基于 Object-Z 的 Web 组件形式化建模

严吉皞 缪淮扣

(上海大学计算机工程与科学学院 上海 200072) (上海市计算机软件评测重点实验室 上海 201112)

摘 要 Web 组件技术是一种解决 Web 服务再利用和扩展问题的方法。Object-Z 是 Z 语言的面向对象补充,它们是基于一阶谓词逻辑和集合论的形式规格说明语言。用形式规格说明语言 Object-Z 对 Web 组件建模,能够保证 Web 组件在异构平台、松散耦合、封装等特性下的一致性和精确性。以 Web 组件为研究对象,以 Object-Z 为形式规格说明语言建立模型,提出了 Web 组件及其组合的建模方法。该方法对包括接口、组件操作在内的 Web 组件静态行为进行了建模,定义了接口、消息的匹配方法。构造了基本组合结构的形式化框架,利用组件的逻辑分解方法将该框架应用于复杂的组件组合过程,并提出了需求驱动的组件组合方法。在此基础上,结合实例对组件的交互、组合进行了建模分析。

关键词 Web 组件,组件组合,Object-Z,定理证明

中图法分类号 TP311 **文献标识码** A

Formal Model of Web Component Based on Object-Z Specification

YAN Ji-hao MIAO Huai-kou

(School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China)

(Shanghai Key Laboratory of Computer Software Testing and Evaluating, Shanghai 201112, China)

Abstract Web component is a solution to reuse and extend Web service. Object-Z is the object-oriented complement of the language Z which is based on set theorem and first-order predicate logic. Modeling with a formal specification language Object-Z could ensure the consistency and accuracy of the Web component which work in heterogeneous, loose coupling and encapsulation conditions. This thesis concentrates on studying Web component based on Object-Z, and poses the related approach to modeling Web component and its compositions. The approach models static behaviors of component including interface and operation, defines the matching methods between interface and message, builds the formal frame of basic composite structure, applies the frame with component decomposition, and describes the composite method based on the requirements. On this basis, a case study for demonstrating our proposed approach, modeling the interactions and compositions is shown.

Keywords Web component, Component composition, Object-Z, Theorem proving

1 概述

根据 W3C 的定义,Web 服务是由统一资源指示符(URI)标识的软件应用,它可以通过 XML 标准定义、描述和发现其接口及绑定,具有松耦合、粗粒度、开放性和可集成的特点。

在 Web 服务中,存在 3 种角色,即服务提供者、服务中介者和服务请求者。他们之间的关系如图 1 所示。

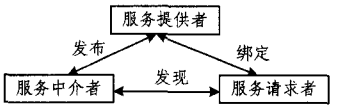


图 1 Web 服务架构

Web 服务常用描述语言是 WSDL[1](Web Services Description Language)和 OWL-S[2](Web Ontology Language for Services)。WSDL 用来描述网络服务或终端的一种 XML 语言,用于定义 Web 服务的调用。OWL-S 是一种用来描述 Web 服务的属性和功能的本体规范。Web 服务组合可以利用 Web 服务资源,从而提供更多、更强的功能。Web 服务组合主要有 3 种方法:基于 BPEL4WS 的工作流方法、基于 AI 规划的方法和基于软件工程的方法[3]。Web 组件组合方法[3]是借鉴软件工程中的复用、泛化、细化和扩展原则,将 Web 服务看成一个 Web 组件,根据对应类的定义将组合逻辑(包括组合类型和消息依赖)封装在类的定义中。Beyer 等人提出了 Web 服务接口理论[4,5]。S. K. Shin 提出了一种使用 Object-Z 从面向对象设计向基于组件设计的转换方法[6]。Alencar 定义了一种基于逻辑的形式化方法用于组件的形式规格的建立[7]。文献[8]提出了一种基于形式规格说明的组件功能匹配方法。文献[9]提出了一种基于反射机制的软件体系结构重用方法。

本文受国家自然科学基金项目(60970007),上海市科学技术委员会(10510704900),上海市重点学科建设项目(J50103)资助。

严吉皞(1987—),男,硕士生,主要研究方向为软件形式方法,E-mail:yanjihao2003@163.com;缪淮扣(1953—),男,教授,博士生导师,主要研究方向为软件形式方法。

国内外对于 Web 服务的形式化研究主要集中在基于 Petri 网的 Web 服务组合模型、基于有限状态自动机的 Web 服务组合模型和基于进程代数的 Web 服务组合模型 3 个方面，本文从 Web 服务组件的角度建立了基于 Object-Z 的形式化模型，为进一步研究 Web 组件交互组合中的性质提供了基础。

Object-Z 编写的需求规格说明能够较容易地发现需求分析中的错误，而且编写者需要精确地使用 Object-Z 的数学符号，这有利于保证规格说明的精确性。此外，Object-Z 编写的规格说明能够便于对系统的性质进行相应的抽取和验证，从而分析系统行为。Web 组件在分布式、松耦合的环境下工作，采用 Object-Z 对其研究有助于发现、验证 Web 组件系统性质，提高其交互的安全性和一致性。

2 Z 语言和 Object-Z 语言简介

Z 语言[10]是基于一阶谓词逻辑和集合论的形式化建模语言，它利用集合、序列、包和函数等数学概念对目标软件系统的结构和行为特征进行抽象描述。

Object-Z[11]是 Z 语言面向对象的扩展，它通过类的概念来封装对象的状态及相关操作。全局定义在所有类的外部，被所有类所共享。类的定义和全局定义的语法与 Z 语言的语法相同。它的结构如图 2 所示。

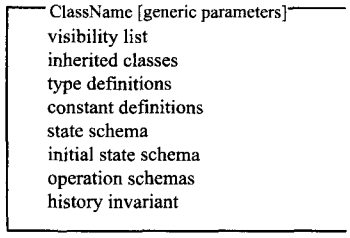


图 2　Object-Z 类的结构

类的定义是一个有名称的框架，也可以带有通用式参数，并可以继承父类。子类与父类的同名变量与操作是合取的关系。visibility list 部分限制了该类对象特性的存取权限，如果该部分缺省，则所有特性外部可见。State schema 是无名称框架，定义了类的属性。Initial state schemas 定义了类的对象允许的初始值。Operation schemas 负责定义对象的操作。

3 Web 组件的形式化建模

将 Web 服务的实体看成是一个 Web 组件。Web 组件中包含了 Web 服务的功能信息，同时也定义了 Web 服务间的交互行为。

Web 组件由消息、接口、功能实体组成。消息是组件之间交互信息的抽象，每个消息都有消息类型。消息是通过接口传递的，组件接口分为输入接口和输出接口，消息通过输入接口传递给组件，通过输出接口传递给其他组件，它们的关系如图 3 所示。接口可以允许多种消息类型的消息通过。

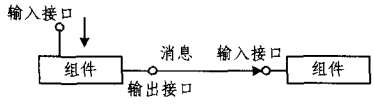


图 3　组件关系

对 Web 组件建模、验证的完整流程如下：

(1)对原子 Web 组件进行建模，包括接口、消息、组件行为等。

(2)对 Web 组件的交互行为建模，分析组件操作的逻辑关系，对特定的需求建立相应的组合模型。

(3)根据建立的模型抽取相应的性质，以保证模型的一致性和安全性。

(4)将推理出的性质带入原模型，补全原模型的非功能性描述。

3.1 接口与消息

接口是组件的重要组成部分，它是组件与外部交互的媒介。对于使用者，组件的内部结构是不可见的，用户只能通过接口来实现组件的相关操作。接口需要能够表述组件的功能，即组件“做什么”，但是又不能涉及到组件的具体实现，所以在接口中需包含表述功能的信息。为了便于描述，本文将该功能信息从接口中剥离出来，单独用组件操作表示。Web 组件通常运行在分布式的系统环境下，不同的组件可能存在于不同的网络地址、不同的平台，故作为组件与外部交互的途径，接口中需要包含地址信息(url)，以便系统的交互寻址。此外，接口还需要确定通过它的消息的类型(messagetypes)。接口中只定义了接口所包含的变量，它的实现在组件的实现中定义，所以接口是无法脱离组件存在的，它是组件的一部分。在形式化建模中，组件操作可以用来描述组件的抽象功能，它和接口可以共同作为组件对外交互的依据。接口的形式化定义如下：

Interface
messagetypes: $\mathbb{P}$ MESSAGETYPE
url: URL
type::= ininterface|outinterface

消息是组件之间通信的媒介，它通过接口传入或传出组件，存在于接口与接口之间，保证了组件操作的独立性。消息有自己的消息类型，并包含消息映射函数 addressmapping，表示该消息的接口地址的映射。其描述如下：

Message
messagetypes: $\mathbb{P}$ MESSAGETYPE
addressmapping: URL$\mathrel{f}$URL

3.2 接口与消息匹配

组件之间的组合交互是通过接口的绑定实现的，是输出接口↔消息↔输入接口三者的关系。例如在图 4 所示的组件 C 中，$Aout_1$ 和 Bin_1 是绑定在一起的，它有两层含义，一是消息要与接口匹配，即消息是能够通过该接口的，二是接口与接口的绑定。接口与消息的匹配可以用公理来描述：

Match: Message × Interface → Boolean
mess: Message
inf: Interface

mess.meesagetypes ⊆ inf.messagetypes
ran mess.addressmapping=inf.url ∨
dom mess.addressmapping=inf.url

接口与接口的绑定描述如下：

Bind[interfaceA,interfaceB]
message: Message

interfaceA.type=ininterface
interfaceB.type=outinterface
Match(message, interfaceA)
Match(message, interfaceB)
interfaceA.url→interfaceB.url∈
messages.addressmapping

接口 A、B 分别为输入接口、输出接口，它们传递的消息要与它们各自匹配。并且消息的地址映射是从接口 A 的地址到接口 B 的地址。这种绑定关系只存在于基于需求的组件交互，并不意味着接口与接口永久绑定。即每一个组件依然可以作为一个逻辑上的功能与其他组件交互；绑定的生命周期始于需求的产生，终于需求的结束；脱离了需求的绑定关系无法满足组件松耦合的特性，也是没有意义的。

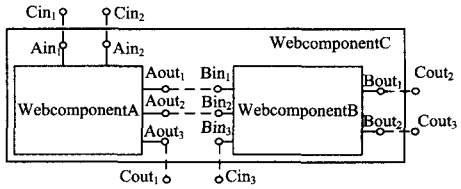
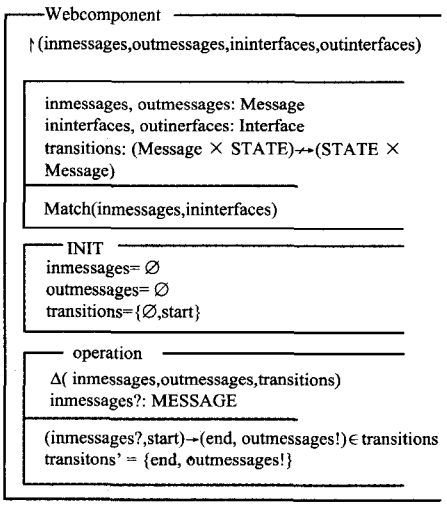


图 4　组件的顺序组合

3.3　Web 组件

原子组件是组件的最小单元，它不是由其他组件组合而成的，多个原子组件能够组合成新的组件。单个组件可以被看作是一个组件类。组件与外部通过接口实现交互，消息通过接口传入或传出组件，接口是具有方向的，一个接口只能接受消息或只能传出消息。原子组件的描述如下：



组件类与外部通过接口 *prointerfaces*、*reqinterfaces* 交互，接口可以接受或传出消息，*inmessages*? 是输入的消息，组件每通过输入接口接受一组消息(可以是单个消息，也可以是多个消息)，组件的状态由此发生了变化，并由输出接口输出一组消息(可以为空消息)。这可以通过状态迁移函数 *transitions* 来实现。*INIT* 描述的是组件的初始状态(*start*)，定义的变量的初始值，此状态下组件没有接受或输出消息。

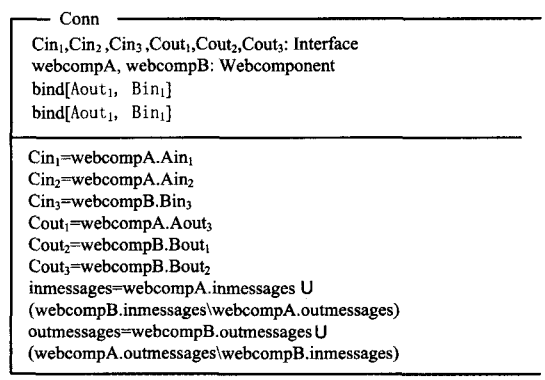
3.4　组件组合

3.4.1　顺序组合结构

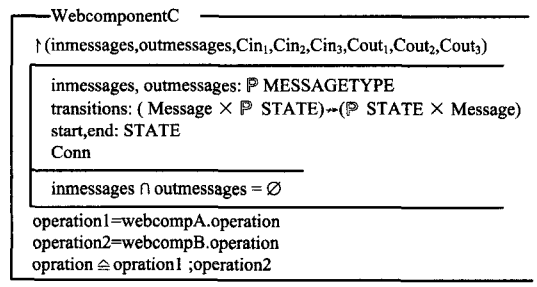
组件组合成新的复合组件，复合组件也可以与其他组件组合，整个系统也可以看作是一个由多个复合组件组合而成的组件。组件的组合最基本的是两个组件的顺序组合，如图 4 所示。

组件 A 和组件 B 组合成组件 C，Ain_1、Ain_2 是组件 A 的输入接口，$Aout_1$、$Aout_2$、$Aout_3$ 是组件 A 的输出接口，Bin_1、Bin_2、Bin_3 是组件 B 的输入接口，$Bout_1$，$Bout_2$ 是组件 B 的输出接口。$Aout_1$、$Aout_2$ 输出的消息由 Bin_1、Bin_2 传递给组件 B，并由 $Bout_1$，$Bout_2$ 输出。那么，对于组件 C 而言，有 3 个输入接口，即 Ain_1、Ain_2、Bin_3，2 个输出接口即 $Bout_1$，$Bout_2$。

图 4 中的组件 A 和组件 B 之间的组合结构表述如下：



组件 C 的形式化表述如下：



组件的组合是多个组件复合生成新的组件的过程。这个过程需要区分哪些是对外接口，哪些是内部接口，哪些是外部消息，哪些是内部消息，对外接口和外部消息对外可见，内部接口和内部消息仅对内部可见，从结构上说就是根据功能需求绑定接口。

在复合组件的形式规格说明中，*Conn* 中表述了组件之间的组合关系，定义了组合后组件的接口和消息结构。复合组件的 *transitions* 保留了所有组件自身的状态变迁，一个复合组件的状态是其下一级组件的状态的集合。在 *transitions* 中，*Message*×*PSTATE* 代表了一个组件接受的消息和接受消息时的状态，*PSTATE* × Message 代表一个组件处理接受消息后的状态以及它所输出的消息。若有组件 X 和组件 Y，它们的状态迁移分别为 X：(输入消息 X，状态 X)→(状态 X′，输出消息 X)，Y：(输入消息 Y，状态 Y)→(状态 Y′，输出消息 Y)，若 X 的输出消息和 Y 输入消息相同，那么它们两者可合并为 XY：(输入消息 X，状态 XY)→(状态 XY′，输出消息 Y)，其中状态 XY 即状态 X 和状态 Y 的合并，状态 XY′即状态 X′和状态 Y′的合取。

3.4.2　其他组合结构

以上是最基本的 2 个组件之间顺序组合的例子，还有以下几种基本的组合接口，更加复杂的组合结构可由这几种基本结构组成，如图 5 所示。

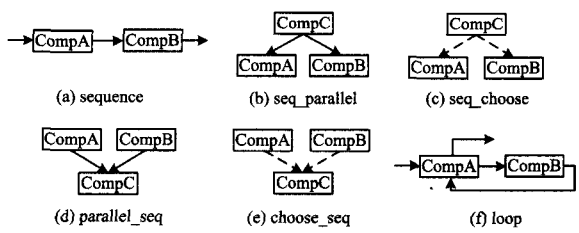


图 5　几种基本组合结构

(a)中组件 A 传出消息传入组件 B。

(b)中组件 C 传出的消息分别传入组件 A 和组件 B,组件 A 和组件 B 并发执行。

(c)中组件 C 传出的消息选择传入组件 A 或组件 B,组件 A 和组件 B 选择执行。

(d)中组件 A 和组件 B 并发执行,两者传出的消息传入组件 C。

(e)中组件 A 或组件 B 被执行,两者传出的消息传入组件 C。

(f)中组件 A 的消息传入组件 B,组件 B 传出的消息重新传回组件 A,并在一定情况下从组件 A 传入其他组件。

例如,在(b)中有 3 个组件,*CompA*、*CompB*、*CompC* 三者组成 *CompD*,消息先通过 *CompC*,再从 *CompC* 传出 2 条消息分别传入 *CompA* 和 *CompB*,此时 *CompA* 和 *CompB* 并发操作,那么根据 Graeme Smith 提出的语义,在 *CompD* 类中加入并行,即 $CompA.operation \parallel CompB.operation$,类似可以定义选择操作 $CompA.operation[]CompB.operation$。

在形如(a)、(c)的并行结构下,并行组件的接口的消息是可以同时存在的,在形如(b)、(d)的选择结构下,选择组件的接口具有互斥性。

在构建复合组件时,可以采用 3.4.1 节中的方法定制复合组件,也可以通过以下给出的这几种结构的基本形式化表述来构建新的组件。

(a)中的顺序结构表述如下:

sequence[compA,compB]

$\upharpoonright$(inmessages,outmessages,ABin,ABout)

inmessages, outmessages: $\mathbb{P}$ MESSAGETYPE
ABin,ABout: INTERFACE
start,end: STATE
bind[compA.Aout,compB.Bin]

ABin=compA.Ain
ABout=compB.Bout
Match(inmessages,ABin)$\wedge$Match(outmessages,ABout)

compAB.operation$\triangleq$compA.operation;compB.operation

(b)和(d)中的组件 A、B 并行结构表述如下:

parallel [compA,compB]

$\upharpoonright$(inmessages,outmessages,$ABin_1$,$ABin_2$,$ABout_1$,$ABout_2$)

inmessages, outmessages: $\mathbb{P}$ MESSAGETYPE
$ABin_1$,$ABin_2$,$ABout_1$,$ABout_2$: INTERFACE
start,end: STATE

$ABin_1$=compA.Ain
$ABin_2$=compB.Bin
$ABout_1$=compA.Aout
$ABout_2$=compB.Bout
Match(inmessages,$ABin_1$) $\vee$ Match(inmessages,$ABin_2$)
Match(outmessages,ABout1) $\vee$ Match(outmessages,$ABout_2$)

compAB.operation$\triangleq$compA.operation$\|$compB.operation

(c)中的选择结构表述如下:

seq_choose [compA,compB]

$\upharpoonright$(inmessages,outmessages,ABin,$ABout_1$, $ABout_2$)

inmessages, outmessages: $\mathbb{P}$ MESSAGETYPE
ABin,$ABout_1$, $ABout_2$: Interface
start,end: STATE

($ABin_1$=compA.Ain) $\vee$ ($ABin_1$=compB.Bin)
$ABout_1$=compA.Aout
$ABout_2$=compB.Bout
Match(inmessages,$ABin_1$)
Match(outmessages,$ABout_1$)
Match(outmessages,$ABout_2$)

compAB.operation$\triangleq$compA.operation[]compB.operation

(e)的结构表述如下:

choose_seq[compA,compB]

$\upharpoonright$(inmessages,outmessages,$ABin_1$,$ABin_2$,ABout)

inmessages, outmessages: $\mathbb{P}$ MESSAGETYPE
$ABin_1$,$ABin_2$,ABout: Interface
start,end: STATE

$ABin_1$=compA.Ain
$ABin_2$=compB.Bin
ABout=compA.Aout $\vee$
compB.Bout
Match(inmessages,$ABin_1$) $\wedge$ Match(inmessages,$ABin_2$)
Match(outmessages,ABout)

compAB.operation$\triangleq$compA.operation[]compB.operation

假设对于模式 *operation*,用$(operation)^n$ 表示 n 个 *operation* 模式的复合,*tag*:*BOOL* 表示循环跳出条件,则(f)的表述如下:

loop[compA,compB]

$\upharpoonright$(inmessages,outmessages,ABin,ABout)

inmessages, outmessages: $\mathbb{P}$ MESSAGETYPE
ABin,ABout: Interface
start,end: STATE
bind[compA.$Aout_1$,compB.Bin_1]
bind[compB.$Bout_1$,compA.Ain_2]

ABin=compA.Ain_1
ABout=compA.$Aout_2$
compA.Ain_1= compA.Ain_2
compA.$Aout_1$= compA.$Aout_2$

operation1=compA.operation
operation2= compB.operation
operation3$\triangleq$ operation2;operation1
$\exists n:Z \cdot$ tag operation $\triangleq$ operation1;$(operation3)^n$

可以用操作 $CompA.operation \circ CompB.operation$ 简化表示(f)中的结构。

在实际应用中,对于单个组件可以将其接口根据以上结构进行复合,例如组件 AB 是组件 A 和组件 B 的复合,对于组件 A 有多个输出接口 $Aout_1$,$Aout_2$,$Aout_3$ 等,但是为了应用以上的基本结构,可将这些输出接口复合成为 $Aout \triangleq Aout_1 \triangleq Aout_2 \wedge Aout_3$。

用 Object-Z 描述的组件都可以继承其他组件,从而新的组件保留了其所继承组件的所有特性,并能够在此基础上描述其特有的谓词约束。

3.4.3 应用组件组合绑定的条件

在对 Web 组件建模的过程中,每一个组件都有其相应的接口、消息类型以及表述其功能的操作,这就使每一个组件都可以作为一个相对独立的功能实体,能够在网络环境中独立存在。以上方法中的每一种绑定方法都是一种逻辑上的组合应用,可将独立地组件组合成一个能解决新问题的复合组件。这种复合组件同样可以独立地在网络中实现相应的功能。

上文中的绑定方法是在组件交互基础上的绑定,组件之间通过消息联系,不需要了解对方的实现,其实质是将组件组合成一个实现某功能的整体,在使用时不需要了解其内部复杂的交互。

在使用基本绑定结构时,绑定的组件两者之间不存在形如图 6 所示的情况,即属于同一个组件的两个操作之间在逻辑顺序上还存在其他组件的操作。

在图 6 中,*opA_1*、*opB* 、*opA_2* 三个操作顺序执行并代表了某种功能上的需求。此时,该功能需求应满足:

(1) $post\_opA\_1 \subseteq pre\_opB$

(2) $post\_opB \subseteq pre\_opA\_2$

这两个式子表示 opB 的执行前提是 opA_1 执行完成，opA_2 的执行前提是 opB 执行完成。

组件操作的前后置条件可以用于描述较复杂的组件交互行为并有助于性质的抽取和验证研究。

若在图 6 的情况下应用组件绑定结构则需将组件 A 分解成两个组件 A_1、A_2，操作分别为 opA_1 和 opA_2，分解后的组件保留了相应的接口定义，此时的复合组件 $compAB$ 为：

$compA\_1\_B \triangleq sequence[compA\_1, compB]$

$compAB \triangleq sequence[compA\_1\_B, compB]$

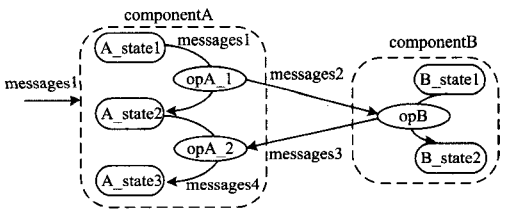


图 6　组件交互图

3.5　组件组合的推导过程

根据图 1 所示 Web 服务构架，服务请求者请求服务，这种需求可以用形式规格说明表示。对组件而言，有的接口处于使用状态，有的处于闲置状态，这取决于消息，即消息的类型要符合接口。可以使用形式化模型的操作谓词推导出组件的组合结构。一个复合组件结构的形成就是不断用不同的组合方式将已有的组件组合，最终使复合组件操作的谓词表达式成立。这个过程涉及了组件类中的操作、接口和消息。操作描述了组件的功能需求，接口和消息为组件的可组合性提供了依据。

3.6　组件组合动态操作

在组件组合的环境中，会出现组件的替换、删除、增加等操作。这些操作的过程是从涉及变化的最小组件所在层次开始的，组件的增加、替换、删除操作改变了组件的组合结构，组件的接口连接随之发生了改变，如图 7 所示。

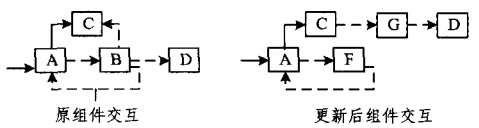


图 7　组件的更新

在例子中，B 组件被组件 G 和组件 F 所替代。这一过程实际上是先删除组件 B，再添加组件 G 和组件 F 到组件组合中。删除 B 以后，B 的关联接口的连接发生了变化，新加入的组件的接口需要与这发生变化的接口关联，在图中用虚线表示。在形式规格说明中，接口的连接发生了变化，迁移函数也相应地发生了变化。组件 B 的状态迁移被组件 F 和 G 的消息所替代，那么原有组件 B 所组合的组件的迁移函数也随之改变。为了保证新的结构的正确性，需要用 3.4.3 节中推导操作前后置条件的方法对新的结构进行正确性的验证。

4　实例研究

某旅游公司为旅客提供旅游服务，它的工作与组织结构如图 8 所示。

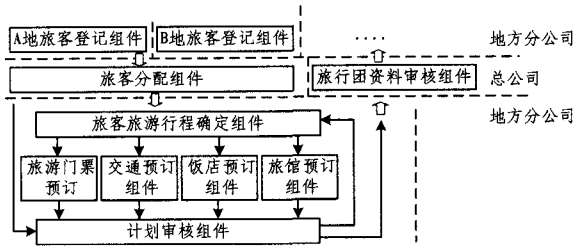
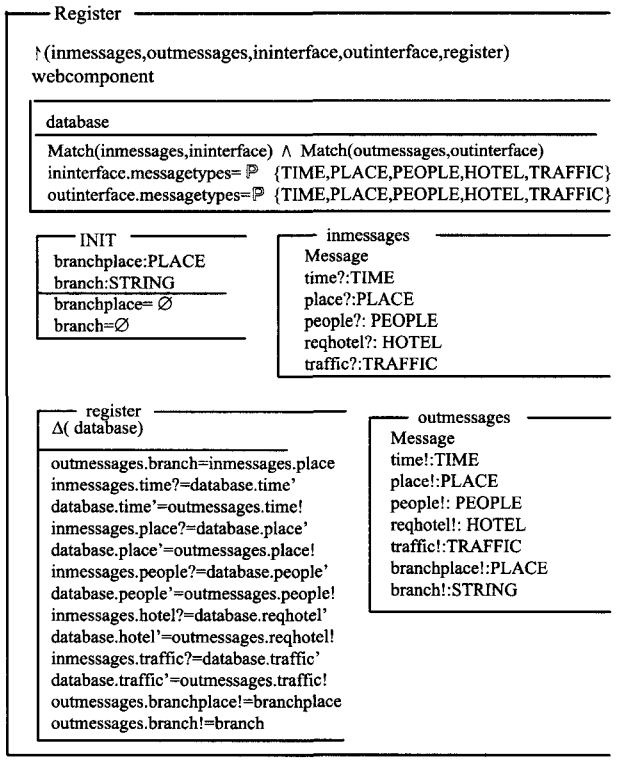


图 8　某旅游公司工作结构示意图

旅客向其所在地的分公司提供旅游申请，该分公司将信息传递给总公司，总公司根据旅客要求将信息分配给旅客将要旅游地区的分公司，被分配任务的分公司协调旅馆、景区门票、交通过程等工作。根据系统要求可以确定给定类型 [*TIME*, *PLACE*, *PEOPLE*, *HOTEL*, *TRAFFICTYPE*, *RESTAURANT*]。

假设 *travelerinformation* 包括了旅客的各种资料及旅游需求信息，而旅客登记组件 Register 的主要功能是对用户的信息进行简单的处理和筛选，加入所在分公司的受理信息 (*branchinfomation*) 后发送给总公司。该组件作为提供登记服务的基本组件，可以被各地分公司的组件所继承，各地分公司的旅客登记服务组件也可以在此基础上根据所在地的实际情况增加其他的功能。



实例中的组件、消息、接口继承了 3.1 节至 3.3 节中定义的基本结构，并根据实际情况增加了约束条件。*Register* 描述中 *database* 定义如下：

travelerinformation
Message
time: TIME
place: PLACE
people: PEOPLE
hotel: HOTELTYPE
traffic: TRAFFICTYPE

branchinformation
branchplace: PLACE
branch: STRING

database
travelerinformation
branchinformation

A地分公司的*RegisterA*组件继承了*Register*并包含了自身的A分公司的相关信息。这种方法重用了已有组件并将其扩展，使得新组件包含了已有组件的特性及新的约束条件。

RegisterA
Register
attribution:BRANCH
attribution=A

*Register*组件的接口和消息是匹配的，根据3.2节的定义，消息的类型包含在接口的类型之中，而且输入消息的*addressmapping*函数的值和输出消息的*addressmapping*函数的自变量的值都与*Register*组件的地址一致。

总公司需要根据从分公司给予的旅客资料，确定旅客的时间、行程、人员等各方面要求，从而将资料派发给负责旅游地相关工作的分公司。对总公司的分配组件来说，它与各地的分公司之间都有接口相连，所以总公司的分配组件中，具有选择接口的功能，从而将旅客资料传送给相应的负责。

对于旅行所在地的分公司而言，它承担着为旅客制定行程计划和预订旅馆、联系景区、组织交通的工作。实现这些功能的是*programe*组件。该组件是由行程确定(*itinerary*)组件、饭店预订(*restaurant*)、旅馆预订(*hotel*)、门票预订(*tickets*)、交通预订(*traffic*)等组件组合而成。其结构如图8所示。*itinerary*组件通过接收总公司发送的在某一时间段内的多位旅客的资料并根据当地景区的实际情况从而确定这些组团旅客的旅行时间。旅行时间确定后，再根据该时间与旅客资料中的旅客要求预订景区、交通、旅馆、饭店等等。其结果通过审核组件确定各项条件是否满足，不满足则返回*itinerary*组件重新确定旅行计划，满足则发回总公司的审核组件(*branchverify*)进行审核。

*branchverify*组件是一个选择分支组件，它实现了旅游计划制定的循环确定过程。

以下是C地分公司相关组件的形式化表示，每个组件都有其*inmessages*和*outmessages*，其中包含了输入输出变量的声明，组件状态(*state schema*)和初始状态(*INIT*)描述了组件的状态变量和其初始化，为了集中研究操作谓词，以下描述中省略初始状态和状态不变式。

itinerary
INIT
state schema
itinerary_operation
itinerary_set'!:TIME×PLACE
tag?:BOOL
time?×place?∈itinerary_set'!
⇒tag?
hotel?=hotel'!
people?=people'!
traffic?=traffic'!

tickets
INIT
state schema
tickets_operation
tickets_op'!:PEOPLE×PLACE⇸TICKETS
dom tickets_op'!=people?×place?

traffic
INIT
state schema
traffic_operation
traffic_op'!: TIME ×PLACE×PLACE⇸TRAFFIC
dom traffic_op'!= itinerary_set? ×place?
ran traffic_op'!⊆reqtraffic?

hotel
INIT
state schema
hotel_operation
hotel_op'!: TIME×PLACE⇸hotel
dom hotel_op'!=time?×place?
ran hotel_op'!⊆reqhotel?

restaurant
INIT
state schema
restaurant_operation
restaurant_op'!: TIME×PLACE⇸restaurant
dom restaurant_op'!=time?×place?

branchverify
INIT
state schema
branchverify_operation
tag:BOOL
tag'! =(∅≠ran tickets_op?∧∉ran traffic_op?∧∅∉∧∅∉ran hotel_op? ran restaurant_op?)

可以采用3.4.2节中的组合范例方法将C地的组件组合成一个制定旅行计划和预约相关服务的复合组件。

parallel[*tickets*,*traffic*]重命名为模式名*tickets_traffic*和*parallel*[*restaurant*,*hotel*]重命名为模式名*restaurant_hotel*。这四者的并行模式为*parallel*[*tickets_traffic*,*restaurant_hotel*]，重命名为模式名*booking*。*booking*与*verify*的关系是顺序，那么其复合模式为*sequence*[*distribution*,*booking*]，重命名为模式名*distribution_booking*。综合已组合的组件，整个复合模式为*loop*[*branchverify*,*itinerary_booking*]。

对于有特殊要求的组件复合，可以使用3.4节中的方法以及*bind*[*interfaceA*,*interfaceB*]等进行定义。

在实例中，C地分公司的复合组件的总功能需求可以表示为：

(0)*operation* ⊢*tag'*!

tag'! 为布尔变量，表示结果是否符合功能需求。

可以将各组件的操作谓词用不同的组合方式组合，从而推导出(0)。

(1)*itinerary_operation* ⊢(*time*? × *place*? ∈*itinerary_set'*!) ∧⇒*tag*?

(2)*tickets_operation* ⊢*dom tickets_op'*! = *people*? × *place*?

(下转第407页)

2008
[2] 高随祥. 图论与网络流理论[M]. 北京:高等教育出版社,2009
[3] 徐俊明. 组合网络理论[M]. 北京:科学出版社,2007
[4] Cayley A. The theory of graphs, graphical representation[J]. Mathematical Paper, Cambridge, 10(1895):26-28
[5] Akers S B, krishnamurthy B. A group-theoretic model for symmetric interconnection networks [J]. IEEE Transactions on Computers, 1989, 38(4):555-565
[6] Lakshmivarahan S, Jwo J-S, Dhall S K. Symmetry in interconnection networks based on Caylay graphs of permutation groups: A survey [J]. Parallel Computing, 1993, 19:361-407
[7] 师海忠. 互连网络的代数环模型[D]. 北京:中国科学院数学与系统科学研究院应用数学研究所,1998
[8] 师海忠,牛攀峰,马继勇,等. 互连网络的向量图模型[J]. 运筹学报,2011,15(3):115-123
[9] Gauyacq G. On quasi-cayley graphs. Discrete Applied Mathematics, 1997(77):43-58
[10] Shi Hai-zhong, Niu Pan-feng. Hamiltonian deconposition of some interconnection networks [A]// Proceed of 3th Aunnal International conference on Combinatorial Optimization and Applications. (Ding-zhu Du etc)[C]. Berlin: Springer, 2009:231-237
[11] 师海忠,路建波. 关于互连网络的几个猜想[J]. 计算机工程与应用,2008,44(31):112-115
[12] Shi Hai-zhong, Niu Pan-feng, Lu Jian-bo. One conjecture of bubble-sort graph[J]. Information Processing Letters, 2011, 111:926-929
[13] Xu Jun-ming, Ma Mei-jie. Survey on path and cycle embedding in some networks[J]. Front. Math. China, 2009, 4(2):217-252
[14] 师海忠,马继勇,牛攀峰. 关于修正冒泡排序网络一簇猜想[J]. 计算机科学,2011,38(10A):265-267
[15] 路建波. Star-网络的 Hamiltonian 圈分解及(l,ω)控制数[D]. 兰州:西北师范大学,2010
[16] 牛攀峰. 冒泡排序网络的一些新结果[D]. 兰州:西北师范大学,2011

(上接第 388 页)

(3) $traffic\_operation \vdash (dom\ traffic\_op'! = (itinerary\_set? \times place?)) \wedge (ran\ traffic\_op'! \subseteq reqtraffic?)$

(4) $hotel\_operation \vdash (dom\ hotel\_op'! = (time? \times place?)) \wedge (ran\ hotel\_op'! \subseteq reqhotel?)$

(5) $restaurant\_operation \vdash dom\ restaurant\_op'! = (time? \times place?)$

(6) $branchverify\_operation \vdash tag'! = ((\emptyset \notin ran\ tickets\_op?) \wedge (\emptyset \notin ran\ traffic\_op?) \wedge (\emptyset \notin ran\ hotel\_op?) \wedge \emptyset \notin ran\ restaurant\_op?))$

在已有的(1)－(6)中,可以根据变量的输入输出得到组件之间的组合关系。为了简便,现假设组件之间都可以进行组合。

例如(1)和(2),(2)的输入变量包含在(1)的输出变量中,所以(1)和(2)可以是顺序组合的关系,即(1);(2)。同理可以推出(1);(3),(1);(4),(1);(5),简写为(1);((2) ‖ (3) ‖ (4) ‖ (5))≙(7)。

由于(2)、(3)、(4)、(5)的输出包含在(6)的输入中,因此可以推出它们与(6)可以顺序连接。(6)的输出是 *tag*!,它的值可能为真也可能为假。若为真,即得到(0)的结论,也就得到了需要的结果;若为假,那么为(1)的输入。由此可得到(1)与(6)之间的选择关系,由此得到(7)与(6)的循环关系。使用 3.4 节说明的简化写法即 $operation^{\circ}(6)^{\circ}(7)$。

如果考虑到组件之间的可组合性,那么每一步推导需要加入组件之间的可组合性的检查。可组合性的检查使用的是 3.2 节中的 *Match* 方法,通过输出接口↔消息↔输入接口关系的匹配,确定是否可组合。

组件系统中的组件的发生更新时,相对应的操作谓词发生了改变,那么根据以上的推导方法可以相应地改变组件的组合结构。

该方法建立的组件系统是一个增量式的组件库,每一个原子组件和根据某种需求复合的新组件都在这个组件库中供新的需求提取,即新的需求可以重用组件库中已有组件并对其扩展。

结束语 本文使用 Object-Z 语言对在 Internet 环境下的 Web 组件进行了形式化建模,对 Web 原子组件及组件的组合等行为进行了形式化的表述,并以此作为进一步使用定理证明来验证 Web 组件的组合、交互行为的基础。进一步的工作是研究组件的组合在功能上的一致性、相关自动化工具的开发以及采用定理证明的方法对 Web 组件组合的验证。

参 考 文 献

[1] Christensen F, Curbera F, Meredith G. Web Service Definition Language[EB/OL]. http://www.w3.org/TR/wsdl
[2] Martin D, burstein M, hobbs J, et al. OWL-S: Semantic Markup for Web Services[EB/OL]. http://www.w3.org/Submission/OWL-S/
[3] Yang Jian, Papazoglou M P. Web component: a substrate for Web servise reuse and composition[J]. Lecture Notes in Computer Science, 2002, 2348:21-36
[4] Beyer D, Chakrabarti A, Henzinger T A. Web service interfaces [C]// Proceedings of the 14th international conference on World Wide Web(WWW '05). 2005:148-159
[5] Beyer D, Chakrabarti A, Henzinger T A. An Interface Formalism for Web Services[C]// Proceedings of the First International Workshop on Foundations of Interface Technologies(FIT 2005, San Francisco, CA, August 21). 2005
[6] Shin S K, Kim S D. A method to transform objectoriented design into component-based design using Object-Z[C]// Proceedings of the International Conference on Software Engineering Research, Management and Applications(SERA). 2005:274-281
[7] Alencar P S C, Cowan D D, Lucena C J P. A Logical Theory of Interfaces and Objects[J]. IEEE Transaction on Software Engineering, 2002, 28(6):548-575
[8] Zaremski A M, Wing J M. Specification matching for software components[J]. ACM Trans Soft Eng Meth, 1997, 6(4):333-369
[9] 罗巨波,应时. 基于 Object-Z 的 ReflectiveArchitecture 形式化研究[J]. 计算机科学,2010(11):126-130
[10] 缪淮扣,李刚,朱关铭. 软件工程语言-Z[M]. 上海:上海科技文献出版社,1999
[11] Smith G. The Object-Z Specification Language [M]. Kluwer Academic Publishers, 2000