支持 MDA 的交互式需求获取方法及辅助工具*)

鱼 滨[1] 张 琛[1] 郝克刚[2]
(西安电子科技大学 西安 710071)[1] (西北大学 西安 710127)[2]

摘 要 为支持 MDA 的开发模式,本文提出了表格驱动的交互式需求获取方法并开发了相应的支持工具。该方法通过填写参与者表、非功能性需求表、用例卡、CRC 卡以及用户反馈表来得到用户需求,为建立计算独立模型 CIM 和平台独立模型 PIM 提供足够的信息。

关键词 模型驱动架构,用例驱动,需求获取,计算独立模型,平台独立模型

Interactive Requirements Elicitation Methods to Support MDA and Assistant Tools

YU Bin[1] ZHANG Chen[1] HAO Ke-gang[2]
(Xidian University, Xi'an 710071, China)[1] (Northwest University, Xi'an 710127, China)[2]

Abstract To support the development mode of MDA, this paper presents the tables-driven of interactive requirements elicitation methods and developes the supporting tools. This approach gets the customer demands by completing the participator table, the non-functional requirements table, use case cards, CRC cards and user feedback table, provides sufficient information for the establishment of calculation independent model CIM and the platform independent model PIM.

Keywords MDA, Use case-driven, Requirements elicitation, CIM, PIM

1 引言

在软件系统开发过程中,有效的需求工程对项目的成败至关重要。软件需求的获取、分析、协商、文档化是需求工程的 4 个主要步骤。在实际开发中,特别是在基于敏捷软件开发方法学的软件项目中,这个过程往往是相互交织、迭代、增量进行的。需求获取是其中一个非常关键、重要却又较困难的活动,通常与需求分析过程交融在一起。需求获取的主要任务是列出候选需求、理解系统的语境、捕获功能性需求、捕获非功能性需求。成功的软件产品建立在成功的需求基础之上,而高质量的需求源于用户与开发人员之间的有效沟通与合作。

MDA(Model Driven Architecture,模型驱动架构)是 OMG(The Object Management Group,国际对象管理组织)提出的新的方法学,是一种新的系统开发方法,它强调整个系统的开发过程由对软件系统的建模行为驱动,完成系统需求分析、架构设计、构建、测试、发布和运行维护工作。MDA 根据不同的抽象层次,定义了 CIM(Computation Independent Model,计算独立模型),即通常所说的业务模型、PIM(Platform Independent Model,平台独立模型)和 PSM(Platform Specific Model,平台相关模型)[1]。不同模型间可通过模型转换技术实现相互转化。在 MDA 开发过程中,系统的业务模型和实现技术是相分离的,业务模型具有持久价值,但是 MDA 中并未包含前端的需求获取。本文就是针对这一问题提出的。

本文阐述了表格驱动的交互式需求获取方式及其在模型驱动开发过程中的重要意义。第 2 节着重讨论了交互式获取方法所使用到的全部表格以及他们的作用;第 3 节描述了需求的模型化表示,以及计算无关模型和平台无关模型的建立;第 4 节简单介绍了需求获取工具的特点;最后对本文工作进行了总结。

2 表格驱动的需求获取方法

软件需求可以定义为:用户解决某一问题或达到某一目标所需的软件功能;系统或系统构件为了满足合同、规约、标准或其他正式实行的文档而必须满足或具备的软件功能。传统的软件开发模式,将系统需求的获取与向形式化规范的转换看作两个完全独立的部分,将需求获取阶段的结束看作向形式化规范转换的起点。

2.1 需求获取技术

需求获取,属于软件工程中的一部分,包括需求来源和获取需求的技术[2]。它是软件开发的第一阶段,其本质主要是人的活动,涉及软件设计人员如何与客户建立有效的沟通。本文详细介绍了支持 MDA 的交互式需求获取方法,通过填写表格的方式能够快速、有效地得到准确的用例描述,从而获得完整的系统需求。

在需求获取过程中,我们可以采用多种不同的技术进行业务理解和信息收集,常见的需求获取技术包括面谈和问卷调查、需求专题讨论会、观察用户工作流程、基于用例的方法[3]、原型化方法等,而选择这些技术需要根据应用类型、开发团队技能、用户性质等因素来决定。随着面向对象技术的发展,基于用例的方法在需求获取和建模方面应用得越来越普遍。这种方法是以任务和用户为中心的,可以使用户更清楚地认识到新系统允许他们做什么。另外,用例有助于开发

*) 基金项目:国家自然科学基金重点项目"框架时序逻辑程序设计"(60433010);十一五装备预先研究项目"高可信软件开发方法及支持环境"(513150501)资助。鱼 滨 博士,副教授,CCF 会员,主要研究领域为基于 Internet 的软件方法与技术;张 琛 博士生,主要研究领域为软件开发方法学、形式化方法。

人员理解用户的业务和应用领域，并可以运用面向对象分析和设计方法将用例转化为对象模型[4,5]。

以用例为中心的需求获取与建模技术的完美结合[6]，为模型驱动的开发模式提供了有利的前提条件。本文通过所设计的交互式图形界面，以提问方式与用户沟通，获取需求、建立CIM模型。界面的设计能够引导用户描述需求及建立PIM模型，并便于向PIM模型转换。需求获取与模型之间的映射关系如图1所示。

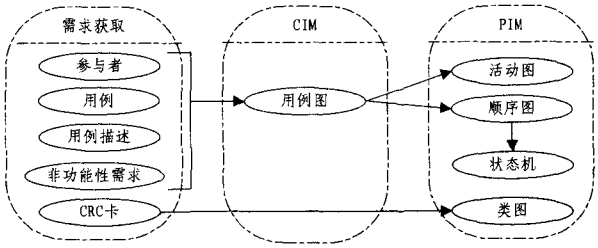

图1 需求获取与模型之间的映射关系

2.2 交互式表格

软件的需求可以分为功能性需求和非功能性需求两个层次。功能性需求定义了开发人员必须实现的软件功能，使得用户能完成他们的任务，从而满足了业务需求，用例模型最能够说明这些需求。软件的非功能性需求包括产品必须遵从的标准、规范和合约；外部界面的具体细节；性能要求；设计或实现的约束条件及质量属性等内容。有些非功能性需求可能是很一般的，不会纳入任何一个用例，这些内容应该独立地作为辅助需求列出来。本文通过填写五种类型的表格来获取详细的系统需求，其中包括：参与者表、非功能性需求表、用例卡、CRC(class，responsibility，collaborator)卡以及用户需求反馈表。

参与者表记录了用例的全部参与者，该表包括参与者名、参与者的简单描述以及该参与者涉及的所有用例。非功能性需求表描述了系统的性能、速度、效率、可用性、可靠性、准确性、响应时间、恢复时间、界面友好性等。不同软件在性能要求上既有相似之处，又有其独特性，所以非功能性需求表的制定应根据不同类型软件的自身特点进行调整。用例卡记录了单个用例的详细信息，其中包括用例表中所涉及的全部内容以及用例之间的关联关系。通过对功能性需求与非功能性需求的分析，将抽取的类信息保存在CRC卡中。一个CRC卡片代表一个独立对象的实例。卡片上记录有类的名称，此类所要实现的功能，以及与此类相关的其它类的名称。

2.3 词汇表

词汇表主要用于定义项目特定的术语，它有助于开发人员对项目中所用术语有统一的理解和使用，以便用户可以正确地解释软件需求规格说明，也是后续阶段中进行对象抽象的基础。在用户与开发人员的沟通过程中，并非每个客户或者项目小组成员都能够明白“用户”、“角色”、“用例”之间的差别，也不见得都能很好地理解“通道”、“管道”、“消息”到底是什么含义，为了让项目模型文档使每个浏览者正确地理解，同时帮助客户更好地利用信息化工具提高工作效率，定义词汇表是非常需要的。

2.4 用例表

用例图只是简单地用图形化描述系统需求，但对于每个用例，还需要有更加详细的说明，这样才可以使其他人对系统有更加深入的了解，因此我们需要写用例描述。在UML2.0[7]中对用例描述有详细的介绍，如表1所示。

表1 用例描述表

用例描述的细节	对用例的详细描述以及用例有用性的说明
用例名称	用例定义的名字
相关需求	用例所能够描述的部分或完整的需求。
全局目标	用例在系统中的作用以及该用例的重要性。
前置条件	在用例启动时，参与者与系统应置于什么状态，这个状态应该是系统能够检测到的、可观测的。
完成目标的结束条件	当用例成功执行时系统所处的状态。
未能完成目标的结束条件	当用例未能成功执行时系统所处的条件。
主要的参与者	参与用例的主要活动者。通常包括用例的触发者或者用例执行后直接获得消息的参与者。
次要的参与者	在用例执行过程中参与用例执行，但非主要参与者。
触发器	由导致用例执行的参与者所触发的事件。
主流程	对用例中常规、预期路径的描述。
扩展流程	主要是对一些异常情况、选择分支进行描述。

在用例的抽取过程中，需要注意：用例必须是由某一个主角触发而产生的活动，即每个用例至少应该涉及一个主角。如果存在与主角不进行交互的用例，就可以考虑将其并入其他用例；或者是检查该用例相对应的参与者是否被遗漏，如果是，则补上该参与者。反之，每个参与者也必须至少涉及到一个用例，如果发现有不与任何用例相关联的参与者存在，就应该考虑该参与者是如何与系统发生对话的，或者由参与者确定一个新的用例，或者该参与者是一个多余的模型元素，应该将其删除。

2.5 用户需求反馈表

在系统分析员对所调研的需求进行填表与分析之后，需求获取工作暂告一段落，此时需要用户对需求进行评估，给出反馈意见。本文仍采用表格驱动的方式，邀请用户填写需求反馈表，对描述需求的每一个用例进行确认，并根据权重给出相应的分数。分数的高低直接体现了该用例所描述的需求是否满足用户的期望。最后用户还需给出用例的优先级和用例变更的可能性，1至5表示了优先级由低向高变化，高、中、低三个等级反映了一个用例变更的可能性大小，如表2所示。

表2 用户需求反馈表

用例描述项	权重	用户打分	修改意见	用例优先级	用例变更可能性
需求描述(准确性、一致性、完整性)	30%				
目标	20%				
参与者	20%				
前置条件/后置条件	20%				
流程	10%				
总分	100%			优先级共分5级	高、中、低三等

针对不同用例的特点，可以调整用例描述项的权重。在需求获取过程中，优先级高的用例往往是用户当前最关注的需求，此类需求面对时间紧、质量要求高等问题需要优先处理。

3 模型表示

3.1 生成用例图

在参与者表、非功能性需求表、用例卡填写完成之后，工具自动将这些信息保存在多个xml文件中，这些文件的组织关系文档如图2所示。生成用例图所需的信息均从这些xml文件中解析、读取。需求获取表格向用例图的转换如图3所示。用例图中的参与者信息来自于描述参与者表的xml文件，而对用例本身所表示的信息以及用例之间的关联关系均来自描述用例卡的xml文件，非功能性需求表提供用例图中显示的一些约束信息。

```
<?xml version="1.0" encoding="GB2312" standalone="yes" ?>
  <Requirement>
    <Function-rq>                    //功能性需求
      <Actors />                     //参与者
      <Use-cases />                  //用例
      <Usecase-Card />               //用例卡
    </Function-rq>
    <Non-Function-rq />              //非功能性需求
    <CRC />                          //CRC 卡
</Requirement>
```

图 2　组织关系文档

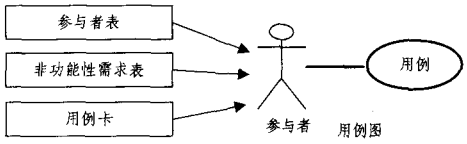

图 3　需求获取表格向用例图的转换

3.2　生成类图

将从参与者表与用例卡中获取的功能性需求，以及非功能性需求表中得到的信息进行分析，可抽取获得若干类，这些类信息是保存在 CRC 卡中的。CRC 卡中的信息，系统仍以 xml 的格式保存，主要包括了类的职责和类之间的协作关系，如图 4 所示。生成类图所需的内容就从这个 xml 文件中抽取，若信息不足时可查询保存用例卡信息的 xml 文件。

```
<?xml version="1.0" encoding="GB2312" standalone="yes" ?>
   <CRCCards>                              //CRC 卡
    <ClassName> </ClassName>               //类名
    <Responsibilitys>
      <Responsibility> </Responsibility>   //职责
    </Responsibilitys>
    <Collaborations>
      <Collaboration> </Collaboration>     //协作
    </Collaborations>
</CRCCards>
```

图 4　保存 CRC 卡的 xml 文件

3.3　向 CIM 和 PIM 转换

通过需求获取工具的引导，完成所有表格的填写之后得到详细的用户需求。全部信息都以 xml 格式保存在系统中。获取工作结束之后，工具将得到的所有内容精加工，抽取生成业务模型和平台独立模型所需的信息，实现用例图以及类图的自动生成，完成了模型驱动开发过程的第一阶段。为第二阶段业务模型向活动图、顺序图、状态机等平台独立模型的自动化转换提供了基础，同时也完善了从前端的需求获取到模型驱动开发的整个流程。

表示系统业务模型的用例图建成之后，协同用例卡中的详细信息，通过预定义的转换规则生成活动图、顺序图，而状态机的生成依赖于顺序图。活动图能够将用例卡中每一阶段的信息直观生动地表现出来，特别是一些分支节点。因此活动图的建立依赖于用例卡中存在的大量信息。参考文献[8]详细介绍了用例向顺序图的转换过程。

3.4　生成需求文档

软件需求文档是对系统开发者要求的正式陈述。它包括系统的用户需求和一个详细的系统需求描述。许多大型机构，例如美国国防部、IEEE，已经为需求文档定义了标准[9]。它的结构包括绪言、引言、术语、用户需求定义、系统体系结构、系统需求描述、系统模型、系统进化、附录、索引等部分。用户需求可通过自然语言的形式给出，而作为软件开发者使用的系统需求描述，在模型驱动的开发模式中可借助业务模型表示。本文参考 IEEE 标准提供的需求文档结构，定义了需求文档模版。在需求获取工作完成之后将会自动生成用户需求文档，该文档也将作为需求获取阶段的里程碑。

4　支持工具

为支持本文介绍的交互式需求获取方法，在开源的 Eclipse 平台下开发了需求获取工具，该工具作为一个 Eclipse 插件使用。用户或系统分析员在需求调研与分析完成之后填入需求描述，根据所填写的内容，使用表格驱动的获取技术，将用户需求转化为相应的用例驱动技术元素，比如：参与者、用例。通过工具自动生成相应的 xml 文档保存用例信息，以供进一步模型建立与转换的使用，以及软件设计与开发过程的应用。图 5 展示了该工具在填写用例卡时所应提供的详细信息。从左边的树状结构可以看到，需求获取类型分为功能性需求、非功能性需求、CRC 卡。中间的用例卡信息表包括了对用例的详细描述，以及用例之间的关联关系。图 6 显示了非功能性需求获取界面，按照粗粒度划分，包括了性能、可靠性、可用性、可支持性、设计属性、设计约束等；按照细粒度的划分，性能又可分为响应时间、硬件配置等诸多元素。

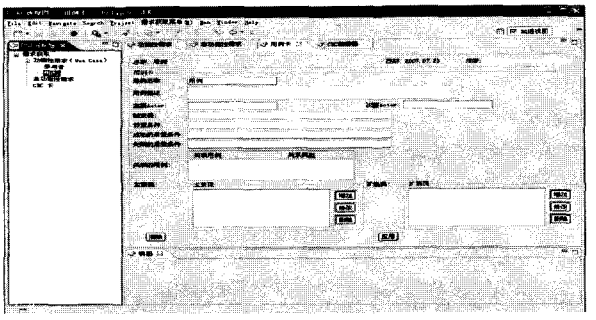

图 5　用例描述界面

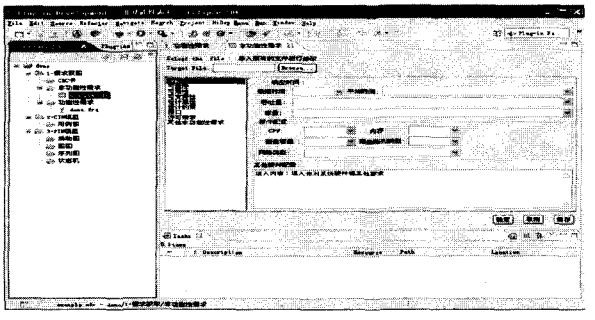

图 6　非功能性需求获取界面

结束语　需求获取是软件开发过程中一个至关重要的阶段。由于人对复杂问题的认识能力有限，不可能在短时间内全面透彻地了解这类问题，而是需要一个渐进反复的过程。此外，由于软件系统的客户与开发人员在背景和知识结构方面的差异，造成了双方沟通的障碍，导致难以获得准确的需求。

本文采用填写表格的交互式手段，借助更易被用户理解的用例驱动的需求获取方式，使开发人员始终站在用户的角度上考虑问题、调研需求，为支持模型驱动的开发模式建立计算独立模型和平台独立模型提供了足够的信息。在进一步的工作中，需要深入研究如何对整个需求获取过程进行度量，并给出模型间转换规则的形式化描述与验证。

参 考 文 献

[1]　Dubielewicz I, Hnatkowska B, Huzar Z. Evaluation of MDA/PSM database model quality in the context of selected non-func-

tional requirements// 2nd International Conference on Dependability of Computer Systems (DepCoS-RELCOMEX'07)
[2] Heise M, Souquières J. A Method for Requirements E-licitation and Formal Specification// Proceedings of the 18th International Conference on Conceptual Modeling. Lecture Notes In Computer Science, 1999: 309-324
[3] Wilson P, Filho P. Quality Gates in Use-Case Driven Development// International Conference on Software Engineering, Proceedings of the 2006 International Workshop on Software Quality. Shanghai, China, 2006: 33-38
[4] Seffah A, Djouab R, Antunes H. Comparing and reconciling usability-centered and use case-driven requirements engineering processes// User Interface Conference, 2001. AUIC 2001. Proceedings. Second Australasian, 2001: 132-139
[5] Behrens T. Capturing business requirements using use cases. Chief Technology Officer, Alpheus Solutions, Dec. 2004. http://www.ibm.com/developerworks/rational/library/dec04/behrens/index.html? S_TACT=105AGX52&S_CMP=cn-a-r
[6] Li Xiaoshan, Liu Zhiming, He Jifeng. Formal and Use-Case Driven Requirement Analysis in UML// 25th Annual International Computer Software and Applications Conference (COMPSAC'01). 2001
[7] Hamilton K, Miles R. Learning UML 2.0. O'Reilly, April 2006
[8] Li Liwu. A Semi-Automatic Approach to Translating Use Cases to Sequence Diagrams// 29th International Conference on Technology of Object-Oriented Languages and Systems. Nancy, France. IEEE Computer Society, June 1999
[9] Sommerville I. Software Engineering (6th Edition). Ian Sommerville. , China Machine Press: 79

(上接第 268 页)

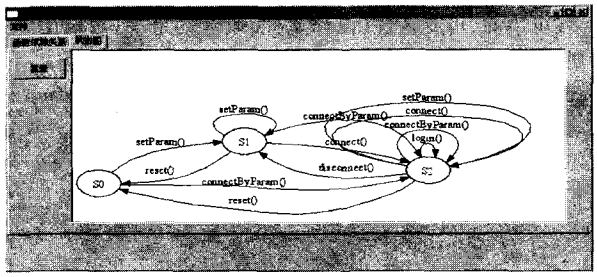

图 4 最终生成的状态图

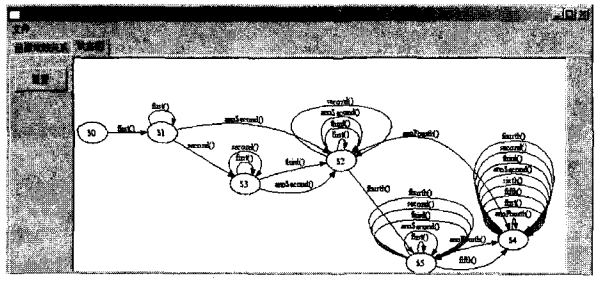

图 5 另一个状态图的例子

4.2 讨论

静态的分析方法对于提取语法级别的依赖来说是十分有效而且准确的。但是我们在实践中也可以明显感觉到静态分析存在的一些弱点。主要的弱点就是静态分析所能提取的信息比较有限，无法对运行时相关的依赖关系进行提取。例如，通过输入参数和一些由分支变量进行控制的状态迁移等语义相关的依赖关系，以及动态绑定等运行时相关的情况，就无法使用静态分析进行处理。因为这些都涉及到运行时变量当时的具体取值，而这些值在编译期是未定义的。但是取值的不同，很可能就导致了程序所采用的行为会有所不同。这些信息将是我们接下来所要关注的。为了弥补静态方法的这些不足，接下来我们会尝试引入一些动态方法进行辅助，比如符号执行(Symbolic Execution)[14]以及程序不变量(Program Invariants)[15]等。这些动态方法也可以提取出前后置条件相关的一些信息。通过动态的执行信息和静态的结构信息相结合，辅助我们的方法，可能提供更强的依赖分析功能。

结束语 本文提出了一种从遗产系统中恢复出程序接口行为协议的方法。最终的接口约束以抽象状态图的形式表示出来。我们的方法通过考察程序中全局变量的共享使用来推测方法之间隐含的依赖关系，恢复出方法调用的行为序列。这是一个自动的静态分析过程，避免了动态方法对测试用例的依赖性。恢复出的抽象状态图保留了协议的约束行为，并减少了复杂度和冗余性。通过实验证明，我们的方法可以有效地对接口方法的协议进行恢复，这对于遗产系统的理解有着重要的帮助。

参 考 文 献

[1] Kim Y, Hong H, Bae D, et al. Test cases generation from UML state diagrams// IEEE Proceedings- Software. 1999, 146(4): 197-192
[2] Briand Y L L C, Penta M D. Assessing and improving state-based class testing: A series of experiments. IEEE Transactions on Software Engineering, 2003, 30(11)
[3] Yang Jinlin, Evans D. Dynamically Inferring Temporal Properties// Proc. the ACM-SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. 2004: 23-28
[4] Yuan Hai, Xie Tao. Automatic Extraction of Abstract-Object-State Machines Based on Branch Coverage// Proceedings of the 1st International Workshop on Reverse Engineering to Requirements at WCRE 2005 (RETR 2005). November 2005: 5-11
[5] Xie Tao, Notkin D. Automatic Extraction of Sliced Object State Machines for Component Interfaces// Proc. 3rd Workshop on Specification and Verification of omponent-Based Systems at ACM SIGSOFT 2004/FSE-12 (SAVCBS 2004). October 2004: 39-46
[6] Corbett J, Dwyer M, Hatcliff J, et al. Bandera: extracting finite-s-tate models from Java source code// 22nd International Conference on Software Engineering. June 2000
[7] Whaley J, Martin M C, Lam M S. Automatic extraction of object-oriented component interfaces// International Symposium on Software Testing and Analysis. July 2002
[8] Olender K M, Osterweil L J. Interprocedural Static Analysis of Sequencing Constraints. ACM Transactions on Software Engineering and Methodology, 1992, 1(1): 21-52
[9] Tang M H, Wang W L, Chen M H. A UML Approach for Software Change Modeling. cs. albany. edu
[10] Abstract state machines and high-level system design and analysis. Theor. Comput. Sci, 2205, 336 (2-3): 205-207
[11] Zhang Yan, Hu Jun, Yu Xiao-feng, et al. 场景驱动的构件行为抽取. Journal of Software, 2007, 18(1)
[12] http://compilers. cs. ucla. edu/jtb/jtb-2003/
[13] https://javacc. dev. java. net/
[14] King J. Symbolic Execution and Program Testing. Communications of the ACM, 1976, 19(7)
[15] Ernst M D, Cockrell J, Griswold W G, et al. Dynamically discovering likely program invariants to support program evolution// IEEE Transactions on Software Engineering. February 2001
[16] Nimmer J W, Ernst M D. Automatic Generation of Program Specifications. ACM SIGSOFT Software Engineering Notes, 2002