一种基于反射机制的OWL-S需求规约演化方法

袁文杰[1] 应 时[1,2] 吴可嘉[1] 姚俊峰[1]
(武汉大学软件工程国家重点实验室 武汉 430072)[1] (武汉大学计算机科学学院 武汉 430072)[2]

摘 要 软件系统面临用户需求、软件资源和系统上下文环境等方面变化的挑战,软件需求不可避免地要求持续的演化。反射目前被用于软件系统的运行时管理和动态演化等方面,但还没有用于软件需求规约的演化。提出了一种基于反射机制的OWL-S需求规约演化方法,通过描述支持OWL-S需求规约演化的元信息,并以合理的方式使用这些元信息,实现OWL-S需求规约的演化。利用这种方法,需求分析人员可以有效地管理需求变更,以一种可控、有序的方式完成需求规约的演化任务。

关键词 需求演化,反射,OWL-S

Approach for Evolution of OWL-S Requirements Specification Based on Reflection Mechanism

YUAN Wen-jie[1] YING Shi[1,2] WU Ke-jia[1] YAO Jun-feng[1]
(State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China)[1]
(College of Computer Science, Wuhan University, Wuhan 430072, China)[2]

Abstract Software system has being faced challenge such as change of user requirements, software resources and system context environment, so software requirements must support sustainable evolution. Reflection mechanism has been successfully applied in the realm of the runtime management and dynamic evolution of software systems, but scarcely applied in the evolution of requirements specification. This paper proposed an approach which supports the evolution of OWL-S requirements specification based on reflection mechanism. Requirements analyst can implement the evolution of OWL-S requirements specification through describing and reasonably using the meta-information of OWL-S requirements specification. By use of this approach analyst can effectively manage the requirements change and finish the evolution task of requirements specification through the controlled and orderly manner.

Keywords Requirement evolution, Reflection, OWL-S

当前软件系统面临用户需求、软件资源和系统上下文环境等方面变化的挑战,软件需求不可避免地要求持续的演化。需求演化是影响软件开发活动以及软件系统特征(如系统可靠性等)的主要因素,已成为软件项目中急需解决的问题。虽然学者和实践者都认识到了需求演化的重要性,但是研究成果和经验仍然不足。传统的需求工程方法对需求演化只提供了有限的支持,缺乏有效的建模方法和手段,导致需求演化受到极大的制约。本文认为难以显式地描述并使用支持需求规约重用操作和过程的信息,缺乏有效的演化方法是导致需求难以演化的最根本原因。

反射[1]技术已成功地应用在软件演化方面。近几年来,人们研究了在软件系统运行时,如何基于反射来实现软件系统的动态可修改性和可维护性[2]。这些研究都把反射技术当作只能在运行时才能发挥作用的技术。事实上,只要定义并描述恰当的元级,反射技术就可以用在需求规约阶段,支持需求规约演化。本文基于需求元建模框架RGPS[3],研究需求规约的演化方法。RGPS需求元建模框架是根据网络式软件用户需求的特点进行构造的,该框架包括4个层次:角色层(R)、目标层(G)、过程层(P)和服务层(S),通过这4个层次之间的关联能够为需求模型的互操作提供有力的支持。基于RGPS的领域需求规约中的过程模型通过OWL-S描述,为网络式软件的用户群体所定制。但是,每个用户都会有一些个性化的需求[4],而这些个性化的需求并没有被描述领域过程模型的领域需求规约所包含。网络式软件的重要特征之一就是具有自我演化的能力,它可以根据网络用户提出的新需求,扩展或者整合原有的领域需求规约,从而得到满足用户新需求的系统需求规约。因此,针对OWL-S[5]需求规约的特点和需求变更管理的要求,设计出一种基于反射的OWL-S需求规约演化方法,支持需求规约以可控、有序的方式完成演化。

1 研究背景

反射是指计算系统能够根据对自身状态和行为的自表示来描述、推理和操纵自身的能力[1]。通过这种能力,系统能够获取关于自己的结构和行为的信息,并动态地调整自己的状

到稿日期:2009-07-02 返修日期:2009-09-28 本文受国家973重点基础研究发展规划(2007CB7310800),国家自然科学基金(60773006),国家教育部博士点基金(20060486045)资助。

袁文杰(1982—),男,博士生,主要研究方向为软件体系结构、反射,E-mail:yuanwenjie@gmail.com;应 时(1965—),男,博士,教授,博士生导师,主要研究方向为面向服务的软件工程方法、基于组件的软件工程方法、软件体系结构和模式、软件的可重用性与互操作性等。

态。反射式系统一般由两个部分组成：元级，对系统的问题域建模并对其进行推理和操纵，以解决问题；基级，对系统自身建模并对其进行推理和操纵，使系统适应某些变化。反射有两方面的操作：自省(introspection)和拦截(intercession)[6]。自省是系统对自身状态的观察和推断，该操作常被用来探测对象运行时的方法和属性等。拦截是系统对自身行为的动态调整，使用拦截可以动态地改变系统的行为。

OWL-S(Web Ontology Language for Services)，是基于OWL语言描述的Web服务的本体，其前身是DAML-S。OWL-S引入了本体来描述服务领域的概念，利用普遍概念来描述服务自身，以及这些服务怎么与领域本体联系起来的(通过输入、输出、前置条件、效果等)，这些丰富的语义描述使Web服务能够被人和机器理解[5]。OWL-S的动机和目标是使服务具有机器可理解性和易用性，从而支持代理程序基于逻辑语义实现对Web服务的自动发现、调用、组合及互操作。

2 支持OWL-S需求规约演化的反射机制

为了有效管理需求的变更，并以一种可控、有序的方式完成需求规约的演化，本文针对OWL-S需求规约的特点提出了一种支持OWL-S需求规约演化的反射机制。该反射机制将需要演化的系统需求规约划分为两层：元级和基级。需要演化的需求规约的基级部分就是使用OWL-S描述的用户需求规约，它描述用户的具体需求。元级部分包括控制需求规约演化过程的元信息模型、演化规约和依赖检查规则。演化规约是用于根据需求变更以及相关元信息制定需求演化的完整步骤，而依赖检查规则是用于在进行需求演化前获取元信息模型中的依赖元信息，检查发生变更的过程是否存在某依赖关系使得其不允许发生变更。

同时，反射机制定义了元级与基级之间的关系，从而为需求规约演化的操作和过程提供支持，如图1所示。在一般的反射机制中，基级和元级之间具有因果关联，该关联使基级的改变能在元级中产生效果，反之亦然，因果关联确保基级和元级之间的一致性。在图1所示的反射机制中，这种因果关联是这样实现的：首先初始化该反射系统，即根据OWL-S描述的需求规约构造基级需求规约信息模型，使用具体化操作从其中抽取元信息构造元信息模型；然后根据演化规约，结合元信息以及依赖检查规则检查是否可以实施该变更操作，若检查结果表明实施变更操作后过程中的依赖信息不会产生冲突，则开始执行演化变更；最后，元级的变更会通过反射操作使得基级发生相应变更从而完成演化。

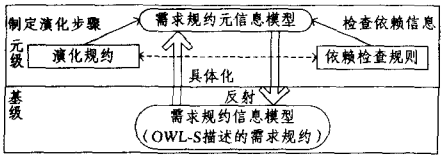

图1 支持OWL-S需求规约演化的反射机制

2.1 支持需求规约演化的元信息模型

元信息[7]是关于信息的信息，它用于描述信息的结构、语义、用途和用法等。反射利用元信息为管理、控制和使用复杂信息提供了一种高效的途径。OWL-S需求规约元信息模型为OWL-S需求规约及其组成元素定义支持演化操作和过程所需的元信息。这些元信息用于支持OWL-S需求变更的管理和控制需求演化的实施。在需求演化过程中，主要发生变更的是OWL-S描述的过程，因此元信息模型中主要部分是过程元信息，而构建元信息模型的工作主要集中在构建过程元信息模型。

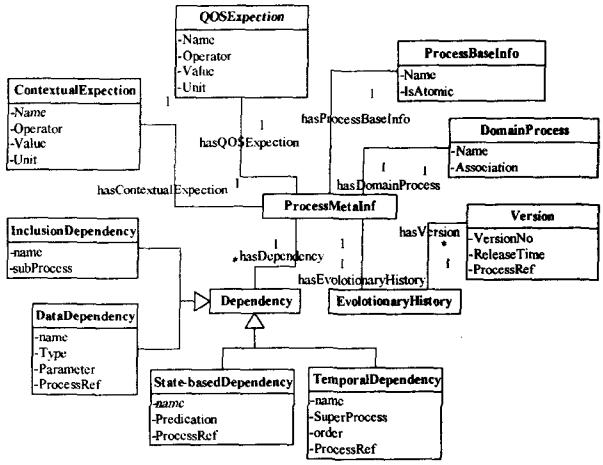

图2 过程元信息模型

如图2所示，过程元信息ProcessesMetaInfo是由多个Process组成，而Process主要包括以下信息：

(1)过程基本信息(ProcessBasicInfo)。包括过程名(Name)、过程的类型(描述是否是原子过程或组合过程的类型)以及该过程的输入输出的数据类型(即接口信息)。

(2)与领域模型中过程的关系信息(DomainProcessInfo)。DomainProcessInfo由多个DomainPro-cess组成，Domain-Process的属性包括：与领域模型中相关的过程的名称(Name)、该过程与领域模型中过程的关系(Association)。关系包括：① 等价关系(Equal)：该过程与领域模型中过程完全相同，领域过程模型的直接重用。② 继承关系(Subclassof)：该过程与领域过程具有相同的结构和行为。③ 精化关系(Refine)：过程在领域模型中过程的基础上进行了精化，修改了过程体，但过程的外部接口和行为保持不变。④ 修改关系(Modified)：在领域模型中过程的基础上，针对系统的具体要求进行了修改，过程的外部接口和行为也发生了变化。⑤ 新过程关系(New)：是领域模型中未定义的新过程。

(3)情境信息(ContextInfo)。情境是可以用于描绘实体(如，人、地点或对象)所处环境的特征的任何信息[35]，其中，实体与用户和需求规约之间的交互有关。典型的情境有人、组织或需求所处的环境、标识及状态。需求在演化过程中需要考虑情境的因素，例如在某些情况下不允许进行某些需求变更。在本文中情境主要是用来描述过程的特征的信息。情境信息可以通过不同的视角进行分类，现有很多分类方法，例如可分为概念视角、度量视角、时间特征视角等。本文从3个不同的视角对情境信息进行分类：用户视角、服务视角(或称为过程视角)、系统视角。

情境信息(ContextInfo)由多个Context组成，Context包括情境名称(Name)和情境类型(ContextType)，情境类型有3种：用户情境、系统情境和过程情境。而每个Context则由多个情境属性(ContextProperty)组成，情境属性包括属性名称(PropertyName)、属性值(Value)、属性类型(PropertyType)以及相关的情景。另外，由于每个过程情境是对某一个特定过程的环境特征的描述，因此它应包括过程引用信息。例如

网络带宽是属于系统情境的一个情景属性名，假设它的值为 k，属性类型指向网络概念里的带宽，由此就可知该属性值的度量单位为 kb/s。

(4)QoS 信息(QoSInfo)。对过程所期望的质量属性的相关元信息。QoSInfo 由多个 QoS 组成，QoS 信息描述的是过程的非功能特性，如响应时间、吞吐量、可靠性、可用性、安全性等。每个 QoS 属性都有自己的名称(Name)，并且有对应的值(Value)以及权重(Weight)。Domain 属性描述该 QoS 所属的领域。QoS 属性可分为效益性属性和成本性属性，效益性的属性，值越大越好，而成本性的属性，值越小越好。该 QoS 属性的偏好类型通过属性 PreferType 来描述，可用 Benifit 表示该 QoS 属性为效益性属性，也可用 Cost 表示 QoS 属性为成本性属性。Domain 属性描述该 QoS 所属的领域。另外，还要对每一个 QoS 属性的值约束和权重约束进行详细说明，即使用 Value 和 Weight 进行说明。

(5)过程依赖信息(Dependency)。定义了过程间的依赖关系。需求规约演化时需要参考这些依赖信息，它可以分为包含依赖(IncludeDependency)、时序依赖(TemporalDependency)、状态依赖(State-basedDependency)和数据依赖(DataDependency)。

① IncludeDependency：描述过程与子过程和父过程之间的关系。它包含：依赖的名称(Name)、子过程引用(subProcessRef)、父过程引用(parentProcessRef)。

② TemporalDependency：描述一个过程的行为在另一个过程之前或之后发生。它包含：依赖名称(Name)；父过程引用(superProcessRef)；如果本过程在某一过程之后发生，记录该过程的引用(afterProcessRef)；如果本过程在某一过程之前发生，记录该过程的引用(beforeProcessRef)。

③ State-basedDependency：描述一个过程的行为发生与否取决于另一个过程是否到达某个状态。它包含：依赖名称(Name)、条件谓词(Predication，描述本过程发生需要依赖另一过程的条件)、依赖的过程的引用(ProcessRef)。

④ DataDependency：描述本过程使用的数据引用另一过程处理过的数据。它包含：依赖名称(Name)、共同使用的数据(SharedData)、依赖的过程的引用(ProcessRef)。

当需求中的过程对要删除、替换或修改的过程存在上述依赖关系时，不允许进行删除操作，如果要进行替换和修改操作，则需要使替换或修改后的过程与原有过程接口一致，保证依赖关系依然成立。

(6)演化历史信息(EvolutionaryHistory)。由一组演化版本组成，演化版本记录了过程的变更信息，包括版本号(VersionNo)、发布时间(ReleaseTime)、发生变更的过程的引用(ProcessRef)。

2.2 需求规约演化操作集

上述元信息用于支持需求规约演化操作。需求规约演化可以通过一系列演化操作完成，反射机制支持的操作集在某种程度上是对反射机制更直接的描述，用于描述演化的方式和内容。表 1 列出了元信息模型支持的需求演化操作。

表 1 需求规约演化操作集

序号	演化操作	操作说明
1	添加过程	在系统需求规约中添加新过程。新过程可能来自于领域模型中定义的已有过程，也可能来自于需求变更中定义的新过程
2	删除过程	删除系统需求规约中的某个过程
3	替换过程	使用一个具有相同接口和行为的新过程替换已有的过程
4	修改过程	对过程模型的接口、过程体等进行修改
5	获取领域过程	从领域模型中查找相关过程并将过程导入到系统的需求规约中

各个演化操作的执行与否取决于依赖检查的结果。依赖检查是通过检查依赖检查规则库中的规则来完成的。依赖检查规则库中的规则是在演化前根据过程的各种特点已定义的一系列规则，这些规则用于进行需求演化前获取元信息，模型中的依赖元信息检查发生变更的过程是否存在某依赖关系使得其不允许发生变更。依赖检查规则使用 event-conditions-checklist 的形式描述，其描述方法如下：

on event if conditions check checklist return True/False

event 描述触发依赖关系检查时调用的变更操作(添加、删除、修改、替换中的一种)以及变更的对象(Process，Goal 或 Role)；conditions 描述当 event 发生时需要进行的验证操作(如验证过程是否具有依赖关系以及依赖关系的类型)；checklist 描述当条件 conditions 满足时需要进行的检查操作，checklist 可以为空，即只要满足 conditions 则直接通过 return 返回检查结果；当 checklist 中所有的检查操作都完成检查返回 True，则该条依赖规则检查完成并使用 return 返回 True，如果 checklist 中有一条检查没有通过，则返回 False。如 VR1：

on Delete a Process p
if Process p has data-dependency
return False

表示当要删除的过程 p 与其他过程有数据依赖关系(在元信息模型中被标识)时，该过程不允许删除。表 2 详细描述了各个依赖检查规则。

表 2 依赖检查规则集

编号	规则描述
VR1	on Delete a Process p if Process p has data-dependency return False
VR2	on Add a new Process p if p is added to a sequence process p1 and p is between p2 and p3 check the output type of p1 is the same with the input type of p2 and the output type of p is the same with the input type of p3 return True
VR3	on Add a new Process p if p is added to a choice process p1 check the input type of p is the same with the input type of p1 and the output type of p is the same with the outtype of p1 return True
VR4	on Modify process p if p has Temporal Dependency with process p1 and p2 check after modifying, the input type of p is the same with the out type of p1 and the output type of p is the same with the input type of p2 return True
VR5	on Replace process op with a new process np check the input type and the output type of op is the same with those of np return True

2.3 需求演化规约

新增的功能需求、新增的业务情境需求和新增的非功能需求等个性化的需求变更将会引起 OWL-S 描述的系统需求规约的演化。由于 OWL-S 语言不支持需求规约演化的建

模，无法描述演化是在何时、何处发生的，也无法描述演化是如何发生的，因此，OWL-S 语言没有足够的能力对个性化的演化需求规约进行建模。在 OWL-S 的基础上定义了 OWL-S^C 语言，该语言能够对需求规约变更的属性进行描述。需求变更描述语言 OWL-S^C 将需求变更分为两个描述部分：

(1)过程描述部分，该部分为变更所涉及的过程的描述。用户可以在已有的领域模型中查找自己需要的过程，描述该领域过程的标识(id)以及领域过程的引用(ProcessRef)；用户也可以自己定义领域模型中不存在的过程，首先需要提供该过程的标识，接着使用 OWL-S 的语法对其过程体进行详细定义。

(2)变更描述部分，该部分描述变更操作。变更操作可以分为 4 类：添加过程、删除过程、修改过程、替换过程，而且每个变更可能只是单独的上述 4 类操作中的一种，也可能是多个上述 4 类操作组合而成的复杂的变更。因此可以将变更分为简单变更(Single)和组合变更(Composite)。在变更描述部分，用户可以限定执行变更操作的前提条件(Condition)。当变更操作类型不同时需要描述的信息也有所不同：

① 当变更操作为添加过程时，需要描述添加的新过程的引用(即引用过程描述部分定义的过程)以及新过程与现有过程之间的一系列的关系(relationship)。如果该新过程要添加到当前某过程 P 中，则该新过程为过程 P 的子过程(subProcessOf)；如果新过程与当前某过程为并行选择关系，则他们之间的关系为平行关系(parallelTo)；如果新过程要添加到已有的两个过程 $P1$ 和 $P2$ 之间，则它与 $P1$ 和 $P2$ 的关系为序列关系(sequence)，并使用 beforeProcessRef 描述对过程 $P1$ 的引用，使用 afterProcessRef 描述对过程 $P2$ 的引用。从这些关系中可以提取出新过程的依赖关系信息。

② 当变更操作为删除过程时，需要描述要删除过程的引用(即引用过程描述部分定义的过程)以及要删除过程的父过程的引用。

③ 当变更操作为修改过程时，需要描述要修改的过程的引用、该过程需要修改的属性以及对该属性需要赋予的新值。

④ 当变更操作为替换过程时，需要描述要替换的过程的引用以及用于替换该过程的新过程的引用。

为了简要说明该语言，将使用 BNF 风格的 pseudo-schemas 来说明 OWL-S^C 的语法结构。Pseudo-schemas 接近 OWL-S^C 文档的最终结构，并采取了类似 BNF 范式的语法符号来描述文档的语法，易于阅读和理解。在 Pseudo-schemas 中："?"表示可选(发生零次或者一次)，"*"表示发生零次或者多次，"+"表示发生一次或者多次，"["和"]"表示一组元素，"|"分开几个可选择项。属性或者包含简单内容的元素通常指定一个表示其类型的值，如："xs:string"表示字符串类型。因此将需求规约变更语言定义如图 3 所示。

```
〈ProcessDefinition〉
  〈DomainProcess id = "xs:string" processRef = "xs:string"/〉*
  〈UserProcess id = "xs:string"〉
    〈! -- 使用 OWL-S 语法描述用户自己定义的过程 --〉
  〈/UserProcess〉*
〈/ProcessDefinition〉
〈ChangeDescription〉
  〈Change name="xs:string" type="Single|Composite" condition=
"xs:string"〉
    〈Add processRef="xs:string"〉
    〈Relationships〉
      〈Relationship type= "subProcessOf" parentProcessRef=
          "xs:string"/〉*
      〈Relationship type="parallelTo" processRef="xs:
        string"/〉*
      〈Relationship type= "sequence" beforeProcessRef = "xs:
            string" afterProcessRef= "xs:string"/〉*
    〈/Relationships〉
    〈/Add〉*
    〈Delete processRef = "xs: string" parentProcessRef = "xs:
        string"/〉*
    〈Modify processRef="xs:string" attribute="xs:string"
        newValue="xs:string"/〉*
    〈Replace oldProcessRef = "xs: string" newProcessRef = "xs:
        string"/〉*
  〈/Change〉*
〈/ChangeDescription〉
```

图 3　需求变更描述语言

3　基于反射机制的需求规约演化实施过程

基于反射机制 RMERS 的需求演化实施过程如图 4 所示。在演化过程中，除了反射技术的机制起作用之外，还必须要借助基于 RGPS 的领域需求模型中的需求资源。需求资源是需求分析过程中所需要的各种资源，以及以前需求分析的结果可以用来重复使用的资源。演化后产生的需求规约也要按照 RGPS 领域需求模型中定义的形式进行存储，以供今后进行需求规约演化时调用。具体的实施过程如下：

①获取并分析用户的实际需求在现有的基于 RGPS 的领域需求模型的需求资源查询中能最大程度满足用户需求 OWL-S 描述的需求规约，同时生成用户定义的 OWL-S^C 描述的需求变更；

②在需求规约演化的系统接收到查询的需求规约以及相应的需求变更后，在反射机制的支持下，抽取演化前 OWL-S 描述的系统需求规约的元信息，构造元信息模型；

③在反射机制的支持下，根据 OWL-S^C 描述的需求变更和元信息模型对需求规约进行演化；

④将演化后的反射式需求规约中的 OWL-S 描述的需求规约存入需求演化资源库，以用于今后的演化。

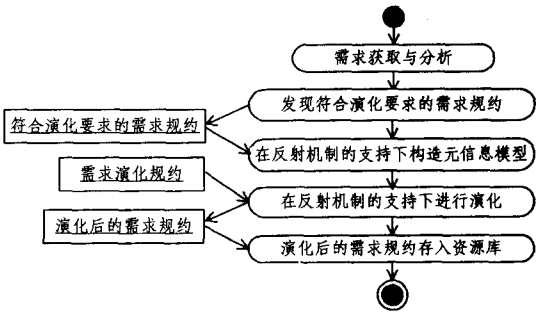

图 4　基于反射的需求演化实施过程

4　案例研究

本部分介绍了一个使用上述方法进行需求规约演化的实际案例。崔某从武汉到北京旅游，他希望由系统为他自动安排旅行行程，并预订到达北京的交通工具。现有的需求资源库中存在需求规约符合该要求，该组合服务 travelPlan 由 4 个服务组成：查询行程安排原子服务(getTime)、选择交通工具并预定组合服务(bookTrans)、查询酒店信息原子服务和预

定酒店原子服务。其中选择交通工具并预定服务是一个choice组合服务，它包括机票预定服务(由查询符合要求的航班信息原子服务和预定机票原子服务组成，bookFlight)和火车票预定服务(由查询符合要求的列车信息原子服务和预定火车票原子服务组成，bookTrain)。由于某种原因，崔某不想搭乘飞机或者火车，而希望能包机前往北京，因此需要对该需求进行修改使得崔某可以顺利前往北京。

(1)描述需求变更

由于原有的旅行行程安排需求规约只具有订机票和预订火车票的功能而无法满足崔某需要，因此，该需求规约必须进行适当的演化，以支持崔某的这条新增的需求，即增加包机预订的组合服务用于获取包机信息并预订合适的包机，该组合服务将加入选择交通工具并预定组合服务中，与原有的机票预订服务和火车票预订服务并行供用户选择。因此该旅行行程安排需求规约变更的OWL-S描述如图5所示。

```
〈ProcessDefinition〉
  〈UserProcess id="Charter Flight"〉
    〈process:CompositeProcess rdf:ID="Charter Flight"〉
      〈process:composedOf〉
        〈process:Sequence〉
          〈process:components rdf:parseType="Collection"〉
            〈! ——查询包机信息，返回可供包机的飞机信息——〉
            〈process:AtomicProcess rdf:resource="# getCharter-
              FlightInfo"/〉
            〈! ——预定可供可供包机的飞机——〉
             〈process:AtomicProcess rdf:resource="# Charter-
              Flight"/〉
          〈/process:components〉
        〈/process:Sequence〉
      〈/process:composedOf〉
    〈/process:CompositeProcess〉
  〈/UserProcess〉
〈/ProcessDefinition〉
〈ChangeDescription〉
  〈Change name="addCharterFlight" type="Composite" 〉
    〈Add processRef = "Charter Flight"〉
    〈Relationships〉
      〈Relationship type= "subProcessOf" parentProcessRef =
        "bookTrans"/〉
       〈Relationship type = "parallelTo" processRef = "book-
          Flight"/〉
       〈Relationship type = "parallelTo" processRef = "book-
          Train"/〉
    〈/Relationships〉
    〈/Add〉
  〈/Change〉
〈/ChangeDescription〉
```

图5 旅行行程安排的需求演化规约

(2)根据基级抽取出元信息构造元级

在这里仅列出旅行行程安排中关键过程的元信息。过程getTime为原子过程，输出参数为日期类型；过程bookTrans为choice类型的组合过程，输入参数为日期类型，输出参数为boolean类型；过程bookFlight为sequence类型的组合过程，输入参数为日期类型，输出参数为boolean类型。同时可以从中抽取出依赖信息，bookFlight过程是bookTrans过程的子过程，因此与其有IncludeDependency关系；getTime和bookTrans与travelPlan有IncludeDependency关系；getTime在bookTrans过程之前执行，因此它们之间有TemporalDependency关系；bookTrans过程需要使用getTime过程获取时间参数，因此它们之间还有DataDependency关系。这些元信息利用xml描述如图6所示。

```
〈MetaInfo〉
  …
    〈ProcessMetaInfo〉
      〈Process〉
        〈ProcessBasicInfo Name="getTime" Type="Atomic"
              InputType="date" OutputType="date"/〉
      〈/Process〉
      〈Process〉
        〈ProcessBasicInfo Name="bookFlight" Type="Sequence"
              InputType="date" OutputType="date"/〉
      〈/Process〉
      〈Process〉
        〈ProcessBasicInfo Name="bookTrans" Type="Choice"
              InputType="date" OutputType="boolean"/〉
          〈DependencyInfo〉
            〈IncludeDependency processRef="bookFlight"/〉
            〈IncludeDependency processRef="bookTrain"/〉
          〈/DependencyInfo〉
      〈/Process〉
      〈Process〉
        〈ProcessBasicInfo Name="travelPlan" Type="Sequence"
              InputType="date" OutputType="boolean"/〉
          〈DependencyInfo〉
            〈IncludeDependency processRef="getTime" /〉
            〈IncludeDependency processRef="bookTrans"/〉
            〈TemporalDependency processRef="getTime"
                beforeProcessRef="bookTrans"/〉
            〈DataDependency processRef="bookTrans"
                dependProcessRef="getTime" shareData
                ="date"/〉
            〈! —— other dependency ——〉
          〈/DependencyInfo〉
        〈/Process〉
    〈/ProcessMetaInfo〉
  …
〈/MetaInfo〉
```

图6 旅行行程安排的过程元信息

(3)根据现有的依赖检查规则检查变更操作是否可以执行。检查依赖规则库，发现存在依赖规则VR3可以对应到当前的添加过程的操作，即当添加新过程p时，如果要将p添加到现有的choice类型的组合过程$p1$中，则检查p的输入输出类型是否与$p1$的输入输出类型一致，若一致则允许添加。

根据变更描述，新过程CharterFlight要添加到已有的choice类型的组合过程bookTrans中作为其子过程，并且和bookFlight服务、bookTrain服务并行供选择。根据规则需要检查新过程的接口信息，新过程CharterFlight的输入参数为

(下转第195页)

[J]. 控制与决策,2006,21(11):1284-1288
[2] Holland JH. Genetic algorithms and classifier systems: foundations and their applications[A]//Proceedings of the Second International Conference on Genetic Algorithms[C]. 1987:82-89
[3] You XM, Liu S, Shuai DX. On parallel immune quantum evolutionary algorithm based on learning mechanism and its convergence[C]//L. Jiao, et al., eds. Proc. of ICNC06, PT1 4221. Berlin Heidelberg: Spring- Verlag, 2006:903-912
[4] Lydyard P M, Whelan A, Fanger M W. Instant notes in immunology[M]. Beijing: Science Press, 2001:1-40
[5] 游晓明,帅典勋,刘升. 基于免疫原理的量子进化算法及收敛性研究[J]. 控制与决策,2007,22(7):749-754
[6] 王磊,潘进,焦李成. 免疫算法[J]. 电子学报,2000,28(7):74-78
[7] 潘正君,康立山,陈毓屏. 演化计算[M]. 北京:清华大学出版社,1998
[8] 张潜,李钟慎,胡祥培. 基于模糊优化的物流配送路径问题研究[J]. 控制与决策,2006,21(6):689-692
[9] 黄席樾,胡小兵. 蚁群算法在 K-TSP 问题中的应用[J]. 计算机仿真,2004,21(12):162-164
[10] 杨兆升. 城市智能公共交通系统理论与方法[M]. 北京:中国铁道出版社,2004
[11] 乘公交,看奥运[EB/OL]. http://www.shumo.com/home

(上接第 145 页)

时间类型,与 bookTrans 过程的输入类型一致,并且新过程 CharterFlight 的输出参数为 boolean 类型,也与 bookTrans 过程的输出类型一致,因此其接口信息不会与已有过程的接口发生冲突,可以添加该过程。

5 相关工作

目前需求演化方面的工作主要集中在需求演化的实证分析和需求演化建模两方面。需求演化的实证分析一般是指需求工程的方法和实践,目前相关工作非常有限,且缺乏对相关需求工程方法学之间的比较分析。这一方面是因为很难收集和分析需求演化数据(软件生命周期的跨度很大);另一方面是目前需求工程方法学缺乏对需求演化分析的支持[8]。Hooks 和 Farry 等人认为需求质量是可以通过分析变更请求(CRs)和差异报告(DRs)来度量的,分析结果将促使需求发生变更[9]。Lutz 和 Mikulski 等人则认为新的需求能从操作异常中产生[10]。

目前虽然存在很多建模方法和建模语言,但是侧重于需求演化建模的很少。现有两种策略进行需求演化建模:一个是使用一种建模需求的方法,该方法可以使需求的演化更简单;另一个是在需求模型中建模演化本身。PROTEU 工程提出了一种形式化框架用于进行需求变更的描述和推理。该形式化描述由一种目标结构框架(goal-structures framework)组成[11]。PROTEUS 的目标结构框架在描述需求和需求之间的交互时考虑到了需求的变更。更重要的是,该框架获取了系统和环境之间的交互,这些交互(例如需求之间的交互和系统与环境之间的交互)形成了分析敏感度和影响的基础。Zowghi 和 Offen 提出了一种逻辑框架用于需求演化建模[12],它包括两个基本操作:① 非单调推理(nonmonotonic inference),通过领域模型来完成需求;② 信念修正(belief revision),通过创建新的需求来完成需求。

结束语 本文将元信息、元建模、反射和需求演化结合起来,构造了一种在需求分析阶段支持 OWL-S 需求规约演化的反射式需求规约。反射式需求规约有助于在需求元建模框架 RGPS 的支持下,高效率、高质量地完成需求演化建模的分析任务。我们的研究工作主要贡献在于提供了一套完整的方法对需求演化进行建模,以一种可控、有序的方式完成需求规约的演化。

今后还需要开发一个工具,以支持需求分析人员使用基于反射机制的需求规约演化方法。这个工具应该能够支持检索、组合、分解、分析和验证 OWL-S 需求规约及其组成元素,支持反射机制的实现即元信息模型的抽取和需求规约演化的完成;还将针对交通领域的具体案例进行更深入的研究,展示反射机制对需求规约演化的支持过程。

参 考 文 献

[1] Maes P. Concepts and Experiments in Computational Reflection [C]//Proceedings of OOPSLA87. ACM Sigplan Notices, New York: ACM Press, 1987:147-155
[2] Huang Gang, Wang Qianxiang, Mei Hong, et al. Research on Architecture-Based Reflective Middleware[J]. Journal of Software, 2003, 14(11):1819-1826
[3] Wang J, He K, Li B. Meta-models of Domain Modeling Framework for Networked Software[C]//Proc. of The Sixth International Conference on Grid and Cooperative Computing (GCC 2007). 2007:878-885
[4] Feng Zaiwen, He K. Towards Individualized Requirements Specification Evolution for Networked Software Based on Aspect[C]// International Conference of Software Process(ICSP'08). 2008: 88-99
[5] Martin D, Ankolekar A, Burstein M, et al. OWL-S: Semantic Markup for Web Services[S]. W3C Candidate Recommendation, 2004, http://www.daml.org/services/owl-s/
[6] Demers F N, Malenfant J. Reflection in logic, functional and object-oriented programming: a short comparative study[C]//Proceedings of the IJCAI95 Workshop on Reflection and Metalevel Architectures and their Applications in AI. 1995:29-38
[7] Costa F M. Combining meta - information management and , reflection in an architecture for configurable and reconfigurable middleware[D]. Lancaster University, 2001
[8] Jarke M, Pohl K. Requirements engineering in 2001: (virtually) managing a changing reality[J]. Software Engineering Journal, 1994:257-266
[9] Hooks I F, Farry K A. Customer - Centered Products : Creating Successful Products through Smart Requirements Management [M]. Amacom, 2001
[10] Lutz R R, Mikulski I C. Operational anomalies as a cause of safety-critical requirements evolution[J]. The Journal of Systems and Software: 2003, 65(2):155-161
[11] Harker S, Eason K, Dobson J. The change and evolution of requirements as a challenge to the practice of software engineering [C]//Proceedings of the IEEE International Symposium on Requirements Engineering. New York: IEEE Computer Society Press, 1999:266-272
[12] Zowghi D, Offen R. A logical framework for modeling and reasoning about the evolution of requirements[C]//Proceedings of the Third IEEE International Symposium on Requirements Engineering. New York: IEEE Computer Society Press, 1997:247-257