, pp. 81-103

[19] A. Pretschner, H. Lötzbeyer, J. Philipps, Model based testing in evolutionary software development [C]. 12th IEEE International Workshop on Rapid System Prototyping, June 25-27, 2001, Monterey, pp. 155-161

[20] T. Isakowitz, E. A. Stohr, and P. Balasubramanian, RMM: A methodology for structured hypermedia design [J]. Communication of the ACM, Vol. 38, No. 8, 1995, pp. 34-44

[21] F. Coda, C. Ghezzi, G. Vigna, and F. Garzotto, Towards a software engineering approach to web site development [C]. Proceedings of 9th International Workshop on Software Specification and Design, April 16-18, 1998, Ise-Shima

[22] Miao, H., Qian, Z., and He, T, Modeling Web browser interactions using FSM [C].

Proceedings of the 2nd IEEE Asia-Pacific Service Computing Conference, December 11-14, 2007, Tsukuba Science City, pp.211-217

[23] H. Gellersen and M. Gaedke, Object-oriented web application development [J]. IEEE Internet Computing, Vol. 3, No. 1, 1999, pp. 60-68

[24] J. Conallen, Modeling web application architectures with UML [J]. Communications of the ACM, Vol. 42, No. 10, 1999, pp. 63-70

[25] Huaikou Miao, Shengbo Chen, Huanzhou Liu, Zhongsheng Qian, An approach to generating test cases for testing component-based Web applications [C]. Workshop on Intelligent Information Technology Application (IITA'07), December 2-3, 2007, Zhang Jiajie, pp. 264-269

[26] D.C. Kung, Chien-Hung Liu, Pei Hsia, An object-oriented web test model for testing web applications [C]. Proceedings of The First Asia-Pacific Conference on Quality Software, October 30-31, 2000, Hong Kong, pp. 111

[27] F. Ricca, Tonella P, Analysis and testing of web applications [C]. Proceedings of the 23rd International Conference on Software Engineering, May 12-19, 2001, Toronto, pp. 25-34

[28] The precise group [EB]. http://www.cs.york.ac.uk/puml/

[29] Rafael Magalhães Borges, Alexandre Cabral Mota, Integrating UML and formal methods [J]. Electronic Notes in Theoretical Computer Science, Vol. 184, 2007, pp. 97-112

[30] Latella D, Majzik I, Massink M, Automatic verification of a behavioral subset of UML statechart diagrams using the SPIN model-checker [J], Formal Aspects of Computing, Vol. 11, No. 6, 1999, pp. 637-664

[31] Traore I, An outline of PVS semantics for UML statecharts [J], Journal of Universal Computer Science, Vol. 6, No. 11, 2000, pp. 1088-1108

[32] H.Storrle, Semantics of interactions in UML 2.0 [C]. Proceedings of the 2003 IEEE Symposium on Human Centric Computing Languages and Environments, Auckland, October 28-31, 2003, pp. 129-136

[33] Ø.Haugen, K.E.Husa, STAIRS towards formal design with sequence diagrams [J]. Software and Systems Modeling, Vol. 4, No. 4, 2005, pp. 355–357

[34] M.V.Cengarle, A.Knapp, UML 2.0 interactions: semantics and refinement [C]. Workshop on Critical Systems Development with UML, Lisbon, October 11-15, 2004, pp. 85-99

[35] J.P.Katoen, L.Lambert, Pomsets for message sequence charts [C]. Workshop of the SDL Forum Society on SDL and MSC, Berlin, June 29- July 1, 1998, pp. 291

[36] Demissie Bediye Aredo, Semantics of UML sequence diagrams in PVS [C]. UML2000 Workshop on Dynamic Behavior in UML Models: Semantic Questions, York, October 2-3, 2000

[37] Xiaoshan Li, Zhiming Liu, He Jifeng, A formal semantics of UML sequence diagram [C]. Australian Software Engineering Conference, Melbourne, April 13-16, 2004, pp. 168

[38] Erich Mikk, Yassine Lakhnech, and Michael Siegel, Hierarchical automata as model for statecharts [C], Proceeding of the 3rd Asian Computing Science Conference on Advances in Computing Science, December 9-11, 1997, Kathmandu, pp. 181-196

[39] Whittle J, Schumann J, Generating statechart designs from scenarios [C]. Proceedings of The 22nd International Conference on Software Engineering, Limerick, June 4-11, 2000, pp. 314-323

[40] Whittle J, Araújo J, Toval. A, Rigorously automating transformations of UML behavior models [C]. UML2000 Workshop on Semantics of Behavioral Models: Semantic Questions, York, October 2-3, 2000

[41] T. Kameda, On the reduction of non-deterministic automata [D]. Doctoral Dissertation, Princeton University, 1968

[42] Tsunehiko Kameda, Peter Weiner, On the state minimization of nondeterministic finite automata [J]. IEEE Transactions on Computers, Vol. C-19, No. 7, 1970, pp. 617-627

[43] Maurizio Damiani, The state reduction of NFSM [J]. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, Vol. 16, No. 11, 1997, pp. 1278-1291

[44] G.Cabodi, S.Quer, Extending equivalence class computation to large FSMs [C]. Proceedings of the 1995 Computer Design: VLSI in Computers and Processors, Austin, October 2-4, 1995, pp. 258

[45] Gordon Fraser, Martin Weiglhofer, Coverage based testing with test purposes [C].

Proceedings of the Eighth International Conference on Quality Software, Oxford, August 12-13, 2008, pp. 199-208

[46] Hsiou-Wen Hsueh, Test directive generation for functional coverage closure using inductive logic programming [C]. Proceedings of High Level Design Validation and Test Workshop, Monterey, November 8-10, 2006, pp. 11-18

[47] Alessandra Cavarra, Jim Davies, Language Specification [EB]. http://www.agedis.de/

[48] C.Jard, T.Jeron, TGV: theory, principles and algorithms: A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems [J]. International Journal on Software Tools for Technology Transfer, Vol. 7, No. 4, 2005, pp. 297-315

[49] Schneider. A, Walter. S, Langer. J, Heinkel. U, Automatic visualization of abstract system specifications [C]. Proceedings of the Sixth International Conference on Quality Software, Beijing, October 26-28, 2006, pp. 167-174

[50] Timothy J. Grose, Gary C. Doney, Stephen A, Brodsky. Mastering XMI: Java Programming With XMI, XML, and UML [M]. John Wiley&Sons, Inc. 2002

[51] Spec Explorer Website http://research.microsoft.com/specexplorer/

[52] Eades P. A, Heuristic for graph drawing. Congressus Nutnerantiunt, Vol. 42, 1984, pp. 149-160

[53] Kamada T, Kawai S, An algorithm for drawing general undirected graphs [J]. Information Processing Letters, Vol. 31, No. 1, 1989, pp. 7-15

[54] Cui Meng, Yuan Hai, A method of transforming UML sequence diagram to statechart diagram based on MDA [J]. Journal of Nanjing University (Natural Science), Vol. 40, No. 4, 2004, pp. 471-482

[55] 赖明志, 尤晋元, 使用时间化自动机形式化带有时间扩展的 UML 状态图[J]. 计算机应用, 2003, 23 (8): 4-6

[56] 胡军, 于笑丰, 张岩, 基于场景归约的构件式系统设计分析与验证[J]. 计算机学报, 2006, 29 (4): 513-525

[57] 胡军, 于笑丰, 张岩, 基于场景构件式实时软件设计的一致性检验[J]. 软件学报, 2006, 17 (1): 48-58

[58] Grady Booch, James Rumbaugh, Ivar Jacobson. The Unified Modeling Language User Guide [M]. Addison-Wesley, 1998

[59] Michael Sipser, Introduction to the Theory of Computation [M]. PWS Publishing Company, 1997

[60] de Alfaro L, Henzinger TA, Interface automata [C]. Proceeding of the 9th Annual ACM Symposium, September 10-14, 2001, Austria, pp. 109-120

作者在攻读硕士学位期间公开发表的论文

[1] Xi Wang, Liang Guo, Huaikou Miao, An approach to transforming UML model to FSM model for automatic testing [C]. Proceedings of 2008 International Conference on Computer Science and Software Engineering, Wuhan, December 12-14, 2008, pp. 251-254

[2] Xi Wang, Huaikou Miao, Liang Guo, Towards automatic transformation from UML model to FSM model for web applications [J]. Journal of Software Engineering and Applications, Vol. 1, No. 1, 2008, pp. 7-15

[3] 郭亮，缪淮扣，王晢，陈圣波，UML 模型到 FSM 模型的转换 [J]. 计算机科学，已录用

作者在攻读硕士学位期间所参加的项目

[1] 国家自然科学基金项目“基于规格说明的 Web 应用测试方法研究”（No. 60673115）

[2] 国家高技术研究发展计划（863 计划）“基于模型的 Web 应用测试技术与工具研究”（No. 2007AA01Z144）

[3] 国家重大基础研究（973）项目“基于 OWL-S 的服务需求规格的检测、验证与工具研究”(No. 2007CB310800)

致　谢

感谢导师缪淮扣教授的悉心指导，正是缪老师的亲切关怀与严格要求，使本论文如期完成。缪老师定期询问课题进度与可能存在的问题，并给予我们学术上的帮助。而缪老师严谨的治学态度与勤勉负责的工作态度更让我们懂得作为一名研究者所应具备的优良品质，这将由我们一直延续下去。在此，谨向缪老师致以深深的敬意和由衷的感谢。

感谢所有帮助过我的老师、同学和朋友，尤其是同一课题组的老师和同学们，在无以计数的讨论会中迸发的智慧火花点燃了我的灵感，引领我进入一个又一个学术新世界。曾红卫老师在课题的研究过程中给予了积极的帮助，经常组织讨论会，与同学们共同解决各阶段面临的问题，并在仔细阅读本文后，提出了宝贵中肯的意见与建议，使我受益匪浅。

最后，感谢我的父母，在我挫折失意时给予我支持与鼓励，轻声的问候与温暖的话语激发出无穷的力量，抵御一切失落与打击；在我骄傲得意时提醒我潜在危机与不利，善意的批评与中肯的建议调节着骄躁的心态，预防任何自满与放任。在此，表达我一如既往的感谢。