一种集成 WS – BPEL 与业务规则的实现机制①

An Implementation Mechanism for WS – BPEL and Business Rules Integration

周进刚　（东软集团 基础软件事业部 辽宁大连 116023）
赵大哲　（东北大学 软件中心 辽宁沈阳 110006）
纪　勇　（东软集团 基础软件事业部 辽宁大连 116023）

摘　要： WS – BPEL（Web Service Business Process Execution Language）已经成为在面向服务计算环境下基于 Web 服务的业务流程编排标准，但它缺少对业务规则的有力描述，而业务规则作为业务建模的一个重要方面，是实现柔性流程与应用的重要技术。为解决 WS – BPEL 中缺少业务规则的问题，提出了一种 XML 规则语言，用来描述业务规则，并基于 WS – BPEL 的扩展要求实现了标准扩展，实现了 WS – BPEL 与业务规则的有效集成，同时分析了此方案的适用场景。

关键词： WS – BPEL　业务规则　XML

业务流程管理技术作为企业日常业务管理的核心技术变得日益重要，为了在实现和修改业务流程时保持一定的灵活性，需要把业务规则[1]从业务逻辑代码中分离出来，这样不仅可以降低在代码中维护这些错综复杂的规则代码的成本，而且可以对独立的规则进行动态变更，大大提高了流程或应用的柔性。随着 Web 服务技术的成熟和面向服务计算的应用，WS – BPEL（Web Services Business Process Execution Language，以下简称 BPEL）[2]成为业务流程管理的事实标准，然而 BPEL 没有提供把业务规则集成到业务流程中的有效方法。文献[3]提出了一种采用 AOP（Aspect – Oriented Programming，面向方面的编程）的方式，将业务规则以方面的方式编织到业务流程中，通过 AOP 感知的 BPEL 引擎来处理；文献[4]通过拦截器的方式在某个活动执行前后调用 Drools、Jess 等规则引擎来处理某个规则，从文章描述看，只是提供对数据的校验，不适用一般性的规则，这两种方式都破坏了 BPEL 流程的标准型；文献[5]采用将 ILog JRules 实现的规则封装为 Web 服务，由 BPEL 流程来调用，这种方式与文献[4]都需要进行 XML 与对象之进行相互转换，带来了性能损失。

1　示例流程

本文采用图 1 所示的流程来阐述本文 BPEL 与业务规则的集成。图 1 描述的是一个订单处理流程，根据用户订单的产品信息来决定需要支付的订单总额，并将订单总额信息发给客户，其中计算订单总额涉及图 1 中所示的 R1、R2 和 R3 三个业务规则。

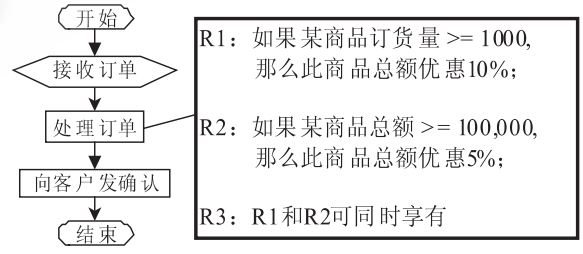

图 1　示例流程及业务规则的自然语言描述

标准的 BPEL 描述如下：

① 基金项目：国家 863 高技术研究发展计划（2006AA04Z166）；国家科技支撑计划（2007BAH19B01）；电子信息产业发展基金（[2007]329）

```
<process . . . >
  <sequence >
    <receive createInstance = " yes"
        variable = " orderRequest" . . . / >
    <sequence name = " orderProcessing" >
    <if > <condition >
        $ orderRequest. order/prodcut/number
> =1000
      </condition >
      <assign > <from >
        $ orderRequest. order/product/total _
price * 0. 9
      </from >
      <to > $ orderResponse. order/product/
total_price
      </to > </assign >
    <else >
     <assgin > <from >
     $ orderRequest. order/product/total_price
     </from >
     <to > $ orderResponse. order/product/total
_price
     </to >
    </assign >
  </else >
  </if >
  <if > <condition >
   $ orderResponse. order/product/total _ price >
=100000
  </condition >
  <assgin > <from >
   $ orderResponse. order/product/total _ price *
0. 95
   </from >
   < to > $ orderResponse. order/product/total
_price
   </to >
  </assign >
  </if >
 </squence >
 <reply variable = " orderResponse" . . . / >
</sequence >
</process >
```

业务规则的处理包含在 sequence **节点**(oederProcessing) **中,其中订单**(order) **的** XML **模式为**:

```
<schema xmlns = " http: //www. w3. org/2001/
XMLSchema"
    targetNamespace = " http: //www. sample. xml-
rules"
    xmlns: tns = " http: //www. sample. xmlrules"
    elementFormDefault = " qualified" >
   <element name = " order" >
     <complexType >
       <sequence >
         <element name = " product" minOccurs
= " 1"
             maxOccurs = " unbounded" >
           <complexType >
            <sequence >
            <element name = " item_price"
              type = " float" / > /
           <element name = " number" type = "
int" / > /
            <element name = " total_price"
              type = " float" / >
            </sequence >
             < attribute name = " name" type = "
string"
              use = " required" / >
            </complexType >
           </element >
         </sequence >
       </complexType >
     </element >
  </schema >
```

2　XML 规则语言

2.1 规则描述

本文描述的规则语言称为 xmlRule，其模式如下：

(1) package ?　　namespace

(2) import *　　xml schema definition

(3) variable *　　variable declaration

(4) rule +　　rule declaration

其中(1)表示此规则定义的命名空间，用来对规则进行分类分包管理。? 表示此条目的基数是 0..1；(2)用来引入一些 XML 处理规则中的自定义 XML 数据类型。xml schema definition 指代具体的 XSD 文件路径，* 代表此类条目的基数是 0..n；(3)用来声明此规则文件中用到的全局变量，变量声明的语法格式为：

variable " $ " name " : " xml schema data type

(4)用来声明真正的 XML 处理规则，+ 表示规则的基数是 1..n。规则声明的语法为：

RuleDeclaration	::= "rule" name **RuleProps** * **RuleBody** "end"	1
RuleProps	::= **Salient** | **ValidDate** | **Loop**	2
Salient	::= "salient" number	3
ValidDate	::= ("expire – date" date) | ("effective – date" date)	4
Loop	::= "loop" ("yes" | "no")	5
RuleBody	::= "if" (" $ " name "in" PathExpr) * **Condition** "then" **Action**	6
Condition	::= **BasicBoolExpr** | **CompBoolExpr**	7
BasicBoolExpr	::= **BoolConst** | **BoolItem**	8
BoolConst	::= "true" | "false"	9
BoolItem	::= **PathExpr LogicOp textValue**	10
LogicOp	::= " = " | " ! = " | " < " | " < = " | " > " | " > = "	11
CompBoolExpr	::= **Condition BoolOp Condition**	12
BoolOp	::= "and" | "or"	13
Action	::= **InsertExpr** | **RenameExpr** | **DeleteExpr** | **ReplaceExpr**	14
InsertExpr	::= "insert" "node" **PathExpr** **PositionChoice PathExpr**	15
PositionChoice	::= (("as" ("first" | "last"))? "into") | "after" | "before"	16
RenameExpr	::= "rename" "node" **PathExpr** **"as" PathExpr**	17
DeleteExpr	::= "delete" "node" **PathExpr**	18
ReplaceExpr	::= "replace" ("value" "of")? "node" **PathExpr "with" PathExpr**	19
PathExpr	::= **XQuery::PathExpr**	20

具体的规则声明分为三部分：名称声明；规则属性；规则主体。

名称(产生式 1 中的 name)是规则命名空间内规则的唯一标识符。规则属性包括：salient、validDate、loop 三个，分别表示：

- 规则优先级。用于规则冲突时的执行选择依据，其值越大表明优先级越高，缺省值为 0；
- 规则有效期限。expire – date 表示规则的失效日期；effective – date 表示规则的起始生效日期；
- 循环执行。表示当前规则是否在本规则处理后的 XML 片段仍然满足的情况下是否执行，以处理一些特定的循环处理情况。

其中产生式 3、4、6、10 中的变量终结符 number、date、name、textValue 分别表示一个整数、日期、变量名称、具体的业务属性值。产生式 14 – 19 表示规则允许的 XML 数据处理操作；产生式 15、17、19 中的前一个 PathExpr 表示新插入的 XML 元素，后一个表示所要插入的目的元素。规则主体的基础是 XQuery[6] 语法中的 PathExpr 产生式，典型的 PathExpr 描述如下(#)：

```
PathExpr# ::= ("/" RelativePathExpr?)
            | ("//" RelativePathExpr)
            | RelativePathExpr
RelativePathExpr ::= StepExpr (("/" | "//") StepExpr)*
StepExp ::= (ForwardStep | "..") Predicate*
ForwardStep ::= ("@" attribute) | element
Predicate ::= "[" (number) |
              (AttrExpr (BoolOp AttrExpr)*) "]"
```

AttrExpr ::= "@" attribute (**LogicOp** textValue)?

上述产生式中的 atttribute 和 element 分别指代 XML 中的属性和元素名; number 指代节点索引; textValue 指代具体属性值。

2.2 订单价格规则

订单处理规则的 xmlRule 描述如表 1 所示,其中 R1 的级别高,可以保障在规则冲突时(两个规则条件都满足)先执行,这符合客户的最大利益,因为 R1 的折扣高。Loop 属性设为"no"确保规则只执行一次。R1 和 R2 的 xmlRule 描述也满足 R3。

表 1　订单规则的 xmlRule 描述

```
rule NumberForDiscount//R1
  salient 1 //优先级高
  loop no //防止循环
if
    $x in order/product : $x/number >= 1000
    then
    replace value of node $/total_price with
  $/total_price * 0.9
```

```
rule TotalPriceForDiscount//R2
  salient 0
  loop no
if
    $x in order/product : $x/total_price >= 1,000,000
then
    replace value of node $/total_price with
     $/total_price * 0.95
```

3 集成机制

3.1 业务规则扩展

扩展的思路是定义一个实现了规则调用功能的新流程节点 rule,并将其置入流程。修改本文的示例流程如下:

```
<process ... >
  <extensions >
    <extension namespace = "http://www.unirule/"
      mustUnderstand = "yes" / >
  </extensions >
  <sequence >
    <receive createInstance = "yes"
      variable = "orderRequest" ... / >
    <extensionActivity >
      <uni:rule xml:uni = "http://www.unirule/"
        uni:ruleFile = "... /order.xrl"
        uni:inputVariable = "orderRequest"
        uni:outputVariable = "orderResponse"
        uni:session = "stateless" / >
      </extensionActivity >
      <reply variable = "orderResponse" ... / >
    <sequence >
      </process >
```

修改后的流程用一个扩展的 rule 节点替换了原来流程中实现规则的 sequence 节点。节点 rule 的 XML Schema 描述如下:

```
<schema targetNamespace = "http://www.unirule/"
    xmlns = "http://www.w3.org/2001/XMLSchema" >
    <element name = "rule" >
  <complexType >
  <attribute name = "ruleFile" type = "string"
    use = "required" / >
  <attribute name = "inputVariable"
    type = "string" use = "required" / >
  <attribute name = "outputVariable"
    type = "string" use = "required" / >
  <attribute name = "session" use = "required" >
    <simpleType >
      <restriction base = "xs:string" >
        <enumeration value = "stateful" / >
        <enumeration value = "stateless" / >
      </restriction >
    </simpleType >
  </attribute >
  </complexType >
```

</element>

</schema>

rule 节点定义了需要引用的规则文件、输入/输出变量、规则调用类型(有状态或无状态的,以区别工作区内数据的有效时限)。

3.2 集成框架

BPEL 流程与业务规则进行集成的框架如图 2 所示,包括:

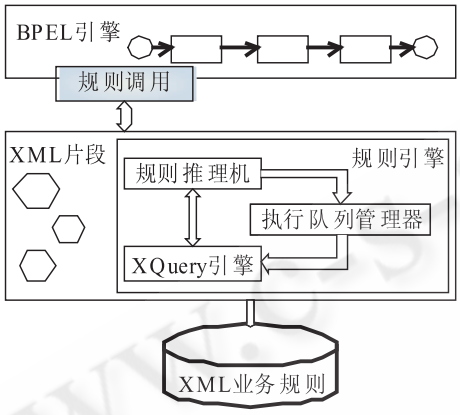

图 2 业务流程与业务规则集成框架

● 业务流程层:该层负责 BPEL 实现的业务流程的执行,其中包括一个规则调用服务,来完成与规则引擎的交互;

● 业务规则层:负责完成工作区内 XML 片段的规则处理,它主要包括规则推理机、执行队列管理器和 XQuery 引擎。规则推理机借助 XQuery 来实现规则的匹配与执行,规则执行由执行队列管理器统一调度,其中 XQuery 引擎是整个规则引擎的基础,它完成 XML 数据处理操作。

3.3 方案评估

业务规则从流程的分离降低了流程的复杂度、提高了可维护性;规则的独立变更,有利于规则的文档化和更方便地使用规则设计工具对其进行编辑。由于目前的企业级应用开发中大规模 XML 数据处理仍是影响应用系统性能的一个重要因素,因此在处理大规模复杂逻辑(业务对象关系复杂、相关对象数目庞大以及相关联的业务规则众多)的情况下,还不能与面向对象的规则技术相媲美。但对于大多数情况下,尤其在多信息系统交互或者集成的应用场景下,BPEL 与基于 XML 数据处理的业务规则的集成为它们实现柔性配置与管理提供了一种简洁、高效的方案。

4 结束语

业务规则的分离使得 BPEL 流程的结构更清晰,维护更简单;业务规则的有效集成必将深化 BPEL 应用,促进业务流程管理技术的发展。未来将引入面向服务的方式对规则进行服务封装,以进一步提高 BPEL 流程的可移植性。

参考文献

1 The Business Rules Group. Defining Business Rules – What Are They Really?. http://rewerse.net/downloads/BRG – whatisBR_3ed.pdf.

2 OASIS. Web Services Business Process Execution Language Version 2.0. http://docs.oasis – open.org/wsbpel/2.0/OS/wsbpel – v2.0 – OS.html. 2007 – 04 – 11.

3 Charfi A, Mezini M. Hybrid Web Service Composition: Business Processes Meet Business Rules In: The 2nd International Conference on Service Oriented Computing. New York: ACM, 2004. 30 – 38.

4 Florian Rosenberg, Schahram Dustdar. Business Rules Integration in BPEL – a service – oriented approach. In: The 7th International IEEE Conference on E – Commerce Technology. Los Alamitos, CA: IEEE Computer Society Press, 2005. 476 – 479.

5 Kevin Geminiuc. A Services – Oriented Approach to Business Rules Development. http://www.oracle.com/technology/pub/articles/bpel _ cookbook/geminiuc.html. 2008 – 3 – 3.

6 W3C. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/, 2007 – 01 – 23.