基于工作流模式的OWL-S过程模型分析及其应用*)

雷丽晖 段振华
(西安电子科技大学计算理论与技术研究所 西安 710071)

摘 要 确定OWL-S过程模型描述Web服务之间交互协议的能力和缺陷,为组合Web服务执行提供有效支持。将组合Web服务视为基于Web服务的工作流,利用OWL-S过程模型的控制结构给出工作流模式的实现方案,分析出OWL-S描述Web服务之间交互协议的能力及缺陷;在此基础上,设计并实现基于OWL-S的组合Web服务执行工具,通过验证Web服务调用的有效性及Web服务之间交互协议的有效性,提高组合Web服务执行的健壮性。

关键词 组合Web服务,OWL-S,工作流模式

Analysis of OWL-S Process Model Based on Workflow Patterns and its Application

LEI Li-Hui DUAN Zhen-Hua
(Institute of Computing Theory & Technology, Xidian University, Xi'an 710071)

Abstract Ascertaining the capabilities and the limitations of the OWL-S process model for describing the interaction protocol between Web services is important for the effective performance of composite Web services. A composite Web service can be regarded as a service-based workflow. The implementations of workflow patterns with the control constructs of the OWL-S process model are presented in this paper. As a result, the capabilities and the limitations of the OWL-S process model for describing the interaction protocols can be clarified. A prototype tool executing composite Web services which are described by OWL-S is designed and implemented. This tool validates the interaction protocol between services and the compatibility of parameters. Therefore, it makes the performance of composite Web services more robust.

Keywords Composite Web services, OWL-S, Workflow patterns

Web服务适于构建跨平台的松耦合分布式应用,是动态电子商务实现方案的关键技术。越来越多的企业在尝试使用Web服务组合实现企业应用集成,即用单个Web服务封装独立的业务功能,通过Web服务之间的协作来完成复杂的业务功能。这种Web服务之间的协作依赖于Web服务之间的交互协议,即Web服务之间消息传递的顺序,由组合Web服务定制。现有许多语言可用来描述Web服务之间的交互协议,如BPSS[1]、BPEL4WS[2]、WSCI[3]和OWL-S[4]。但这些语言没有明确地定义形式化执行语义,使得组合Web服务执行比较困难。解决问题的一种方法是全面分析这些语言对Web服务之间交互协议的描述,确定这些语言的表达能力和缺陷;以此为基础,设计实现组合Web服务执行工具。

本文分析了OWL-S对Web服务之间交互协议的描述。OWL-S是描述Web服务属性和功能的本体,目的是使Web服务发现、组合和调用自动化。OWL-S过程模型(Process Model)是OWL-S的三个组成部分[4]之一,定义了Web服务之间的交互协议。OWL-S对Web服务之间交互协议的描述就是OWL-S过程模型对Web服务之间交互协议的描述。由于OWL-S过程模型可视为过程组建的工作流,因此本文使用工作流模式(Workflow Patterns, WPs)[6]深入系统地分析OWL-S过程模型对Web服务之间交互协议的描述。

1 OWL-S过程模型

OWL-S过程模型定义了一组过程及其执行顺序。过程的定义包括三个部分[4]:输入、执行条件(preconditions)和结果(results);结果又包含输出和执行效果(effects)两部分。过程的定义隐含描述了过程执行:如果过程执行条件可满足,过程根据输入和当时运行环境的状态产生输出和执行效果,并且从执行前状态转移到执行后状态。从功能角度来看,过程可以分为两类:原子过程和组合过程。原子过程描述了Web服务之间一次不可分割的交互;组合过程描述了Web服务之间一系列的交互。Web服务之间的交互协议由组合过程来定制。

组合过程是用控制结构(control constructs)将原子过程分层组合而成的。OWL-S过程模型给出九种控制结构,用来定义组合过程的各个组成部分(原子过程或其它组合过程)的执行顺序。其中Iterate是个抽象概念,不能在OWL-S过程模型中实例化[4],所以不考虑。其余控制结构执行语义的直观解释如下:假设一个组合过程由以下控制结构组成,

- *Sequence*:其组成部分是一组顺序执行的过程。
- *Split*:其组成部分是一组将要并发执行的过程;组合过程在激活其组成部分之时结束。用*Split*组合在一起的这些并发执行的过程,有着相同的前驱(可为空)。
- *Split*+*Join*:其组成部分是一组并发执行的过程;组合过程在其所有组成部分执行结束之时结束。用*Split*+*Join*组合在一起的这些并发执行的过程,有着相同的后继(可为空)。
- *Anyorder*:其组成部分是一组以任意顺序执行的过程,而且在组合过程的执行过程中,不允许组成部分的执行有重叠,即原子过程不能并行执行,组合过程不能交错执行。

*)基金项目:国家自然科学基金(No. 60373103),国家自然科学基金重点项目(No. 60433010)资助。雷丽晖 博士生,主要研究领域为网络计算,语义Web;段振华 博士生导师,CCF会员,主要研究领域为网络计算和时序逻辑程序验证。

• *Choice*:其组成部分是一组过程,组合过程从中选择一个过程执行;选择条件没有明确定义,选择的依据是组成部分的特性(例如:过程的状态或者功能[7])。

• *If-then-else*:其组成部分是两个过程,组合过程根据明确定义的条件选择其中之一执行。

• *Repeat-while* 和 *Repeat-until*:其组成部分只有一个过程;前者表示当外界条件可满足时重复执行这个过程;后者表示重复执行这个过程直到外界条件可满足。

此外,OWL-S 过程模型还定义了 Web 服务交互过程中的数据流,即过程之间输入和输出的关系。因本文研究的是 Web 服务之间的交互协议,故暂不讨论服务交互过程中的数据流。

2 工作流模式的表示

活动(Activity)是组成工作流的原子单位。工作流描述了一组活动及其执行顺序。OWL-S 过程模型中的原子过程可视为工作流中的活动;组合过程由原子过程分层组合而成,描述了原子过程的执行顺序,可视为原子过程组建的工作流。工作流模式描述了工作流中反复出现的二十种典型的活动执行顺序,已用于工作流描述语言评估[5]。本节用 OWL-S 过程模型的九种控制结构给出这些模式的实现方案。为便于描述,实现方案用简化的 OWL-S 语法表示。

WP1:顺序(Sequence)

顺序模式描述了工作流中各个活动按顺序依次执行。控制结构 Sequence 可直接给出该模式的实现方案。如图 1 所示,过程 A、B 和 C 顺序执行。

```
<Sequence>
  <Process> A</Process>
  <Process> B</Process>
  <Process> C</Process>
</Sequence>
```

图 1 顺序模式

WP2:平行拆分(Parallel split)

平行拆分模式记为 *And-split*,描述了工作流中某个活动结束时,有一组活动(两个或者两个以上)被激活且并发执行。控制结构 *Split* 可以直接给出该模式的实现方案。如图 2 所示,假设过程 A 结束时 B 和 C 被激活且并发执行。显然,B 和 C 有同一前驱 A。

```
<Sequence>
<Process> A</Process>
<Split><Process>B</Process>
       <Process>C</Process>
 </Split>
</Sequence>
```

图 2 平行拆分模式

WP3:同步(Synchronization)

同步模式记为 *And-join*,描述了工作流中一组并发执行的活动通过阻碍同步,即并发执行的活动全部结束时,其后继活动才能被激活执行。控制结构 *Split*+*Join* 可以直接给出该模式的实现方案。如图 3 所示,假设过程 A 和 B 并发执行,两者都结束时 C 被激活执行。显然,A 和 B 有共同的后继 C。

```
<Sequence>
 <Split-Join><Process>A</Process>
          <Process>B</Process>
</ Split-Join >
<Process>C</Process>
</ Sequence >
```

图 3 同步模式

图 4 给出了 *And-split* 连接 *And-join* 的实现方案,过程 B、C 和 D 一起激活,并发执行且阻碍同步。如果将虚线部分删除,那么图 4 描述了部分并发过程同步[1],即 B、C 和 D 一起激活且并发执行,但只有 B 和 D 阻碍同步。即只要 B 和 D 执行结束,就可激活执行过程 E。

```
<Sequence>
  <Process>A</Process>
  <Split>
     <Process>B</Process>
     <Process>C</Process>
     <Process>D</Process>
  </Split><Split-Join>
     <Process>B</Process>
     <Process>C</Process>
     <Process>D</Process>
  </ Split-Join >
  <Process>E</Process>
</Sequence>
```

图 4 平行拆分模式连接同步模式

```
<Sequence>
   <Process>A</Process>
   <If-then-else>
      <If-condition> f1</If-condition>
      <then><Process>B</Process></then>
      <else>< If-then-else>
            <If-condition> f2 </If-condition>
            <then><Process>C</Process></then>
            <else><Process>D</Process></else>
      </if-then-else></else>
   </ If-then-else>
    <Process>E</Process>
</Sequence>
```

图 5 独占式选择模式连接简单合并模式

WP4:独占式选择(Exclusive Choice)

独占式选择模式记为 *Xor-split*,描述了工作流中某个活动结束后,有一组可选的后继活动;在实际的工作流运行过程中,选择条件是互斥的,只能选择一个后继活动执行。假设过程 A 结束后可执行 B、C 或 D;三者的选择条件互斥,在工作流实例中只有一个被选中执行。

WP5:简单合并(Simple Merge)

简单合并模式记为 *Xor-join*,描述了工作流中一组可选活动有着相同的后继活动,且在实际的工作流运行过程中,这组活动中只能有一个活动处于激活状态,该活动结束时后继活动被激活执行。假设过程 B、C 和 D 都有可能处于激活状态,E 是其共有的后继;在工作流实例中 B、C 和 D 只能有一个处于激活状态,并在执行结束时激活 E。

控制结构 *If-then-else* 可给出上述两种模式的实现方案。如图 5 所示,f_1 成立时执行过程 B,$\neg f_1 \wedge f_2$ 成立时执行过程 C,$\neg f_1 \wedge \neg f_2$ 成立时执行过程 D;f_1、$\neg f_1 \wedge f_2$ 和 $\neg f_1 \wedge \neg f_2$ 是互斥的。如果删除虚线部分,图 5 给出 *Xor-split* 的

[1] *OWL-S*1.1 定义了部分同步(*partial synchronization*),即在一组并发执行的过程中只有一部分过程同步。

实现方案：过程 A 执行结束后，根据条件选择 B、C 或 D 中一个过程执行。若加上虚线部分，则产生 *Xor-join* 的实现方案：B、C 或 D 中只有一个是激活状态，该过程结束时执行 E。从整体上看，图 5 给出 *Xor-split* 连接 *Xor-join* 的实现方案：A 执行结束后，选择 B、C 或 D 中的一个执行，该过程结束时激活执行 E。

WP6：多选(Multi-Choice)

多选模式记为 *Or-split*，描述了工作流中某个活动结束后有一组可选的后继活动；在实际的工作流运行过程中，允许选择其中若干(一个或者多个)活动执行。选择条件不一定互斥；如果选择了多个活动，那么这些活动并发执行。

WP7：平行合并(Synchronizing Merge)

平行合并模式记为 *Or-join*，描述了工作流中一组可选活动有相同的后继活动；这组活动中若只有一个活动处于激活状态，该活动结束时后继活动被激活执行；若有多个活动并发执行，并发执行的活动全部结束时后继活动被激活执行。

OWL-S 的控制结构不能直接给出上述两种模式的实现方案。*Split* 和 *If-then-else* 的组合可以给出 WP6 的两种实现方案；*Split*＋*join* 和 *If-then-else* 的组合可以给出 WP7 的两种实现方案。

• WP6 两种实现方案：先用 *If-then-else* 为 B、C 和 D 分别定义选择条件 f_1、f_2 和 f_3，使之成为组合过程；再用 *Split* 将这些组合过程连接，使其并发执行，如图 6；第二种方案是先用 *Split* 将需要并发执行的过程组合，再用 *If-then-else* 将这些组合过程连接。

• WP7 两种实现方案：先用 *If-then-else* 为 B、C 和 D 分别定义选择条件 f_1、f_2 和 f_3，使之成为组合过程，再用 *Split*＋*join* 将这些组合过程连接，等待并发过程结束，如图 7；第二种方案是先用 *Split*＋*join* 将需要同步的过程组合，再用 *If-then-else* 将这些组合过程连接。

实现 *Or-split* 连接 *Or-split*，WP6 的两种实现方案和 WP7 的两种实现方案可以交错搭配。WP6 根据选择条件判定哪些过程需要并发执行；WP7 根据选择条件判定哪些过程需要同步。

```
<Sequence>
  <Process>A</Process>
  <Split>
   <If-then-else><If-condition> f1 </If-condition>
      <then><Process>B</Process></then></If-then-else>
   <If-then-else><If-condition> f2 </If-condition>
      <then><Process>C</Process></then></If-then-else>
   <If-then-else><If-condition> f3 </If-condition>
      <then><Process>D</Process></then></if-then-else>
  </Split>
</Sequence>
```

图 6　多选模式

```
<Sequence>
  <Split-Join>
   <If-then-else><If-condition> f1 </If-condition>
      <then><Process>B</Process></then></If-then-else>
   <If-then-else><If-condition> f2 </If-condition>
      <then><Process>C</Process></then></If-then-else>
   <If-then-else><If-condition> f3 </If-condition>
      <then><Process>D</Process></then></if-then-else>
  </Split-Join>
  <Process>E</Process>
</Sequence>
```

图 7　平行合并模式

WP8：多合并(Multi-Merge)

多合并模式描述了工作流中一组可选活动有着相同的后继活动；若这组活动中只有一个活动处于激活状态，那么该活动执行结束时，其后继活动被激活执行；若这组活动中有多个活动并发执行，那么这些活动不需要同步，每个活动执行结束时，其后继活动都要激活执行一次。假设过程 A 结束后，如果 B 或 C 中只有一个处于激活状态，那么该过程结束时 D 被激活；如果 B 和 C 中有多个过程并发执行，每个过程结束时都要激活 D。由以上的描述不难看出，D 总在处于激活状态的过程结束后执行。所以，在这组可选过程的每一个过程后，都可直接连接 D。多合并模式不能直接给出实现方案，但可给出其等价形式的实现方案，如图 8 所示。

```
<Sequence><Process>A</Process>
 <Split> <If-then-else><If-condition>f1</If-condition>
          <then><Sequence><Process>B</Process>
                <Process>D</Process></Sequence></then></If-then-else>
      <If-then-else><If-condition>f2</If-condition>
          <then><Sequence><Process>C</Process>
                <Process>D</Process></Sequence></then></If-then-else>
 </Split></Sequence>
```

图 8　多合并模式

WP9：鉴别器(Discriminator)

鉴别器模式描述了工作流中一组可选活动有着相同的后继活动；若这组活动中只有一个活动处于激活状态，那么该活动执行结束时，其后继活动被激活执行；若这组活动中有多个活动并发执行，只要其中的一个活动结束，其后继活动就可激活执行；剩余活动虽然继续执行，但执行结果却被忽略。由于无法表示哪个活动先结束，因此不能给出该模式的实现方案。

WP10：任意循环(Arbitrary Cycles)

任意循环模式描述了工作流中若干活动反复执行，类似 GOTO 语句。假设过程 A、B、C、D 和 E 可顺序执行；也可在 C 结束时跳转执行 B；也可在 D 结束时跳转执行 C。该模式不能直接给出实现方案，须将任意循环转换为等价结构化循环[8]，再给出等价循环的实现方案。

WP11：隐式终止(Implicit Termination)

隐式终止模式描述了工作流的一种终止方式，即某些活动执行结束后，没有别的活动在执行且没有可被激活的活动时(排除死锁的情况)，工作流结束。隐式终止模式没有明确的定义某个活动结束后工作流终止。平行拆分模式提供了该模式实现的基础。如图 2，过程 B 或 C，哪个结束后工作流终止，一定是随机的。

WP12：非同步的多实例(Multiple Instances Without Synchronization)

非同步的多实例模式描述了在同一个工作流实例中，某个活动可以有多个实例在并发执行，且在后继活动执行之前无需同步。该模式实现的前提是：这个活动实现的机制允许产生多个实例来处理多个调用请求，且调用者不必等待调用结果的返回。在这个前提下，*Repeat-while* 可给出该模式的实现方案，如图 9。假设 f_1 成立时过程 A(满足上述前提)被

反复调用。

```
<Repeat-while>
  <whileCondition> f1
  </whileCondition>
  <whileProcess>
    <Process>A</Process>
  </whileProcess>
</Repeat-while >
```

图 9　非同步的多实例模式

WP13-15：同步的多实例(Multiple Instances with Synchronization)

同步的多实例模式描述了在同一个工作流实例中，某个活动可以有多个实例在并发执行，且在后继活动执行之前需要同步。由于需要同步，实例个数需要考虑。

• WP13 描述了工作流设计期间，实例个数可确定的同步多实例模式。该模式可用 *Split* 和 *Split*＋*join* 实现。假设工作流中需要 N 个过程 A 的实例，实现方法是：先用 *Split* 将 N 个 A 连接使之并发执行，再用 *Split*＋*join* 将这 N 个并发执行的 A 连接使之同步。

• WP14 描述了工作流运行期间，在实例产生之前，实例个数可以确定的同步多实例模式。WP15 描述了工作流运行期间，在实例产生之前，实例个数无法确定的同步多实例模式。两者的区别在于：对于后者而言，只要还有该活动的实例在运行，新实例就可以被创建。OWL-S 过程模型还没有考虑到过程执行这一层，所以不能给出这两种模式的实现方案。

WP16：延迟选择(Deferred Choice)

延迟选择模式描述了工作流中一组可选的活动，只能有一个活动被选中执行。该模式的选择条件没有明确定义，且选择可以延迟到选择条件有效时执行。例如：收到订单后有两种运输工具可送货，选择取决于相关资源可用性(选择可推迟到至少其中一种运输工具可用为止)。*Choice* 可直接给出延迟选择模式的实现方案，如图 10，过程 A、B 和 C 中选择一个执行。

WP17：交替平行路由(Interleaved Parallel Routing)

交替平行路由模式描述了工作流中的一组活动以任意的顺序执行；每个活动都只执行一次，执行顺序在运行时决定，且任意一个时刻都不会有两个或两个以上的活动在执行。控制结构 *Anyorder* 可直接给出该模式的实现方案，如图 11，过程 A、B 和 C 可以任意顺序执行。

```
<Choice>
  <Process>A</Process>
  <Process>B</Process>
  <Process>C</Process>
</Choice >
```

图 10　延迟选择模式

```
<Anyorder>
  <Process>A</Process>
  <Process>B</Process>
  <Process>C</Process>
</Anyorder>
```

图 11　交替平行路由模式

```
<Sequence>
 <Process>B</Process>
 <Repeat-while><whileCondition> f1</whileCondition>
   <whileProcess><Choice> <Process>A</Process>
        <Process>C</Process> // 修改f1使其为假
  </Choice> </whileProcess>
 </Repeat-while >
</Sequence>
```

图 12　里程碑模式

WP18：里程碑(Milestone)

里程碑模式描述了工作流中某个活动能否执行，取决于某个特定的状态。假设活动 A 只有在 B 结束之后，C 开始之前，才能被激活运行，即 B 和 C 之间的状态决定了 A 能否被激活。*Repeat-while* 和 *Choice* 组合可以给出该模式的实现方案。如图 12 所示，组合过程根据 C 的状态选择：只要 C 没有激活，就选择执行 A。f_1 可满足时重复执行选择过程；当 C 被激活使 f_1 不再满足时，循环过程结束。

WP19-20：取消活动/取消实例(Cancel Activity and Cancel Case)

取消活动模式描述了工作流中的一个正在执行的活动被强制结束，与该活动相关的正在等待执行的活动被全部取消。而取消实例模式描述了整个工作流实例被消除的情况。由于 OWL-S 过程模型还没有考虑到过程执行这一层，因此不能给出这两个模式的实现方案。

3　表达能力分析

文[5]基于工作流模式分析了 BPEL4WS 描述 Web 服务之间交互协议的能力；两者的比较如表 1 所示。“＋”表示实现方案用控制结构直接给出；“－”表示实现方案用控制结构组合给出；“＋/－”表示某些条件满足时，可提供实现方案；“×”表示不能提供实现方案。

• OWL-S 过程模型的控制结构可直接给出 WP1-WP5 的实现方案，且控制结构组合可给出 WP6-WP8 的实现方案。就表达 WP1-WP8 而言，OWL-S 过程模型比 BPEL4WS 简单。

• OWL-S 过程模型没考虑工作流实例中，过程内部的数据及其计算，因而无法给出 WP14、WP15、WP19 和 WP20 的实现方案；而可执行的 BPEL4WS 可给出这些模式的实现方案。

• OWL-S 过程模型实现 WP12 需要满足附加条件：在工作流实例中，产生多个并发执行实例的活动，其实现机制应该允许产生多个实例来处理多个调用请求，而且调用者不必等待调用结果的返回。这样的条件在 OWL-S 过程模型中无法完全定义。

• 因为 OWL-S 过程模型没考虑过程的执行，WP17 可用 OWL-S 过程模型给出实现方案；而 BPEL4WS 却只能在限制条件[5]满足的情况下给出 WP17 的实现方案。

• 现有的 OWL-S 过程模型和 BPEL4WS 都无法实现 WP9。

由上述分析可知，OWL-S 过程模型定义了 Web 服务之间交互的所有可能顺序，但是没有定义与组合 Web 服务执行相关的技术细节。实现组合 Web 服务执行时，有较多的灵活性。

[2] 原子过程的 URL 是 Web 服务访问地址；原子过程的 IOPE 是原子过程输入、输出、执行条件和执行效果。

表 1 OWL-S 过程模型和 BPEL4WS 的对比

Workflow Patterns	OWL-S Process Model	BPEL4WS
WP1:Sequence	+	+
WP2:Parallel Split	+	+
WP3:Synchronization	+	+
WP4:Exclusive Choice	+	+
WP5:Simple Merge	+	+
WP6:Multi-Choice	−	+
WP7:Synchronizing Merge	−	+
WP8:Multi-Merge	−	−
WP9:Discriminator	×	×
WP10:Arbitrary Cycles	−	−
WP11:Implicit Termination	+	+
WP12:MI without Synchronization	+/−	+
WP13:MI with a Priori Design Time Knowledge	−	+
WP14:MI with a Priori Runtime Knowledge	×	−
WP15:MI without a Priori Runtime Knowledge	×	−
WP16:Deferred Choice	+	+
WP17:Interleaved Parallel Routing	+	+/−
WP18:Milestone	−	−
WP19:Cancel Activity	×	+
WP20:Cancel Case	×	+

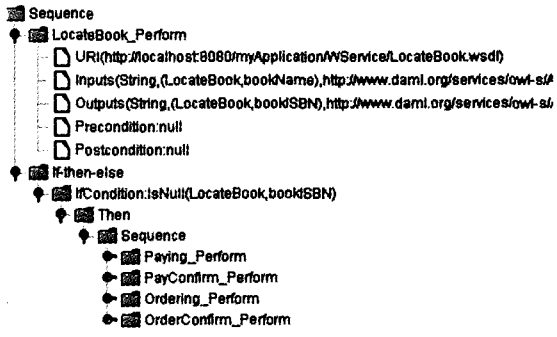
4 OWL-S 执行工具

本文设计了一个基于 OWL-S 的组合 Web 服务执行工具,实现了原型系统。系统主要由三部分组成:解析 OWL-S 过程模型;验证 Web 服务调用的有效性以及 Web 服务之间交互协议的有效性;执行 OWL-S 描述的组合 Web 服务。

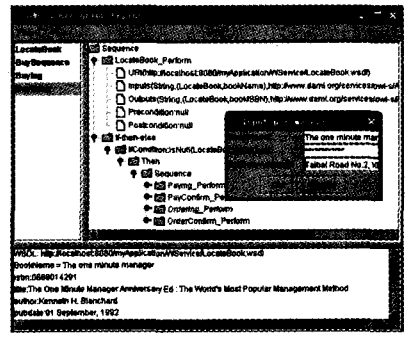
4.1 解析

解析 OWL-S 过程模型,得到一棵表达工作流的树,如图 13(a)。树的结点可分为四类:控制结构、原子过程、判断条件以及原子过程的 URL 和 IOPE[2]。如果 OWL-S 过程模型包含组合过程,那么根结点是控制结构;否则根结点是原子过程。叶子结点是原子过程的 URL 和 IOPE。原子过程的孩子结点只可能是原子过程的 URL 和 IOPE;判断条件一定是控制结构的孩子结点;原子过程可是控制结构的孩子结点,反之不可;控制结构可以是控制结构的孩子结点。

从树的结点上,可确定原子过程输入和输出的绑定,得到描述原子过程执行条件和执行效果的逻辑公式,获取访问 Web 服务的 URL。从树的结构上,还可以确定工作流执行顺序。所以,该树是验证 Web 服务调用有效性,验证 Web 服务之间交互协议的有效性,以及执行组合 Web 服务的基础。



(a) 工作流树



(b) 过程执行

图 13 原型系统运行截图

4.2 验证及执行

在执行条件可以满足的情况下,过程输入和输出的有效性决定了过程执行的有效性,即 Web 服务调用的有效性。OWL-S 过程模型描述的 Web 服务之间的交互协议必须是有效的,即能转换为可执行的工作流。因为 OWL-S 过程模型没有定义组合 Web 服务执行错误及错误的处理,这两类验证保证了组合 Web 服务执行不会出现服务调用失败和工作流逻辑错误。

验证 Web 服务调用的有效性,即验证用户输入和输出是否满足调用 Web 服务的要求。如果 Web 服务输入和输出是简单数据类型,那么判定的过程比较简单,只需证明 Web 服务输入和输出与用户输入和输出:数据类型一致且语义一致(即用户输入的语义等于或包含于 Web 服务输入的语义;并且 Web 服务输出的语义等于或包含于用户输出的语义)。如果过程输入和/或输出不是简单数据类型,那么判定过程比较复杂,除了上述证明,还要判定数据转换能否完成。例如:某查询 animal 的 Web 服务要求输入是 animal 类型,而用户提供了 bird 类型的输入,所以在服务调用之前需要进一步将 bird 类型转换为 animal 类型。转换的依据是 animal 和 bird 属性值的数据类型和语义。如果转换不成功,Web 服务也不能调用。

(下转封四)

(上接第 133 页)

验证 OWL-S 过程模型描述的 Web 服务之间交互协议的有效性,就是验证 OWL-S 过程模型能否转换为可执行的工作流。关键问题在于判定工作流发散之后(被分为若干分支),能否有效地聚合(排除不需要聚合的情况)。假设 OWL-S 过程模型的描述没有语法错误,在 OWL-S 过程模型描述中有一组过程分别包含在先后两个模式中(为便于讨论,设 *And-split* 和 *And-join* 中的过程没有选择条件;若要讨论有选择条件的情况,转换为 *Or-split* 和 *Or-join*):

• *And-split* 连接 *And-join*:该模式组合可以转换为可执行的工作流。

• *Xor-split* 连接 *Xor-join*,*Xor-split* 连接 *Or-join* 和 *Or-split* 连接 *Xor-join*:如果这组过程的选择条件互斥,且在前后两个模式中不变,那么该模式组合可以转换为可执行的工作流。

• *Or-split* 连接 *Or-join*:只要过程的选择条件不变,该模式组合就可转换为可执行工作流。

• *And-split* 连接 *Xor-join*:该模式组合不能转换为可执行的工作流。*And-split* 组合的一组过程要全部激活且并发执行,而 *Xor-join* 要求这组活动中只有一个活动处于激活状态,执行语义矛盾。事实上,*Xor-join* 决定了在创建工作流时,不会出现这样的模式组合。

• *And-split* 连接 *Or-join* 和 *Or-split* 连接 *And-join*:当 *Or-join* 中所有过程的选择条件都为真,前一个模式组合可以转换为可执行的工作流;当 *Or-split* 中所有过程的选择条件都为真,后一个模式组合可以转换为可执行的工作流。

• *Xor-split* 连接 *And-join*:该模式组合不能转换为可执行的工作流,因为 *Xor-split* 组合的一组过程只有一个被激活执行,而 *And-join* 却要等待这组过程全部结束。

图 13(b)给出系统运行时的屏幕截图。系统运行时,首先读入描述 Web 服务的 OWL 文档,解析该文档,并产生描述工作流的树;然后系统接收用户提供的 Web 服务输入;接着系统验证 Web 服务调用的有效性及组合过程模型能否转换为可执行的工作流;如果验证通过,那么执行该工作流,跟踪执行过程并显示执行结果,否则提示验证出现的问题。

4.3 未解决的问题

系统需要进一步解决组合 Web 服务执行时的错误定义及错误处理问题。虽然系统通过验证保证了组合 Web 服务执行不会出现服务调用失败及工作流逻辑错误,但不能保证执行过程中不出现其它错误,例如:Web 服务调用超时。此外,由于控制结构 *Choice* 没有明确的定义选择条件,组合 Web 服务执行时,执行者(人或软件代理)需要根据组合 Web 服务组成部分的信息或状态来选择,本系统暂时没有完全实现 *Choice* 的执行。

总结 本文分析了 OWL-S 过程模型描述 Web 服务之间交互协议的能力及缺陷,在此基础上,设计并实现了一个基于 OWL-S 的组合 Web 服务执行工具,有效提高了组合 Web 服务执行的稳定性。由于 OWL-S 是语义 Web 服务描述语言,故该工具可用于语义 Web 服务应用。此外,系统需要进一步解决组合 Web 服务执行时的错误及错误处理问题。如果需要组合 Web 服务实现工作流模式 WP12、WP14、WP15、WP19 或 WP20,则要在创建组合 Web 服务的时候,使用一些具有辅助功能的 Web 服务(与组合 Web 服务实现的业务逻辑无关)来实现。例如,组合 Web 服务可能调用这样的 Web 服务:一旦它被调用,组合 Web 服务强制结束。该 Web 服务与组合 Web 服务业务逻辑无关,但可用于实现 WP20。

参考文献

1 The OASIS Business Process TC. ebXML Business Process Specification Schema version 2.0.3. http://www.oasis-open.org/committees/documents.php? wg_abbrev=ebxml-bp, Jun. 2006

2 Andrews T, Curbera F, Dholakia H, et al. Business Process Execution Language for Web Services version 1.1[OL]. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, Jun. 2006

3 Arkin A, Askary S, Fordin S, et al. Web Service Choreography Interface version 1.0[OL]. http://www.w3.org/TR/wsci/, Jun. 2006

4 Martin D, Burstein M, Hobbs J, et al. OWL-S: Semantic Markup for Web Services[OL]. http://www.w3.org/Submission/OWL-S/, Jun. 2006

5 Wohed P, van der Aslst W M P, Dumas M, ter Hofstede A H M. Pattern Based Analysis of BPEL4WS[G]:[Technique Report]. Faculty of IT, Queensland University of Technology, Dec. 2002

6 van der Aalst1 W M P, ter Hofstede A H M, Kiepuszewski B, Barros A P. Workflow Patterns[G]:[Technique Report]. Faculty of IT, Queensland University of Technology, Jul. 2002

7 Paolucci M, Srinivasan N. Amazon.com Web Services[OL]. http://www.daml.org/services/owl-s/AmazonWS/1.1/AWSProcess.owl, Jun. 2006

8 Kiepuszewski B, ter Hofstede A H M, Bussler C. On Structured Workflow Modeling[A]. Wangler and L. Bergman. In: Proceedings of the CAiSE'2000, LNCS[C], Sweden: Springer-Verlag, Jun. 2000. 431～445

计算机科学

(1974 年 1 月创刊)

第 34 卷第 05 期 (月刊)

2007 年 5 月 25 日出版

国际标准连续出版物号 ISSN 1002－137X

国内统一连续物出版号 CN50—1075/TP

定价:30.00 元 国外定价:5 美元

邮发代号:78—68

发行范围:国内外公开

主管单位:国家科学技术部

主办单位:国家科技部西南信息中心

编辑出版:《计算机科学》杂志社

重庆市北部新区洪湖西路 18 号 邮政编码:401121

电话:(023) 63500828 E-mail:jsjkx@swic.ac.cn

网址:www.jsjkx.com

社 长:牟炳林

总 编:彭 丹

主 编:朱宗元

主编助理:徐书令

印 刷 者:重庆科情印务有限公司

总发行处:重庆市邮政局

订 购 处:全国各地邮政局

国外总发行:中国国际图书贸易总公司(北京 399 信箱)

国外代号:6210—MO