基于状态方面的Web服务动态替换

窦文生[1] 吴国全[1,2] 魏 峻[1] 刘绍华[1]
(中国科学院软件研究所软件工程技术研究中心 北京 100190)[1]
(中国科学技术大学计算机科学技术系 合肥 230026)[2]

摘 要 随着面向服务计算技术的成熟,服务复合已成为Internet上开发企业间业务协作的一种新模式,WS-BPEL是服务复合事实上的标准。但是由于复合服务所依赖的第三方伙伴服务的分布、自治和松散耦合等特性,在执行过程中易受到伙伴服务失效的影响,可靠性无法得到保证,因此需要支持在运行时对伙伴服务进行动态替换。目前的BPEL规范只提供有限的服务替换功能,当与伙伴服务的交互涉及到一系列有状态的会话操作时,服务替换就更加复杂。通过对面向方面的研究,提出面向BPEL语言的状态方面扩展。通过状态方面,记录与伙伴服务交互过程中的会话信息。在伙伴服务失效时,通过透明地替换伙伴服务,使得与当前伙伴服务的会话信息传播到功能等价的另一个伙伴服务上,以保证流程的正常执行。通过该方法,使得BPEL流程具有一定的自愈能力,增强了流程执行的可靠性。

关键词 Web服务,WS-BPEL,状态方面,动态替换
中图法分类号 TP311.56 **文献标识码** A

Dynamic Substitution of Web Services through Stateful Aspect Extension

DOU Wen-sheng[1] WU Guo-quan[1,2] WEI Jun[1] LIU Shao-hua[1]
(Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)[1]
(University of Science & Technology of China, Department of Computer Science & Technology, Hefei 230026, China)[2]

Abstract As the service-oriented computing technology matures, service composition has became a new model for business cooperation among enterprises in internet era. WS-BPEL is the de-facto standard for service composition. However, because Web services are loosely coupled, distributed, and autonomous, the reliability of BPEL processes can't be assured when partner services fail. As a result, dynamic runtime substitution of partner services is needed. But BPEL provides only limited service substitution. And when the partner services are stateful, the problem gets more complex. This paper presented an stateful aspect extension of WS-BPEL. Stateful aspects were used to store the conversational details of partner services. In case of partner service failures, the conversational details can be replayed on another equivalent partner service transparently for the BPEL process. Our approach makes WS-BPEL have some ability of self-healing and improves the reliability of processes.

Keywords Web service, WS-BPEL, Stateful aspect, Dynamic substitution

1 引言

服务是一种构建面向服务架构(SOA)的分布式计算技术,已经成为构建下一代软件"网构软件"的核心技术[1]。在面向服务架构中,单个服务提供的功能有限,服务复合通过组合现有的Web服务从而创建新的、高层的Web服务以获取新的功能,是构建Web服务应用的重要手段,已成为开发企业内部业务应用和企业间协作的一种新模式。WS-BPEL[2]已经成为开发这类业务流程的事实标准。

复合服务所依赖的第三方伙伴服务的分布、自治和松散耦合等特性,使得伙伴服务本身是不可靠而且易于出错的。这使得复合服务执行过程中易受到伙伴服务失效的影响,复合服务的可靠性就无法得到保证。复合服务如何灵活应对伙伴服务失效的问题,就显得尤为突出。

在伙伴服务失效的情况下,用另一个功能等价的伙伴服务,替换原有的伙伴服务,使得复合服务能够继续正确完成自身的业务逻辑。WS-BPEL自身提供了伙伴服务替换的功能,但是需要复合服务开发者自己定义错误处理逻辑,增加了复合服务开发的复杂度;而且这种替换功能也相当有限。当前有不少关于伙伴服务替换的研究,如文献[7]利用面向方面编程(AOP[9])的方法,动态选择最优的服务调用,运行时绑定到具体的伙伴服务;文献[15,16]利用动态代理的方法,当伙

到稿日期:2009-01-16 返修日期:2009-03-02 本文受国家高技术研究发展计划项目(No. 2006AA01Z163,2007AA010301),国家科技支撑研究发展项目(No. 2006BAH02A02)资助。

窦文生(1984-),男,硕士研究生,主要研究方向为网络分布计算、软件工程,E-mail:wsdou@otcaix.iscas.ac.cn;吴国全(1979-),男,博士研究生,主要研究方向为网络分布计算、软件工程;魏 峻(1970-),男,博士,研究员,主要研究方向为网络分布计算、软件工程;刘绍华(1976-),男,博士,助理研究员,主要研究方向为网络分布计算、软件工程。

伴服务出错时，利用代理从注册库中选择一个功能等价的服务来完成错误恢复。

当前的这些研究，都是假设复合服务的伙伴服务是无状态的。但是在实际应用中，很多伙伴服务是基于会话的。复合服务和伙伴服务的交互，要通过一定的交互协议来进行[17]。当前这些关于服务替换的研究不能适应有状态伙伴服务替换的情况。在复合服务和有状态成员服交互过程中，如果伙伴服务失效，对该伙伴服务的替换要利用服务会话过程信息，使新的伙伴服务具有原有伙伴服务的会话状态，使得复合服务能够继续运行。这就需要一种新的服务替换方法。

对于伙伴服务的错误恢复逻辑，希望能够从复合服务设计中分离出来，AOP 是一种比较好的方法。例如文献[7]中使用 AOP 实现服务替换。对于面向 BPEL 的方面扩展，已经由 AO4BPEL[4]和 Padus[12]做了相关工作，实现对流程活动的横切关注点。但是他们对于复合服务中活动执行历史的状态方面(stateful aspect)都很少关注，而面向活动执行历史的状态方面是由状态服务替换所需要的。

本文首先提出一个面向 BPEL 的状态方面扩展方法。将与复合服务交互的伙伴服务视为抽象的服务，并利用状态方面定义实际的伙伴服务，负责复合服务与伙伴服务的交互。在流程和伙伴服务交互过程中，如果出现伙伴服务失效，则利用状态方面的处理机制，替换失效的伙伴服务，并将当前交互过程中该伙伴服务的会话信息传播到新的伙伴服务上。从而在复合服务运行过程中，动态透明地替换服务，提高流程的可靠性。

本文第 2 节介绍研究动机和状态 AOP；第 3 节介绍提出的面向 BPEL 的状态方面扩展；第 4 节介绍基于状态方面的动态服务替换方法；第 5 节介绍原型实现；第 6 节说明相关工作；最后进行总结，并指出进一步的研究工作。

2 研究动机和状态 AOP

复合服务与第三方伙伴服务的交互通常涉及到一系列有状态的会话操作。例如银行转账服务，将购物款(cost)从 ClientBA 账户转入 ShopBA 帐户。其抽象 BPEL 流程简单描述如图 1 所示。

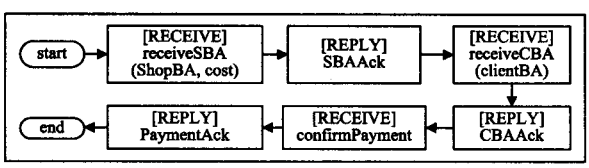

图 1 银行转帐抽象流程

银行转帐包括以下步骤：

1)收到网上商店的银行帐号信息 ShopBA 和要转入的金额 cost，对网上商店的银行帐户进行确认；

2)收到用户的银行账户信息 clientBA，并进行检查确认；

3)收到用户的确认信息，执行转账。

从上面过程看出，银行转帐服务是有状态服务。当一个网上书店将这个银行转帐服务编排到网上书店的复合服务时，网上书店服务根据银行转帐服务要求的协议与其进行交互。但是银行转帐服务是一个自治的第三方服务，在与其进行交互过程中可能会出现各种故障。例如在交互过程中，网上书店流程调用了 receiveSBA，并得到确认结果，而在调用 receiveCBA 时，网上银行服务由于网络连接等原因而不再可用。为了能够满足用户网上购书的功能性需求，网上书店服务不希望一个伙伴服务的失败而导致购书流程失败。这就需要网上书店流程能够动态替换掉这个有状态的伙伴服务，调用另外一个提供相同功能的 Web 服务。

面向方面的软件开发方法(AOSD[9])是一种新的软件开发范型，它将软件系统的一些横切关注点描述为方面。在程序执行过程中，当程序执行到由方面定义的某个特定点时，方面定义的横切关注点行为将会执行。程序执行中的某些特定点称为连接点；定义连接点的表达式称为切入点；方面定义的横切关注点行为称为通知。

通常切入点只关注程序执行的当前行为，而无法表达依据程序执行历史而执行相应通知的行为。定义特定的切入点，表达对程序执行历史的关注，并根据执行历史执行相应的横切关注点行为，这种方面称为有状态方面，这种特性已经在 JAsCo[6]和 tracematches[13]中有相关支持，它们是针对 Java 语言有状态方面扩展。

我们提出了一个面向 BPEL 语言的状态方面扩展，通过状态方面，记录网上商店和银行转帐服务交互过程。在当前银行服务失效后，在另一个银行转帐服务上恢复伙伴服务的会话信息，从而使流程继续执行，提高网上商店的可用性。

3 面向 BPEL 的状态方面扩展

当前面向方面的方法很少支持基于协议的方面。为了解决这个问题，Douence[5]等人提出了一种根据协议触发条件的形式化方面模型，称为状态方面。在这一节中，基于这个形式化模型，根据 BPEL 语言的特点，提出一种面向 BPEL 语言的状态方面扩展，增强切入点的表达能力，以支持复合服务运行时伙伴服务的动态替换。

在所提出的状态方面扩展中，用户可以灵活定义与伙伴服务进行交互的协议切入点，并定义相应协议切入点的通知操作。

图 2 是一个状态方面扩展的示例。下面详细描述所提出的状态方面。首先介绍切入点和协议切入点，以及协议切入点的语义状态机；最后定义相应的通知机制。

```
1   <aspect name="ProtocolAspect">
2       <partnerlink name="partnerlink1" ... />
3       <variable name="message1" .../>
4       <protocolpointcuts>
5           <complement />
6           <transition name="tran1">
7               <pointcut>
8                   //invoke[@pt="bank" @op="receive"]
9               </pointcut>
10              <destinationTransitions>
11                  <transition ref="Tran2" />
12              </destinationTransitions>
13          </transition>
14          <transition name="tran2">...</transition>
15      </protocolpointcuts>
16      <advices>
17          <advice transition="tran1" type="after fault">
18              ...
19          </advice>
20          ...
21          <complement type="around">...</complement>
22      </advices>
23  </aspect>
```

图 2 BPEL 状态方面扩展代码示例

3.1 连接点、切入点和协议切入点

BPEL 流程包含一系列活动，每个活动都有可能是一个连接点。切入点是捕捉连接点的结构，它用来匹配连接点，在

潜在的连接点中挑选出符合规则的连接点来织入方面。活动的属性可以被作为选择相应连接点的条件。BPEL 流程是采用 XML 描述的文档，则利用 XPath[18] 描述切入点是一种自然的方式。本文采用 XPath 作为切入点的描述语言。

在大多数的面向 BPEL 的方面扩展中，切入点描述连接点和相应条件的限制，不能描述出流程协作过程中的状态信息，不支持有状态方面。我们提供一个基于状态机的切入点扩展机制，描述服务交互过程的状态和协议。这种扩展明确描述在服务协作过程中有状态服务执行历史。

对图 2 的状态方面示例，说明本文定义的协议切入点。

协议切入点：协议切入点定义协议中引起状态变化的转换。在图 2 中，6－13 行定义一个协议切入点的示例。第 6 行描述转换的名字；7－9 行描述该切入点发生的条件，使用 XPath 定位到一个活动切入点；第 10－12 行表示 tran1 发生之后接着会发生的转换，按示例的要求，接着是 tran2 发生。

在协议执行过程中，如果活动没有按照协议规定执行，则引起对协议的违背，complement 表示对协议违背的切入点。在图 2 的第 5 行定义了协议违背切入点。如示例中，tran1 发生之后，希望 tran2 发生；但是 tran2 没有发生而 tran3 发生，则与该协议违背。

在 invoke 活动执行过程中，第三方伙伴服务的失效等将导致 invoke 活动失败。定义异常处理 fault，表示该活动执行错误，如图 2 中第 17 行所示。

3.2 协议切入点状态机

上面定义的面向 BPEL 的状态扩展是与确定有限自动机（DFA[14]）等价的。协议切入点明确描述了一个以转换为中心的状态机。将状态方面构造成一个 DFA，随着 BPEL 流程的执行、驱动 DFA 的执行，实现对协议的解释，并记录下协议执行历史。

当流程执行到某个协议切入点时，根据通知的设计规则，执行相应的通知，并激活状态机中设定的下一步转换，为下一步转换事件做准备。DFA 检查当前的状态和转换事件，驱动自动机的执行。

3.3 通知（Advice）

流程执行到特定的协议切入点时，要执行的动作称之为通知。为了与目标语言保持一致，在所提出的状态方面，选择 BPEL 语言作为通知语言。所定义的通知操作只在一个特定的流程实例上起作用，因此动态替换只作用于出现故障的特定流程实例。

和 AO4BPEL 类似，状态方面扩展允许定义变量、伙伴链接等，并在通知内使用。通知可能需要复合服务执行过程中的上下文信息，提供变量 MYMProcess 保存这些信息，比如流程名字等，变量 MYMActivity 记录活动相关的信息。

定义了 4 种通知类型（type）：before，around，after，after fault，表示通知执行的时机；分别表示在转换动作执行前、替换转换动作、转换执行后、活动错误发生后。如果同时定义一个切入点的 4 种类型的通知，采用 AspectJ[22] 的处理方法，执行顺序如下：around->before->after fault->after。如果同一个切入点有多个不同的通知、有同样的类型，则按定义的顺序执行这多个通知。

在本文定义的通知类型中，around 表示替换原有的转换活动，用通知的动作代替转换活动。在本文中，引入一个特殊的活动 proceed，它被应用于 around 类型的通知中，表示原有活动继续执行；或者在同一个切入点上定义多个 around 时，将执行下一个 around 通知，这样就构成一个 around 通知链（around advice chain[19,22]）。

BPEL 语言本身提供的表达能力是有限的，为了完成会话恢复，要重新执行原有执行过的活动。本文定义 replayActivity 活动，表示对活动重新执行。

replayActivity(S)：该活动将重复活动 S（用 XPath 表达式表示）的执行。如〈replayActivity activity=" //invoke[@pt='bank' @op='receive']"〉表示重新执行该服务调用。

4 基于状态方面的服务动态替换

在复合服务执行过程中，如果伙伴服务由于网络等原因而失效，用另一个功能等价的伙伴服务替换原有的服务，从而使复合服务具有错误处理能力，提高复合服务的可用性、可靠性，同时也能灵活应对伙伴服务的演化。

在 BPEL 流程定义时，将定义的伙伴服务都视为抽象服务，它们没有指定具体的第三方伙伴服务。针对 BPEL 流程中定义的抽象伙伴服务，对每一个具体的服务定义服务方面，该方面描述具体的伙伴服务执行的细节，比如服务接口适配、服务协议描述、服务调用地址等。利用状态方面，截取流程对抽象服务的调用，选择一个方面定义的具体服务完成服务调用，如图 3 所示，对一个抽象服务，定义 3 个具体的服务调用方面。利用状态方面执行机制，当一个具体的伙伴服务失效后，将由另一个具体的伙伴服务方面替换原有服务，完成同样的服务功能。下面利用状态方面实现有状态和无状态的服务替换。

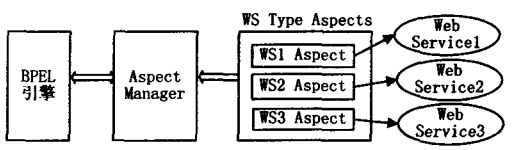

图 3 服务替换的概念架构

4.1 无状态服务动态替换

如图 4 所示，用方面描述示例银行服务的 receiveSBA 的具体调用处理（这里假设 receiveSBA 是无状态的）。第 4－10 行定义了协议切入点；对于无状态服务的协议切入点来说，就是一系列单独的转换，这些转换之间没有任何关系，如 5－9 行定义的转换和 10 行定义的转换是没有关系的。

针对 receiveSBATran 协议切入点，在 13－25 行指定针对该协议切入点的通知。对该通知的代码解释如下：

全局的 faultHandlers 处理：如果该服务方面不能处理调用，则通过调用 proceed 操作，将服务调用处理转交给另一个服务方面来完成；

对抽象服务的消息进行适配操作，如在 20 行的 assign 等，将抽象服务消息转化为具体的伙伴服务要求的消息格式；

调用具体服务的操作，这里也可以通过聚合几个服务的功能，完成抽象服务要求的功能；在图 4 的示例中，伙伴服务 backupbank 将取代原有抽象服务；

将具体伙伴服务调用的结果做适配操作，并设置到抽象服务的调用结果中。

```
1   <aspect name="statelessAspect">
2       <partnerlink name=" backupbank " ... />
3       <variable name="backupvariable" .../>
4       <protocolpointcuts>
5           <transition name="receiveSBATran">
6               <pointcut>
7                   //invoke[@pt="bank" @op="receiveSBA"]
8               </pointcut>
9           </transition>
10          <transition name="tran2">...</transition>
11      </protocolpointcuts>
12      <advices>
13          <advice transition="receiveSBATran" type="around">
14              <scope>
15                  <faultHandlers>
16                      <catchAll>
17                          <proceed />
18                      </catchAll>
19                  <faultHandlers>
20                  <assign> ... </assign>
21                  <invoke partnerlink="backupbank"
22                      operation="receiveShopBA" />
23                  <assign>...</assign>
24              </scope>
25          </advice>
26      </advices>
27  </aspect>
```

图 4　无状态服务替换方面

对于一个抽象的伙伴服务，如果部署多个具体的伙伴服务方面，则这些方面构成一个 around 通知链。当具体服务的方面(WS1 Aspect)由于失效不能使用后，另外一个具体服务方面(WS2 Aspect)接着完成抽象服务的功能，从而完成服务的动态替换。

4.2　有状态服务动态替换

无状态服务替换是将对伙伴服务的调用透明地转换为对另外一个伙伴服务的调用，这就不能保证将几个请求定位到同一个特定的伙伴服务上。如果对一个抽象服务的调用没有遵循制定的交互协议，也没有办法检测出来。在上面给出的网上银行转帐示例中，receiveCBA 和 confirmPayment 都必须按协议要求定位到同一个伙伴服务上。利用面向 BPEL 的状态方面扩展，保证服务调用定位到同一个服务上。

当复合服务调用一个有状态的伙伴服务时，以后复合服务对该有状态服务的请求都要定位到这个特定的服务实例上。针对每一个复合服务实例，将有一个特定的方面实例，管理流程会话信息，并由多个具体服务状态方面，处理实际的服务调用。

4.2.1　会话信息处理状态方面

为了记录有状态伙伴服务交互的过程，并在有状态伙伴服务失效时，将原有有状态服务交互过程的会话信息传播到新的第三方服务上，定义一个针对抽象服务的状态方面，记录抽象服务的会话协议，并记录抽象服务会话的内容和交互过程。当具体的有状态服务交互失败后，这个有状态方面将会话重新传播到一个新的第三方伙伴服务上，并在新的第三方服务上完成抽象服务流程，从而实现有状态服务的动态切换，提高复合服务执行的可靠性。

在图 5 中描述了银行转帐抽象服务对应的状态会话处理方面。第 6－29 行定义网上银行有状态服务的协议，从描述中看出网上银行转帐服务的 3 个操作是顺序执行的，receiveSBA->receiveCBA->confirmPayment。针对协议切入点，32－37 行描述了在协议切入点的处理。在这里，33－34 行将传入的消息备份；35 行表示将由具体服务方面来执行服务调用操作；36 行对结果备份，并将协议执行历史保存。38－41 行描述服务调用操作错误时的恢复方法，示例中如果当前服务在调用 receiveCBA 时失效，则服务恢复执行调用了两个 replayActivity 活动，在另一个具体服务上调用 receiveSBA 和 receiveCBA，实现会话信息恢复。第 43－45 行则说明，如果当前方面不能处理协议动作，则由下一个服务方面进行处理。这样就可以保证服务在特定服务方面上有效执行。

```
1   <aspect name="BankProtocolAspect">
2       <partnerlink name="backupbank" ... />
3       <variable name="sbamessage" .../>
4       <protocolpointcuts>
5           <complement />
6           <transition name="tran1">
7               <pointcut>
8                   //invoke[ @pt="bank"
9                       @op="receiveSBA"]
10              </pointcut>
11              <destinationTransitions>
12                  <transition ref="tran2" />
13              </destinationTransitions>
14          </transition>
15          <transition name="tran2">
16              <pointcut>
17                  //invoke [@pt="bank"
18                      @op="receiveCBA"
]9              </pointcut>
20              <destinationTransitions>
21                  <transition ref="tran3" />
22              </destinationTransitions>
23          </transition>
24          <transition name="tran3">
25              <pointcut>
26                  //invoke [@pt="bank"
27                      @op="confirmPayment"]
28              </pointcut>
29          </transition>
30      </protocolpointcuts>
31      <advices>
32          <advice transition="tran1" type="around">
33              <assign>...
34                  <to variable="sbamessage"></assign>
35              <proceed />
36              <assign> ... <assign/>
37          </advice>
38          <advice transition="tran2" type="after fault">
39              <replayActivity activity="//invoke SBA" />
40              <replayActivity activity="//invoke CBA" />
41          </advice>
42          ...
43          <complement type="around">
44              <proceed />
45          </complement>
46      </advices>
47  </aspect>
```

图 5　会话保存状态方面

4.2.2　具体服务状态方面

对于有状态的伙伴服务，定义特定的状态方面描述对该具体服务的调用处理。

具体服务状态方面与会话信息状态方面采用的方法是类似的，只是每一个转换切入点的通知采用无状态伙伴服务中类似的机制，调用具体的服务，并在服务调用过程中做服务适配。

针对每一个有状态的伙伴服务，定义一个会话信息处理状态方面和若干具体服务状态方面，从而实现具体服务的方面的动态替换和错误恢复。在复合服务运行时，通过添加和删除状态方面，实现伙伴服务的添加和删除，从而实现复合服务伙伴服务的动态绑定和管理。比如开发一个特殊的具体服务状态方面，它查找服务注册库中的服务并在服务方面中调用，从而实现服务运行时的动态查找和绑定。

对于原有出错的伙伴服务，一般需要对它进行补偿或者事务回滚，已经有关于事务处理相关的规范，如 WSTF，WS-CAF 等。这些不在本文的讨论范围内，不过用面向 BPEL 的状态方面也很容易完成这些工作。

5　原型

OnceBPEL 是由中科院软件所软件工程技术中心设计开发的业务流程执行环境，它为基于 WS-BPEL 规范业务流程

的开发、部署、运行和管理提供了集成化的支撑平台。OnceBPEL 为构建企业环境下的业务流程提供了完整的基础设施，能较好地满足企业应用快速地响应业务变化、集成异构系统和易于维护的需求。

我们的基于方面的动态服务替换原型是基于 OnceBPEL 系统的，包括方面部署和运行时方面扩展，如图 6 所示。

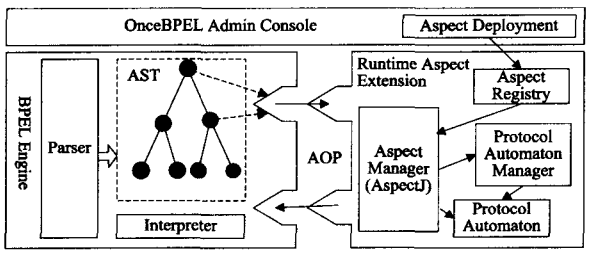

图 6　状态方面实现结构图

BPEL 引擎(BPEL Engine)

BPEL 引擎将 BPEL 文档解析为抽象语法树(AST)，并解释该语法树来执行复合服务。语法树支持基于 DOM 的 XPath 节点选择，在 BPEL 引擎执行过程中，根据活动执行信息，激活与该流程相关的方面。

方面部署(Aspect Deployment)

方面部署工具作为 OnceBPEL 管理控制台的一部分，针对特定的流程进行方面的部署和反部署。利用方面部署，实现运行时流程候选伙伴服务的添加和删除。在 OnceBPEL 系统中，用一个 Web 应用实现方面部署管理。

运行时方面扩展(Runtime Aspect Extension)

运行时方面扩展使得流程在执行过程中，监视特定的协议切入点，驱动服务协议的执行，并将通知动态编织到 BPEL 流程中，这种机制也可以在其他 BPEL 引擎上实现。在 OnceBPEL 系统中，运行时动态方面扩展本身也作为方面编织在 OnceBPEL 核心代码中。

Aspect Registry 保存部署的方面信息。当状态方面部署时，在 BPEL 抽象语法树上计算切入点表达式，返回匹配切入点表达式的一系列结点，并把这些值存入到内部数据结构中。

Aspect Manager 管理被编织到流程中的方面，当它被编织到流程引擎中后，它可以访问流程的结构和执行状态，并负责管理协议自动机的执行。

Protocol Automaton Manager 管理所有部署的协议状态机实例，针对每一个流程实例，建立一个协议状态机实例。协议状态机实例的信息存储在协议状态机(Protocol Automaton)中。

当一个 BPEL 活动执行时，Aspect Manager 查找到相应的切入点，执行该切入点的通知，同时驱动协议状态机的执行。在协议状态机执行过程中，协议切入点相关的通知将被执行。

在 OnceBPEL 系统中，用户部署服务替换功能的方面，复合服务运行期间就会对执行过程以及协议进行检查，并执行服务的动态替换，实现功能等价伙伴服务的适配与无状态和有状态服务的动态替换，提高复合服务执行过程的容错性和可靠性。

6　相关工作

关于面向 BPEL 方面扩展，已经有不少的研究，如 AO4BPEL[4]，Padus[12]等，但是对状态方面的研究却很有限；面向 BPEL 的状态扩展，如，使用正则表达式表示切入点[10]，并采用 Padus 静态编织方法，不能灵活对协议切入点做灵活的处理；文献[21]利用状态方面扩展，实现流程监控，但对伙伴服务执行协议与恢复没有关注。本文提出面向 BPEL 的状态方面扩展，可以更好处理伙伴服务交互协议信息，灵活处理复合服务执行状态。

对于服务的动态替换，已经有不少相关的工作。在文献[7]中，Oliver Moser 等人利用 AOP 的方法，截取特定流程引擎实现中的服务调用操作，并在此基础上实现服务的适配和替换。这种方法利用 Java 语言本身的特性，没有采用和 BPEL 一致的适配与替换风格，不够灵活。同时采用消息层面的适配和服务替换，不能处理有状态伙伴服务的替换方法。

文献[6,8]利用有状态的 JAsCo，从调用有状态服务客户端的角度，针对生成的 Java 客户端，实现对有状态服务的动态替换，这里的服务的状态性是用基层的状态协议(比如 SOAP-Conversation[3])来维持的。我们采用的方法和它的方法有些类似，但是我们从流程协作和协议的角度实现有状态服务的动态替换，用面向 BPEL 的状态方面扩展，实现流程的伙伴服务的动态替换。

文献[15,16]通过静态和动态代理的方法，透明地将服务的自治行为引入到 BPEL 流程中，当伙伴服务失效时，利用代理完成服务替换，代理是静态编织到复合服务中的。文献[20]也是用代理方法，复合服务直接与代理交互，实现服务的动态绑定，而代理负责服务调用错误恢复。当服务是有状态时，它们都不能有效处理。而且通过代理实现服务替换，增加了复合服务与代理之间的通讯代价，该面向方面的方法，去除复合服务与代理之间通信的代价，能实现高效的服务动态切换。

结束语　本文提出了面向 BPEL 状态方面扩展，显式定义有状态服务交互协议，规范复合服务与有状态服务的交互；并利用该扩展机制实现无状态和有状态伙伴服务的动态替换、错误恢复，增强了复合服务执行的动态性、容错性、可演变性，复合服务具有更好的鲁棒性。

下一步的工作，将深入分析基于业务规则的服务替换机制，以及如何根据服务的 QoS 要求，动态选择最优的服务，实现流程执行的优化和容错性。为基于状态方面的服务替换方法提供开发工具，提供更高层次的抽象，使针对 BPEL 服务替换的方面开发更加简便。

参 考 文 献

[1] 杨芙清. 软件工程技术发展思索[J]. 软件学报，2005，16(1)：1-7

[2] OASIS. Web services business process execution language(wsbpel) v2. 0[OL]. http：// docs. oasis-open. org /wsbpel/2. 0/wsbpel-v2. 0. pdf

[3] Bau D，Orchard D. SOAP Conversation Protocol. Whitepaper specification 1. 0 BEA，June 2002

[4] Charfi A，Mezini M. Ao4Bpel：An Aspect-oriented extension to bpel[J]. World Wide Web，2007，10(3)：309-344

[5] Donence R，Fradet P，Sudholt M. Composition，Reuse and Interaction Analysis of Stateful Aspects[C]// Proceedings of the 3th International Conference on Aspect-Oriented Software Development. Lancaster，UK，March 2004

[6] Wim V，Davy S，Cibr'an M A，et al. Stateful aspects in JAsCo

[C]// Workshop on Software Composition at ETAPS. 2005

[7] Moser O, Rosenberg F, Dustdar S. Non-intrusive monitoring and service adaptation for WS-BPEL[C]// WWW'08 Proceedings. 2008

[8] Verheecke B, Jonckers V. Stateful aspects for conversational messaging with stateful web services[C]// Proceedings. International Conference on Next Generation Web Services Practices (NWeSP05). Seoul, Korea, August 2005

[9] Kiczales G, Lamping J, Mendhekar A, et al. Aspect - oriented programming[C]// Proceeding of the European Conference on Object-Oriented Programming. Springer-Verlag, 1997

[10] Braem M, Gheysels D. History - based aspect weaving for WS - BPEL using Padus[C]// Fifth European Conference on Web Services(ECOWS). Germany, November 2007

[11] Papazoglou M P, Traverso P, Dustdar S, et al. Service-Oriented Computing: State of the Art and Research Challenges[J]. IEEE Computer, 2007, 4(11): 38-45

[12] Braem M, Verlaenen K, Joncheere N, et al. Isolating process-level concerns using Padus[C]// Proceedings of the 4th International Conference on Business Process Management (BPM 2006). Vienna, Austria: Springer-Verlag, September 2006

[13] Allan C, Avgustinov P, Christensen A S, et al. Adding trace matching with free variables to AspectJ[C]// OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on object oriented programming systems languages and applications. ACM Press, 2005: 345- 364

[14] Hopcroft J E, Motwani R, Ullman J D. Introduction to Automata Theory. Addison Wesley, 2st edition, 2001

[15] Ezenwoye O, Sadjadi S. Enabling robustness in BPEL processes [C]// Proceedings of the 8th International Conference on Enterprise Information Systems(ICEIS-06). 2006

[16] Ezenwoye O, Sadjadi S. RobustBPEL - 2: Transparent autonomization in aggregate web services using dynamic proxies[R]. FIU-SCIS-2006-06-01. School of Computing and Information Sciences, Florida International University, 11200 SW 8th St., Miami, FL 33199, June 2006

[17] Benatallah B, Casasti F, Toumani F. Web Service Conversation Modeling: A Cornerstone for E-Business Automation[J]. IEEE Internet Computing, 2004, 8(1): 46-54

[18] XML Path Language (XPath) 2. 0. [OL]. http: // www. w3. org/TR/xpath20/, 2007

[19] Pawlak R, Duchien L, Seinturier L. Compar: Ensuring safe around advice composition[C]// Proceedings of Formal Methods for Open Object-Based Distributed Systems. Athens, Greece, June 2005

[20] Penta M D, Esposito R, Villani M L, et al. WS Binder: a Framework to enable Dynamic Binding of Composite Web Services// Proceedings of the International Workshop on Service-oriented Software Engineering(SOSE'06). 2006: 74-80

[21] Wu Guoquan, Wei Jun, Huang Tao. Flexible pattern monitoring for WS-BPEL through stateful aspect extension[C]// ICWS'08. 2008

[22] Kiczales G, Hilsdale E, Hugunin J, et al. An Overview of AspectJ[C]// Proceedings of the 15th European Conference on Object-Oriented Programming. June 2001: 327-353

(上接第 75 页)

据叠加噪声和干扰设计了一组仿真数据，通过对实验数据的处理验证设计的算法。

在分析采集的泥浆压力信号基础上[8]，给出了一压力传感器对井口泥浆压力进行 40 次测量并人为地对数据叠加一定的噪声和干扰而得到的 5 组数据，如表 1 所列。

表 1 叠加噪声和干扰的实验数据

	数据 1	数据 2	数据 3	数据 4	数据 5	数据 6	数据 7	数据 8
第一组	530	531	526	532	528	536	557	535
第二组	532	526	527	532	528	533	497	531
第三组	530	534	506	535	529	538	527	526
第四组	560	530	536	537	525	532	533	525
第五组	500	533	528	531	529	526	527	535

对表 1 中的每一组数据按照 8 点平均值处理，结果分别为 534mV，526mV，528mV，535mV，526mV。

对表 1 中每一组数据按照本文算法动量进行融合处理，结果分别为 531 mV，530 mV，530 mV，531 mV，529 mV。

由处理结果可知，此时压力传感器采集的压力对应数据真值为 530mV。本文算法对几组数据结果比用 8 点平均处理效果好，误差和方差都比较小，提高了单传感器的测量准确度，有效地抑制了干扰和噪声。

结束语 本文在统计方法理论的基础上讨论信号间的相互支持度含义和计算方法，提出了关系矩阵的模糊化确定方法，使得数据间的相互支持关系不再绝对化。并利用每一个传感器的综合支持度进行数据融合，不仅充分利用所有数据，而且使最终融合效果具有更好的稳定性，保证信息融合具有更高的抗干扰性。通过对泥浆压力信号采集的数据特点进行分析，设计 5 组实验数据，验证了本文提出的融合算法的可靠性，并对随钻测井现场采集的数据进行处理，取得了良好的处理结果。

参 考 文 献

[1] Henderson T C, Shilcrat E. Logical Sensor Systems[J]. Rabotics Systems, 1984, 1(2): 169-193

[2] 陈福增. 多传感器数据融合的数学方法[J]. 数学的实践与认识, 1995(2): 11-16

[3] 蒋正新，施国梁. 矩阵理论及其应用[M]. 北京：北京航天航空大学出版社，1998: 371-378

[4] 涂国平，邓利钊. 多传感器数据的统计融合方法[J]. 感器技术，2001, 28(3): 28-29

[5] 罗中良. 不确定信息的数字滤波其设计及应用[J]. 传感器技术，2002, 21(5): 24-26

[6] 王威，周军红，王润生. 多传感器数据融合的一种方法[J]. 传感器技术，2003, 22(9): 39-40

[7] Klein L A. Sensor and data Fusion Concepts and Application [M]. 北京：北京理工大学出版社，2004

[8] 张恒，李安宗，李传伟. 钻井液波信号处理方法比较[J]. 石油钻采工艺，2007, 29(2): 84-86