计算机科学
COMPUTER SCIENCE

基于模型驱动的Web服务建模与三阶段模型转换方法

王昌晶, 丁希龙, 陈茜, 罗海梅, 左正康

引用本文

王昌晶, 丁希龙, 陈茜, 罗海梅, 左正康基于模型驱动的Web服务建模与三阶段模型转换方法[J]. 计算机科学, 2022, 49(11A): 211100055-14.

WANG Chang-jing, DING Xi-long, CHEN Xi, LUO Hai-mei, ZUO Zheng-kang. Web Service Modeling Based on Model-driven and Three-stage Model Transformation Method [J]. Computer Science, 2022, 49(11A): 211100055-14.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.211100055

基于模型驱动的 Web 服务建模与三阶段模型转换方法

王昌晶[1,2]　丁希龙[1]　陈　茜[1]　罗海梅[3]　左正康[1]
1 江西师范大学计算机信息工程学院　南昌 330022
2 江西师范大学管理科学与工程研究中心　南昌 330022
3 江西师范大学物理与通信电子学院　南昌 330022
(wcj771006@163.com)

摘　要　精确的描述 Web 服务的语义对 Web 服务的发现、执行、动态组合和交互至关重要。为支持 Web 服务建模,提出从抽象到具体 4 个模型:Radl-WS 服务需求模型、Apla 服务设计模型、Java 可执行代码、WSDL/RESTful API。为支持模型转换,进一步提出一种三阶段转换生成 Web 服务可执行代码的方法:第一阶段将 Radl-WS 服务需求建模语言转换为 Apla 服务设计语言;第二阶段将 Apla 服务设计语言通过相关转换工具生成可执行代码;第三阶段将可执行代码封装成服务。进而研究了三阶段模型转换的语义正确性,最后通过实例,展示了所提方法的实际效果。

关键词:代数规范;模型驱动;Web 服务;建模语言;模型转换
中图法分类号　TP301

Web Service Modeling Based on Model-driven and Three-stage Model Transformation Method

WANG Chang-jing[1,2],DING Xi-long[1],CHEN Xi[1],LUO Hai-mei[3] and ZUO Zheng-kang[1]
1 College of Computer Information and Engineering,Jiangxi Normal University,Nanchang 330022,China
2 Management science and Engineering Research Center,Jiangxi Normal University,Nanchang 330022,China
3 College of Physics and Communication Electronics,Jiangxi Normal University,Nanchang 330022,China

Abstract　Describing the semantics of web services accurately plays a crucial role in service discovery,execution,dynamic composition and interaction. In order to support web service modeling,this paper proposes four models from abstract to concrete:Radl-WS service requirement model,Apla service design model,Java executable code,and WSDL/RESTful API. To suppor model transformation,a three-phase method that generates an executable code by transformation is further proposed. The first stage transforms the Radl-WS service requirement modeling language into the Apla service design language,the second stage uses the Apla service design language to generate executable codes through related conversion tools,the third stage encapsulates the executable codes into services. Then the semantic correctness of the three-stage model transformation is studied. Through examples,the actual effect of the proposed method is demonstrated.

Keywords　Algebraic specification,Model-driven,Web services,Modeling language,Model transformation

1　引言

面向服务的计算(Service Oriented Computing,SOC)是一种利用服务作为分布式计算基础元素的计算范式。在该范式下,服务的发现、执行、动态组合和交互必须依赖对其语法和语义的精确理解。绝大多数服务语义的描述方法,如支持 Big Web Service 的 OWL-S[1] 和支持 RESTful Web 服务的 WADL[2],都是基于本体的。在基于本体的方法中,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。该描述方法具有易于开发者理解和计算机处理的优点,但是,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。代数规范于 1970 年被首次提出,其目的是为抽象数据类型(Abstract Data Type,ADT)提供一种与实现无关的规格说明语言。过去的几十年里,代数规范已经发展到了行为代数[3] 和余代数(co-algebra)[4-6],可以对并发系统、基于状态的系统和软件组件进行规格说明。

相比其他形式化方法,代数规范抽象层次很高且与实现细节无关。另一个十分吸引人的是它们能够支持(半)自动地生成代码[7-9]。这个特征对服务工程尤其重要,因为当服务进行动态组合时,代码必须能够(半)自动地即时生成与执行。

在前期研究工作中,我们提出了支持算法、软件组件、数据库系统形式化(半)自动开发的 PAR 方法及其支撑平台[10-15]。我们设计了建模语言 Radl,并扩展 Radl 为 Radl-WS,它能对基本数据类型、ADT、类和软件组件进行规格说明,并研制了相应的一系列(半)自动工具,支持从 Radl 建模语言通过

基金项目:国家自然科学基金(11804133,61862033);江西省教育厅科技重点项目(GJJ210307)
This work was supported by the National Natural Science Foundation of China(11804133,61862033) and Science and Technology Key Project of Education Department of Jiangxi Province(GJJ210307).
通信作者:左正康(kerrykaren@126.com)

模型转换逐步生成可执行语言程序，如 Java，C++，Python 等。

本文的创新点如下：

(1)为支持 Web 服务建模，提出从抽象到具体 4 个模型：Radl-WS 服务需求模型、Apla 服务设计模型、Java 可执行代码、WSDL/RESTful API。

(2)为支持模型转换，进一步提出了一种三阶段转换生成 Web 服务可执行代码的方法，第一阶段将 Radl-WS 需求建模语言转换为 Apla 服务设计语言，第二阶段利用转换器将 Apla 设计语言转换为可执行代码，最后将可执行代码封装为重量级 WSDL 和轻量级 RESTful API 服务。

本文第 2 节介绍模型驱动基本概念及三阶段 Web 服务生成方法；第 3 节对具体三阶段生成过程进行详细阐述，重点是第一阶段将 Radl-WS 服务需求建模语言转换为 Apla 服务设计语言；第 4 节对三阶段模型转换语义的正确性进行分析；第 5 节通过实例说明本文提出的三阶段模型转换方法；第 6 节是相关工作的比较；最后总结全文并展望未来。

2 模型驱动的三阶段 Web 服务转换生成方法

2.1 模型驱动架构

MDA(Model Drive Architecture)由对象管理组织(Object Management Group，OMG)提出。它以模型为核心，将实际问题抽象化并建立相关模型，然后对模型进行转换和精化，直至生成 Java，C++，Python 等具体的可执行代码，该过程是一个由抽象到具体的过程。MDA[16]建议应当产生以下 3 种类型的抽象系统模型：计算无关模型(Computational Independent Model，CIM)，平台无关模型(Platform Independent Model，PIM)和平台相关模型(Platform specific Model，PSM)。如图 1 所示。

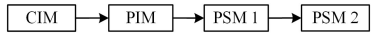

图 1 模型驱动的 3 个阶段

Fig. 1 Three stages of model-driven

2.2 三阶段 Web 服务转换生成方法

本文提出的 Web 服务生成方法包括 4 个模型和 3 个模型转换阶段。

4 个模型：Radl-WS 服务需求模型、Apla 服务设计模型、Java 可执行代码、WSDL/RESTful API。

3 个模型转换阶段：

(1)Radl-WS 服务需求模型→Apla 服务设计模型：通过提炼的等价变换规则转换得到。

(2)Apla 服务设计模型→Java 可执行代码：通过 PAR 方法及其支撑平台程序转换器自动生成。

(3)Java 可执行代码→WSDL/RESTful API：其中 Java 可执行代码封装为 WSDL 自动生成，Java 可执行代码封装为 RESTful API 半自动生成。

如图 2 所示，本文提出的 4 个模型可以很好地对应模型驱动的三类系统抽象模型。Radl-WS 服务需求模型对应 CIM，Apla 服务设计模型对应 PIM，经过 PAR 方法及其支撑平台系列转换器(Apla→Java，Apla→C++，Apla→Python 等)生成的可执行代码对应 PSM1，最后封装成的服务 WSDL/RESTful API 对应 PSM2。

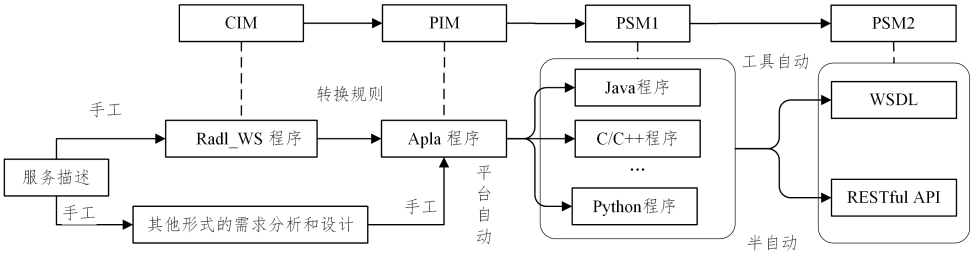

图 2 模型驱动与三阶段转换方法

Fig. 2 Model-driven and three-stage transformation method

3 基于模型驱动的三阶段生成过程

3.1 第一阶段：Radl-WS 服务需求模型到 Apla 服务设计模型

3.1.1 基于 Radl-WS 代数规范的 Web 服务建模语言

Radl-WS 是一种代数规范，使用 BNF 描述 Radl-WS 代数规范如下：

```
〈spec〉::={〈spec unit〉}*
〈spec unit〉::=Spec 〈sort name〉
        [where 〈generic and its constraints〉]
        [extends 〈extend sort list〉]
        [imports 〈import sort list〉]
        〈signature unit〉;
        [〈axioms unit〉]
end-spec
```

其中〈signature〉和〈axioms〉共享〈sort name〉，分别定义如下：

```
〈signature unit〉::=[sorts 〈sort list〉];
    ops 〈operation list〉;
〈axioms unit〉::=[axioms 〈axioms〉]
```

其中使用〈where〉子句，来指定一个规格说明单元的泛型及泛型约束；利用〈extends〉和〈imports〉子句，由已存在的规格说明单元来构造一个新的规格说明单元。

将〈where〉〈extends〉和〈imports〉子句结合起来，服务开发者就可以以一种可组合的方式来增量地开发(或验证)通用的 Web 服务，这尤其便于 Web 服务的按需动态组装。

使用我们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-WS 规格说明。可以进一步将操作集中的操作划分为 3 类：创建子、转换子和观察子。前两类统称为构造操作，该类操作用于创建和修改规格说明中定义的类型实体，如例 1 中的操作 Create，Cons 和 Tail；后面一类称为检查操作，该类操作用于计算规格说明中定义的类型属性，如例 1 中的操作 Head 和 Length。

例 1 使用 Radl-WS 代数规范描述 LIST。

```
1. Spec LIST(sometype elem)
```

```
2. where(Comparable〈elem〉)
3. imports integer
4.     sorts list
5. ops
6. creator:
7.     Create:－＞ list//生成一个新列表
8.       |[in L:list(elem);]|
9.       {Q:true}
10.      {R:L={}}
11. transformer:
12.    Cons:list,elem ->list//列表增加一元素
13.      |[in L:list(elem);e:elem;out L:list(elem);]|
14.      Q:#(L)＜n}
15.      {R:L′=L↑v}
16. observer:
17.    Length:list－＞integer//取列表长度
18.      |[in L:list(elem);out n:elem;]|
19.      {Q:true}
20.      {R:n=#(L)}
21. axioms
22. end-Spec
```

注意 LIST 规格说明中使用关键字 sometype 来定义泛型基本数据类型 elem;使用关键字 where 来定义泛型约束 Comparable,它刻画了一组可比较类型,Comparable 是 PAR 平台预定义泛型约束,存放在泛型约束库中,具体可参考文献[17]。

3.1.2 Apla 服务设计语言

本文使用 Apla 来对 Web 服务进行详细设计,Apla 是本团队研究开发的一种抽象程序设计语言(Abstract Programming Language)。Apla 的主要优点是便于程序的开发,简单实用,充分体现了抽象(包括数据抽象和操作/算法抽象)这一程序设计思想,并且由 Apla 设计的程序容易理解阅读和验证,也易于被转换成 Java,C++,Python 等可执行的程序设计语言。其详细设计的是对数据结构、算法及数据库应用程序的具体说明。Apla 这一过程设计语言不仅可以完成这些基本任务,而且还能对程序的正确性进行验证。

3.1.3 Radl-WS 服务需求模型到 Apla 服务设计模型的转换规则

对于 Radl-WS 服务需求模型转换为 Apla 服务设计模型,我们总结了如下的转换规则:

(1)名称转换规则:Radl-WS 代数规范的主类名,转换为 Apla 程序名。

(2)类型转换规则:Radl-WS 类型(包括简单类型与复杂类型)与 Apla 类型没有什么区别,Radl-WS 类型可以直接转换为 Apla 类型。需要注意的是 Radl-WS 自定义简单类型(包括数组、记录、子界和枚举类型)、预定义 ADT 类型(包括序列、集合、树和图类型)和自定义 ADT 类型(如数据库表)是位于主类名之前,由 import 关键字导入;而 Apla 记录类型需紧接着程序名,并且需给出详细定义。

(3)操作子转换规则:Radl-WS 操作子中没有返回值的操作转换为 Apla 中的过程(Procedure),有返回值的操作转换为 Apla 中的函数(Function);Radl-WS 的中关键字 in 声明的类型转换为过程或者函数的输入值类型,关键字 out 声明的类型转换为过程或者函数输出值类型;过程或函数的主要功能由 Radl-WS 的前后置断言转换而成(后文 4.1 节具体说明)。

3.2 第二阶段:Apla 服务设计模型生成为可执行代码

我们在 PAR 方法及其支撑平台上,研发了各种基于 Apla 的自动程序转换系统,如 Apla→C++,Apla→Java,Apla→Python 等[18],并进一步部署到云平台开放[1)]。本文使用 Apla 到 Java 的自动程序转换系统,其能够将 Apla 程序转换为对应的 Java 代码

3.3 第三阶段:可执行代码封装为服务

3.3.1 Java 代码封装为 WSDL

由 Java 语言函数生成 WSDL。WSDL 是一门基于 XML 的语言,用于描述 Web Services 以及如何对它们进行访问。本文以 MyEclipse2017 为开发工具来生成 WSDL,其生成步骤为:1)右键单击项目,并选择 New＞Other;2)在过滤器字段中输入 Web 服务,选择 Web 服务,然后单击 Next;3)选择 JAX-WS 2.0 为框架以及 Botton-up scenario 为策略,单击 Next;4)最后选择需要生成 WSDL 的 Java 类并且进行相应的设置就可生成 WSDL。

3.3.2 Java 代码封装为 RESTful API

由 Java 语言函数生成 RESTful API。目前 RESTful 架构风格服务已经成为提供 Web 服务的最主流技术。本文以 MyEclipse2017 为开发工具来生成 RESTful API,生成步骤为:1)右键单击项目,并选择 New＞Other;2)在过滤器字段中输入 Web 服务,选择 Web 服务,然后单击 Next;3)选择 REST 为框架、Botton-up scenario 为策略并且勾选 Create new Java bean,单击 Next;4)填写好信息,单击 Add;5)对访问接口进行相应的设置,点击 Finish,生成 RESTful API;6)可以根据实际需求对生成的代码进行修改。

4 三阶段模型转换的语义正确性

本节侧重于 Radl-WS 服务需求模型到 Apla 服务设计模型的正确性以及 Apla 服务设计模型到可执行代码的正确性。可执行代码封装为服务这一阶段是通过 MyEclipse 2017 开发工具自动/半自动完成的,在此不进行阐述。

4.1 Radl-WS 服务需求模型到 Apla 服务设计模型的正确性

Apla 服务设计语言一般对应于 Radl-WS 签名部分的具体实现,主要是签名部分中的操作(算法)实现。因此,本节将主要讨论如何将 Radl-WS 代数规范签名部分中的操作生成为 Apla 服务设计语言。

其主要思路是同时组合代数公理方法和 Hoare 公理方法,其中描述整个 Web 服务使用代数公理方法,即使用代数规范;而描述 Web 服务中的操作则使用 Hoare 公理方法,即使用前后置断言。代数公理方法易于建立整个 Web 服务的语义框架,如前文所述;而 Hoare 公理方法便于指导操作(算法)的形式化推导和验证。对 Radl-WS 代数规范中的操作使用 Hoare 公理方法,可以结合我们在长期研究中提出的 PAR 方法及其支撑平台,来自动化(半自动化)地生成可执行代码[10-15]。

在本文中,我们将操作(算法)划分为两类:第一类是复杂的操作(算法),它涉及到重复的计算;第二类是简单的操作

1) http://219.229.250.13:8081/runcloud/

(算法),它不涉及重复的计算。第一类包括循环和递归,由于其涉及到复杂数据结构和循环不变式,十分适合使用本节介绍的形式化推导和形式化验证的方法,这是本文的特色和重点着墨之处;而第二类由于不涉及循环和递归,可直接写出Apla代码,并使用简单程序的验证方法来便捷地进行形式化验证,只需使用到Hoare公理的几条公理和推导规则(主要是赋值公理、复合语句规则和条件语句规则)。如不特别说明,本文提到的操作(算法)主要是指第一类。

下面从Hoare公理出发,通过两个例子来分别说明如何对操作(算法)进行形式化推导和验证,以生成Apla代码。

4.1.1 Radl-WS操作的形式化推导生成方法

下面以例子1中Length操作为例,从该操作的前后置断言出发,给出该操作的形式化推导过程,其余操作可以通过类似的方法来推导得到。Length操作的标识符说明部分如下:

|[in L:list(elem);out m:integer;]|

(1)分划子问题

将原问题记为 $p(k)$,根据这个问题的属性,可分划如下:

$$p(k)=F(p(k-1),k) \tag{1}$$

(2)构造递推关系 F

$$p(k)=k=\sharp(L)=(\Sigma i:0\leqslant i\leqslant k-1:1) \tag{2}$$

≡{范围分裂}

$$(\Sigma i:0\leqslant i\leqslant k-2:1)+(\Sigma i:i=k:1) \tag{3}$$

≡{单点范围}

$$(\Sigma i:0\leqslant i\leqslant k-2:1)+1 \tag{4}$$

≡{使用 p 的定义}

$$p(k-1)+1 \tag{5}$$

≡由此得到递推关系:

$$p(k)=p(k-1)+1 \tag{6}$$

(3)生成Radl-WS算法和循环不变式

假设list是使用字符数组来实现,由初始化条件和递推关系,使用Radl-WS语言表达算法如图3所示(若list使用链表来实现,则主要需将数组索引操作 $k++$ 修改为指针直接操作 $kp++$。)。

```
1. ALGORITHM:Length(elem a[])
2. |[var p:integer;k:integer;]|
3. {Q∧R}
4. BEGIN:p=0,k=0++1;
5. TERMINATION:a[k]='\\0';
6. RECUR:p(k)=p(k-1)+1
7. END
```

图3 Length操作的Radl-WS程序

Fig. 3 Radl-WS program of length operation

由循环不变式新开发策略[19]给出 k 的变化范围,且每个子问题的解用变量 T 来存放,可得到循环不变式如下:

$$\rho:T=p(k)\wedge 0\leqslant k \tag{7}$$

转换生成可执行代码使用Radl到Apla转换工具[20-21],可以生成Apla程序,其关键代码如图4所示。

```
1. p=0;k:=0;
2. do(a[k]≠'\\0')
3.    p=p+1;k:=k+1;
4. od
```

图4 Length操作的Apla程序

Fig. 4 Apla program of length operation

4.1.2 Radl-WS操作的形式化验证生成方法

对于基于线性数据结构(如集合、序列、包)上Web服务的操作(算法),一般可以通过形式化推导来生成;但是对于基于非线性数据结构(如树、图)上Web服务的操作(算法),则需要通过形式化验证的方法来生成。为便于说明,我们先使用Radl-WS代数规范给出二叉树BTREE的形式化描述,如算法1所示(这里只给出主要的)。

算法1 二叉树RTREE

```
1. SpecBTREE(elem)(sometype elem)
2. where(Comparable〈elem〉)
3. importselem,void,boolean,list
4. sortsbtree
5. op
6. creator:
7. SetEmpty:-> btree;//构造一棵空树
8. |[out T:btree(elem);]|
9. {Q:true}
10. {R:T=void }
11.          ……//(此处省略了部分代码)。
12. transformer:
13. Pre:btree->list //前序遍历二叉树
14. |[in T:btree(elem);out X:list(elem);]|
15. {Q:true}
16. {R:X=Pre(T)}
17. {Q:true}
18. {R:X=Post(T)}
19. axioms
20. end-spec
```

下面以前序遍历操作为例,从该操作的前后置断言出发,给出该操作的形式化验证过程,其余操作可以通过类似的方法验证得到。首先使用非形式化的方法导出前序遍历的非递归算法,然后使用Dijstra-Gries的标准程序证明方法[22]来进行形式化验证。设 T 是一棵有穷树,$Pre(T)$ 表示前序遍历 T。我们使用Radl-WS中两个ADT:二叉树btree和序列list来描述操作 $Pre(T)$ 的前后置断言,并将 $Pre(T)$ 存放在一序列变量 X 中。由此,得到Radl-WS语言描述的前后置断言如下:

|[in T:btree(elem);out X:list(elem);]|

{Q:true}

{R:X=Pre(T)}

(1)构造循环不变式

根据后序遍历二叉树的定义,若 $T=\%$,则 $Pre(T)=[\,]$;若 $T\neq\%$,分划 $Pre(T)$:

$$Pre(T)=T.d\uparrow Pre(T.l)\uparrow Post(T.r) \tag{8}$$

基于式(8)设计一个递归算法十分方便,但效率很低。为了得到一个非递归的算法程序,我们继续前序遍历子树 $T.l$ 和 $T.r$,即:

$$\begin{aligned}Pre(T)&=T.d\uparrow Pre(T.l)\uparrow Pre(T.r)\\&=T.d\uparrow T.l.d\uparrow Pre(T.l.l)\uparrow Pre(T.l.r)\uparrow Pre(T.r)\\&=T.d\uparrow T.l.d\uparrow T.l.l.d\uparrow Pre(T.l.l.l)\uparrow Pre(T.l.l.r)\uparrow Pre(T.l.r)\uparrow Pre(T.r)\end{aligned}$$

$$= T.d \uparrow T.l.d \uparrow T.l.l.d \uparrow T.l.l.l.d \uparrow Pre(T.l.l.l.l) \uparrow Pre(T.l.l.l.r) \uparrow Pre(T.l.l.r) \uparrow Pre(T.l.r) \uparrow Pre(T.r) \quad (9)$$

由上述推导可得到设计 $Pre(T)$ 操作非递归算法的总策略。引入 3 个变量 X，q 和 S，其中 X 用来存放所遍历结点的序列，比如 $T.d \uparrow T.l.d \uparrow T.l.l.d \uparrow$；$q$ 用来存放将要访问的 T 的子树，比如 $T.l.l.l$；S 是一个起堆栈作用的序列变量，用于存放 T 的右子树，比如 $Pre(T.l.l.r) \uparrow Pre(T.l.r) \uparrow Pre(T.r)$。为开发循环不变式，我们定义函数 $F(S)$ 如下：

$$F[\,] = [\,] \quad (10)$$

$$F(S \uparrow q) = Pre(q) \uparrow F(S) \quad (11)$$

基于循环不变式新的开发策略[19]，得到循环不变式如下：

$$\rho: Pre(T) = X \uparrow Pre(q) \uparrow F(S) \quad (12)$$

(2)生成非递归 Apla 程序

根据上述设计非递归算法的总策略和循环不变式 ρ，可以得到如图 5 所示的操作 $Pre(T)$ 的 Apla 程序。

```
1. Program Pre
2. X:list(elem);T,q:btree;
3. S:list(btree);
4. begin
5.    X,q,S:=[ ],T,[ ];
6.    do;
7.       q≠%→ X,q,S:=X↑q.d,q.l,S↑q.r;
8.       []q=%∧S≠[ ]→ q,S:=S.t,S[h…t-1];
9.    od;
10. end.
```

图 5　Pre(T)的 Apla 程序

Fig. 5　Apla program of Pre(T)

4.2　Apla 服务设计语言生成为可执行代码的正确性

模型驱动的 Web 服务开发，是对具体问题进行抽象建模，然后通过逐步的转换、精化，生成具体的可执行代码的过程。其中每一步的精化过程都可以看成是抽象模型 M_A 到具体模型 M_C 一个精化演算过程。

本节主要阐述抽象 Apla 服务设计语言通过模型精化演算，生成可执行的 Java 代码的正确性理论。下面给出模型精化演算的各个概念。

定义 1　软件系统的模型规范是一种基于状态的抽象数据类型，它可以表示为三元组 $M(\beta, \beta_0, \Omega)$。其中 β 为 M 的有效状态集合，$\beta \subseteq \beta_0$ 为系统初始状态的集合；Ω 是若干运算的集合，Ω 中的每个运算 op 可表示为一个二元组 $op=(S,R)$。其中，$S \subseteq \beta$ 是一个状态子集，$R \subseteq \beta \times \beta$ 是状态间的关系，满足条件：$(dom\ R = \{S \in \beta \mid \exists\, S' \in \beta: \langle S, S' \rangle \in R\}): S \subseteq dom\ R$。

定义 2　对于抽象模型 M_A 和具体模型 M_C，$M_A=(A\beta, A\beta_0, A\Omega)$，$M_C=(C\beta, C\beta_0, C\Omega)$，$M_C$ 关于 M_A 的一个抽象函数是一个自 $C\beta$ 到 $A\beta$ 的函数 $Abs: C\beta \rightarrow A\beta$，满足：

(1)Abs 是全函数。

(2)$Abs(C\beta_0) \subseteq A\beta$。

定义 3　具体模型 M_C 的运算 $C_{op} \in C_\Omega$ 是抽象模型 M_A 中对应运算 $A_{op} \in A_\Omega$ 的正确实现，即 C_{op} 在函数 Abs 解释下满足 A_{op}，也就是满足以下两个条件：

(1) $\forall C_s \in C\beta \cdot Pre\_A_{op}(Abs(C_s)) \Rightarrow Pre\_C_{op}(C_s)$。

(2) $\forall C_s, C_s' \in C\beta \cdot Pre\_A_{op}(Abs(C_s)) \wedge Post\_C_{op}(C_s', C_s) \Rightarrow Post\_A_{op}(Abs(C_s'), Abs(C_s))$。

定义 3 的条件说明：

条件(1)的说明：在抽象函数 Abs 的解释下，作用在 $A\beta$ 所描述的状态空间上的操作模式 A_{op} 终止时，作用于 $C\beta$ 所描述的状态空间上的操作模式 C_{op} 也能够终止，即要求 C_{op} 的前置条件比 A_{op} 的前置条件弱。

条件(2)的说明：在抽象函数 Abs 的解释下，作用在 $A\beta$ 所描述的状态空间上的操作模式 A_{op} 终止和作用于 $C\beta$ 所描述的状态空间上的操作模式 C_{op} 都终止时，$C\beta$ 作用结束后的状态也能对应于 A_{op} 作用终止后的状态空间上，即要求 C_{op} 的后置条件比 A_{op} 的后置条件强。

5　实例研究

本例选自 PayPal，定义了两个类：交易记录类 TransRecord 和支付人记录类型 PayerInfoType。案例功能实现：实现 TransRecord 的新增、删除、修改和查询操作；实现业务流程 setExpressCheckout，getExpressCheckoutDetail 和 doExpressCheckout。其中 setExpressCheckout 根据买卖信息获得 token 值、交易金额、支付状态(正在进行或者取消)，以及支付人信息，并根据这些信息创建新的记录并返回 token 值。getExpressCheckoutDetail 指定服务返回支付人信息。doExpressCheckout 通过最后的支付信息来更新记录，完成服务，业务流程如图 6 所示。

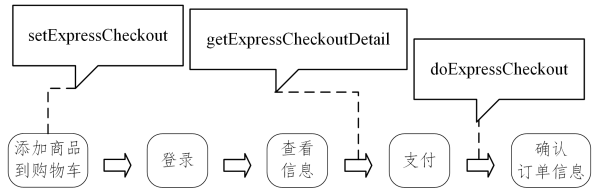

图 6　PayPal 业务流程

Fig. 6　Business process of PayPal

(1)顾客将商品加入购物车后点击支付，启动 setExpressCheckout 服务。

(2)然后登录，确认用户信息。

(3)查看支付相关信息，点击继续，调用 getExpressCheckoutDetail 服务返回支付人信息。

(4)确认订单并支付，调用 doExpressCheckout 服务完成订单。

(5)收到订单确认信息。

首先通过 Radl-WS 服务需求建模语言定义交易记录 TransRecord 和支付人信息 PayerInfoType，并对交易记录 TransRecord 进行增删改查的数据建模以及基于增删改查的 3 个服务建模；再通过转换规则将 Radl-WS 服务需求建模语言转换为 Apla 服务设计语言；然后由 Apla 转换工具转换为 Java 可执行代码；最后封装成服务。

5.1　Radl-WS 服务需求模型到 Apla 服务设计模型

通过转换规则，可以将 Radl-WS 服务需求建模语言转换为 Apla 服务设计语言。限于篇幅，这里只给出数据模型中的新增记录(createRecord)和服务模型中的 setExpressCheckout，由 Radl-WS 服务需求建模语言转换为 Apla 服务设计语言。具体转换如图 7 和图 8 所示。

```
typeTransRecord=record
   token:string;
   transAmount:real;
   paymentStatus:string;
   pid:integer;
end;
typePayerInfoType=record
   age:integer;
   name:string;
   sex:string;
   pid:integer;
end;
specPayPalDM
Imports:TransRecord,PayerInfoType
sorts:transrecord:table(TransRecord);
ops:
creator:
   //新增记录//
   createRecord:→ table(TransRecord)
   |[in rec: TransRecord, out result. transrecord: table
(TransRecord)]|
   {Q:true}
   {R:result. transrecord=this. transrecord{rec}
end-spec
```

转换规则

```
programPayPalDM;
type PayerInfoType=record
      name:string;
      sex:string;
      age:integer;
      pid:integer;
end;
type TransRecord=record
      token:string;
      transAmount:real;
      paymentStatus:string;
      pid:integer;
end;
var transrecord:table(TransRecord);
   //新增记录//
procedurecreateRecord(rec:TransRecord);
begin
   transrecord:=transrecord∪{rec};
writeln(transrecord);
end;
```

图 7 createRecord 的转换

Fig. 7 Transformation of createRecord

```
Spec PayPalDC
Imports: PayPalDM, TransRecord, PayerInfo-
Type
   sorts:ppdm:PayPalDM;rec:TransRecord;
ops:
   transformer;
   setExpressCheckout:real→ string
   |[in sPaymentAmount: real; out result:
string;]|
   {Q:true}
      {R: rec. token ≠ null ppdm. findRecord-
ByKey(rec. token)=null}
      {R: rec. pid ≠ null rec. transAmount =
sPaymentAmount result=rec. token}
      {R:rec. paymentStatus=InProgress}
{R:ppdm′=ppdm. createRecord(rec)}
      end-spec
```

转换规则

```
programPayPalDC;
type PayerInfoType=record
      name:string;
      sex:string;
      age:integer;
      pid:integer;
end;
type TransRecord=record
      token:string;
      transAmount:real;
      paymentStatus:string;
      pid:integer;
end;
var transrecord:table(TransRecord);
   rec:TransRecord;
token:string;
   pid:integer;
//通过主键查询//
functionfindRecordByKey(key:string):TransRecord;
begin
transrecord:=∏{token,transAmount,paymentStatus,pid}(σ{token=
key}(transrecord));
   findRecordByKey:=transrecord;
end;
//新增记录//
procedurecreateRecord(rec:TransRecord);
begin
   transrecord:=transrecord∪{rec};
end;
//setExpressCheckout//
function setExpressCheckout(sPaymentAmount:real):string;
begin
   token:=1;
   pid:=1;
   rec. token:=token;
   rec. pid:=pid;
if
rec. token≠null∧findRecordByKey(rec. token)=null∧rec. pid≠null
∧rec. paymentStatus:="Inprogress"∧rec. transAmount:=sPaymen-
tAmount;
createRecord(rec);
fi
   setExpressCheckout:=rec. token
end;
```

图 8 setExpressCheckout 的转换

Fig. 8 Transformation of setExpressCheckout

5.2 Apla 服务设计模型生成为可执行代码

在该阶段使用 Apla 到 Java 的自动程序转换系统,从而实现 Apla 服务设计语言到 Java 可执行代码的转换。数据模型的 Apla 服务设计语言转换为 Java 可执行代码过程如下。

```
//数据模型//
program PayPalDM;
  type PayerInfoType=record
    name:string;
    sex:string;
    age:integer;
    pid:integer;
  end;
  type TransRecord=record
    token:string;
    transAmount:real;
    paymentStatus:string;
    pid:integer;
  end;
  var transrecord:table(TransRecord);
//新增记录//
procedure createRecord(rec:TransRecord);
begin
  transrecord:=transrecord∪{rec};
writeln(transrecord);
end;
//删除记录//
procedure deleteRecord(key:string);
begin
  deleteTuple(transrecord,token,key);
writeln(transrecord);
  end;
//更新记录//
procedure updateRecord(rec:TransRecord);
begin
  deleteRecord(rec.token);
  transrecord:=transrecord∪{rec};
writeln(transrecord);
end;
//通过主键查询//
function findRecordByKey(key:string):TransRecord;
begin
  transrecord:=∏{token,transAmount,paymentStatus,pid}(σ{token=key}
(transrecord));
  findRecordByKey:=transrecord;
writeln(transrecord);
end;
begin
//业务逻辑//
end.
```

数据模型

```
import java.util.*;
import java.sql.*;
import java.io.*;
import java.lang.*;
import strulib.*;
import leeko.Table;
import javax.transaction.xa.Xid;
public class PayPalDM{
  //新增记录//
public void createRecord( record rec) {
  t.insert("transrecord",);
    System.out.println(transrecord);
  }
//删除记录//
public void deleteRecord(String key) {
  t.deleteTuple("transrecord","token","key");
    System.out.println(transrecord);
  }
//更新记录//
public void updateRecord( record rec) {
  this.deleteRecord(rec.token);
  t.insert("transrecord",);
    System.out.println(transrecord);
  }
//通过主键查询//
public record findRecordByKey(String key) {
  record returnvalue;
    sql=t.project("token,transAmount,paymentStatus,pid",t.choose("token
        =key","transrecord"));
    System.out.println("[关系代数对应的 SQL 语句为:"+sql+"]");
    System.out.println();
    t.assign("transrecord",sql);
    returnvalue=transrecord;
    System.out.println(transrecord);
    return returnvalue;
  }
public void maincall( ){
//业务逻辑//
  }
public static void main(String arg[]){
    PayPalDM aco=new PayPalDM( );
    aco.maincall();
    System.out.println();
  }
private class PayerInfoType {
  String name;
  String sex;
  int age;
  int pid;
  }
private class TransRecord {
  String token;
  double transAmount;
  String paymentStatus;
  int pid;
  }
}
```

服务模型的 Apla 服务设计语言转换为 Java 可执行代码过程如下。

```
//契约建模//
program PayPalDC;
  type PayerInfoType=record
    name:string;
    sex:string;
    age:integer;
    pid:integer;
  end;
  type TransRecord=record
    token:string;
    transAmount:real;
    paymentStatus:string;
    pid:integer;
```

```
  end;
  var transrecord:table(TransRecord);
    payerinfotype:table(PayerInfoType);
    payerInfo:PayerInfoType;
    rec:TransRecord;
//新增记录//
procedure createRecord(rec:TransRecord);
begin
  transrecord:=transrecord∪{rec};
writeln(transrecord);
end;
//删除记录//
procedure deleteRecord(key:string);
begin
  deleteTuple(transrecord,token,key);
  writeln(transrecord);
end;
//更新记录//
procedure updateRecord(rec:TransRecord);
begin
  deleteRecord(rec.token);
  transrecord:=transrecord∪{rec};
  writeln(transrecord);
end;
//通过主键查询//
function findRecordByKey(key:string):TransRecord;
begin
  transrecord:=∏{token,transAmount,paymentStatus,pid}(σ{token=key}
(transrecord));
  findRecordByKey:=transrecord;
writeln(transrecord);
end;
//setExpressCheckout//
function setExpressCheckout( sPaymentAmount:real):string;
begin
  rec.pid:=1;
  rec.token:=String.valueOf(123322);
  rec.transAmount:=transAmount;
  rec.paymentStatus :="InProcessed";
  createRecord(newrec);
writeln(rec.token);
  setExpressCheckout:= rec.token;
end;
//getExpressCheckoutDetails//
function getExpressCheckoutDetails(g_token:string):PayerInfoType;
begin
  rec:=findRecordByKey(g_token);
  payerInfo:=∏{*}(σ{pid=rec.pid}(payerinfotype));
writeln(payerInfo);
  getExpressCheckoutDetails:=payerInfo;
end;
//doExpressCheckout//
function doExpressCheckout (dPaymentAmount :real;dToken:string):boole-
an;
begin
//只考虑成功的情况//
  rec.token := dToken;
  rec.transAmount:=dPaymentAmount ;
  rec.paymentStatus := "Processed";
  rec.pid:= 1;
  createRecord(newrec);
  doExpressCheckout:=true;
writeln(true);
end;
begin
  //业务逻辑//
end.
```

契约模型

```
import java.util.*;
import java.sql.*;
import java.io.*;
import java.lang.*;
import strulib.*;
import leeko.Table;
import javax.transaction.xa.Xid;
public class PayPalDC{
    PayerInfoType payerInfo=new PayerInfoType( );
    TransRecord rec=new TransRecord( );
  //新增记录//
public void createRecord( record rec) {
     t.insert("transrecord","'"+rec.token+"'"+","+rec.transAmount
     +","+"'"+rec.paymentStatus+"'"+","+rec.pid);
    System.out.println(transrecord);
  }
//删除记录//
public void deleteRecord(String key) {
  t.deleteTuple("transrecord","token","key");
    System.out.println(transrecord);
  }
//更新记录//
public void updateRecord( record rec) {
  this.deleteRecord(rec.token);
  t.insert("transrecord","'"+rec.token+"'"+","+rec.transAmount+","
  +"'"+rec.paymentStatus+"'"+","+rec.pid);
    System.out.println(transrecord);
  }
//通过主键查询//
public record findRecordByKey(String key) {
  record returnvalue;
    sql=t.project("token,transAmount,paymentStatus,pid",t.choose("token
        =key","transrecord"));
    System.out.println("[关系代数对应的 SQL 语句为: "+sql+"]");
    System.out.println();
    t.assign("transrecord",sql);
    returnvalue=transrecord;
    System.out.println(transrecord);
    return returnvalue;
  }
//setExpressCheckout//
public String setExpressCheckout(double sPaymentAmount) {
  String returnvalue = "";
  rec.pid=1;
  rec.token=String.valueOf(123322);
  rec.transAmount=transAmount;
  rec.paymentStatus="InProcessed";
  this.createRecord(newrec);
  System.out.println(rec.token);
    returnvalue=rec.token;
  return returnvalue;
  }
//getExpressCheckoutDetails//
public record getExpressCheckoutDetails(String g_token) {
  record returnvalue;
  rec=findRecordByKey(g_token);
    sql=t.project("*",t.choose("pid=rec.pid","payerinfotype"));
    System.out.println("[关系代数对应的 SQL 语句为: "+sql+"]");
    System.out.println();
    System.out.println(payerInfo);
    returnvalue=payerInfo;
  return returnvalue;
}
//doExpressCheckout//
public boolean doExpressCheckout(double dPaymentAmount,String dToken) {
  boolean returnvalue = false;
  //只考虑成功的情况//
  rec.token=dToken;
  rec.transAmount=dPaymentAmount;
  rec.paymentStatus="Processed";
```

```
  rec.pid=1;
  this.createRecord(rec);
    System.out.println("true");
    returnvalue=true;
    return returnvalue;
    }
public void maincall( ){
//业务逻辑//
  }
public static void main(String arg[]){
  PayPalDC aco=new PayPalDC( );
  aco.maincall();
  System.out.println();
  }
private class PayerInfoType {
  String name;
  String sex;
  int age;
  int pid;
  }
private class TransRecord {
  String token;
  double transAmount;
  String paymentStatus;
  int pid;
  }
}
```

5.3 可执行代码封装为服务

根据 3.3.2 节和 3.3.3 节中的操作,可以将本案例的 Java 代码封装成相应的 WSDL 服务和 RESTful API,然后通过 RESTful API 和客户需求,给出对应的案例接口说明。数据模型生成的 WSDL 文件如下,限于篇幅,这里只给出 createRecord和 setExpressCheckout 的 RESTful API 代码以及对应的接口说明。

```
〈?xmlversion="1.0"encoding="UTF-8"?〉〈! -- Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.3-hudson-390-. --〉〈definitionsxmlns="http://schemas.xmlsoap.org/wsdl/"xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"xmlns:tns="http://daoImpl/"xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"xmlns:xsd="http://www.w3.org/2001/XMLSchema"name="PayPalDMService"targetNamespace="http://daoImpl/"〉
〈types〉
〈xsd:schema〉
〈xsd:importnamespace="http://daoImpl/"schemaLocation="PayPalDMService_schema1.xsd"/〉
〈/xsd:schema〉
〈/types〉
〈messagename="updateRecord"〉
〈partelement="tns:updateRecord"name="parameters"/〉
〈/message〉
〈messagename="updateRecordResponse"〉
〈partelement="tns:updateRecordResponse"name="parameters"/〉
〈/message〉
〈messagename="updateSeller"〉
〈partelement="tns:updateSeller"name="parameters"/〉
〈/message〉
〈messagename="updateSellerResponse"〉
〈partelement="tns:updateSellerResponse"name="parameters"/〉
〈/message〉
〈messagename="createRecord"〉
〈partelement="tns:createRecord"name="parameters"/〉
〈/message〉
〈messagename="createRecordResponse"〉
〈partelement="tns:createRecordResponse"name="parameters"/〉
〈/message〉
〈messagename="deleteRecord"〉
〈partelement="tns:deleteRecord"name="parameters"/〉
〈/message〉
〈messagename="deleteRecordResponse"〉
〈partelement="tns:deleteRecordResponse"name="parameters"/〉
〈/message〉
〈messagename="updatePayer"〉
〈partelement="tns:updatePayer"name="parameters"/〉
〈/message〉
〈messagename="updatePayerResponse"〉
〈partelement="tns:updatePayerResponse"name="parameters"/〉
〈/message〉
〈messagename="findPayerInfoByid"〉
〈partelement="tns:findPayerInfoByid"name="parameters"/〉
〈/message〉
〈messagename="findPayerInfoByidResponse"〉
〈partelement="tns:findPayerInfoByidResponse"name="parameters"/〉
〈/message〉
〈messagename="findSellerInfoByid"〉
〈partelement="tns:findSellerInfoByid"name="parameters"/〉
〈/message〉
〈messagename="findSellerInfoByidResponse"〉
〈partelement="tns:findSellerInfoByidResponse"name="parameters"/〉
〈/message〉
〈messagename="findRecordByCriteria"〉
〈partelement="tns:findRecordByCriteria"name="parameters"/〉
〈/message〉
〈messagename="findRecordByCriteriaResponse"〉
〈partelement="tns:findRecordByCriteriaResponse"name="parameters"/〉
〈/message〉
〈messagename="creatToken"〉
〈partelement="tns:creatToken"name="parameters"/〉
〈/message〉
〈messagename="creatTokenResponse"〉
〈partelement="tns:creatTokenResponse"name="parameters"/〉
〈/message〉
〈messagename="findRecordByKey"〉
〈partelement="tns:findRecordByKey"name="parameters"/〉
〈/message〉
〈messagename="findRecordByKeyResponse"〉
〈partelement="tns:findRecordByKeyResponse"name="parameters"/〉
〈/message〉
〈portTypename="PayPalDMDelegate"〉
〈operationname="updateRecord"〉
〈inputmessage="tns:updateRecord"/〉
〈outputmessage="tns:updateRecordResponse"/〉
〈/operation〉
〈operationname="updateSeller"〉
〈inputmessage="tns:updateSeller"/〉
〈outputmessage="tns:updateSellerResponse"/〉
〈/operation〉
〈operationname="createRecord"〉
〈inputmessage="tns:createRecord"/〉
〈outputmessage="tns:createRecordResponse"/〉
〈/operation〉
〈operationname="deleteRecord"〉
〈inputmessage="tns:deleteRecord"/〉
〈outputmessage="tns:deleteRecordResponse"/〉
〈/operation〉
〈operationname="updatePayer"〉
〈inputmessage="tns:updatePayer"/〉
〈outputmessage="tns:updatePayerResponse"/〉
〈/operation〉
〈operationname="findPayerInfoByid"〉
〈inputmessage="tns:findPayerInfoByid"/〉
〈outputmessage="tns:findPayerInfoByidResponse"/〉
〈/operation〉
〈operationname="findSellerInfoByid"〉
〈inputmessage="tns:findSellerInfoByid"/〉
〈outputmessage="tns:findSellerInfoByidResponse"/〉
```

```
</operation>
<operationname="findRecordByCriteria">
<inputmessage="tns:findRecordByCriteria"/>
<outputmessage="tns:findRecordByCriteriaResponse"/>
</operation>
<operationname="creatToken">
<inputmessage="tns:creatToken"/>
<outputmessage="tns:creatTokenResponse"/>
</operation>
<operationname="findRecordByKey">
<inputmessage="tns:findRecordByKey"/>
<outputmessage="tns:findRecordByKeyResponse"/>
</operation>
</portType>
<bindingname="PayPalDMPortBinding"type="tns:PayPalDMDelegate">
<soap:bindingstyle="document"transport="http://schemas.xmlsoap.org/
soap/http"/>
<operationname="updateRecord">
<soap:operationsoapAction=""/>
<input>
<soap:bodyuse="literal"/>
</input>
<output>
<soap:bodyuse="literal"/>
</output>
</operation>
<operationname="updateSeller">
<soap:operationsoapAction=""/>
<input>
<soap:bodyuse="literal"/>
</input>
<output>
<soap:bodyuse="literal"/>
</output>
</operation>
<operationname="createRecord">
<soap:operationsoapAction=""/>
<input>
<soap:bodyuse="literal"/>
</input>
<output>
<soap:bodyuse="literal"/>
</output>
</operation>
<operationname="deleteRecord">
<soap:operationsoapAction=""/>
<input>
<soap:bodyuse="literal"/>
</input>
<output>
<soap:bodyuse="literal"/>
</output>
</operation>
<operationname="updatePayer">
<soap:operationsoapAction=""/>
<input>
<soap:bodyuse="literal"/>
</input>
<output>
<soap:bodyuse="literal"/>
</output>
</operation>
<operationname="findPayerInfoByid">
<soap:operationsoapAction=""/>
<input>
<soap:bodyuse="literal"/>
</input>
<output>
<soap:bodyuse="literal"/>
</output>
</operation>
<operationname="findSellerInfoByid">
<soap:operationsoapAction=""/>
<input>
<soap:bodyuse="literal"/>
</input>
<output>
<soap:bodyuse="literal"/>
</output>
</operation>
<operationname="findRecordByCriteria">
<soap:operationsoapAction=""/>
<input>
<soap:bodyuse="literal"/>
</input>
<output>
<soap:bodyuse="literal"/>
</output>
</operation>
<operationname="creatToken">
<soap:operationsoapAction=""/>
<input>
<soap:bodyuse="literal"/>
</input>
<output>
<soap:bodyuse="literal"/>
</output>
</operation>
<operationname="findRecordByKey">
<soap:operationsoapAction=""/>
<input>
<soap:bodyuse="literal"/>
</input>
<output>
<soap:bodyuse="literal"/>
</output>
</operation>
</binding>
<servicename="PayPalDMService">
<portbinding="tns:PayPalDMPortBinding"name="PayPalDMPort">
<soap:addresslocation="http://localhost:8080/WS14/PayPalDMPort"/>
</port>
</service>
</definitions>
```

服务模型生成的 WSDL 文件如下。

```
<?xmlversion="1.0"encoding="UTF-8"?><!-- Generated by JAX-WS RI at
http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.3-hudson-390-.
--><definitionsxmlns="http://schemas.xmlsoap.org/wsdl/"xmlns:soap="ht-
tp://schemas.xmlsoap.org/wsdl/soap/" xmlns: tns = "http://daoImpl/"
xmlns: wsu = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" xmlns: xsd = "http://www.w3.org/2001/
XMLSchema" name = "PayPalDCService" targetNamespace = "http://daoIm-
pl/">
<types>
<xsd:schema>
<xsd:importnamespace="http://daoImpl/"schemaLocation="PayPalDCSer-
vice_schema1.xsd"/>
</xsd:schema>
</types>
<messagename="SetExpressCheckout">
<partelement="tns:SetExpressCheckout"name="parameters"/>
</message>
<messagename="SetExpressCheckoutResponse">
<partelement="tns:SetExpressCheckoutResponse"name="parameters"/>
</message>
<messagename="GetExpressCheckoutDetails">
<partelement="tns:GetExpressCheckoutDetails"name="parameters"/>
</message>
```

```
〈messagename=“GetExpressCheckoutDetailsResponse”〉
〈partelement = “ tns: GetExpressCheckoutDetailsResponse” name = “ parameters”/〉
〈/message〉
〈messagename=“RefundTransaction”〉
〈partelement=“tns:RefundTransaction”name=“parameters”/〉
〈/message〉
〈messagename=“RefundTransactionResponse”〉
〈partelement=“tns:RefundTransactionResponse”name=“parameters”/〉
〈/message〉
〈messagename=“DoExpressCheckout”〉
〈partelement=“tns:DoExpressCheckout”name=“parameters”/〉
〈/message〉
〈messagename=“DoExpressCheckoutResponse”〉
〈partelement=“tns:DoExpressCheckoutResponse”name=“parameters”/〉
〈/message〉
〈portTypename=“PayPalDCDelegate”〉
〈operationname=“SetExpressCheckout”〉
〈inputmessage=“tns:SetExpressCheckout”/〉
〈outputmessage=“tns:SetExpressCheckoutResponse”/〉
〈/operation〉
〈operationname=“GetExpressCheckoutDetails”〉
〈inputmessage=“tns:GetExpressCheckoutDetails”/〉
〈outputmessage=“tns:GetExpressCheckoutDetailsResponse”/〉
〈/operation〉
〈operationname=“RefundTransaction”〉
〈inputmessage=“tns:RefundTransaction”/〉
〈outputmessage=“tns:RefundTransactionResponse”/〉
〈/operation〉
〈operationname=“DoExpressCheckout”〉
〈inputmessage=“tns:DoExpressCheckout”/〉
〈outputmessage=“tns:DoExpressCheckoutResponse”/〉
〈/operation〉
〈/portType〉
〈bindingname=“PayPalDCPortBinding”type=“tns:PayPalDCDelegate”〉
〈soap:bindingstyle = “document” transport = “http://schemas. xmlsoap. org/soap/http”/〉
〈operationname=“SetExpressCheckout”〉
〈soap:operationsoapAction=“”/〉
〈input〉
〈soap:bodyuse=“literal”/〉
〈/input〉
〈output〉
〈soap:bodyuse=“literal”/〉
〈/output〉
〈/operation〉
〈operationname=“GetExpressCheckoutDetails”〉
〈soap:operationsoapAction=“”/〉
〈input〉
〈soap:bodyuse=“literal”/〉
〈/input〉
〈output〉
〈soap:bodyuse=“literal”/〉
〈/output〉
〈/operation〉
〈operationname=“RefundTransaction”〉
〈soap:operationsoapAction=“”/〉
〈input〉
〈soap:bodyuse=“literal”/〉
〈/input〉
〈output〉
〈soap:bodyuse=“literal”/〉
〈/output〉
〈/operation〉
〈operationname=“DoExpressCheckout”〉
〈soap:operationsoapAction=“”/〉
〈input〉
〈soap:bodyuse=“literal”/〉
〈/input〉
〈output〉
〈soap:bodyuse=“literal”/〉
〈/output〉
〈/operation〉
〈/binding〉
〈servicename=“PayPalDCService”〉
〈portbinding=“tns:PayPalDCPortBinding”name=“PayPalDCPort”〉
〈soap:addresslocation=“http://localhost:8080/WS14/PayPalDCPort”/〉
〈/port〉
〈/service〉
〈/definitions〉
```

RESTful API 代码如图 9 和图 10 所示,接口说明如表 1 和表 2 所列。

```
1. @POST
2. @Path(“TransRecord/{paymentStatus}/{transAmount}/{pid}”)
3. @Produces(“application/json”)
4. @Consumes(“application/json”)
5. public String createRecord(@PathParam(“paymentStatus”) String paymentStatus, @ PathParam (“transAmount”) floattransAmount, @ PathParam(“pid”) intpid) {
6.      String token=ppdm. creatToken();
7.       TransRecord record = newTransRecord(token, transAmount, paymentStatus,pid);
8.      booleanb=ppdm. createRecord(record);
9.      if (b==true)
10.           return“succeed”;
11.     else
12.           return“fail”;
13. }
```

图 9　createRecord RESTful API 代码

Fig. 9　Code of createRecord RESTful API

```
1. @GET
2. @Path(“setExpressCheckout”)
3. @Produces(“application/json”)
4. @Consumes(“application/json”)
5. public String setExpressCheckout(@QueryParam(“sPaymentAmount”) floatsPaymentAmount) {
6.      String token=ppdc. setExpressCheckout(sPaymentAmount);
7. returntoken;
8. }
```

图 10　setExpressCheckout RESTful API 代码

Fig. 10　Code of setExpressCheckout RESTful API

表 1　createRecord RESTful API 接口说明

Table 1　Interface description of createrecord RESTful API

接口属性	属性说明
接口名称	创建某条记录的接口
接口描述	通过输入参数在创建新纪录
传值字段名称	token,transAmount,paymentStatus 和 pid
返回值	成功返回 succeed,失败返回 fail

表 2　setExpressCheckout RESTful API 接口说明

Table 2　Interface description of setExpressCheckout RESTful API

接口属性	属性说明
接口名称	启动服务的接口
接口描述	通过交易金额启动服务
传值字段名称	sPaymentAmount
返回值	返回记录的 token 值

6　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三大类:基于本体的方法、基于模型驱动架构

(MDA)或 UML 的方法,以及基于代数规范的方法。

(1)基于本体的方法。大多数服务的描述方法都是基于本体的,如支持 Big Web Service 的 OWL-S 和支持 RESTful Web 服务的 WADL。该方法一般使用本体描述语言,从用户需求中抽取领域知识的本体描述和做为配置文件的服务语义[23-25]。虽然本体描述具备可搜索和机器可读的优点,但基于本体方法一般都缺乏对服务功能进行验证的支持。

(2)基于模型驱动架构或 UML 的方法。基于模型驱动架构的方法通常采用主流的商业过程建模表示法[26-27],并支持从系统高层模型逐步转换为最终实现;而基于 UML 的转换方法,以 OMG 组织设计的统一建模语言 UML 作为国际标准,在 UML 的几类图与使用 XML 描述的 WSDL 和 BPEL 之间进行相互转换[28-29]。由于很多模型可以看成 UML 的扩展,因此可以将它们归为一类。它们一般可以生成 Web 服务的抽象接口描述或代码框架,但难以生成 Web 服务可执行代码。

基于模型驱动架构的方法,典型的如 Gao 等[30]扩展了 IBM 提出的面向服务的建模与分析方法(SOMA),允许将 Web 服务集成到 UML 模型中,由此流图风格表示的 BPMN 模型最终可转换为可实现的 BPEL 服务编排。该方法有利于对基于服务的软件系统进行动态建模,但是存在一个很多研究者都指出的缺点,即模型缺乏形式化语义。

Miao 等[31]提出了一种对基于服务的软件系统进行建模的形式工程方法。该方法通过服务发现、过滤和选择 3 个步骤,将初始的系统需求逐步转换为形式化设计规约。其中初始的系统需求采用条件数据流图的形式,它是数据流图的扩展,与流图风格表示的 BPMN 类似。该方法采用 VDM 方法来建立 Web 服务中每个操作基于前后置断言的形式化语义,但是并未对 Web 服务本身及其组合进行形式化建模。

Mohsen 等[32]提出一种基于 Web 服务建模语言的模型驱动的语义 Web 服务建模方法。该方法通过一种模型驱动体系结构对语义 Web 服务进行建模,并将其从统一建模语言等高级建模语言转换为 Web 服务建模语言等低级语义描述。而本文是在需求分析阶段,采用抽象程度较高的 Radl-WS 服务需求建模语言描述 Web 服务的需求,将形式化方法应用于需求建模,进而提出一种三阶段转换生成 Web 服务可执行代码的方法。

David 等[33]提出一种快速构建一致性 Web 服务的模型驱动方法。文中提到将大部分基于模型驱动的工程方法应用于开发 Web 服务还存在着不能同时适合快速原型设计、模型验证和与通用编程语言兼容的问题。在快速原型设计上本文的 Web 服务建模主要基于 PAR 平台,能够快速地对 Web 服务进行建模;在模型验证上本文采用 Radl-WS 对 Web 服务需求进行建模,该建模语言包含一个公理单元,可以用于后续对模型的验证工作(本文主要工作在于 Web 服务建模方法,公理部分用于验证,没有过多的讨论);在 PAR 平台中,Apla 服务设计语言可以转换为 C++,Java 和 Python 等多种通用的编程语言。以上分析,说明本文的方法可以很好地解决模型驱动运用于 Web 服务开发的缺陷。

本文借鉴了模型驱动架构的思想,但没有采用 UML 或 UML 扩展作为标准建模语言,而是设计自定义的建模语言 Radl-WS。该建模语言融合了 Hoare 公理方法和代数公理方法,具有严格的形式化语法和语义。进而提出了一种三阶段转换生成 Web 服务的方法,第一阶段将 Radl-WS Web 服务建模语言转换为 Apla 服务设计语言,第二阶段将 Apla 服务设计语言通过转换工具转换为可执行代码。第三阶段将可执行代码封装为服务。

(3)基于代数规范的方法。代数规范已经发展到了行为代数[3]和余代数[4-6],可以对并发系统、基于状态的系统和软件组件进行规格说明。但是,国内外仍鲜有工作使用代数规范来对 Web 服务进行建模。

Smith[8]设计了 Specware 生成系统,它使用 MetaSlang 代数规范语言来对软件系统需求进行建模。MetaSlang 中的一个代数规范代表一个理论(Theory),代数规范还可以进行组合和精化,但是 MetaSlang 代数规范一般表示一个算法或软件组件,并不支持对 Web 服务进行建模和转换。

Specware 生成系统可形式化生成可执行代码,它主要是通过应用算法设计模板库来实现的;而本文使用的 Radl-WS 语言支持由操作(算法)生成可执行代码,这主要是通过统一的算法设计方法 PAR 方法及其支撑平台来实现的,既可以通过形式化推导生成,也可以通过形式化验证来生成。Zhu 等[34-35]提出了 SOFIA 代数规范语言以支持对 Web 服务的描述,但该语言并不具备丰富的类型系统,也不支持泛型及其约束机制;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。

本文使用的 Radl-WS 语言,是在我们前期研究中设计的 Radl 语言基础上进行扩展得到的,它具备很多 Radl 语言优点,支持丰富的类型系统、泛型及其约束机制;将操作集简化为构造操作和查看操作两类,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 中的类型进行转换时,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。更重要的是,Radl-WS 语言支持由操作(算法)生成为可执行代码。SOFIA 代数规范语言设计的目的是用来进行 Web 服务测试,主要面向服务使用者;而本文使用的 Radl-WS 语言设计目的是用来进行 Web 服务开发和验证,主要面向服务开发者。

结束语 本文充分利用模型驱动的思想。为支持 Web 服务建模,提出从抽象到具体 4 个模型:Radl-WS 服务需求模型、Apla 服务设计模型、Java 可执行代码、WSDL/RESTful API。为支持模型转换,进一步提出一种三阶段转换生成 Web 服务可执行代码的方法。第一阶段将 Radl-WS 服务需求建模语言转换为 Apla 服务设计语言;第二阶段将 Apla 服务设计语言通过相关转换工具生成可执行代码;第三阶段将可执行代码封装成服务。通过实例,展示了本文提出的方法的实际效果。

工业界一般直接设计网络用户所需的操作(算法)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。相比工业界直接设计操作(算法)的可执行代码,采用三阶段模型转换方法带来的好处主要有两个:1)提高了 Web 服务开发的可靠性,这是由于生成过程中不仅采用了形式化推导和验证技术,而且充分体现了保持语义正确性的数据求精和操作求精;2)提高了 Web 服务开发的生产效率,这是

由于在相关模型转换工具支持下,三阶段模型转换生成中绝大部分工作都可以(半)自动化。

未来有两个值得进一步研究的工作:1)如何将本文提出的方法应用到 Web 服务组合建模与模型转换;2)将本文提出的方法应用到可交易服务上。

参 考 文 献

[1] XIA Q,JIANG C X,YANG C,et al. A Semantic Demand-Service Matching Method based on OWL-S for Cloud Testing Service Platform[C]// 2020 3rd International Conference on Algorithms,Computing and Artificial Intelligence. 2020:1-6.

[2] LING J,JIANG L Y. Semantic description ofinternet of things services:A transformation method from WSDL to OWL-S [J]. Computer Science,2019,46(4):89-94.

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

[4] HENNING K,BARBARA K. Coalgebraic trace semantics for continuous probabilistic transition systems [J]. Logical Methods in Computer Science,2013,9(4):2320-2346.

[5] DIEKERT V,DRISTE M,MUSCHOLL A,et al. Logic,Algebra and Fromal Verification of Concurrent Systems[C]// Schloss Dagstuhl-L eibniz-Zentrum für Informatik. 2021:1-26.

[6] BONCHI F,MONTANARI U. A coalgebraic theory of reactive systems [J]. Electronic Notes in Theoretical Computer Science,2008,209:201-215.

[7] QIU B,YANG Z B,ZHOU Y,et al. AADL multi paradigm modeling and automatic code generation method forIMA [J]. Journal of Chinese Computer Systems,2021,42(10):2223-2233.

[8] SMITH D R. Composition by colimit and formal software development[M]// Algebra,Meaning,and Computation. Berlin:Springer,2006:317-332.

[9] CHEN X,YANG G,CUIi Z Q,et al. Overview of automatic code annotation generation methods [J]. Journal of Software,2021,32(7):2118-2141.

[10] XUE J Y,HUANG J W,YOU Z,et al. Research on virtual reality modeling mechanism based on Apla language [J]. Journal of Huazhong University of science and Technology(Natural Science Edition),2021,49(2):62-67.

[11] XUE J Y,YANG B,ZUO Z K,et al. A linear in-situ algorithm for the power of cyclic permutation[C]// International Workshop on Frontiers in Algorithmics. Berlin:Springer,2008:113-123.

[12] XUE J Y. PAR method and its supporting platform[C]// Procedure of the 1st International Workshop on Asian Working Conference on Verified Software. Macao:UNU-IIST,2006. 10-20.

[13] SHI H H,XUE J Y,et al. PAR-based formal development of algorithms [J]. Chinese Journal of Computers,2009,32(5):982-991.

[14] WANG C J,XUE J Y,et al. Research on relative correctness of Radl formal specification [J]. Journal of Software,2013. 24(4):715-729.

[15] WANG C J. Verifying the correctness of loop optimization based on extended logictransformation system μ TS [J]. Journal of Computer Research & Development,2012,49(9):1863-1873.

[16] SSEBASTIAN G,GALLUD J A,TESORIERO R. Code generation using model driven architecture:A systematic mapping study [J]. Journal of Computer Languages,2020,56:100935.

[17] ZUO Z K,XUE J Y. Research on generic constraints of Apla [J]. Journal of Software,2015,26(6):1340-1355.

[18] SHI H H. Apla-Java automatic program transformation system supporting generic programming[D]. Nanchang:Jiangxi Normal University,2004.

[19] HU Q M,XUE J Y,YOU Z. Research on formal development of non-recursive algorithms of graph search[C]// International Workshop on Structured Object-Oriented Formal Language and Method. Cham:Springer,2015:165-178.

[20] XIE W P,XUN J Y. Research of Radl->Apla program generation system and its reliability[D]. Nanchang:Jiangxi Normal University,2009.

[21] XIE W P,XUN J Y. Research on generation system from Radl algorithm toApla program [J]. Journal of Computer Research and Development,2014,51(4):854-864.

[22] ZUO Z Z,FANG Y,HUANG Q,et al. Non-recursive algorithm derivation and formal proof of binary tree traversal class problems[C]// 2020 IEEE 20th International Conference on Software Quality,Reliability and Security Companion. IEEE,2020:670-671.

[23] LIU D,YANG Y F,CHEN Y,et al. Evaluating the ontological semantic description of web services generated from algebraic specifications[C]// 2016 IEEE Symposium on Service-Oriented System Engineering(SOSE). IEEE,2016:211-220.

[24] PUSHPA C N,DEEPAK G,KUMAR A,et al. OntoDisco:improving web service discovery by hybridization of ontology focused concept clustering and interface semantics[C]// 2020 IEEE International Conference on Electronics,Computing and Communication Technologies(CONECCT). IEEE,2020:1-5.

[25] REN F L,SHEN J K,SUN B B,et al. A review of ontology technology in the field of construction from the text [J]. Journal of Computer,2019,42(3):654-676.

[26] HYOUNG D K. BPMN-based modeling of B2B business processes from the neutral perspective of UMM/BPSS[C]// 2008 IEEE International Conference on e-Business Engineering. IEEE,2008:417-422.

[27] SADOVYKH A,DESFRAY P,ELVESAETER B,et al. Enterprise architecture modeling with soaML using BMM and BPMN-MDA approach in practice[C]// 2010 6th Central and Eastern European Software Engineering Conference (CEE-SECR). IEEE,2010:79-85.

[28] ANISHA V,NARY S. Transforming functional requirements from UML into BPEL to efficiently develop SOA-based systems [C]// OTM Confederated International Conferences. Berlin:Springer,2009:337-349.

[29] KNAPP A,MOSSAKOWSKI T,ROGGENBACH M. An institutional framework for heterogeneous formal development in UML [J]. Springer International Publishing,2014:215-230.

[30] GAOH Y,ZHANG J Y,POVALEJ R,et al. Service-oriented modeling method for the development of an e-Commerce platform[C]// 2009 International Conference on E-Business and Information System Security. IEEE,2009:1-5.

[31] MIAO W K,LIU S Y. A formal engineering framework for

service-based software modeling [J]. IEEE Transactions on Services Computing, 2012, 6(4): 536-550.

[32] MOHSEN M, MOHAMMAD K S, MORTEZAD, et al. A model-driven approach for semantic web service modeling using web service modeling languages [J]. Journal of Software: Evolution and Process. 2021, 33(7): e2364.

[33] DAVID S, JERME R, CHRISTIANH A, et al. A model-driven method for fast building consistent web services in practice [C]// MODELSWARD. 2018: 1-12.

[34] ZHU H, YU B. Algebraic specification of web services[C]// 2010 10th International Conference on Quality Software. IEEE, 2010: 457-464.

[35] LIU D M, ZHU H, BAYLEY I, et al. SOFIA: An algebraic specification language for developing services[C]// 2014 IEEE 8th International Symposium on Service Oriented System Engineering. IEEE, 2014: 70-75.

WANG Chang-jing, born in 1977, Ph.D, Professor, Ph.D supervisor, is a senior member of China Computer Federation. His main research interests include software formal method, trustworthy software and Web services.

ZUO Zheng-kang, born in 1980, Ph.D, professor, is a senior member of China Computer Federation. His main research interests include software formal method and functional programming.