ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.5, May 2006, pp.1182−1194
DOI: 10.1360/jos171182

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

WSC/ADL:Web Services 组合系统体系结构描述语言*

杨 鑫+, 陈俊亮

(网络与交换技术国家重点实验室(北京邮电大学),北京 100876)

WSC/ADL: An Architecture Description Language for Web Services Composition System

YANG Xin+, CHEN Jun-Liang

(State Key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications), Beijing 100876, China)

+ Corresponding author: Phn: +86-10-58552135, Fax: +86-10-58502170, E-mail: aq_yangxin@263.net

Yang X, Chen JL. WSC/ADL: An architecture description language for Web services composition system. *Journal of Software*, 2006,17(5):1182−1194. http://www.jos.org.cn/1000-9825/17/1182.htm

Abstract: Web services composition is the hotspot research in the field of Web services. Many methods are proposed, however, composition starting from architecture is a new perspective. BPEL4WS is a de facto standard description language for Web services composition. An architecture style of Web services composition system based on BPEL4WS is presented. Then, an architecture description language, WSC/ADL (Web services composition/architecture description language), is designed according to the architecture style. WSC/ADL consists of service components describing Web services, connectors describing the interaction between Web services and configures setting up the relations between instances of service components and connectors, providing the support for top-down development of Web services composition based on architecture. WSC/ADL is discussed in detail with an example and is compared with related work.

Key words: Web services composition; software architecture; architecture description language (ADL); BPEL4WS; CSP (communicating sequential processes)

摘 要: Web services 组合是 Web services 领域的研究热点,虽然已经提出了很多组合的方法,但从体系结构方面去研究 Web services 组合,则是一个新的研究角度.BPEL4WS 是当前工业界主流的 Web services 组合描述语言.给出了基于 BPEL4WS 的 Web services 组合系统体系结构风格,并针对这种风格设计了体系结构描述语言 WSC/ADL(Web services composition/architecture description language),WSC/ADL 是基于体系结构的、自顶向下的 Web services 组合开发的研究基础,其组成包含描述 Web services 的服务构件、描述 Web services 之间交互的连接件以及建立服务构件和连接件实例联系的配置.给出了 WSC/ADL 的详细分析介绍和实例说明,并与相关工作进行了比较.

关键词: Web 服务组合;软件体系结构;体系结构描述语言;BPEL4WS;通信顺序进程
中图法分类号: TP393 文献标识码: A

* 本文为 2005 年中国计算机大会推荐优秀论文.Supported by the National Natural Science Foundation of China under Grant No.60432010 (国家自然科学基金)

Received 2005-06-15; Accepted 2006-01-18

当前,工业界和学术界从不同角度对 Web services 组合进行了大量研究,提出了多种 Web services 组合方法[1,2].总的来说,工业界的方法侧重于提出组合描述语言,开发相关编辑工具和执行引擎;学术界的方法侧重于从语义、人工智能规划等方面研究自动组合,并通过形式化方法验证组合系统的性质.但是,这些方法大多是站在服务或者构件层面上.实际上,Web services 组合也是一种软件开发,还需站在系统级的高层上去研究它.目前,这方面的研究刚刚起步[3,4].而软件体系结构是对系统的全局组织、控制结构、全局成员之间的高层交互等问题的较高抽象层次描述,为系统性质分析、系统开发提供帮助,我们可以将软件体系结构研究方法引入到 Web services 组合中来.

体系结构的研究基础是体系结构描述语言(architecture description language,简称 ADL).ADL 形式化地描述体系结构,可以精细化为系统实现,并为系统性质的分析验证等提供支持.本文提出一种基于 BPEL4WS (business process execution language for Web services,简称 BPEL)[5]的 Web services 组合系统体系结构描述语言 WSC/ADL(Web services composition/architecture description language).WSC/ADL 给基于体系结构、自顶向下的 Web services 组合开发提供基础.

本文第 1 节分析介绍 Web services 组合主流描述语言 BPEL,给出基于 BPEL 的 Web services 组合系统体系结构风格.第 2 节详细介绍我们提出的 WSC/ADL.第 3 节给出一个 WSC/ADL 实例说明.第 4 节对 WSC/ADL 与相关工作进行比较.最后是全文总结和展望.

1 BPEL 与基于 BPEL 的体系结构风格

目前,存在多种 Web services 组合描述语言.在工业界成为事实标准的是 IBM、微软等公司提出的 BPEL.BPEL 是一种基于工作流、可执行的组合描述语言,通过一个流程将多个 Web services 组合起来,流程的每一步称作活动(activity).BPEL 定义了原子活动和结构化活动控制流程,定义了伙伴(partner)和伙伴链接(partnerLink),用于将多个 Web services 纳入流程,BPEL 流程是组合系统集中控制点.另外,BPEL 还支持异步消息处理、异常处理、事务处理和生命周期管理.并且,BPEL 流程本身也作为一个 Web services 发布.如图 1 所示,BPEL 流程组合 4 个 Web services A,B,C,D.

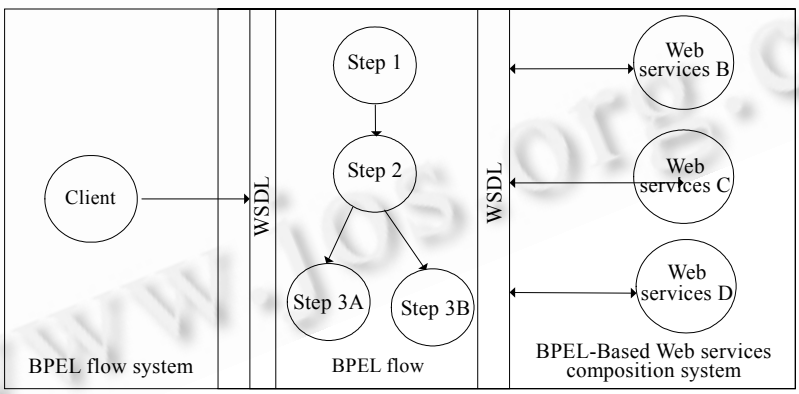

Fig.1 BPEL Flow and BPEL-Based Web services composition system

图 1 BPEL 流程与基于 BPEL 的 Web services 组合系统

在本文中,我们定义 BPEL 流程系统为包含 BPEL 流程以及参与流程的 Web services 在内的系统.对于 BPEL 流程系统来说,每一个 Web services 均被认为是流程中对等的某一方,并不区分客户端、服务端.如果认为某一方是客户端(例如图 1 中的 A),那么,不包括客户端的 BPEL 流程系统的其余部分,我们在本文中定义为基于 BPEL 的 Web services 组合系统.基于 BPEL 的 Web services 组合系统包括 BPEL 流程和为客户端服务的其他 Web services.对于同一个 BPEL 流程,视角不同(视不同的 Web services 为客户端),基于 BPEL 的 Web services 组合系统包含的组合成员也不同.

体系结构风格是一系列具有类似组织结构的体系结构抽象,在本质上反映了一些特定的元素按照特定的方式组成一个特定的结构.Web services 本质上是一种采用 XML 描述数据的消息中间件,基于 BPEL 的 Web services 组合系统的体系结构风格如图 2 所示.各个 Web services 挂接在 BPEL 流程上,BPEL 流程根据客户端输入消息,启动流程执行,流程负责参与组合的各个 Web services 之间消息的转发和处理,控制各个 Web services 顺序、并发、选择等方式的执行,如果参与组合的某一个 Web services 被定义为客户端的话,那么流程中与该 Web services 交互的接口称为 BPEL 流程客户接口(简称客户接口),客户接口是基于 BPEL 的 Web services 组合系统与客户端对应的外部接口.组合系统中的 Web services 可以分为两种:不再包含其他 Web services 的原子 Web services;基于 BPEL 的包含多个 Web services 的复合 Web services,这种复合根据需要可以扩展到多个层次.图 2 的体系结构风格与张世琨等人提出的层次消息总线风格[6]的主要区别在于,图 2 中的 BPEL 流程强调对挂接在上面的 Web services 执行有控制功能,包括顺序、循环、并行等控制执行方式,而层次消息风格中的消息总线强调的是消息的登记、分派、过滤.类似地,我们可以定义基于 BPEL 的 Web services 组合系统体系结构风格是一种层次控制总线风格.BPEL 流程以 Web services 形式发布,所以基于 BPEL 的 Web services 组合系统也以 Web services 形式发布.

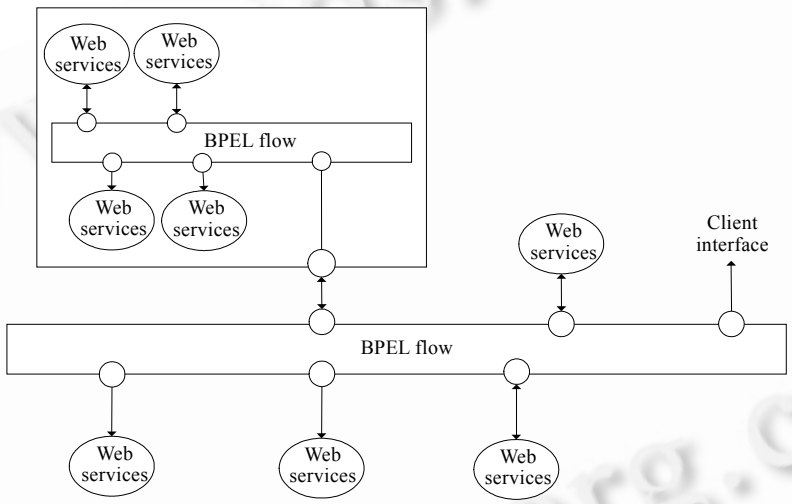

Fig.2 Architecture style of BPEL-Based Web services composition system

图 2 基于 BPEL 的 Web services 组合系统体系结构风格

2 WSC/ADL

针对上述基于 BPEL 的 Web services 组合系统体系结构风格,我们设计了 WSC/ADL 对组合系统进行高层规范化的描述.WSC/ADL 设计时,遵循以下几个基本原则:

- WSC/ADL 统一描述 Web services 组合系统和组成系统的 Web services;
- WSC/ADL 易于转化或者细化为 BPEL 流程代码;
- WSC/ADL 显式地描述 Web services 之间的交互.

WSC/ADL 分为 3 部分:服务构件定义、连接件定义、配置定义.服务构件是用于描述组合系统中的某个 Web services 的构造实体,区分为复合构件与原子构件:复合构件描述基于 BPEL 的组合多个 Web services 的复合 Web services;原子构件描述直接可用而不关注其内部构成的原子 Web services.整个 Web services 组合系统也是一个复合构件.连接件是用于建立服务构件组合交互规则的构造实体,描述 BPEL 流程如何控制 Web services 之间的组合交互.配置定义系统中的服务构件和连接件实例,并且建立这些实例之间的联系,对应着组合系统 BPEL 流程部署时与具体参与交互的 Web services 绑定.

下面采用 BNF 范式来描述 WSC/ADL,其中:"//"标记注释;"⟨…⟩"表示非终结符,"[…]"表示出现 0 次或者多次,"[…]+"表示出现 1 次或者多次.

2.1 服务构件

服务构件:每个服务构件至少包含一个端口、一个行为描述以及数据类型定义,端口是服务构件与外界的一个相对独立的交互点.如果服务构件是复合构件,那么服务构件还将包括内部结构,可选的 *as System* 表示该服务构件是描述整个组合系统的复合 Web services.服务构件的定义如下所示:

⟨ServiceComponent⟩::=ServiceComponent⟨ServiceComponent_Name⟩[as System]
[⟨DataType_Def⟩]+
[⟨Port⟩]+
[⟨ServiceComponent_Behavior⟩]+
[⟨ServiceComponent_Structure⟩]
End ServiceComponent

数据类型:Web services 消息交互的数据类型定义非常丰富、灵活,一般采用 XML Schema 在 WSDL 中定义,所以,WSC/ADL 直接引入 XML Schema(以及相应的 XML 命名空间)进行数据类型定义.一方面,这样与 Web services 数据类型定义能力等价;另一方面,也不用去另创一套数据类型定义形式.数据类型定义如下所示:

⟨DataType_Def⟩::=DataType
[⟨DataType⟩]+
End DataType
⟨DataType⟩::=⟨XMLNameSpace⟩|⟨XML_Schema_DataType⟩
⟨XMLNameSpace⟩::=xmlns:⟨XMLNameSpace_Name⟩=⟨URLString⟩ //⟨URLString⟩为命名空间对应的 URL 格式字符串
⟨XMLSchema_DataType⟩::=⟨simpleType⟩|⟨complexType⟩ //包含简单类型和复杂类型
⟨simpleType⟩::=⟨XMLNameSpace_Name⟩:simpleType name=⟨XMLSchema_DataType_Name⟩
… //参考 XML Schema 简单数据类型定义
End simpleType
⟨complexType⟩::=⟨XMLNameSpace_Name⟩:complexType name=⟨XMLSchema_DataType_Name⟩
… //参考 XML Schema 复杂数据类型定义
End complexType

端口:每个端口至少包含一条消息和一个操作以及端口的行为描述.端口以及端口中的消息和操作与 WSDL 的端口类型以及消息和操作可以很容易地互相映射.端口定义如下:

⟨Port⟩::=Port ⟨Port_Name⟩
[⟨Message⟩]+
[⟨Operation⟩]+
⟨Port_Behavior⟩
End Port

消息:每条消息可以包含若干参数,参数类型在上面的数据类型中已定义.消息定义如下:

⟨Message⟩::=Message ⟨Message_Name⟩
[⟨Contained_Parameter_Def⟩] //消息包含的参数定义,可能不包含任何参数
End Message
⟨Contained_Parameter_Def⟩::=⟨DataType_Name⟩:⟨Parameter_Name⟩[,⟨Parameter_Name⟩]
//引用前面定义的数据类型,包含 XML 命名空间和 XML Schema 自定义的数据类型名称,
//并且为防止混淆,这两者的连接符号是"."而不是":"

〈DataType_Name〉::=〈XMLNameSpace_Name〉.〈XMLSchema_DataType_Name〉

操作:操作区分为输入、输出、输入-输出、输出-输入 4 种类型.服务构件的操作不仅包含了服务构件本身提供的操作,还包括其自身需要的由其他服务构件提供的操作,通过标记标明(前者为 Provided,后者为 Required).这样,服务构件就不仅仅描述了其代表的 Web services,还描述了该 Web services 对外界的服务要求.这种做法类似于文献[7]中提到的接口连接式体系结构中的构件接口定义.操作定义如下:

〈Operation〉::=Operation〈Operation_Name〉〈Tag〉
Type: 〈Operation_Type〉
Message: 〈Contained_Message〉//操作包含的消息
End Operation

〈Tag〉::=Provided|Required

〈Operation_Type〉::=Input|Output|Input-Output|Output-Input

〈Contained_Message〉::=〈Message_Name〉|〈Message_Name〉, 〈Message_Name〉

端口行为:端口的行为通过服务构件端口与外界的消息交换协议而定义.通信顺序进程(communicating sequential processes,简称 CSP)[8]是一种适合描述并发和通信系统的常用形式化描述技术.CSP 的基本单元是进程,进程描述参与通信事件的实体.每个进程可以分解为许多子进程,子进程又可以进一步分解为更深一层的子进程.CSP 本身描述进程间同步通信,但可以引入消息缓冲进程实现进程间的异步通信,从而也可以用于描述异步通信,而 Web services 本质上是一种基于消息的中间件.在本文中,我们选 CSP 作为 WSC/ADL 中的形式化描述,涉及到的 CSP 符号包括:"aP"表示进程 P 的事件集;"$a\rightarrow P$"表示事件 a 然后进程 P;"$c!v$"表示通道 c 输出数据 v;"$c?v$"表示通道 c 输入数据 v;"$P;Q$"表示顺序执行进程 P 和 Q;"$P\Pi Q$"表示内部选择进程 P 或者 Q;"$P\Box Q$"表示外部选择 P 或者 Q.端口行为的 CSP 描述包括进程描述和事件集合两部分定义;至少包含一个与端口同名的进程 P 和 P 的事件集合(aP),并且存在一个双射函数 $f:aP\rightarrow Set(Message)$,$Set(Message)$为端口的消息集合.对于每个事件,当所对应的为输入消息时,那么表示为 $c?v$;所对应的为输出消息时,那么表示为 $c!v$.其中:c 为消息的名称;v 为消息的内容取值.进程 P 可以进一步表示为其子进程的表达式,表达式采用 CSP 符号.如图 3 所示,一个旅行代理服务构件提供订票服务,旅行代理客户发送订票请求.如果客户要求的票存在,那么发送确认消息给客户;如果没有符合要求的票,那么发送拒绝消息.旅行代理服务构件行为描述的 CSP 进程描述为

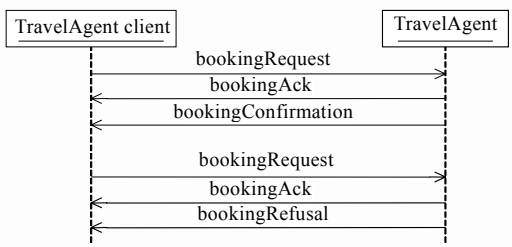

Fig.3 Behavior of service component TravelAgent
图 3 旅行代理服务构件行为

$$TravelAgent=bookingRequest?(\ldots)\rightarrow bookingAck!(\ldots)\rightarrow(bookingConfirmation!(\ldots)\rightarrow TravelAgent \Pi bookingRefusal!(\ldots)\rightarrow TravelAgent)$$

其中:*bookingRequest*,*bookingAck*,*bookingConfirmation*,*bookingRefusal* 为旅行代理服务构件的消息;(…)表示对应消息的取值.综上所述,端口行为定义如下所示:

〈Port_Behavior〉::=ServiceComponent_Behavior
〈CSP_EventSet〉 //事件集合
〈CSP_ProcessDesc〉 //进程描述
End Port_Behavior

〈CSP_EventSet〉::=set 〈EventSet_Name〉={〈Contained_Event〉}

〈Contained_Event〉::=〈event〉[,〈event〉]

〈event〉::=〈Message_Name〉?〈Message_Value〉|〈Message_Name〉!〈Message_Value〉

〈Message_Value〉::=(…)//对应消息的各参数的取值,或者为空表示取值未定

⟨CSP_ProcessDesc⟩::=[⟨CSP_ProcessDef⟩]+

⟨CSP_ProcessDef⟩::=proc ⟨Process_Name⟩=⟨Process⟩

⟨Process⟩::=⟨Process_Name⟩|(⟨Process⟩)|⟨event⟩→⟨Process⟩|⟨Process⟩;⟨Process⟩|

⟨Process⟩Π⟨Process⟩|⟨Process⟩□⟨Process⟩|...

服务构件行为:服务构件的行为同样采用 CSP 描述,并且 CSP 描述只包含一个与服务构件同名进程的定义.如果服务构件只有一个端口,那么服务构件的行为即是端口行为,服务构件进程等同于端口进程;如果服务构件包含多个端口,那么服务构件的行为是各个端口的行为的并行组合,服务构件进程通过并行算子组合各个端口进程.服务构件行为定义如下:

⟨ServiceComponent_Behavior⟩::=ServiceComponent_Beavior

⟨ServiceComponent_CSP_ProcessDef⟩

End ServiceComponent_Behavior

⟨ServiceComponent_CSP_ProcessDef⟩::=proc ⟨Process_Name⟩=⟨Process⟩

⟨Process⟩::=⟨Process_Name⟩|(⟨Process⟩)|⟨Process⟩||⟨Process⟩

内部结构:复合构件与原子构件的外部接口定义相同,不同在于多了内部结构,内部结构由一个连接件和若干服务构件构成,可以用一个体系结构配置描述.内部结构定义如下:

⟨ServiceComponent_Structure⟩::=ServiceComponent_Structure

Configure: ⟨Configure⟩ //指向一个描述内部结构的体系结构配置

End ServiceComponent_Structure

2.2 连接件

连接件:包括数据类型、消息、操作、角色、粘合剂,其定义如下:

⟨Connector ⟩::=Connector ⟨Connector_Name⟩

[⟨DataType_Def⟩]+

[⟨Message⟩]+

[⟨Operation⟩]+

[⟨Role⟩]+

⟨Glue⟩

End Connector

连接件中的数据类型、消息和操作分别包括挂接在连接件上所有服务构件使用的数据类型、参与交互的消息和参与交互时提供的和要求的操作,它们的定义形式与服务构件中的数据类型、消息、操作定义相同.

角色:角色规定了挂接在连接件上的交互方的单个接口的消息、操作和行为,其中角色的消息是引用已经在上面定义的连接件消息序列中的消息,操作是引用已经在上面定义的操作,行为的定义也是采用 CSP 描述,可选的 *as Client* 表示该角色与客户接口对应,这样的角色被称为客户角色.角色定义如下所示:

⟨Role⟩::=Role⟨Role_Name⟩[as Client]

Message: ⟨Contained_Message_Name⟩[,⟨Contained_Message_Name⟩]

Provided_Operation: ⟨Contained_Operation_Name⟩[,⟨Contained_Operation_Name⟩]

Required_Operation: ⟨Contained_Operation_Name⟩[,⟨Contained_Operation_Name⟩]

⟨Role_Behavior⟩

End Role

⟨Role_Behavior⟩::=Role_Behavior

⟨CSP_EventSet⟩ //事件集合,定义形式与服务构件端口行为中的相同

⟨CSP_ProcessDesc⟩ //进程描述,定义形式与服务构件端口行为中的相同

End Role_Behavior

粘合剂:粘合剂用于说明各个角色是如何组合在一起协作工作的.同样采用 CSP 描述,定义形式与上面角色行为的定义形式相同,其中进程描述至少包含一个描述整个协作的进程 *Glue*(被称为粘合剂进程,并可以进一步表示为其子进程的表达式).

组合系统内部服务构件之间,或者组合系统客户端与其内部服务构件之间的交互,一般建立在它们接口操作的功能可以匹配的情况下,但这时仍然可能存在多种类型的不匹配,消除不匹配是连接件的重要任务之一.接口操作功能匹配情况下可能存在的交互不匹配包括以下几种类型:消息类型、消息结构、消息交换协议、服务质量.消息类型不匹配是指一个服务构件输出消息的全部或者部分作为另一个服务构件的输入消息时,两条消息中至少存在一个参数的数据类型不同,并且前者的类型不是后者兼容类型;消息结构不匹配是指一个服务构件输出消息的全部或者部分作为另一个服务构件的输入消息时,两条消息的参数顺序、参数个数等消息格式上的不一致;消息交换协议不匹配是指两个服务构件的交互行为不能达成一致,例如前文所述的旅行代理,假设它是包括航运、航空等多个服务的组合系统,它的外部行为由前面的 CSP 表达式所描述,旅行代理的客户端行为与其外部接口行为兼容(CSP 表达式类似,只是消息的输入输出方向正好相反).如果旅行代理客户端定购船票,那么,旅行代理将调用航运代理的服务,假设航运代理行为的 CSP 描述为 *ShipAgent*=*shipbookingRequest*?(…)→*shipbookingReply*!(…)→*ShipAgent*,旅行代理客户端在发出订票请求后,将首先期待接收一个应答消息,然后再接收多个可能的订票结果消息.但是,航运代理接收到订票请求后并不发送应答消息,而是直接返回包含订票结果的消息.此时,旅行代理客户端与航运代理的行为是不一致的;服务质量不匹配是指请求服务的服务构件与提供服务的服务构件交互时,对服务的响应时间、可靠性等服务质量特性要求不一致,后者不能提供或者没有考虑要求的服务质量.

具体到基于 BPEL 的 Web services 组合,BPEL 提供原子活动 *assign* 灵活转换不同 Web services 之间消息的参数顺序、参数个数,但要求消息类型必须兼容.也就是说,BPEL 建立在消息类型匹配的基础之上,提供了消除消息结构不匹配的操作.BPEL 流程是 Web services 组合系统的集中控制点,协调组合中的消息交互,例如图 3 所示的例子,旅行代理 BPEL 流程将消除旅行代理客户端与旅行代理组合系统成员服务之间不匹配的消息交换协议,如图 4 所示.BPEL 目前不考虑服务质量,默认服务提供者可以满足请求者的服务质量要求.综上所述,我们只需考虑消息结构不匹配和消息交换协议不匹配的情况.由于消除消息结构不匹配的描述比较底层,体系结构高层描述时可以省略;而消息交换协议是组合交互的主体,是体系结构描述的重点,所以在 WSC/ADL 连接件定义中,连接件粘合剂主要用于消除消息交换协议不匹配的描述.并且,粘合剂进程描述中如果可通过包含 CSP 赋值算子来扩展进程定义形式,从而也可在粘合剂中提供消除消息结构不匹配的描述.

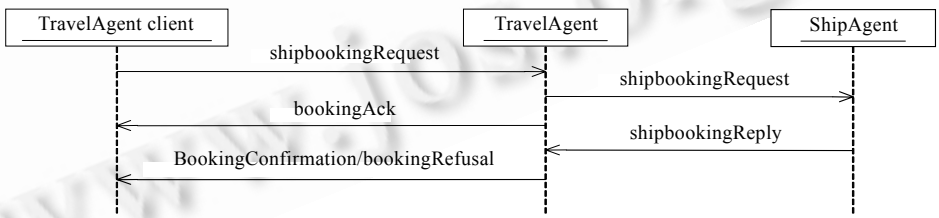

Fig.4 Message interaction coordinated by TravelAgent BPEL flow

图 4 旅行代理 BPEL 流程协调消息交互

另外需要指出的是,Web services 实际交互时还需考虑底层绑定协议的匹配,因为 Web services 可以使用 Http,SMTP,JMS 等多种访问协议,只有当绑定同一个访问协议时,它们之间才能交互.但是,Web services 描述可以区分为抽象定义和具体绑定两个层次,具体绑定是在实际部署和运行时,并且可以采用 WSIF(Web services invocation framework)[9]实现具体访问协议的灵活选择,具备了底层绑定协议的灵活匹配能力;而 WSC/ADL 中的服务构件是建立在抽象层次水平上,可以不用考虑底层绑定协议的不匹配,所以服务构件定义中也不包括底层协议绑定相关部分.

2.3 配　置

体系结构配置包括服务构件实例定义、连接件实例定义、服务构件实例的端口与连接件实例的角色(不包含客户角色,客户角色对应的是整个组合系统的客户端端口)之间的绑定.绑定并不一定要求服务构件端口的接口和接口行为与连接件角色的接口和接口行为相同,它们可以分别看作是系统的实现和规范,从而利用 CSP 进程行为关系验证它们是否兼容、是否可以绑定.配置定义如下:

⟨Configure⟩::=Configure⟨Configure_Name⟩
[⟨ServiceComponent_Instance_Def⟩]+
[⟨Connector_Instance_Def⟩]+
[⟨ServiceComponent_Role_Binding⟩]+
End Configure

⟨ServiceComponent_Instance_Def⟩::=⟨ServiceComponent_Name⟩:⟨ServiceComponent_Instance_Name⟩

⟨Connector_Instance⟩::=⟨Connector_Name⟩:⟨Connector_Instance_Name⟩

⟨ServiceComponent_Role_Binding_Def⟩::=⟨Service_Component_Instance_Name⟩ as
⟨Connector_Instance_Name⟩.⟨Role_Name⟩

3 WSC/ADL 实例

本节给出一个基于 BPEL 的 Web services 组合系统的 WSC/ADL 实例描述.如图 5 所示,旅行代理的 BPEL 流程组合了铁路、航运、航空、金融、信用代理服务,其中航空代理服务是一个复合 Web services,同样采用 BPEL 组合了多家航空公司服务.除了航空代理服务,其他均是原子 Web services.旅行代理接受客户端的订票请求,根据票务类型(铁路、航空还是航运)分别调用铁路、航空、航运的代理服务.如果接受到航空订票请求,那么,首先调用信用代理提供的个人信用查询服务,只有当查询返回个人信用等级良好才接受订票请求,进而调用航空代理.旅行代理的行为在图 3 的基础上增加了在线支付方面的操作.如果可以订票,那么客户可以请求在线支付订票费用,此时,旅行代理将调用金融代理服务,从客户的在线金融账号中扣除订票的服务费.

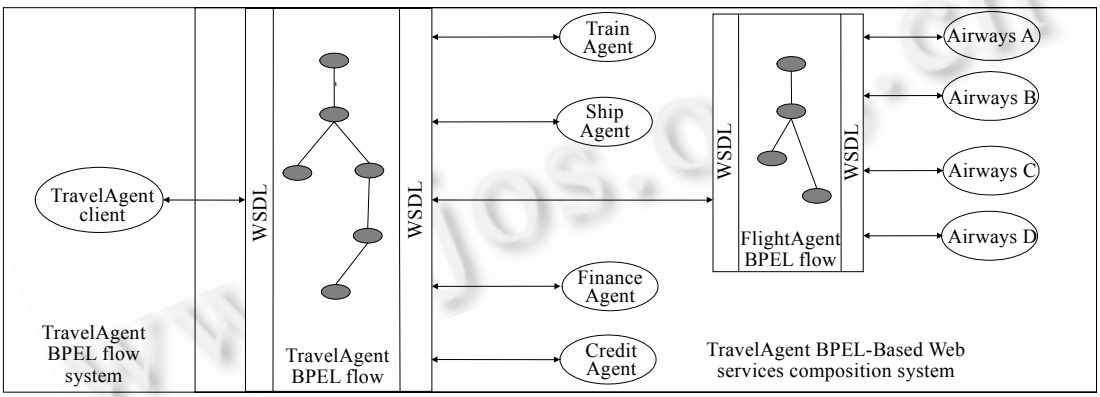

Fig.5　Composition system of *TravelAgent*

图 5　旅行代理组合系统

WSC/ADL 包括服务构件、连接件、配置 3 部分的定义.由于配置部分比较简单,所以下面只给出服务构件和连接件的定义示例.

本例中的服务构件分为 3 个层次:第 1 层次服务构件旅行代理描述整个组合系统的复合 Web services;第 2 层次服务构件包括了组成旅行代理的航空、航运、铁路、金融、信用代理这 5 个服务构件;而航空代理仍然是一个复合构件,所以还存在组成航空代理的第 3 层次服务构件航空公司代理.限于篇幅,这里只给出旅行代理 *TravelAgent* 的服务构件定义.*TravelAgent* 只有一个端口 *TravelAgentPort*,并且内部结构描述指向体系结构配置

TravelAgentConfi.类似地,可以给出其他服务构件定义.服务构件 *TravelAgent* 定义如下:

```
ServiceComponent  TravelAgent  as System
    DataType
        //定义命名空间
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:travelAgentNS="http://www.example.com/travelAgent"
        //定义数据类型 travelType,相当于枚举类型,取值为字符串类型表示的 3 种出行方式.
        travelAgentNS:simpleType name=travelType
            〈xsd:restriction base="xsd:String"〉
            〈xsd:enumeration value="flight"〉
            〈xsd:enumeration value="ship"〉
            〈xsd:enumeration value="train"〉
        End simpleType
        //XML Schema 预定义了字符串类型 String、日期类型 dateTime 等,所以这些数据类型无须再
        //定义
    End DataType
    Port  TravelAgentPort
        Message  clientbookingRequest
            //消息包含的参数:客户身份证号、出行方式、起始地、目的地、
            //出发日期、到达日期、备注等
            xsd.String: identifyID, departure, arrival, remark
            xsd.dateTime: departuredate, arrivaldate
            travelAgentNS.travelType: traveltype
        End Message
        …//类似定义了其他消息 clientbookingAck, clientbookingConfirmation, clientbookingRefusal
        …//clientpayingRequest, clientpayingAck, clientpayingConfirmation, clientpayingRefusal
        Operation  TicketBooking Provided
            Type: Input-Output
            Message:clientbookingRequest, clientbookingAck
        End Operation
        Operation  TicketbookingConfirmation Required
            Type: Input
            Message:clientbookingConfirmation
        End Operation
        …//类似定义了其他操作 TicketBookingRefusal,
        …//TicketPaying, TicketPayingConfirmation, TicketPayingRefusal
        //端口行为定义
        Port_Behavior
            //事件定义
            set TravelAgentEventSet={clientbookingRequest?(), clientbookingAck!(),
                                     clientbookingConfirmation!(), clientpayingAck!()
                                     clientbookingRefusal!(), clientpayingRequest?(),
```

clientpayingConfirmation!(), *clientpayingRefusal*!()}

//进程 *TravelAgentPort* 和 *PayingPort* 描述 *TravelAgentPort* 端口的行为,()表示对应消息值

proc *TravelAgentPort*=*clientbookingRequest*?()→*clientbookingAck*!()

→(*clientbookingConfirmation*!()

→*PayingPort*Π*clientbookingRefusal*!()→*TravelAgentPort*)

proc *PayingPort*=*clientpayingRequest*?()→*clientpayingAck*!()→

(*clientpayingConfirmation*!()→*TravelAgentPort*

Π*clientpayingRefusal*!()→*TravelAgentPort*)

End Port_Behavior

End Port

ServiceComponent_Behavior //服务构件行为

proc *TravelAgent*=*TravelAgentPort*

End ServiceComponent_Behavior

ServiceComponent_Structure //内部结构定义

Configure:*TravelAgentConfi*

End ServiceComponent_Structure

End ServiceComponent

本例中包含两个连接件:一个是组合航空、航运、铁路、金融、信用代理服务构件的连接件 *TravelAgentConn*;另一个是组合多个航空公司代理服务构件的连接件 *FlightAgentConn*.本文下面给出 *TravelAgentConn* 定义:

Connector *TravelAgentConn*

//数据类型定义,与服务构件中的定义相同

//与服务构件中的消息定义类似地定义了以下消息:

//*TrainAgentRole* 的消息 *trainbookingRequest*, *trainbookingAck*, *trainbookingConfirmation*,

//*trainbookingRefusal*; *FlightAgentRole* 的消息 *flightbookingRequest*, *flightbookingAck*,

//*flightbookingConfirmation*, *flightbookingRefusal*; *ShipAgentRole* 的消息 *shipbookingRequest*,

//*shipbookingReply*; *CreditAgentRole* 的消息 *creditRequest*, *creditReply*;

//*FinanceAgentRole* 的消息 *financeRequest*, *financeReply*

//*TravelAgentClientRole* 的消息 *clientbookingRequest*, *clientbookingAck*,

//*clientbookingConfirmation*, *clientbookingRefusal*, *clientpayingRequest*,

//*clientpayingAck*, *clientpayingConfirmation*,*clientpayingRefusal*

//与服务构件中的操作定义类似地定义了以下操作:

//*TicketBooking*, *TicketPaying*, *TicketBookingConfirmation*, *TicketBookingRefusal*,

//*TicketPayingConfirmation*,*TicketPayingRefusal*, *FlightTicketBooking*,

FlightTicketBookingConfirmation,

//*FlightTicketBookingRefusal*,*TrainTicketBooking*, *TrainTicketBookingConfirmation*,

//*TrainTicketBookingRefusal*, *ShipTicketBooking*, *ShipTicketReply*, *CreditQuerying*, *FinancePaying*

Role *FlightAgentRole*

Message:*flightbookingRequest*, *flightbookingAck*,

flightbookingConfirmation, *flightbookingRefusal*

Provided_Operation:*FlightTicketBooking*

Required_Operation:*FlightTicketBookingConfirmation*, *FlightTicketBookingRefusal*

Role_Behavior

set *FlightAgentRoleEventSet*={*flightbookingRequest*?(), *flighttbookingAck*!(),
flightbookingConfirmation!(), *flightbookingRefusal*!()}
proc *FlightAgentRole*=*flightbookingRequest*?()→*flightbookingAck*!()→
(*flightbookingConfirmation*!()→*FlightAgentRole*Π
flightbookingRefusal!()→*FlightAgentRole*)
End Role_Behavior
End Role
//类似地定义了其他角色 *TravelAgentClientRole*,*ShipAgentRole*,*TrainAgentRole*,
//*FinanceAgentRole*, *CreditAgentRole*
Glue
//对应事件定义
//描述各个角色组合交互的 5 个进程
proc *Glue*=*clientbookingRequest*?()→*clientbookingAck*!()→
(*ClientFlightGlue*Π*ClientShipGlue*Π*ClientTrainGlue*)
proc *ClientpayingGlue*=*clientpayingRequest*?()→*clientpayingAck*!()→
financeRequest!()→*financeReply*?()→
((*clientpayingConfirmation*!()→*Glue*)Π
(*clientpayingRefusal*!()→*Glue*))
proc *ClientFlightGlue*=*creditRequest*!()→*creditReply*?()→
((*flightbookingRequest*!()→*flightbookingAck*?()→
((*flightbookingConfirmation*?()→*clientbookingConfirmation*!()→
ClientpayingGlue)□(*flightbookingRefusal*?()→
clientbookingRefusal!()→*Glue*)))Π(*clientbookingRefusal*!()→*Glue*))
proc *ClientShipGlue*=*shipbookingRequest*!()→*shipbookingReply*?()→
((*clientbookingConfirmation*!()→*ClientpayingGlue*)Π
(*clientbookingRefusal*!()→*Glue*))
proc *ClientTrainGlue*=*trainbookingRequest*!()→*traintbookingAck*?()→
((*trainbookingConfirmation*?→*clientbookingConfirmation*!()→
ClientpayingGlue)□(*trainbookingRefusal*?()→
clientbookingRefusal!()→*Glue*))
End Glue
End Connector

4 WSC/ADL 与相关工作比较

WSC/ADL 是一种基于 BPEL 的 Web services 组合系统体系结构描述语言.实际上,体系结构描述有两种不同的思路:一种是使用专门的体系结构描述语言(ADL);另一种是采用通用建模语言 UML,例如文献[10]提出采用 UML 活动模型对 Web services 组合建模,组合模型可以被转化为 BPEL 等多种可执行语言.这两种思路方法各有利弊:UML 具有软件开发者比较熟悉、到具体实现的映射更为接近、有较成熟的工具支持等优点,但 UML 不直接支持体系结构风格的建模,缺少构件-连接件的体系结构显式描述特征,而这正是包括 WSC/ADL 在内的 ADL 特点.UML 作为一种面向对象的符号表示语言进行体系结构描述是否在本质上有充分的表达能力,目前也尚未有定论.另外,UML 缺少一个明确的形式语义,不利于系统性质的分析验证.

传统的软件体系结构研究中已经提出很多 ADL[11−13],具有代表性的 ADL 包括 C2,Rapide,Wright,Darwin 以

及国内研究机构提出的基于框架和角色模型的 FRADL:针对多智能体系统的 A-ADL、基于主动连接件的 Tracer、基于时序逻辑的 XYZ/ADL、基于层次消息总线的 JB/ADL、支持基于体系结构、面向构件软件开发方法的 ABC/ADL.其中,Wright[14]同样是一种采用 CSP 的体系结构描述语言.WSC/ADL 借鉴了 Wright 的部分设计思路,与 Wright 的主要不同包括:Wright 采用 CSP 贯穿整个系统的描述,包括构件的计算和连接件,而 WSC/ADL 对构件的行为和粘合采用 CSP 描述,但另外详细定义了数据类型、消息、操作、构件的内部结构等.所以,WSC/ADL 对组合系统的描述遵循抽象和具体相结合的原则,有利于组合系统的体系结构描述向可执行规范转化;Wright 支持对多种体系结构风格的建模,而 WSC/ADL 是针对特定的基于 BPEL 的层次控制总线风格定义组合系统的体系结构,强调构件(Web services)基于消息的交互、复合构件具有层次性,整个系统是一个复合构件等特征.

文献[3]提出了一个面向服务的 ADL,但该文将服务定义为角色集合之间的交互模式,采用消息序列图(message sequence charts,简称 MSC)对交互模式建模.该 ADL 的重点在于描述实体之间的交互,但这里的服务的含义不同于 Web services,实体的交互与基于 BPEL 的 Web services 组合交互也不尽相同.

综上所述,WSC/ADL 与上述 ADL 的最根本不同在于:WSC/ADL 是专门描述基于 BPEL 的 Web services 组合系统体系结构描述语言,充分考虑了 Web services 自身和基于 BPEL 进行 Web services 组合的特点,而其他 ADL 要么考虑的是一般软件系统(如 XYZ/ADL),要么针对的是特定对象(如针对智能体的 A-ADL)和或者某种特定风格(如针对层次消息总线风格的 JB/ADL)等,并不完全适合基于 BPEL 的 Web services 组合系统体系结构描述.

5 总结与展望

本文在 Web services 组合研究开发中引入软件体系结构研究中的概念和方法,设计了基于 BPEL 的 Web services 组合系统体系结构描述语言 WSC/ADL.WSC/ADL 定义了服务构件、连接件以及配置,可以更加清晰地刻画 BPEL 流程,并且引入形式化描述方法 CSP,有利于系统性质分析验证.目前,我们已经采用形式化验证工具实现了 WSC/ADL 描述的服务组合系统活性、安全性、一致性等性质的验证,并且建立了服务组合系统 WSC/ADL 描述至 BPEL/WSDL 的转换规则,实现了 Web services 组合系统半自动化生成.这种基于体系结构的开发方法提升了基于 BPEL4WS 的 Web services 组合系统生成效率和质量.我们接下来的工作是继续完善这种基于体系结构的 Web services 组合系统开发方法.

References:

[1] Milanovic N, Malek M. Current solutions for Web service composition. IEEE Internet Computing, 2004,18(6):51−59.

[2] Koehler J, Srivastava B. Web service composition: Current solutions and open problems. In: Proc. of the 1st Int'l Conf. on Automated Planning and Scheduling (ICAPS 2003) Workshop on Planning for Web Services. Trento: AAAI Press, 2003. 28−35.

[3] Kruger IH, Mathew R. Systematic development and exploration of service-oriented software architectures. In: Proc. of the 4th Working IEEE/IFIP Conf. on Software Architecture (WICSA 2004). Oslo: IEEE, 2004. 177−187.

[4] Milanovic N, Malek M. Architectural support for automatic service composition. In: Proc. of the IEEE Int'l Conf. on Services Computing (SCC 2005). Orlando: IEEE, 2005. 133−140.

[5] Khalaf R, Mukhi N, Weerawarana S. Service-Oriented composition in BPEL4WS. In: Proc. of the Int'l World Wide Web Conf. (WWW 2003). Budapest: ACM, 2003.

[6] Zhang SK, Wang LF,Yang FQ. Hierarchical message bus-based software architecture style. Science in China (Series E), 2002,32(6):393−400 (in Chinese with English abstract).

[7] Zhang SK, Zhang WJ, Chang X, Wang LF, Yang FQ. Building and assembling reusable components based on software architecture. Journal of Software, 2001,12(9):1351−1359 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/12/1351.htm

[8] Hoare CAR. Communicating Sequential Processes. London: Prentice Hall International Ltd., 1985.

[9] Mukhi N, Khalaf R, Fremantle P. Multi-Protocol Web services for enterprises and the grid. In: Proc. of the EuroWeb Conf. Oxford: St Anne's College. 2002.

[10] Skogan D, Grønmo R, Solheim I. Web service composition in UML. In: Proc. of the 8th Int'l Enterprise Distributed Object Computing Conf. (EDOC 2004). Monterey: IEEE, 2004. 47−57.

[11] Sun CA, Jin MZ, Liu C. Overviews on software architecture research. Journal of Software, 2002,13(7):1228−1237 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/13/1228.htm

[12] Zhang SK, Wang LF, Chang X, Yang FQ. Hierarchical message bus-based software architecture description language. Acta Electronica Sinica, 2001,29(5):581−584 (in Chinese with English abstract).

[13] Wang XG, Feng YD, Mei H. ABC/ADL: An XML-Based software architecture description language. Journal of Computer Research and Development, 2004,41(9):1521−1531 (in Chinese with English abstract).

[14] Allen R, Garlan D. A formal basis for architectural connection. ACM Trans. on Software Engineering and Methodology, 1997,6(3): 213−249.

附中文参考文献:

[6] 张世琨,王立福,杨芙清.基于层次消息总线的软件体系结构风格.中国科学(E 辑),2002,32(6):393−400.

[7] 张世琨,张文娟,常欣,王立福,杨芙清.基于软件体系结构的可复用构件制作和组装.软件学报,2000,12(9):1351−1359. http://www.jos.org.cn/1000-9825/12/1351.htm

[11] 孙昌爱,金茂忠,刘超.软件体系结构研究综述.软件学报,2002,13(7):1228−1237. http://www.jos.org.cn/1000-9825/13/1228.htm

[12] 张世琨,王立福,常欣,杨芙清.基于层次消息总线的软件体系结构描述语言.电子学报,2001,29(5):581−584.

[13] 王晓光,冯耀东,梅宏.ABC/ADL:一种基于 XML 的软件体系结构描述语言.计算机研究与发展,2004,41(9):1521−1531.

杨鑫(1978－),男,安徽安庆人,博士,主要研究领域为下一代网络,新一代互联网服务体系.

陈俊亮(1933－),男,教授,中国科学院院士,中国工程院院士,博士生导师,主要研究领域为电信网络,通信软件.