一个基于服务请求语言的统一 Web 服务框架*)

彭 鑫 赵文耘 吴毅坚 薛云皎

(复旦大学计算机科学与工程系软件工程实验室 上海 200433)

摘 要 提出一种基于服务请求语言的统一 Web 服务框架 UWSF。该框架通过统一的服务发布构件对外提供 Web 服务,并以服务项为单位进行组织,客户端通过对服务项的灵活组合获得各种定制的服务。引入服务请求语言作为客户端服务请求以及交互策略的描述手段。服务请求在服务端解释执行,使得双方的交互过程能够在会话环境中进行,并且减少了由于中间结果传输造成的带宽占用。服务端执行基于状态图的流程级交互控制以及基于权限检查接口的服务项级权限控制,保证了会话过程的合法性与完整性。

关键词 Web 服务,企业应用集成,面向服务架构,会话

A Unified Web Services Framework Based on Service Request Language

PENG Xin ZHAO Wen-Yun WU Yi-Jian XUE Yun-Jiao

(Department of Computer Science and Engineering, Fudan University, Shanghai 200433)

Abstract This paper presents a unified Web Services framework based on service request language. In this framework, Web Services are provided through a unified Web publishing component and organized by service items. The client can obtain various customised services by flexible combination of those service items. The service request language is introduced to serve as the description method for the service request and interoperation policy of the client. The service request is interpreted and executed on the server, this can provide conversation support for the interoperation and greatly decreases the occupation of bandwidth. Conversation control based on state diagrams and authority check of service items are performed on the server. Thus the validity and integrality of the conversation can be ensured.

Keywords Web Service, EAI, SOA, Conversation

1 引言

传统的企业应用集成(EAI)方法都很复杂,并且缺乏必要的灵活性,无法适应快速变化的业务需求。面向服务架构(Service-oriented architecture,SOA)为 EAI 提供了一种新的解决方案。SOA 是一种组件模型,它通过平台无关的接口将应用程序的不同功能单元联系起来,使得各种服务可以以一种统一和通用的方式进行交互。SOA 可以在不修改现有系统架构的情况下将系统和应用迅速转换为服务。

Web 服务技术被广泛用于 SOA,它为 SOA 提供了更高的灵活性。采用 Web 服务的一个很常见的方式是在已有系统之上增加 Web 服务代理,从而将系统转化为 Web 服务[1]。文[2]认为 Web 服务应该在业务逻辑之上单独构成一个层次,角色与用户接口层类似,并且提出了一个在已有应用基础上自动构造 Web 服务层的工具。

Web 服务可以看作一个"程序到程序"的表示层[2]。服务提供形式的转变也带来了业务服务角色的转变。传统的软件开发中的业务服务面向本地发布给表示层,服务接口按照自身需要定制。以 XML Web 形式提供的业务服务面向外部应用,需要面对的是多样的、变化的服务需求。而目前的 Web 服务体系中,服务接口的粒度一般都比较大,且不支持客户服务定制。另一方面,在许多企业应用集成场景中,交互双方需要通过会话进行多次消息通信。无状态的简单交互无法支持典型的业务交互场景下的多步骤这一根本特性[3]。与其它通信方式相比,Web 服务需要占用更多的网络带宽[2]。交互过程中的大量中间结果传输进一步加剧了这种状况。

由此可见,面向 EAI 的 Web 服务体系必须满足以下要求:

(1)方便在已有应用基础上构造 Web 服务。

(2)能够为客户端的服务调用提供灵活的可定制手段,使服务请求者能在基本的服务项基础上根据特定需求进行服务的整合。

(3)为 Web 服务交互提供会话支持,同时能够尽量减少会话过程中的带宽占用。

(4)完备的交互控制机制,使得服务提供者能够描述并执行自己的交互策略、权限控制等。交互技术不能对核心业务过程产生约束,业务过程不需要为特定的交互服务,而且二者的变化也是不同步的[3]。因此还必须将交互控制和业务逻辑加以分离。

针对这些需求,本文提出了一个基于服务请求语言的统一 Web 服务框架 UWSF(Unified Web Services Framework)。在这个框架中,已有的企业应用通过统一的 Web 服务发布构件对外提供服务。以服务项为 Web 服务的原子单位,客户端通过服务请求语言进行服务调用。服务请求是从请求者的角度出发的交互过程描述。服务器端根据请求描述控制本次会话并与本地业务逻辑交互,然后向客户端返回最终结果。Web 服务代理组件负责执行交互策略,以独立于业务逻辑的方式控制交互过程。本文第 2 节将介绍一些相关的研究工

*)本文受国家 863 计划(课题编号 2004AA112070,2004AA113030,2004AA113050)和国家自然科学基金(批准号 60473061)资助。彭 鑫 博士研究生,主要研究方向为软件工程、电子商务、企业应用集成。赵文耘 教授,博导,主要研究方向为软件工程、电子商务、企业应用集成。

作；第 3 节对总体服务框架进行了描述；服务请求的描述和执行，以及交互控制策略将分别在第 4 和第 5 节中详细介绍；第 6 节将结合一个数字化校园中的 EAI 实例说明该框架在服务定制、会话支持等方面的有效性和灵活性；最后对本文的工作进行了总结并展望了下一步研究工作。

2 相关工作

Web 服务有静态调用和动态调用两种调用方式[4]。动态调用是利用描述 Web Services 的元数据（例如 WSDL）在抽象层次上对服务进行调用，而在运行时刻完成绑定等工作[4]，例如 IBM 提出 WSIF（Web Services Invocation Framework）等。因此动态调用可以提供一定的服务定制和交互控制能力。文[4]基于 WSDL 文件和流程控制中的元数据进行动态 SOAP 调用和服务流的控制，从而实现 Web Services 的动态整合。在这个整合过程中，执行业务逻辑的前端服务器根据客户的具体请求以及存储在数据库中的远程服务元数据对多个远程 Web 服务进行集成。这种动态整合方法的交互控制能力较弱（例如要求服务接口提供下一步骤的控制信息），而且没有将业务逻辑与交互逻辑区分开来。

近年来，移动代理（mobile agent）[5~6]技术在分布式计算中得到了广泛应用。移动代理能够在复杂的网络系统中的异质主机间自主地移动，完成特定的分布式计算任务，具有节约带宽、支持服务定值和实时远程交互等优点。文[7]利用工作流的控制流机制实现基于移动 Agent 的多 Web 应用系统的集成控制。其中，工作流结点由一系列与子任务相关的移动 Agent 构成。整个系统通过移动 Agent 之间以及移动 Agent 与相应的 Web 系统之间这两种信息交互方式实现特定任务下多 Web 系统之间的应用集成。这样的多 Web 应用系统具有较高的统一控制性，而且需要工作流服务器和移动 Agent 服务器的支撑，因此适合于一系列紧密协作的 Web 应用系统集成（例如文[7]的运行实例中所介绍的工业过程控制领域）。

许多研究工作是关于如何在已有系统（例如遗产系统）基础上构造 Web 服务层[1~2]。Nicholas[2] 对基于角色交互的 Web 服务通用功能模式进行了研究，提出了一个在已有应用基础上构造 Web 服务层的工具。这个工具通过一系列的通用功能实现与 Web 服务通信，例如获取和发送数据。一系列基本交互可以根据特定的用户场景进行包装，例如通过数据获取接口传入一组方法名和相应的一组参数来实现类似于事务的执行模型。文[2]设计了两种数据服务接口（.NET 接口），分别针对单个方法调用和方法序列的调用（图 1）。其中的 Get 和 Post 方法分别用于获取和设置数据，而动态的服务（或服务序列）执行使用 .NET 中的反射（reflection）库来实现。这种接口使得已有应用的集成接口能够很容易地通过 Web 服务发布出去，而方法序列的调用方式提高了交互的粒度，减少了带宽占用。这种方式只能实现一组方法的顺序执行，缺少交互策略控制。

```
public interface IdataService
{
   object GetData (string method,object[] values);
   bool PostData (string method,object[] values);
}
public interface ISequenceDataService
{
   object[] GetSequencedData (string[] methods,object[][] values);
   bool[] PostSequencedData (string[] methods,object[][] values);
}
```

图 1 数据服务接口[2]

还有许多研究工作关注于如何在业务过程集成过程中提供会话支持[3,8]。Hanson[3] 提出了一种基于会话的异步消息机制用于业务过程集成。交互过程中的每条消息的具体语义都取决于会话中的上一条消息。交互策略用状态机描述，交互双方各自执行自己的交互策略并控制与业务过程之间的数据交互，直至会话结束。这种集成机制很好地将业务逻辑与交互逻辑加以分离，但每次交互需要进行多次消息通信，带宽占用较大。交互控制策略仅在会话状态控制层面上，没有涉及每条交互消息的控制策略，例如权限控制。

如果在系统中引入 Web 服务，那么 Web 服务应该单独构成一层，扮演着与用户接口层类似的角色[2]。因此，应该在原来的应用系统基础上部署代理层，完成数据的接收和发送，同时进行权限控制和服务定制。文[2]中的数据服务接口只是多个方法的简单组合，无法支持会话过程。Hanson[3] 所提出的会话机制基于双方的多次消息通信，交互开销较大。移动代理[5] 具有节约带宽、支持服务定值和实时远程交互等优点，这些都是通过具有自主性的对象的移动实现的。借鉴其中的思想，我们提出在 SOA 环境下基于服务请求语言实现会话交互以及服务的定制。文[2]中的数据服务接口只能为客户端提供简单的方法组合，而服务请求语言则要能提供交互过程描述能力。通过这种交互过程描述，客户端的请求可以由服务器端解释并执行，完成交互后再把最终结果返回给客户端。

3 统一服务框架 UWSF

UWSF 的总体结构如图 2 所示，其中的核心部分是服务发布构件，它构成了本地业务逻辑的服务发布层。服务发布构件负责接收并解释客户机的服务请求，然后按照服务请求流程驱动本地服务，并返回最终结果。本地业务逻辑以服务项为单位对外提供服务，并在服务发布构件中注册相应的接口。客户端通过服务请求描述对服务项以及会话逻辑进行组合和描述。另一方面，服务发布构件还负责执行交互策略和权限控制。从图 2 中可以看出，交互控制主要包括两部分：流程级交互策略和服务项级权限控制。前者从流程上保证服务请求符合服务器端交互策略，而后者针对单个服务项进行角色权限控制。服务发布构件不仅是业务逻辑的 Web 发布代理，而且还实现交互控制。这样，Web 服务的交互策略就可以独立于业务逻辑进行定义和执行，当交互需求变化时只需要修改交互策略。而服务提供者只需要根据自身在 EAI 中的服务角色提供通用的业务服务，不需要考虑各种特定的交互需求。

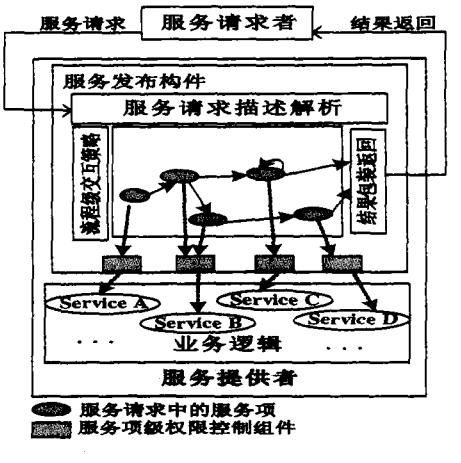

图 2 统一 Web 服务框架 UWSF 总体结构

服务请求描述除了指定需要调用的服务项外，还需要描述交互过程。我们采用 XML 作为服务请求的描述方法，具体的细节将在第 4 节中介绍。流程级交互策略用状态图描述，而服务项级权限控制将通过服务项的权限验证接口进行。交互控制将在第 5 节中详细介绍。

4 服务请求的描述和执行

UWSF 的服务发布构件对外提供统一的 Web 服务调用接口，接口中的参数包含两部分：服务请求描述和参数对象数组。参数对象数组传递服务请求的实参对象列表，而服务请求描述指定所请求的服务项的执行流程以及参数引用。服务的返回有两种可能：(1)服务执行到服务请求中指定的返回点，此时返回服务请求中指定的对象；(2)服务执行过程中出现例外，此时返回错误信息。

4.1 服务请求描述需求

在 UWSF 中，服务请求作为客户端对 Web 服务项的组合执行逻辑描述，将由服务端服务发布构件解释执行。服务请求描述包括所调用的服务项、相应的参数、执行流程等。其中，执行流程用来表示与服务端的会话过程。通过服务请求描述，客户端可以预定义本方的会话控制策略，并由服务端代理执行过程，从而实现会话过程并减少因中间结果传输而造成的带宽占用。

在 Hanson[3] 提出的过程集成机制中，交互双方各自基于状态图执行自己的交互策略。而在 UWSF 中，客户端的会话策略需要预定义并在服务端执行，因此服务请求语言必须满足以下需求：(1)丰富的逻辑表达能力，例如流程控制、消息的上下文关系等；(2)机器可读的；(3)足够的灵活性，能够满足客户端对于服务定制的需要。

4.2 服务请求语言的描述

文[9]介绍了一种支持 Web 服务组合的可视化编程语言 VINCA，这种语言可以供业务用户通过业务端编程创建业务应用。VINCA 的过程模型中包括服务结点和控制结点，控制结点用来描述业务服务之间的控制逻辑关系。而在 UWSF 中，服务请求语言用于 EAI 环境下客户端定制与 Web 服务端的交互过程。我们用 XML 作为服务请求语言的描述手段，服务请求 serviceRequest 的 XML schema 定义如图 3 所示。

服务请求语言描述了一系列服务项以及执行逻辑。其中，唯一的 initStep 元素定义服务请求的初始服务项，0 个或多个 step 元素定义若干后续步骤。step 是服务请求的基本构成，包含一个用来标识该步骤的名称属性(name)以及对应的服务项名属性(serviceName)。每个 step 包含任意个参数(param)，以及至少一个转换关系(transition)。参数有两种类型：一种是客户端传入的对象数组中的某一个对象，用数组的下标描述(paramIndex)；另一种是前面某一步骤的中间结果对象或该对象的一个成员，用该步骤的名称(stepName)以及成员名称(member，空则代表使用该对象本身作为参数)表示。每一个转换关系由条件(fixCondition 或 condition)和转换动作(gotoStep 或 return)组成。条件可以是固定的(常量 true)，表示永远有效的转换。非固定条件是对当前步骤返回对象的判断语句，由服务端在运行期进行判断。gotoStep 指定下一步骤的名称，可以是当前步骤，这样就可以实现服务项的循环调用。return 表明请求结束，服务端向请求者返回此前任一服务项的返回对象(用步骤名表示)。由于每一步骤的中间结果都可能被后续服务作为参数引用，因此服务发布构件在执行服务请求期间将保留各步骤的中间结果对象，如果某一步骤反复执行那么只需要保持最后一次执行的结果。

```
<complexType name="serviceRequest">
  <sequence>
    <element name="initStep" type="stepType" />
    <element name="step" type="stepType" maxOccurs="unbounded"
      minOccurs="0" />
  </sequence>
</complexType>
<complexType name="stepType" id="name">
  <sequence>
    <element name="param" type="paramType" maxOccurs="unbounded"
      minOccurs="0" />
    <element name="transition" type="transType" maxOccurs="unbounded"
      minOccurs="1" />
  </sequence>
  <attribute name="name" type="string" use="required" />
  <attribute name="serviceName" type="string" use="required" />
</complexType>
<complexType name="paramType">
  <choice>
    <element name="paramIndex" type="integer" />
    <sequence>
      <element name="stepName" type="string" />
      <element name="member" type="string" />
    </sequence>
  </choice>
</complexType>
<complexType name="transType">
  <sequence>
    <choice>
      <element name="fixCondition" fixed="true" type="string" />
      <element name="condition" type="string" />
    </choice>
    <choice>
      <element name="return" type="string" />
      <element name="gotoStep" type="string" />
    </choice>
  </sequence>
</complexType>
```

图 3 服务请求语言 Schema

4.3 服务请求的执行

UWSF 中服务请求描述由服务端的服务发布构件解释并使用反射机制执行，与文[2]类似。与 Web 服务一样，UWSF 本身独立于特定的平台和语言。但为了论述方便，本文采用 Java 作为接口定义以及实例研究的实现语言。

Web 服务端的各服务项都要向服务发布构件注册服务接口(包括方法名、参数列表等)和相应的权限控制类(5.2 节)。服务发布构件按照解析得到的执行流程调用相关服务项。各服务项并不需要提供 Web 服务接口，服务发布构件使用 Java 反射机制将运行时服务名和参数对象列表转化为对实际服务项的调用，并保存返回结果。服务描述中的非常量条件可以是任意的 Java 逻辑运算语句，可以使用 #stepname# 特殊标记代替此前某一步骤的结果对象。在运行时，服务发布构件负责解析条件语句，并使用反射机制引用对象实例进行条件判断，然后根据结果确定转换方向。

执行过程中必须同时进行服务端交互策略控制，首先必须保证服务请求状态转换符合流程级交互策略(5.1 节)，其次还要检查各服务项调用(包括参数)是否符合当前请求者的角色权限(5.2 节)。如果违反服务端交互策略则结束本次服务并返回出错信息。如果服务请求中的 return 转换条件满足，那么服务将正常结束并返回指定的服务项返回对象。

5 交互控制策略

服务请求描述了客户端的交互策略，而服务端也需要相应的机制来描述并执行服务端的交互策略。UWSF 的交互策略分为流程级和服务项级两个方面。流程级交互策略根据请求消息的时序特征保证服务交互序列的合法性，而服务项级则根据客户端的角色判断其是否对于某服务项具有访问权以及参数是否超出其权限范围。

5.1 流程级交互策略

服务端以服务项的方式提供 Web 服务，而客户端通过服

务项的有机组合构造服务请求。因此，服务端还需要在交互流程上进行控制，因为各服务项之间可能存在相关性。Hanson[3]认为会话策略是一种机器可读的会话消息交换模式描述，并提出使用状态机描述会话策略。在状态机中，消息的发送对应会话状态的迁移。业务端编程语言 VINCA[9]面向终端业务用户，以业务补充规则的方式控制业务服务的上下文，包括服务依赖规则、服务输入规则和服务选取规则。

在 UWSF 中，服务端流程级交互策略使用状态图描述。服务端可以根据业务逻辑的相关约束定义流程级交互策略，将所允许的会话上下文关系用状态图描述出来。流程级交互策略的目标是保障服务的合法性和完整性，主要包含两部分工作：(1)在执行客户端服务描述的同时保证会话状态的合法性；(2)按照交互逻辑执行一些关联操作，例如图 4 中的项目检查在报销标准检查后由服务端自动执行，不需要请求者指定该服务。

图 4 是一个简单的流程级会话策略实例，描述了数字化校园中财务系统对外提供的报销相关服务的流程级交互策略，这个服务可以提供给校园办公平台作为教职工差旅管理的一部分。这个简单的流程控制主要包括申请、审查、发放等步骤。服务请求者的请求消息 applyForExpense 启动报销流程并传入申请单对象，applyForExpense 服务项返回报销标准审查结果，如果不通过则不予报销。通过报销标准检查后，checkProject 消息调用项目检查(例如项目余额、科目控制等)。checkProject 前的 self 关键字表示这条消息由服务发布构件自动发出，也就是说项目检查服务由服务端隐式调用，对请求者透明。项目检查通过后，根据请求者的 payByCash 或 payByCard 消息调用相应的现金支付和进银行卡处理，如果进卡失败那么自动转为待发放现金状态。在会话过程中，如果服务请求偏离状态则结束本次会话。到达终点状态后当前状态图控制结束，如果还有服务请求并未返回，那么继续根据请求消息开始另一个状态图控制过程，直至服务请求返回或者发生例外。

服务请求描述是从客户端角度出发的会话描述，而交互策略状态图是从服务端角度出发的会话描述。例如图 4 中请求者的请求序列可以是 applyForExpense — payByCard(或者 payByCash)，而服务端会话策略中则还包含标准检查、项目检查等一系列状态控制。因为 Web 服务的交互双方都只需要从各自角度参与并控制会话过程。self 消息定义了服务端自动执行的操作，这些操作都是从服务端的角度出发的一些保障会话合法性和完整性的必要步骤。这种方式使得交互控制在保持一定灵活性的同时，对本地业务逻辑的影响较小。例如当交互策略发生变化，不需要进行项目检查时，只需要在交互策略中去掉相关状态即可，本地基本逻辑不受影响。

需要指出的是，服务请求描述中的执行流程可以是状态图中的一部分，也可以跨越多个状态图。也就是说服务请求可以在某个中间步骤上返回，还可以在完成一个状态图中的完整流程后继续开始另一个流程。将申请报销和进行支付作为两个独立服务项提供给请求者是因为这两种服务具有一定的独立性。例如支付是一种通用服务，不仅可以用于报销支付，还可以用于奖金等的支付；而申请报销服务可以单独使用(暂存在个人财务账户上)。

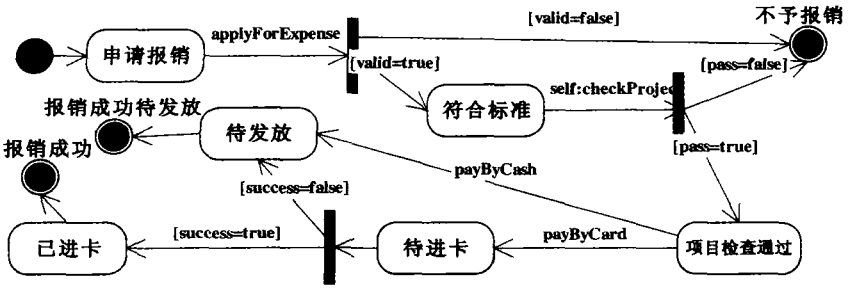

图 4 交互策略状态图示例

5.2 服务项级权限控制

UWSF 中，服务项是服务调用的原子单位，每个服务项均对应一定的权限控制策略。权限控制与请求者的角色相关，包括方法和参数两个层次。方法权限用来判断请求者是否对该服务项有访问权限，而参数权限用来判断请求参数是否超出了角色权限范围。例如，在数字化校园的 SOA 体系中，财务系统负责为相关部门提供学生缴费信息的 Web 服务，该服务本身以费用项代码为参数，与权限无关。那么教务系统和后勤系统都具有该服务项的访问权限，但所允许参数范围不同(例如教务系统只能查询学费，而后勤系统只能查询住宿费)，由服务项级交互策略控制。以权限无关的方式提供 Web 服务使得业务逻辑能够保持与特定交互策略的独立性。

服务项级权限控制通过相应的接口实现，如图 5。IserviceCheck 接口定义了两个方法、checkMethod 和 checkParams，分别代表方法和参数权限检查。其中，roleList 参数代表请求者的角色列表(同一请求者可能具有多种角色)，values 代表请求该服务项时的参数列表(按实际接口中的参数顺序排列)。在 UWSF 中，每个注册到服务发布构件的服务项都必须指定与之对应的权限检查类(实现 IServiceCheck 接口)。这些权限检查类将在相应的服务项被调用时由服务发布构件通过 Java 反射机制调用。方法权限和参数权限检查将在服务项执行前先后进行，如果未通过则向请求者返回出错消息。

```
public interface IServiceCheck
{
 public static boolean checkMethod (String [] roleList)
 public static boolean checkParams (String [] roleList,Object[] values)
}
```

图 5 服务项目级权限控制

6 实例研究

在 UWSF 中，参与企业应用集成的各系统根据自身业务角色提供一定的服务，同时也向其它服务提供者请求服务。这一节将结合数字化校园中的一个应用介绍 UWSF 在企业应用集成中的有效性。

6.1 应用场景

数字化校园的一个重要目标是整合学校中各相关部门的

信息系统，为用户提供一体化的信息服务。电子公务平台(OA系统)是数字化校园的重要组成部分。OA系统面向学校师生提供日常公务服务，例如差旅管理服务。差旅管理服务的整个流程包括差旅费计划申请、审批、报销等，涉及多个部分。现在，OA系统中将此一系列服务整合为差旅管理提供给用户，其中的报销相关流程由财务系统提供。财务系统目前提供的相关服务有报销申请、支付等，另外还提供有扣除欠款等财务服务。因此OA系统需要根据需要定制报销相关流程(服务请求描述如图6)。

```
<?xml version="1.0" encoding="gb2312" ?>
<serviceRequest>
  <initStep name="apply" serviceName="applyForExpense">
    <param>
      <paramIndex>0</paramIndex>
    </param>
    <transition>
      <fixCondition>true</fixCondition>
      <gotoStep>pay</gotoStep>
    </transition>
  </initStep>
  <step name="pay" serviceName="payByCard">
    <param>
      <stepName>apply</stepName>
      <member>payForm</member>
    </param>
    <transition>
      <condition>#pay#.result</condition>
      <gotoStep>allowance</gotoStep>
    </transition>
    <transition>
      <condition>!#pay#.result</condition>
      <return>pay</return>
    </transition>
  </step>
<step name="allowance" serviceName="payByCard">
    <param>
      <paramIndex>1</paramIndex>
    </param>
    <transition>
      <fixCondition>true</fixCondition>
      <return>pay</return>
    </transition>
  </step>
</serviceRequest>
```

图6　报销服务请求描述实例

6.2　应用实例

图6是报销服务请求serviceRequest的XML描述实例。这个请求包含三个步骤：报销申请(apply)、支付(pay)和支付出差补贴(allowance)。出差补贴的标准由OA系统计算并通过支付服务由财务系统执行。这个服务请求从OA系统的角度出发，将这三个服务步骤整合为出差报销服务。而服务端按照交互逻辑还会自动执行相应的项目检查等(图4)。与此服务请求对应的还有一个包含两个对象的参数数组，第一个代表报销申请对象，第二个代表补贴支付对象，这两个对象分别被apply和allowance步骤作为参数引用。步骤pay的参数将来自于apply步骤的返回结果中所包含的payForm对象。服务请求中还指定了报销支付失败后返回，如果成功则继续执行补贴支付。

总结和展望　近年来，基于Web服务的EAI一直是一个研究热点。一方面，Web服务必须能够很容易地在已有应用上构造；另一方面，必须在提供一定的服务定制以及会话能力的同时保障交互过程的合法性和完整性。UWSF正是这样的一个Web服务框架，它具有以下特点：

(1)通过统一的Web服务接口对外提供服务，使得在已有应用的基础上构造Web服务变得简单。

(2)以服务项为原子单位，通过服务项的灵活组合完成每次服务请求，同时将交互控制与业务逻辑加以区分，使得Web服务具有更高的灵活性。

(3)以服务请求描述为基础，为交互双方提供了会话支持，同时交互过程中的中间结果传输大大减少，节约了带宽占用。

(4)服务提供者在交互会话流程和原子服务请求两个层面进行交互控制，使得服务策略和权限控制的执行得到保障。

UWSF的不足主要体现在客户端需要了解服务端各服务项的参数、返回对象等信息，这使得这个框架更适合于企业内的应用集成，例如数字化校园。UWSF的另一不足主要是交互能力较弱，服务请求只能描述请求者的基本交互流程。

UWSF的基本思想已经在一些高校财务服务平台中得到应用，例如浙江大学、上海交大等。进一步的研究工作主要包括：基于UWSF框架提供本地业务服务的自动发布支持机制，使得业务用户可以方便地进行服务发布的定制；增强服务请求描述的交互能力，例如引入服务回调支持，允许在交互过程中反向调用服务请求者的服务，使得请求者能够表达更加复杂的交互策略。

参 考 文 献

1　Litoiu, M. Migrating to Web services - latency and scalability, Web Site Evolution, 2002. In: Proc. Fourth Intl. Workshop on , Oct. 2002

2　Nicoloudis N, Mingins C. XML Web services automation: a software engineering a proach. Software Engineering Conference, 2002. Ninth Asia-Pacific , Dec. 2002

3　Hanson J E, Nandi P, Kumaran S. Conversation support for business process integration. Enterprise Distributed Object Computing Conference, 2002. EDOC '02. In: Proc. Sixth International , Sept. 2002. 65～74

4　潘长胜，于浩海，王光兴. Web Services动态整合的体系结构和算法. 东北大学学报(自然科学版)，2003，24(5)：445～448

5　朱淼良，邱瑜. 移动代理系统综述. 计算机研究与发展，2001，38(1)：16～24

6　冯新宇，吕建，曹建农. 通用的移动Agent通信框架设计. 软件学报，2003，14(5)：984～990

7　周光辉，江平宇，丘礼平. 基于工作流和移动Agent的多Web应用系统的集成研究. 西安交通大学学报，2002，36(9)：933～937

8　Yano K, Hara H, Uehara S. Collaboration Management Framework for Integrating B-to-B and Internal Processes. Enterprise Distributed Object Computing Conference, 2002. EDOC '02. Proceedings. Sixth International , Sept. 2002. 75～83

9　熊锦华，李厚福，韩燕波，耿晖. 支持即时构造的业务端编程语言VINCA. 计算机辅助设计与图形学学报，2004，16(2)：180～185