文章编号:1000-5862(2018)03-0298-06

WSDL→Radl-WS 生成方法及自动转换系统

张 琦[1],王昌晶[1*],罗海梅[2],左正康[1],石海鹤[1],郭 帆[1]
(1. 江西师范大学计算机信息工程学院,江西 南昌 330022;2. 江西师范大学物理与通信电子学院,江西 南昌 330022)

摘要:精确地描述 Web 服务语义对 Web 服务的发现、执行、动态组合和交互至关重要. 在 Radl 语言基础上,扩展 Radl 为 Radl-WS,提出一种新型的基于 Radl-WS 代数规范的 Web 服务建模语言. 该建模语言统一了基于代数公理方法和 Hoare 公理方法来描述软件规格说明. 为支持模型转换,提出了将 WSDL 语言描述的 Web 服务通过等价变换生成为 Radl-WS 建模语言的方法,并设计了相应的 WSDL→Radl-WS 自动转换系统. 最后采用真实案例,验证了该方法的有效性.

关键词:代数规范;Web 服务;模型转换;自动转换
中图分类号:TP 311 **文献标志码**:A **DOI**:10.16357/j.cnki.issn1000-5862.2018.03.13

0 引言

面向服务的计算(SOC)是一种基于 Internet 的全新计算模式,具有松散耦合、平台无关、互操作性强等特点,受到国内外工业界和学术界的广泛认可与支持. Web 服务是服务计算的重要研究内容,对于 Web 服务的初期研究主要是从服务组合的相关问题着手,如服务组合的形式化建模、服务组合的相关性质研究、服务组合编码与测试等[1]. 对于服务语义的描述方法,国内外的工作大多是基于本体的方法,该方法通过使用本体中定义的词汇对服务的功能及其输入输出参数作注解,以此来描述服务的语义[2-3]. 该描述方法具有易于开发者理解和计算机处理的优点,但绝大多数基于本体的方法缺乏对服务功能进行验证的支持,这是由于本体本身只能定义概念及其实例之间的关系.

代数规范通常用来定义子系统接口,它描述了子系统存取的数据及其操作. 精确的子系统接口描述很重要,因为在子系统的服务尚未实现之前,其它子系统的开发者可能就要在代码中使用该子系统的服务了. 进一步地,一个子系统接口描述可以通过组合的方法得到,即对构成子系统接口所有组件的描述来构成子系统的接口描述. Web 服务可以看作一种网络化支撑环境下的子系统,因此,使用代数规范可以精确地描述 Web 服务的语义,这对 Web 服务的发现、执行、动态组合和交互十分重要.

本文以前期研究工作中提出的 Radl 建模语言为基础,扩展 Radl 为 Radl-WS,提出一种新型的基于 Radl-WS 代数规范的 Web 服务建模语言. Radl-WS 建模语言主要对 Radl 语言进行了 3 部分扩展:(i)Radl 语言主要针对算法进行形式化建模,即采用 Hoare 公理方法来描述算法的前后置断言,并支持算法的形式化开发以及验证,而 Radl-WS 建模语言是针对 Web 服务提出来的,它不仅能在细粒度层次上使用 Radl 语言来形式化描述算法,还能从粗粒度层次上使用代数规范来描述 Web 服务;(ii)可以在 Radl-WS 建模语言中使用 < extends > 和 < imports > 子句增量构造新的 Web 服务,即进行服务组合;(iii)Radl-WS 语言允许操作的定义域和协域都是非单一的.

Web 服务通过 WSDL 语言进行描述,其描述文档相当于软件中的"接口"说明,描述了 Web 服务做什么、如何通讯,以及在那里能找到它. 但是由 XML 语言书写的 WSDL 文档并不包含语义说明,此时需要对 Web 服务赋予语义,基于此,本文提出一种将 WSDL 转换为 Radl-WS 的方法. 为遵循等价变换的原则,根据 WSDL 语言特点提炼出 5 条转换规则,进

收稿日期:2018-01-06
基金项目:国家自然科学基金(61762049,51567011,61462039,61662035,61562040),江西省科技课题(20171BAB202013,20151BBG70062)和江西省教育厅科技课题(GJJ14255,GJJ150349)资助项目.
通信作者:王昌晶(1977-),男,江西南昌人,教授,博士,主要从事软件形式化方法、形式规格说明方法、Web 服务方面的研究. E-mail:wcj771006@163.com

而以这 5 条转换规则为理论指导设计并实现了 WSDL→Radl-WS 自动转换系统.

1　基于 Radl-WS 代数规范的 Web 服务建模语言

1.1　Radl-WS 代数规范简介

根据 J. Misra 等[4] 提出的 Web 服务即站点(site)思想,设计基于 Radl-WS 代数规范的建模语言是模块化的,即一个 Radl-WS 代数规范由若干规格说明单元组成,每一个规格说明单元代表软件系统中的一个实体类型(类似 site). 一个软件实体类型可以是一个基本数据类型、ADT、类、组件、文档、XML 文档模式、SOAP 消息或 Web 服务等,将它们抽象为一个规格说明单元后,可将其视为一类对象,建模时不加以区别.

每个规格说明单元包括一个 < sort name > ,它是用来命名规格说明单元的标识符,对应到需求规格说明的软件实体,称为规格说明单元的主类(main sort). 每个规格说明单元可以进一步分解为 2 个更小的单元:签名单元(signature unit),它使用签名来定义其语法;公理单元(axiom unit),它使用若干必须满足的公理来定义其语义. 使用 BNF 描述 Radl-WS 代数规范如下:

```
< spec > ::= { < spec unit > }*
< spec unit > ::= spec  < sort name >
                  [extends  < extend sort list >]
                  [imports  < import sort list >]
                  < signature unit > ;
                  [< axioms unit >]
                 end-spec
```

其中签名单元(< signature unit >)和公理单元(< axioms unit >) BNF 描述如下:

```
< signature unit > ::= [sorts  < sort list > ] ;
                       ops  < operation list > ;
< axioms unit > ::= [axioms  < Axioms > ] ;
```

有 2 种方式可以由已存在的规格说明单元来构造一个新的规格说明单元,分别是 < extends > 和 < imports > 子句. 一个规格说明单元能够扩展(extends)其它规格说明的元素,这类似于面向对象中的继承关系. 一个规格说明单元能够使用(imports)其它规格说明单元,这类似于面向对象中的关联关系. 使用 < extend > 和 < import > 子句使得服务开发者可以以一种可组合的方式增量地开发(或验证)Web 服务.

1.2　Radl-WS 代数规范描述实例

形式化建模语言 Radl 能对基本数据类型、ADT、类和软件组件进行规格说明,并研制了相应的一系列(半)自动化工具,支持从 Radl 建模语言通过模型转换逐步生成可执行语言程序,如 Java、C# 等. 可以描述 Radl 软件规格说明如下[5-6]:

| 标志符声明 |

Q: 谓词表达式;

R: 谓词表达式;

其中标识符说明主要用于说明前、后置断言中出现的函数和变量的类型及属性. 以 Q 开头的谓词表达式称为算法的前置断言,以 R 开头的谓词表达式称为算法的后置断言,均为一阶谓词逻辑公式,分别用于表示算法的输入、输出参数必须满足的条件.

可以使用前期研究中提出的基于问题模式的形式化软件规格说明生成方法来指导书写 Radl 规格说明[7]. 可以进一步将操作集中的操作划分为如下 3 类:创建子(Creator)、转换子(transfer)和观察子(observer).

把前 2 类统称为构造操作,该类操作分别用来创建和修改规格说明中定义的类型实体,如上例中的操作 Create、Cons 和 Tail;后面一类称为检查操作,该类操作用来计算规格说明中定义的类型属性. 如上例中的操作 Head、Length.

使用 Radl-WS 代数规范描述 LIST 如下:

```
Spec LIST(sometype elem)
    sorts list
    imports integer
ops
creator:
    Create:→list//生成一个新列表
transformer:
    Cons:list,elem→list//列表增加一个元素
    Tail:list→list//取列表尾部结点
observer:
    Length:list→integer//取列表长度
    Head:list→integer//取列表头结点
axioms:
    var L:list,v:elem;
    Length(Create) = 0
    Length(Cons(L,v)) = Length(L) + 1
Head(Create) = Exception
Head(Cons(L,v)) = if L = Create then v else
```

Head(L)
Tail(Create) = Create
Tail(Cons(L,v)) = if L = Create then Create else
Cons(Tail(L),v)
end-spec.

2 使用 Radl-WS 代数规范对 Web 服务进行建模

2.1 Web 服务描述语言 WSDL

服务和 Web 服务可定义如下:服务是松散耦合的、封装了离散功能的可复用软件组件,它可以是分布的,根据标题来访问. Web 服务是这样一个服务,使用标准的 Internet 和基于 XML 的协议进行访问[8]. WSDL 文档是对 Web 服务的描述文档,其中包含了一系列描述某个 Web 服务的定义.

当服务使用者使用 Web 服务时,需要知道服务在哪里(URI)以及接口细节,这些信息会在服务描述中说明. 服务描述使用 XML 语言来书写,称为 WSDL,即 Web 服务描述语言. WSDL 的描述定义了有关 Web 服务的 3 个方面:

(i)服务做什么(what),称为接口,指定服务所支持的操作,并且定义服务发送和接受的消息的格式,包括 < types > 、< message > 和 < portType > 3 个部分;

(ii)如何通信(how),称为一个绑定,把抽象接口映射到一组具体的协议上,它指定了如何与一个 Web 服务通信的技术细节,对应包括 < binding > 部分;

(iii)在哪里能找到它(where),描述一个特定的 Web 服务实现的位置(它的端口),对应 < service > 部分. 在 WSDL 中,一个 Web 服务具有以下结构:

```
< definitions >
< types > < ! - - types definitions - - > < /types >
< message > < ! - - message definitions - - >
    < /message >
< portType >
< operation > < ! - - operation definition - - >
    < /operation >
< input > < ! - - input definition - - > < /input >
< output > < ! - - output definition > < /output >
< /portType >
< binding > < ! - - binding definition - - >
    < /binding >
< service > < ! - - service definition - - >
< port > < ! - - port definition > < /port >
< /service >
< /definition >
```

其中 < types > 类型定义是可选的,它描述了服务交换消息所使用的一些类型; < message > 消息定义描述了服务所处理的输入和输出消息部分,有 3 类消息,分别是 IN、OUT 和 INOUT; < portType > 端口类型定义是服务接口的描述部分,即服务为其它的服务或服务使用者提供的操作. < binding > 是服务所使用绑定的描述部分(如发送和接受消息的消息传输协议),默认是 SOAP,但也可以指定其它绑定. < service > 是端口描述部分,这是服务的物理位置,表示为统一资源标识符(URI)[8].

2.2 WSDL→Radl-WS 等价变换规则

Radl-WS 代数规范能够从 Web 服务的抽象接口描述中生成,包括类型定义、消息定义和端口类型定义,它是 WSDL 做什么的描述,相当于软件组件的提供接口. 下面给出从 WSDL→Radl-WS 代数规范的等价变换规则:

(i)名称变换规则:服务名称转换为 Radl-WS 代数规范的主类名;

(ii)类型定义变换规则:对于简单类型定义,将 WSDL 中每一个类型定义看成一个代数规范,每一个类型定义的类型名称转换为 Radl-WS 代数规范的类型,每一个类型名称也分别添加到 Radl-WS 代数规范的 import 列表中. 同时,将 WSDL 中每一个类型分别增加操作 setters 和 getters 到 Radl-WS 代数规范签名部分的 transformer 和 observer 中;对于复合(嵌套)类型定义,这在 XML 文档中十分常见,可以先将其变换为 Radl 语言中的记录类型;若是多重复合(嵌套)类型定义,可递归地进行变换. 变换之后再按照简单类型定义来进行处理;

(iii)消息定义变换规则:将 WSDL 中每一个消息定义看成一个代数规范,每一个消息定义的类型名称转换为 Radl-WS 代数规范的类名;每一个类型名称添加到整个 Radl-WS 代数规范的 import 列表中;

(iv)端口类型定义变换规则: 将 WSDL 中端口类型定义中的操作集转换为 Radl-WS 代数规范中的签名. 其中操作集中每一个操作的名称转换为签名中的操作名,操作集中的输入和输出消息类型名称转换为签名的定义域和协域[9]. 操作集中的输入 IN 和输出消息 OUT 消息名分别转化为 Radl 算法规格说明中标识符说明部分中的 in 和 out 变量名,操

作集中的输入和输出消息类型名称分别转换为标识符说明部分中的 in 和 out 变量类型;

(v)公理构造规则:将签名操作集中的所有操作分成 2 类:构造操作和检查操作. 由此可以根据结构化构造法来生成 Radl-WS 代数规范中所有的公理,即写下每一个构造操作上的每个检查操作的公理. 若有 m 个构造操作和 n 个检查操作,则就会有 $m \times n$ 个公理要定义. 结构化构造法保证了 Radl-WS 代数规范中公理集的完备性.

对于公理构造需要分为 2 种类型:第 1 种类型是相关操作只有对数据的读取(get)和写入(set),此时公理生成可以由自动转换系统来完成;第 2 种类型是服务内的操作互相复合,此种类型下的操作只能根据自然语言描述来半自动化生成.

3　WSDL→Radl-WS 自动转换系统

3.1　转换系统结构图

进行转换之前需要解析 WSDL,由于 WSDL 由 XML 语言描述,解析 XML 文档主要有 2 种方法:SAX(Simple API for XML)解析法、DOM 解析法[10-11](Document Object Model). SAX 是一个用于处理 XML 事件驱动的"推"模型,它是一个得到了广泛认可的 API. SAX 解析器,不像 DOM 那样建立一个完整的文档树,而是在读取文档时激活一系列事件,这些事件被推给事件处理器,然后由事件处理器提供对文档内容的访问,这种解析方式尤其适用于系统处理能力不足或 XML 文档过大(超过 500 K)的情形.

WSDL→Radl-WS 采用基于 DOM 的解析方式. DOM 实际上是以面向对象方式描述的文档模型,它首先将 XML 文档加载到内存中生成一颗 DOM 树,树的每个节点对应到 XML 中的标签元素,使用 XPath 定位技术可以快速地找到目标节点,非常适用于查找 WSDL 中的关键节点[12].

WSDL→Radl-WS 转换系统的系统结构图[13]如图 1 所示.

在技术上使用 Java 语言实现转换逻辑,整个工程属于 Java Web 应用,使用 Maven 进行工程管理. 其中转换系统分为 5 个模块:WSDL 导入、词法分析、语法分析、转换和出错处理模块,该系统的操作方式按照下面 4 个步骤来进行.

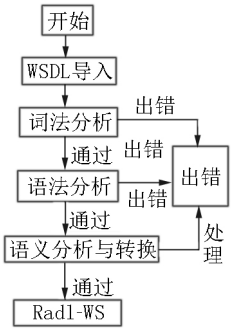

图 1　WSDL→Radl-WS 自动转换系统结构图

(i)WSDL 导入. 转换系统支持读取本地文件、手动输入以及读取云文件夹下的样例 3 种方式来导入 WSDL 文档. 其中第 1 种方式是先定位到本地文件,然后以文件流的形式读取到内存后上传到服务器端,若在云文件夹下没有同名文件则保存在云文件夹中,若存在同名文件则进行覆盖,然后将文件流解析成对应的 XML 文档显示到页面;打开云文件夹中的样例同样是以文件名作为参考,首先定位到工程目录下的指定文件夹,读取文件夹下所有文件名称生成列表显示到前端页面,然后根据选择的文件进行再次定位、读取.

(ii)词法分析. 这一步是检查文档中是否有拼写错误导致 XML 中节点无法配对. 具体做法是将 XML 的所有节点名称提取出来并依次压入栈中[13],若栈顶元素与新入栈的元素内容相同则 2 个元素同时弹出栈,若最终栈不为空则说明 XML 节点无法配对,将此信息封装成异常抛到上层进行页面显示.

(iii)语法分析. 这一步是使用 XPath 技术查找 DOM 树中的关键节点,即 WSDL 关键元素,例如 portType、service 等,若关键节点没有缺失,则尝试进行转换,否则抛出缺失信息.

(iv)转换. 转换模块的理论基础是前文提到的 5 项等价变换规则,根据 5 项规则又可将 Radl-WS 的生成分为 3 个模块,即类型声明、签名生成、公理构造. 类型声明是将复杂类型进行详解,列出复杂类型中的基本类型以及个数,信息提取自 < complexType > 节点;类型声明是对操作集的形式化说明,此处分为转换子与观察子 2 个部分,转换子对应复杂类型的写操作,观察子对应复杂类型的读操作;公理构造分为 2 种情况:第 1 种是只涉及到数据读取、写入的简单逻辑的 Web 服务,如查询天气预报,此时提取 < operation > 节点中 < input > 与 < output > 的内容放入 2 个集合,对 2 个集合进行双重遍历即可构造出公理,可以确保结果集的完备性. 第 2 种情况是具有复杂逻辑的 Web 服务,涉及到类型构造,此类 Web 服务不属于完全独立工作的服务,通常作为

软件系统的构件,多个服务进行组合从而构成完全独立的 Web 服务,对于这种情况的处理方式是将公理单独写入到 XML 文件中与 < operation > 同级,进行公理构造时将此部分内容直接读取到 Radl-WS 中.

出错管理模块是将错误、异常信息(包括控制台堆栈信息以及自定义异常信息)进行前台显示和日志记录. 一方面如果是用户操作不当引起的错误,可以使用户看到错误信息进行操作调节;另一方面是可以使平台管理人员通过日志信息清晰地定位到错误类型,方便调试.

3.2 转换实例

下面使用 WSDL→Radl-WS 转换工具举例说明如何将 WSDL 描述的 Web 服务通过等价变换转换为 Radl-WS 代数规范,该例选文献[14],转换工具已部署在阿里云服务器(http://106.14.226.54:8080/jsp/index.jsp). 该例给出了一个简单 Web 服务的抽象接口描述,其功能是:给定一个城市、所属国家和日期,返回指定城市和日期所记录的最高和最低温度. 这里只给出类型定义和端口定义,省略消息定义.

```
< definitions >
< types >
< xs: schema targetNameSpace = " http://.../weathns" xmlns: weathns = "http://…/weathns" >
< xs: element name = " PlaceAndDate" type = "pdrec" / >
< xs: element name = " MaxMinTemp" type = "mmtrec" / >
< xs:element name = "InDataFault" type = "errmess" / >
< xs:complexType name = "pdrec" >
< xs:sequence >
< xs: element name = " town" type = " xs: string" / >
< xs: element name = " country" type = " xs: string" / >
< xs:element name = "day" type = "xs:date" / >
< /xs:complexType >
//此处省略类型 mmtrec 和 errmess 的具体定义
< /schema >
< /types >
< interface name = "weatherInfo" >
< operation name = "getMaxMinTemps" pattern = "wsdlns: in-out" >
< input messageLabel = " In" element = "weathns:PlaceAndDate" / >
< output messageLabel = " Out" element = "weathns:MaxMinTemp" / >
< outfault messageLabel = " Out" element = "weathns:InDataFault" / >
< /operation >
< /interface >
< /definitions >
```

根据模型转换思想研制出的转换系统采用非常简洁的主界面,界面设计参考了文献[16]. 左侧为 WSDL 输入区域,右侧为 Radl-WS 输出区域. 从左至右功能性按钮分别为选择 WSDL 文件、打开文件、语法检查、进行转换、清空 WSDL、字体增大、字体缩小.

选择 WSDL 文件按钮是从本地打开 WSDL 文件,也支持手动输入在下方的文本框内;打开文件是根据选择的文件路径进行尝试读取文件内容,如果是不支持的文件类型或者文件位置有误则给出错误提醒;语法检查是进行词法、语法分析来检查读入的文档,如果出现错误则给出错误提醒;转换模块是将 WSDL 转换为 Radl-WS,如果出错则给出错误提醒,如果没有错误则将转换成功的 Radl-WS 代数规范填入右侧文本框;字体增大、缩小按钮的功能是将 Radl-WS 代数规范进行字体大小调整.

4 相关工作比较

在 Web 服务建模和模型转换方面,国内外的研究工作主要可以划分为 3 大类:基于本体的方法、基于模型驱动架构(MDA)或 UML 的方法,基于代数规范的方法. 其中比较典型的是 H. Zhu 等[17-18]提出的 SOFIA 代数规范语言以支持对 Web 服务的描述,该语言以行为代数和协代数为理论基础,对 WSDL 中的操作进行形式化说明,并研制出了相关工具(GoGrid)形成自动化过程,但该语言并不具备丰富的类型系统,也不支持泛型;SOFIA 语言的公理集也主要依靠启发性策略来指导,难以保证其完备性和极小性. 而本文提出的 Radl-WS 语言,是在 Radl 语言基础上进行扩展得到的,具备很多 Radl 语言优点,它支持丰富的类型系统、泛型机制;将操作集简化为构造操作和查看操作 2 类,由结构化构造法可以生成完备且最小的公理集. SOFIA 语言在对 WSDL 中类型进行转换时,并没有考虑到对复合类型(这在 XML 文档中十分常见)的转换,而借助 Radl 语言丰富的类型机制,Radl-WS 语言将复合类型定义为记录类型,十分方便地解决了这个问题.

5　总结与展望

本文提出一种新型的基于 Radl-WS 代数规范的 Web 服务建模语言,该语言是在 Radl 语言基础上进行扩展,使之更契合于 Web 服务的描述. 根据模型驱动的思想,依据 WSDL 到 Radl-WS 等价变换规则,提出了将 WSDL 语言描述的 Web 服务通过等价变换生成为 Radl-WS 建模语言的方法,并研制了 WSDL→Radl-WS 自动转换系统.

工业界一般直接设计所需的操作的可执行代码,然后通过 WSDL 描述后封装为 Web 服务供查询与匹配,本研究的下一步工作是通过模型转换的思想进一步将 Radl-WS 建模语言中操作生成为可执行代码,其目标是尽可能地提高转换的可靠性与效率.

6　参考文献

[1] 彭焕峰,黄纬,范大娟,等. Web 服务演化综述 [J]. 科学技术与工程,2015,15(30):63-70.

[2] Mcilraith S, Narayanan S. OWL-S: semantic markup for Web services [C] // Proceedings of the International Semantic Web Working Symposium (SWWS),2001.

[3] Hadley M J. Web application description language (WADL) [EB/OL]. [2017-10-17]. http://wadl.java.net/wadl20051116.pdf.

[4] Misra J, Cook W R. Computation Orchestration [J]. Software & Systems Modeling,2007,6(1):83-110.

[5] 王昌晶,薛锦云. Radl 形式规格说明相对正确性研究 [J]. 软件学报,2013,24(4):715-729.

[6] 左正康,薛锦云. Apla 中泛型约束机制研究 [J]. 软件学报,2015,26(6):1340-1355.

[7] 王昌晶,罗海梅,左正康. 基于问题模式的形式化软件规格说明生成方法 [J]. 计算机研究与发展,2013,50(2):352-360.

[8] MartinKalin. Java Web 服务:构建与运行 [M]. 南京:东南大学出版社,2010.

[9] Diaconescu R, Ţuţu I. Foundations for structuring behavioural specifications [J]. Journal of Logical and Algebraic Methods in Programming,2014,83(3/4):319-338.

[10] Li Chengkai. XML parsing, SAX/DOM [M]. New York: Springer US,2009.

[11] Friesen J. Parsing and creating XML documents with DOM [M]. New York: Springer US,2016.

[12] Berglund A, Boag S, Chamberlin D, et al. XML path language (XPath) 2.0 [J]. Optical Materials,2007,30(1):129-131.

[13] 徐华珍,薛锦云,朱小征. Apla→Java 程序生成系统中泛型机制实现方法研究 [J]. 江西师范大学学报:自然科学版,2017,41(1):52-55.

[14] Sommerville. 软件工程 [M]. 9 版. 程成,译. 北京:机械工业出版社,2011.

[15] 田方,石海鹤,左正康,等. 一种抽象泛型机制的新型 Java 实现 [J]. 江西师范大学学报:自然科学版,2016,40(1):77-82.

[16] 谢武平. Radl→Apla 程序生成系统及其可靠性研究 [D]. 南昌:江西师范大学,2009.

[17] Zhu Hong, Yu Bo. Algebraic specification of Web services [C] // International Conference on Quality Software, IEEE,2010:457-464.

[18] Liu Dongmei, Zhu Hong, Bayley I. SOFIA: an algebraic specification language for developing services [C] // IEEE, International Symposium on Service Oriented System Engineering, IEEE Computer Society,2014:70.

The Generation Method and Automatical Transformation System of WSDL→Radl-WS

ZHANG Qi[1], WANG Changjing[1*], LUO Haimei[2], ZUO Zhengkang[1], SHI Haihe[1], GUO Fan[1]

(1. College of Computer Information Engineering, Jiangxi Normal University, Nanchang Jiangxi 330022, China;
2. College of Physics and Communication Electronics, Jiangxi Normal University, Nanchang Jiangxi 330022, China)

Abstract: Describing the semantics of Web services accurately plays a crucial role in service discovery, execution, dynamic composition and interaction. Based on the Radl language, the paper extend it into the Radl-WS, and propose a novel Web service modeling language upon the Radl-WS algebraic specification. The modeling language unifies algebraic axiom approach and Hoare´s axiom approach to describe software specification. Moreover, in order to support modeling transformation, we transform Web service described by WSDL language into Radl-WS modeling language by a series of equivalence transformation rules. And then we develop an automatically transformation system to support that. Finally, running some real cases to demonstrate the practice effect of the method.

Key words: algebraic specification; Web service; model transformation; automatical transformation

(责任编辑:冉小晓)