基于追踪矩阵获取完整性需求的研究

李 潇 魏长江

(青岛大学数据科学与软件工程学院 山东 青岛 266071)

摘 要 分布式系统自提出以来,逐渐发展成为软件工程中一个重要的研究领域,因此分布性需求成为软件系统的主要特征,同时系统的分布性需求与功能需求又紧密相关。目前,通常使用RUP(Rational Unified Process)推荐的"4+1"视图方法分别将两种需求建模在不同的模型中,此方法在软件工程实践中已经取得了良好的效果,但是也在一定程度上导致了功能需求和分布性需求的分割性,这不利于获取完整的系统需求。针对以上问题,文中首先给出需求追踪的整体框架,从3个层面阐述需求在软件生命周期各个阶段间追踪关系的演变。其次,通过分析需求到其他制品的传播途径,得到需求追踪关系,建立需求追踪矩阵。最后,凭借矩阵计算,描述需求变化追踪的具体实现。通过上述研究,在功能需求模型和分布性需求模型间建立可追踪性链接,不仅能够获取完整性需求,还解决了由需求建模分割性导致的需求变更困难的问题。

关键词 分布性需求,功能需求,RUP,"4+1"视图,需求追踪矩阵,需求变更

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.06.028

Study on Complete Requirement Acquiring Based on Tracking Matrix

LI Xiao WEI Chang-jiang

(Department of Data Science and Software Engineering, Qingdao University, Qingdao, Shandong 266071, China)

Abstract Distributed system has gradually developed into an important research field in software engineering since it was proposed. Nowadays, distributed requirements become the main features of the software systems, and distributed requirements and functional requirements are closely related. Currently, the "4+1" views method recommended by RUP (Rational Unified Process) is usually used to model two kinds of requirements in different models. This method has already produced positive feedback and achieved good results in software engineering practice. However, distributed requirements and functional requirements are modeled separately, which leads to the segregation of functional and distributed requirements to a certain degree. This segmented requirements modeling method is not conducive to obtain complete software requirements when doing requirements analysis work. In response to the above questions, first of all, this paper gave the overall framework of requirements tracking. It illustrated the evolution of requirements tracking relationships across all phases of the software life cycle from three levels. Secondly, by analyzing the transmission route from requirements to other artifacts, requirements tracking relationships were obtained and requirements tracking matrices were established. Finally, with the matrix calculation, the specific implementation of the requirements change tracking was described. Therefore, through the above research, this paper established tracking links between distributed requirements models and functional requirements models eventually, which not only captures requirements completely, but also solves the problem of difficult requirements changes caused by requirements segmentation.

Keywords Distributed requirements, Functional requirements, Rational unified process, 4+1 Views, Requirements tracking matrix, Requirements change

1 引言

要开发出用户满意的软件,则必须要满足用户对系统的所有需求,而不同的系统对需求的内容、种类和形式的要求是有差异的,甚至差别很大。目前,需求工作者通常使用RUP(Rational Unified Process)[1]推荐的"4+1"视图方法,对软件生命周期各个阶段的需求进行建模,从而确保重要的需求被满足。"4+1"视图模型从5个不同的视角对需求分别建

到稿日期:2018-05-22 返修日期:2018-09-10

李 潇(1993-),女,硕士生,主要研究方向为软件需求工程,E-mail:839779044@qq.com;魏长江(1961-),男,博士,教授,硕士生导师,主要研究方向为软件工程、软件需求工程、海量数据处理,E-mail:chj_wei@163.com(通信作者)。

模[2]，其中用例视图侧重于描述系统的功能需求；逻辑视图从对象角度构建对象模型，用以确立逻辑分层、模块划分等；开发视图从开发的角度，描述软件在开发环境下的静态组织；进程视图从过程角度描述系统的并发性和同步性设计；物理视图则从部署角度描述软硬件的映射关系，以及系统在分布/部署上的设计，旨在解决系统部署等问题。由此可见，“4+1”视图方法采用“分而治之”的思想从不同的视角分别建模，不仅使得需求建模过程更科学和明确，还为软件架构的理解、交流和归档提供了方便。

随着软件系统规模和复杂性的不断增大，作为控制软件复杂性、提高软件系统质量的手段之一，分布式系统自提出以来逐渐发展成为软件工程中一个重要的研究领域[3]。分布性需求是系统的主要特征，功能需求的分析离不开分布性，并且与分布性需求的联系越来越紧密。然而，“4+1”视图方法仍然将两种需求分别建模和分析，功能需求和分布性需求建模的分割性导致无法获取完整的需求，从而造成了需求变更困难的问题。

目前，针对需求获取不完整、需求变更困难等问题的解决方法有很多，但被需求工作者广泛使用的是基于需求追踪的方法。国内外需求研究人员对此方法做了大量研究。2010 年，Winkler 等[4]介绍了关于可追踪性的概况研究，以及需求工程和模型驱动开发中可追踪性的发展，让人们开始了解可追踪性的相关知识。2013 年，Cimatti 等[5]提出了验证功能需求的方法，实现了将非正式需求转换为正式、可追踪、自动化、可扩展的需求。2014 年，Goknil 等[6]在处理需求变更问题时详细研究了可追踪性，对需求变更进行了分类、验证。2017 年，Tekinerdogan 等[7]构建了可追踪性元模型，在元模型的基础上分析了可追踪性的关键要求以及可追踪性分析方法。关于获取一致性、易于变更且可追踪的需求，国内也有相关的研究。朱雪峰等[8]对需求的不一致性进行了研究，并提出了一种需求不一致性管理框架和需求建模的方法。王映辉等[9]研究了一种追踪需求变化的方法，他们使用关系矩阵记录功能需求变化传递中的信息，然后通过矩阵运算实现变化信息的过程追踪，并利用可达矩阵辅助确定 SA(Software Architecture)中变化构建所波及和影响的其他构件。综上所述，需求可追踪性方法可以用来保持模型一致性以及支持彼此之间导出的模型间的变化，能够有效解决需求建模的分割性以及需求变更困难的问题。

通过简要分析“4+1”视图方法在需求获取及需求变更方面的缺陷，本文结合需求追踪矩阵方法，在需求模型间建立可追踪关系，并且通过矩阵运算在功能需求模型和分布性需求模型间建立可追踪链接。

2 相关工作的分析与研究

2.1 软件需求的建模与分析

一个软件系统成功与否的衡量标准就是这个软件系统能否很好地满足它的用户以及周围环境对它的期望，这种期望被称为软件需求[10]。软件需求包括功能需求(Functional Requirements，FRs)和非功能需求(Non-Functional Requirements，NFRs)两个方面[11-12]。

2.1.1 功能需求的建模与分析

功能需求是指开发人员必须实现的软件功能或软件系统应具备的外部行为[13]。目前，需求工程师通常使用 RUP 推荐的 UML 用例技术进行需求获取与分析，通过构建用例模型获取用户需求。用例是目前功能需求获取和建模比较理想的工具，用例建模虽然不能穷尽所有的需求，但却为功能需求提供了良好的支持，因此其得到了广泛的应用。

2.1.2 非功能需求的建模与分析

非功能需求主要包括两个方面的需求[14-15]：软件质量的要求和软件系统的分布性需求。

对于软件质量的要求，用户和需求工程师可以根据经验和专业知识非正式地从需求文档中识别非功能性需求[11]。RUP 提供了两份模板，一份是用例补充规约，另一份是软件需求规约。两份规约所涉及的范围不同，用例补充规约专门针对特定用例的非功能需求；软件需求规约则是针对整个系统的非功能需求。

然而，采用自然语言描述需求往往会导致理解上的偏差、书写的随意性，同时无法满足需求建模的完整性和准确性。因此，需求工作者目前通常使用一种正式的工具表示非功能需求，例如 NFR(Non-Functional Requirements)[16-17]框架。NFR 框架是一种面向目标的方法，通过“软目标”依赖关系图表示非功能需求。其中，i^*[18]，Tropos[19]和 GRL(Goal Requirements Language)[20]都是面向目标的方法，它们继承了 NFR 框架的“软目标”的概念，旨在处理“软目标”和非功能相关属性。

软件系统的分布性需求的本质是将系统拆分成多个子系统然后部署在不同的硬件设备上，它关注“目标程序依赖的系统软件”最终如何安装或部署到物理机器上[21]。在“4+1”视图方法中，物理视图用来帮助理解系统的分布性需求，并使用基于 UML 的部署图和构件图[22]具体实现系统中软硬件的映射关系，即通过把软件构件部署到相应的硬件节点上来展示系统的分布性。

2.2 RUP 建模方法存在的问题

目前，使用 RUP 推荐的建模方法对系统需求进行建模与分析的技术仍然存在以下问题。

2.2.1 需求建模的分割性

在基于 UML 并结合 RUP 建模方法的迭代式生命周期中，建立软件系统的可视化模型，例如用例模型、分析模型、设计模型、实现模型和部署模型，以及用来表示这些模型的相关 UML 制品。这些模型之间存在一定的关联关系，例如每个用例至少可以与一个分析模型中的分析类有可追踪的联系；一个分析类又可映射到实现模型中的某个构件；同时，构件又被部署到不同的节点上。然而，用例模型和部署模型之间并没有直接的关联关系，这就导致了功能需求模型和分布性需求模型之间的分割性问题。

功能需求和分布性需求建模的分割性使得需求工作者无

法获取完整的系统需求，进而导致最终开发软件项目的失败。因此，获取软件完整性需求是软件项目成功的关键。

2.2.2　需求变更的影响

根据 RUP 的建模思想，需求工作者通常先将复杂的需求进行划分，然后将各个需求模块进行合并。这就导致当模型中的某一处需求发生变更时，会涉及到其他模型的很多方面。因此需求变更不能单一地修改一个部分，要把需求修改部分所涉及到的其他部分同时进行修改；否则无法满足需求的一致性原则，也可能会因为需求变更而导致需求错误。

本文为了解决需求建模分割性问题以及需求变更困难的问题，进行了如下研究。

3　需求及其制品间追踪关系的整体描述

3.1　需求追踪的整体框架

需求追踪的整体框架如图 1 所示。从纵向上看，图 1 包含 3 个层面：第 1 个层面是需求模型层，通过从用例模型，到系统分析模型，最后到软件架构(SA)设计模型的建模来完成软件需求信息的记录和表示；第 2 个层面是需求制品层，表示软件需求在软件生命周期各个阶段产生的制品，软件需求的变化是通过变化的场景，到变化的用例，到变化的分析类，到变化的构件，最后到变化的节点来传递的；第 3 个层面是需求追踪层，表示需求信息追踪的具体实现依赖需求追踪矩阵以及矩阵运算。

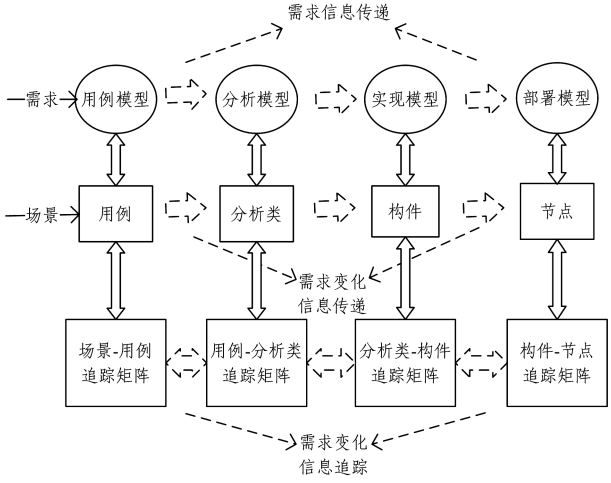

图 1　需求追踪的整体框架

Fig. 1　Framework of requirements tracking

从横向上看，图 1 也包含 3 个层面，这 3 个层面贯穿软件生命周期的 3 个过程：需求分析、系统分析以及系统设计过程。

接下来，本文将从需求传播途径、需求追踪关系以及需求追踪矩阵 3 个方面介绍需求以及需求在软件生命周期各个阶段的制品间追踪关系的演变。

3.2　需求传播途径

本文在用例模型、分析模型、实现模型以及部署模型之间建立需求变化追踪关系，具体分为 4 个步骤：1)根据场景变化，分析受场景影响的用例；2)根据变化的用例，确定与此对应的分析模型中分析类的更改；3)根据变化的分析类，确定哪些构件发生了更改；4)确定构件在节点上的部署。因此，需求变化可以根据场景 s(scenario)→用例 u(use case)→分析类 a(analysis class)→构件 c(component)→节点 n(node)的传播路径进行追踪，由此得到如图 2 所示的需求传播途径。

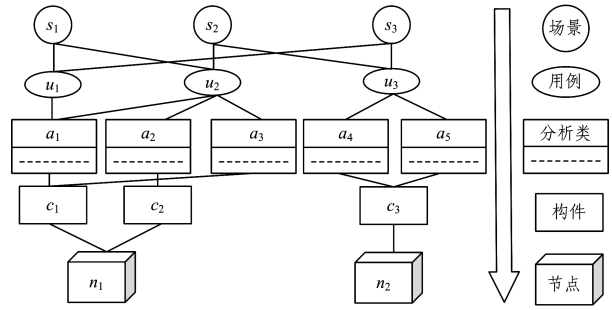

图 2　需求传播途径

Fig. 2　Route of requirements transmission

3.3　需求追踪关系

根据上述需求传播途径，将元素间连线关系用具有一定语义的数值表示，其中 1 表示元素之间有关联，0 表示元素之间无关联。由此得到场景-用例、用例-分析类、分析类-构件以及构件-节点的追踪关系，如图 3－图 6 所示。

	s_1	s_2	s_3
u_1	1	0	1
u_2	1	1	0
u_3	0	1	1

图 3　场景-用例追踪关系

Fig. 3　Scenario-use case trace relationships

	u_1	u_2	u_3
a_1	1	1	0
a_2	0	1	0
a_3	0	1	0
a_4	0	0	1
a_5	0	0	1

图 4　用例-分析类追踪关系

Fig. 4　Use case-analysis class trace relationships

	a_1	a_2	a_3	a_4	a_5
c_1	1	0	1	0	0
c_2	0	1	0	0	0
c_3	0	0	0	1	1

图 5　分析类-构件追踪关系

Fig. 5　Analysis class-component trace relationships

	c_1	c_2	c_3
n_1	1	1	0
n_2	0	0	1

图 6　构件-节点追踪关系

Fig. 6　Component-node trace relationships

3.4 需求追踪矩阵

需求追踪矩阵是管理需求变更和验证需求是否得到实现的有效工具,可以跟踪每个需求的状态[23]。

根据上述 4 个追踪关系可分别定义用例-场景追踪矩阵 $\boldsymbol{M}_s$、分析类-用例追踪矩阵 $\boldsymbol{M}_u$、构件-分析类追踪矩阵 $\boldsymbol{M}_a$、节点-构件追踪 $\boldsymbol{M}_c$。得到 $\boldsymbol{M}_s$,$\boldsymbol{M}_u$,$\boldsymbol{M}_a$ 和 $\boldsymbol{M}_c$ 的追踪矩阵分别为:

$$\boldsymbol{M}_s=\begin{bmatrix}1&0&1\\1&1&0\\0&1&0\end{bmatrix}$$

$$\boldsymbol{M}_u=\begin{bmatrix}1&1&0\\0&1&0\\0&1&0\\0&0&1\\0&0&1\end{bmatrix}$$

$$\boldsymbol{M}_u=\begin{bmatrix}1&0&1&0&0\\0&1&0&0&0\\0&0&0&1&1\end{bmatrix}$$

$$\boldsymbol{M}_c=\begin{bmatrix}1&1&0\\0&0&1\end{bmatrix}$$

设追踪关系矩阵 $\boldsymbol{M}_s=(m_{s(i,j)})$,$i,j=1,2,3,\cdots,n$。其中 $m_{s(i,j)}$ 表示场景 s_i 和用例 u_j 之间的关联关系。

$$m_{s(i,j)}=\begin{cases}1, & \text{当场景 } s_i \text{ 和用例 } u_j \text{ 有关联}\\0, & \text{当场景 } s_i \text{ 和用例 } u_j \text{ 无关联}\end{cases}$$

同理得到:

$$m_{u(i,j)}=\begin{cases}1, & \text{当用例 } u_i \text{ 和分析类 } a_j \text{ 有关联}\\0, & \text{当用例 } u_i \text{ 和分析类 } a_j \text{ 无关联}\end{cases}$$

$$m_{a(i,j)}=\begin{cases}1, & \text{当分析类 } a_i \text{ 和构件 } c_j \text{ 有关联}\\0, & \text{当分析类 } a_i \text{ 和构件 } c_j \text{ 无关联}\end{cases}$$

$$m_{c(i,j)}=\begin{cases}1, & \text{当构件 } c_i \text{ 和节点 } n_j \text{ 有关联}\\0, & \text{当构件 } c_i \text{ 和节点 } n_j \text{ 无关联}\end{cases}$$

最终得到的追踪矩阵能够有效描述分析、设计和实现等不同开发阶段制品间的关联,并且除了逐层的追踪外,还可以实现跨层追踪,旨在在用例模型和部署模型间建立关联关系。

设用例模型和部署模型间的追踪关系矩阵为:

$$\boldsymbol{M}_{u-a-c}=\boldsymbol{M}_c\times\boldsymbol{M}_{u-a}=\boldsymbol{M}_c\times(\boldsymbol{M}_a\times\boldsymbol{M}_u)$$
$$=(m_{u-a-c(i,j)}) \tag{1}$$

$$m_{u-a-c(i,j)}=\sum_{x=1}^{k}(m_{c(i,x)}\times m_{u-a(x,j)}),i,j=1,2,3,\cdots,n \tag{2}$$

其中,m_c 和 m_{u-a} 分别是矩阵 $\boldsymbol{M}_c$ 和 $\boldsymbol{M}_{u-a}$ 中的元素。

例如,当某个用例发生变更时,可通过 $\boldsymbol{M}_{u-a}$ 直接追踪到相关的构件,其中矩阵 $\boldsymbol{M}_{u-a}=\boldsymbol{M}_a\times\boldsymbol{M}_u$:

$$\boldsymbol{M}_{u-a}=\boldsymbol{M}_a\times\boldsymbol{M}_u=\begin{bmatrix}1&0&1&0&0\\0&1&0&0&0\\0&0&0&1&1\end{bmatrix}\times\begin{bmatrix}1&1&0\\0&1&0\\0&1&0\\0&0&1\\0&0&1\end{bmatrix}$$
$$=\begin{bmatrix}1&2&0\\0&1&0\\0&0&2\end{bmatrix}$$

根据计算结果,当矩阵 $\boldsymbol{M}_{u-a}$ 中元素 $m_{u-a(i,j)}\neq0$ 时,用例 u_i 和构件 c_j 之间具有追踪关系,用例更改必将引起构件的变化。

通过计算 $\boldsymbol{M}_{u-a-c}$,可以判断用例变化时会关联到哪些节点:

$$\boldsymbol{M}_{u-a-c}=\boldsymbol{M}_c\times\boldsymbol{M}_{u-a}=\begin{bmatrix}1&1&0\\0&0&1\end{bmatrix}\times\begin{bmatrix}1&2&0\\0&1&0\\0&0&2\end{bmatrix}$$
$$=\begin{bmatrix}1&3&0\\0&0&2\end{bmatrix}$$

根据计算结果,当矩阵 $\boldsymbol{M}_{u-a-c}$ 中元素 $m_{u-a-c(i,j)}\neq0$ 时,用例 u_i 和节点 n_j 之间具有追踪关系。

因此,通过计算需求追踪矩阵,最终在用例和节点间建立可追踪关系,进而将功能需求和分布性需求结合在一起,可以有效避免需求分割性导致的需求获取不完整的问题。

4 自动柜员机系统(ATM)实例分析

本节将在 ATM 系统中分步骤探讨上述方法的可行性。

首先,以 ATM 系统部分功能为例,分别对场景-用例、用例-分析类、分析类-构件以及构件-节点之间的需求传播途径进行分析,如图 7—图 10 所示。

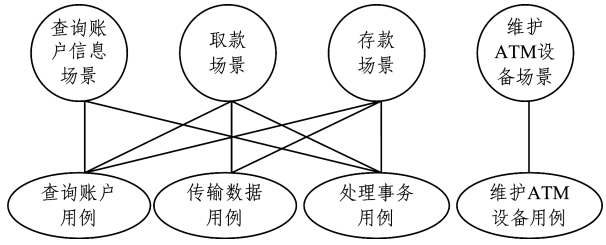

图 7 ATM 系统中场景-用例间需求传播途径

Fig. 7 Requirements transmission route between scenario and use case in ATM

用例通常用于描述可发生的所有事件序列,而场景则是描述其中的一部分。例如在图 7 中,处理事务用例是查询账户信息、取款以及存款 3 种场景的集合。

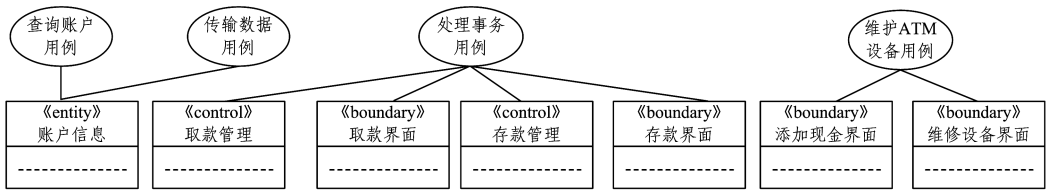

图 8 ATM 系统中用例-分析类间需求传播途径

Fig. 8 Requirements transmission route between use case and analysis class in ATM

用例模型只考虑提供什么功能,而对这些功能的内部运作情况不予考虑,为了揭示系统内部的设计和协作情况,要对类模型进行描述。例如在图 8 中,使用取款管理、存款管理控制类以及取款界面、存款界面界面类来描述处理事务功能的内部结构。

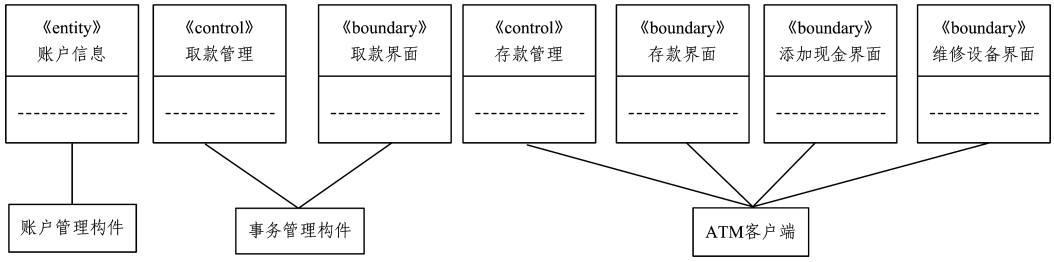

图 9　ATM 系统中分析类-构件间的需求传播途径

Fig. 9　Requirements transmission route between analysis class and component in ATM

构件是具有一定功能的黑盒子,每个构件都具有特定的功能实现。例如,在图 9 中事务管理构件具有取款管理和存款管理的功能实现,ATM 客户端构件则是取款界面、存款界面等界面的集合。

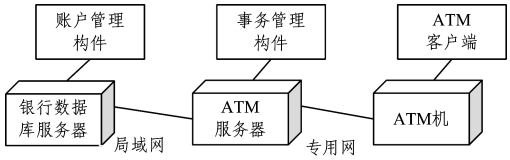

图 10　ATM 系统中构件-节点间需求传播途径

Fig. 10　Requirements transmission route between component and node in ATM

部署图表示构件在节点上的分布情况。例如在图 10 中,账户管理构件部署在银行数据库服务器上,事务管理构件部署在 ATM 服务器上,ATM 客户端构件部署在 ATM 机上。最后,节点间通过特定的链接关联在一起,形成了一个完整的网络。

将需求传播途径图中的元素用字母代替,即用场景 s、用例 u、分析类 a、构件 c 以及节点 n 表示;元素连接关系则用具有一定语义的数字表示。ATM 系统场景-用例、用例-分析类、分析类-构件以及构件-节点的追踪关系分别如图 11－图 14 所示。

	s_1	s_2	s_3	s_4
u_1	1	1	1	0
u_2	0	1	1	0
u_3	1	1	1	0
u_4	0	0	0	1

图 11　ATM 系统中场景-用例追踪关系图

Fig. 11　Scenario-use case trace relationships in ATM

	u_1	u_2	u_3	u_4
a_1	1	1	0	0
a_2	0	0	1	0
a_3	0	0	1	0
a_4	0	0	1	0
a_5	0	0	1	0
a_6	0	0	0	1
a_7	0	0	0	1

图 12　ATM 系统中用例-分析类追踪关系图

Fig. 12　Use case-analysis class trace relationships in ATM

	a_1	a_2	a_3	a_4	a_5	a_6	a_7
c_1	1	0	0	0	0	0	0
c_2	0	1	0	1	0	0	0
c_3	0	0	1	0	1	1	1

图 13　ATM 系统中分析类-构件追踪关系图

Fig. 13　Analysis class-component trace relationships in ATM

	c_1	c_2	c_3
n_1	1	0	0
n_2	0	1	0
n_3	0	0	1

图 14　ATM 系统中构件-节点追踪关系图

Fig. 14　Component-node trace relationships in ATM

接着,根据上述 ATM 系统追踪关系可分别定义 ATM 系统用例-场景追踪矩阵 $\boldsymbol{M}_s$、分析类-用例追踪矩阵 $\boldsymbol{M}_u$、构件-分析类追踪矩阵 $\boldsymbol{M}_a$ 节点-构件追踪矩阵 $\boldsymbol{M}_c$。

$$\boldsymbol{M}_s=\begin{bmatrix}1&1&1&0\\0&1&1&0\\1&1&1&0\\0&0&0&1\end{bmatrix}$$

$$\boldsymbol{M}_u=\begin{bmatrix}1&1&0&0\\0&0&1&0\\0&0&1&0\\0&0&1&0\\0&0&1&0\\0&0&0&1\\0&0&0&1\end{bmatrix}$$

$$\boldsymbol{M}_a=\begin{bmatrix}1&0&0&0&0&0&0\\0&1&0&1&0&0&0\\0&0&1&0&1&1&1\end{bmatrix}$$

$$\boldsymbol{M}_c=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$$

例如,通过 $\boldsymbol{M}_s$ 矩阵可以判断:当 $i=2,j=2$ 时,$m_{s(2,2)}=1$。这说明取款场景的改变会直接导致处理事务用例的变化。同理,通过分析其他矩阵,可查找更改某个元素所能影响到的相关元素。

除此之外,通过跨层追踪可得到用例-节点间需求传播途径,如图 15 所示。

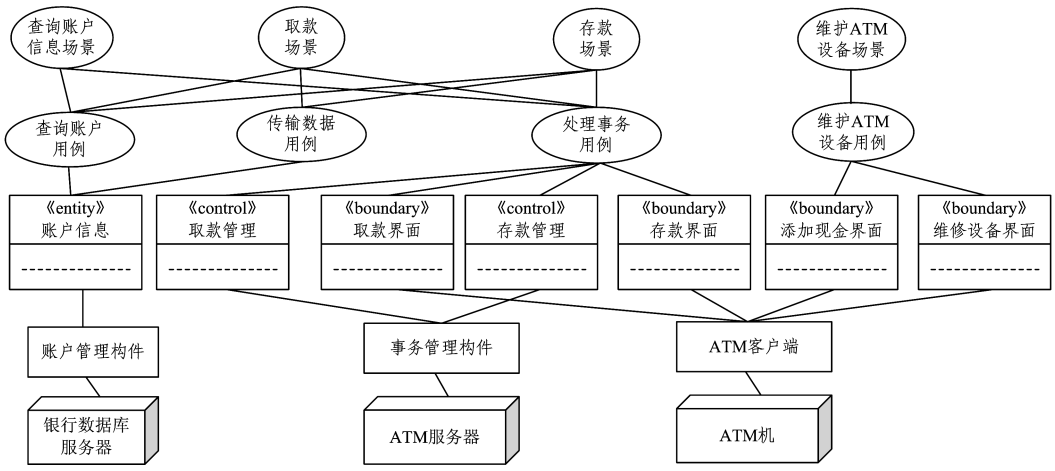

图 15 ATM 系统中用例-节点间需求传播途径

Fig. 15 Requirements transmission route between use case and node in ATM

根据式(1)和式(2)两个矩阵运算公式,得到 ATM 系统中用例和节点间的追踪矩阵为:

$$\boldsymbol{M}_{u-a-c}=\boldsymbol{M}_c\times(\boldsymbol{M}_a\times\boldsymbol{M}_u)$$

$$=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\times\left\{\begin{bmatrix}1&0&0&0&0&0&0\\0&1&0&1&0&0&0\\0&0&1&0&1&1&1\end{bmatrix}\times\begin{bmatrix}1&1&0&0\\0&0&1&0\\0&0&1&0\\0&0&1&0\\0&0&1&0\\0&0&0&1\\0&0&0&1\end{bmatrix}\right\}$$

$$=\begin{bmatrix}1&1&0&0\\0&0&2&0\\0&0&2&2\end{bmatrix}$$

根据 $\boldsymbol{M}_{u-a-c}$ 的计算结果,得到 ATM 系统中用例-节点追踪关系,如图 16 所示,其中非零元素表示用例与节点元素之间具有关联关系。

	u_1	u_2	u_3	u_4
n_1	1	1	0	0
n_2	0	0	2	0
n_3	0	0	2	2

图 16 ATM 系统中用例-节点追踪关系图

Fig. 16 Use case-node trace relationships in ATM

最后,以取款用例 u_3 为例,验证 ATM 系统中取款在用例-节点追踪关系中的准确性。由图 15 可知,取款用例更改所影响的构件与取款管理控制类 a_1、取款边界类 a_2 相关的构件一致,即事务管理构件 c_2 和 ATM 客户端构件 c_3,而这两个构件又分别部署在 ATM 服务器 n_2 和 ATM 机 n_3 上。这意味着 u_3 与 n_2,n_3 之间具有关联关系。显然,这与图 16 中 ATM 系统用例-节点追踪关系的第 3 列一致。同理,可知图 16 的其他列也是正确的。

因此,通过计算需求追踪矩阵,在 ATM 系统的功能需求模型和分布性需求模型间建立可追踪关系,一方面可有效获取 ATM 系统的完整性需求,另一方面可有效管理需求变更。

结束语 本文主要研究了一种基于需求追踪矩阵运算将功能需求模型和分布性需求模型联系在一起的方法。通过计算需求及制品间的追踪矩阵,一方面在用例模型和部署模型间建立可追踪关系,另一方面在修改某一模型元素时,可通过追踪关系寻找其他相关元素,一并更改。这不仅解决了以往功能需求和分布性需求的建模分割性的问题,能够获取更完整的需求,还有效地解决了需求变更困难的问题,从而提高了处理需求变更的效率。本文还结合 ATM 系统实例对此方法进行了可行性验证。然而,本文对基于追踪矩阵获取完整性需求的研究仅限于模型层面,并未深入到元模型层面,因此,接下来的研究方向将从元模型层面的追踪性展开。

参 考 文 献

[1] CAMPOS C,FERNANDES J E,MACHADO R J. Business Modeling and Requirements in RUP:A Dependency Analysis of Activities,Tasks and Work Products [C]// International Conference on Computational Science and Its Applications. Springer International Publishing,2016:595-607.

[2] WEI B. A Comparison of Two Model Transformation Frameworks for Multiple-viewed Software Requirements Acquisition [C]// The International Conference on Software Engineering and Knowledge Engineering. 2017:207-212.

[3] MICHAEL A,JEFFREY G,BRADLEY C,et al. Distributed System for Monitoring Information Events:U. S. Patent Application 12/130,569[P]. 2008.

[4] WINKLER S,PILGRIM J. A Survey of Traceability in Requirements Engineering and Model-driven Development[J]. Software & Systems Modeling,2010,9(4):529-565.

[5] CIMATTI A,ROVERI M,SUSI A,et al. Validation of requirements for hybrid systems:A formal approach[J]. Acm Transactions on Software Engineering & Methodology,2013,21(4):1-34.

[6] GOKNILA,KURTEV I,BERG K V D,et al. Change Impact A-

nalysis for Requirements: A Metamodeling Approach[J]. Information & Software Technology, 2014, 56(8): 950-972.

[7] TEKINERDOGAN B, ERATA F. Modeling Traceability in System of Systems[C]// Symposium. 2017: 1799-1802.

[8] ZHU X F, JIN Z. Inconsistent Management in Software Requirements[J]. Journal of Software, 2005, 16(7): 1221-1231. (in Chinese)
朱雪峰,金芝. 关于软件需求中的不一致性管理[J]. 软件学报, 2005, 16(7): 1221-1231.

[9] WANG Y H, WANG L F, ZHANG S K, et al. A Software Requirements Change Tracking Method[J]. Acta Electronica Sinica, 2006, 34(8): 1428-1432. (in Chinese)
王映辉,王立福,张世琨,等. 一种软件需求变化追踪方法[J]. 电子学报, 2006, 34(8): 1428-1432.

[10] WEI B, DELUGACH H S. A Framework for Requirements Knowledge Acquisition Using UML and Conceptual Graphs[M]// Software Engineering Research, Management and Applications. Springer International Publishing, 2016.

[11] ALSALEH S, HARON H. The Most Important Functional and Non-Functional Requirements of Knowledge Sharing System at Public Academic Institutions: A Case Study[C]// Lecture Notes on Software Engineering, 2016, 4(2): 157.

[12] GNAHO C, SEMMAK F, LALEAU R. Modeling the Impact of Non-functional Requirements on Functional Requirements [M]// Advances in Conceptual Modeling. Springer International Publishing, 2013: 59-67.

[13] GLINZ M. On Non-Functional Requirements[C]// IEEE International Requirements Engineering Conference. IEEE, 2007: 21-26.

[14] HAMMANI F Z. Survey of Non-Functional Requirements Modeling and Verification of Software Product Lines[C]// IEEE Eighth International Conference on Research Challenges in Information Science. IEEE, 2014: 1-6.

[15] LI F L, HORKOFF J, MYLOPOULOS J, et al. Non-functional Requirements as Qualities, with a Spice of Ontology[C]// Requirements Engineering Conference. IEEE, 2014: 293-302.

[16] BOUAIN A, FAZZIKI A E, SADGAL M. Integration of Non-functional Requirements in a Service-oriented and Model-driven Approach[C]// IEEE Eighth International Conference on Research Challenges in Information Science. IEEE, 2014: 1-8.

[17] CHUNG L, LEITE J C P. On Non-Functional Requirements in Software Engineering [M]// Non-functional requirements in software engineering. Kluwer Academic, 2000: 363-379.

[18] AMYOT D. Consistency Analysis for User Requirements Notation Models[C]// International I* Workshop. 2016.

[19] CASTRO J, KOLP M, MYLOPOULOS J. Towards Requirements-driven Information Systems Engineering: the Tropos Project[J]. Information Systems, 2002, 27(6): 365-389.

[20] AMYOT D, MUSSBACHER G. URN: Towards a New Standard for the Visual Description of Requirements[C]// International Conference on Telecommunications and Beyond: the Broader Applicability of Sdl and Msc. Springer-Verlag, 2002: 21-37.

[21] HSIEH Y. Culture and Shared Understanding in Distributed Requirements Engineering[C]// IEEE International Conference on Global Software Engineering. IEEE Computer Society, 2006: 101-108.

[22] MOHAMMADI R G, BARFOROUSH A A. Enforcing Component Dependency in UML Deployment Diagram for Cloud Applications[C]// International Symposium on Telecommunications. IEEE, 2015: 412-417.

[23] WANG Y H, ZHANG S K, LIU Y, et al. Analysis of Evolutionary Spreading Effects of Software Architecture Based on Reachability Matrix[J]. Journal of Software, 2004, 15(8): 1107-1115. (in Chinese)
王映辉,张世琨,刘瑜,等. 基于可达矩阵的软件体系结构演化波及效应分析[J]. 软件学报, 2004, 15(8): 1107-1115.