基于规则引擎的企业服务开发模式

陶晓俊,朱 敏
(华东师范大学 信息科学技术学院,上海 200062)

摘 要:围绕规则引擎技术分离业务流程和业务规则的思想,使用现代软件工程方法和工具,旨在研究并提出一套有效使用规则引擎技术开发企业服务应用系统时所遵循的开发模式。通过将规则引擎思想与现代软件工程常用开发模式、开发方法的改进和融合,设计了一种基于规则引擎的企业服务模型;提出了实际开发过程中紧密结合规则引擎思想,使用规则引擎技术实现企业服务应用的具体步骤和实施方法。在实际的企业服务开发工程中,使用文中所提出的基于规则引擎的企业服务开发模式将提高开发效率同时提高软件质量。

关键词:规则引擎;JSR-94;Rete算法;Drools
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2008)02-0115-04

Pattern of Building Enterprise Services Based on Rule Engine

TAO Xiao-jun, ZHU Min
(Sch. of Info. Sci. & Tech., East China Normal University, Shanghai 200062, China)

Abstract: Study and advance a pattern which is based on the idea of rule engine for practical building enterprise services that using rule engine technology effectively and handy. The pattern includes a model of enterprise services based on rule engine, and the step and process of developing enterprise services using rule engine technology. It will help to improve the efficiency of exploitation and the quality of software if the pattern is used in the practical exploitation of enterprise services.

Key words: rule engine; JSR-94; Rete algorithm; Drools

0 引 言

规则引擎[1,2]是用以管理和自动实现业务规则的软件系统,其主要实现的功能是存储、分类和管理规则,验证规则的一致性,通过规则推断其它规则、联系规则和执行这些规则的应用程序,其中的规则主要是指企业或商务业务逻辑、法律条款、企业政策等。规则引擎概念的思想是从软件的应用逻辑中分离出商业规则,以实现商业应用的灵活性。在传统的企业服务应用程序开发模式[3]下,业务逻辑[1,2]被直接固定在应用程序代码中,这使得应用程序维护复杂并且代价昂贵,变化的商业规则和业务流程总是引起对应用程序的频繁修改,尤其面临动态商业模型和业务流程的挑战时,传统模式下开发的应用程序往往面临全面和代价高昂的修改,甚至设计变化。解决这个问题就需要采用新的开发模式,将业务逻辑从代码层剥离。使用规则引擎恰恰提供了一个将业务处理和业务规则处理分离、共用和统一管理维护业务规则的系统开发构架。以下探讨的就是基于规则引擎的企业服务应用开发模式,其中包括基于规则引擎的企业服务模型和基本的开发步骤和方法。

收稿日期:2007-05-06

作者简介:陶晓俊(1982-),男,上海人,硕士研究生,研究方向为现代软件技术;朱 敏,高级工程师,硕士生导师,研究方向为信息系统、现代软件技术、软件中间件技术。

1 基于规则引擎的企业服务模型

设计明确和有效的系统模型是企业服务系统得以顺利进行的前提。图1设计了一个简单的基于使用符合JSR-94[4]标准的规则引擎及其模式开发的企业服务应用程序的体系结构。

图1描述的企业服务体系分为三个部分[4]:

* 应用程序/数据获得系统:捕获和存储应用程序提交的所有数据,是业务服务的使用者。主要功能是提交业务请求和处理业务判定。

* 业务服务:通过具体实现的可调用的网络服务器或者API,调用选定的规则引擎来执行业务规则逻辑或对业务规则逻辑进行运算,产生反馈信息和数据。同时也提供方便和有效维护业务规则逻辑的功能。

* 支持服务:提供业务服务使用者所提交的相关数据,即规则引擎执行业务规则或运算业务规则所需

要的相关数据或应用程序或服务接口。

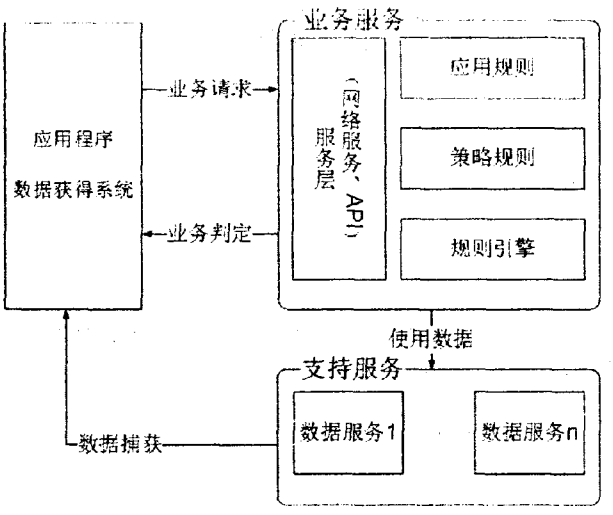

图 1 基于规则引擎的企业服务模型

2 基于规则引擎的企业服务开发模式中的步骤和方法

在基于规则引擎的企业服务开发模式中，至关重要的原则是：分离工作流程和业务规则；形式化地描述业务规则。在这个开发模式过程中，这两个原则贯穿始终。分离工作流程和业务规则的目的在于将关键的业务判断规则和业务事件响应提取出来，置于系统的公共部位(业务服务)，供不同的应用程序工作流程使用，并且便于维护和管理。这是此模式下开发系统得以顺利进行的前提。形式化地描述业务规则的目的则是将业务规则以一种能够被规则引擎处理的形式描述和表示，使业务规则可被运算化，使应用程序可以按照即定义的约定通过一个服务层来访问这些规则。

以下提及和讨论的开发模式中的方法和原则都围绕着以上两个原则展开。

2.1 使用决策表提取规则

在基于规则引擎的企业服务开发模式中首先要解决的问题是明确企业应用中有哪些规则以及对应的业务判定。商业事务中条件元素的集合构成规则，规则决定判定和反馈[1,2,4,5]，在散乱的企业业务中初步提取规则和对应的判定可以使用决策表的方法。

表 1 给出的是决策表的一般格式。参照图 1 所示模型，其中条件将成为系统中的数据域，条件值则是对应的数据，是支持服务管理和维护的对象，同时是应用程序/数据获得系统捕获和提交的对象。若干条件及其值的集合构成规则，是业务服务管理和维护的对象，也是规则引擎处理的根据。判定是业务服务通过规则引擎处理给出的反馈，最后由应用程序/数据获得系统接收。

表 1 决策表一般格式

	规则 1	规则 2	……	规则 i
条件 1	条件值			
条件 2				
……				
条件 j				
判定 1	是否反馈			
判定 2				
……				
判定 k				

下面使用一个企业决定销售人员当月销售奖金的例子来进一步说明决策表的使用(见表 2)。

表 2 决策表使用样例

	A	B	C	D	E	F	G
1		rule1	rule2	rule3	rule4	rule5	rule6
2	achieveTarget	N	N	N	Y	Y	Y
3	saleVolume	<10k	>preMonth	>=10k	<10k	<=preMonth	>=10k
4	warn	Y	N	N	N	Y	N
5	amortization	0%	1%	2%	2%	2%	3%

表 2 中的“achieveTarget”表示条件“完成销售指标”；“saleVolume”表示条件“销售额”；“amortization”表示判定“提成”；“warn”表示判定“提出警告”；“preVolume”表示数据“上月销售额”。

在表 2 中，列 A 描述了应用中所涉及的条件和系统处理后的判定，与各个条件或者判定同行的单元格中的值组合构成列 B 至列 G，列 B 至列 G 中的每一列都是通过决策表得以分离的规则，规则通过条件值集合和判定反馈值的形式得以描述。例如规则 2(rule2)描述的业务规则是：若没有完成销售指标，且销售额大于上月销售额，那么不给警告，并且判定提成比率为 1%。规则 5(rule5)描述的业务规则是：若完成销售指标，且销售额小于或等于上月销售额，那么给予警告，并且判定提成比率为 2%。在实际的应用中可以通过对企业成文或不成文商业规则的总结和归纳来生成决策表，在必要时，例如商业规则不明确时也可以通过对条件组合的穷举来制定决策表，在以后的开发过程中再进行精简。

分离商业规则是基于规则引擎开发企业应用的前提，它是基于规则引擎的企业服务开发模式中的第一步，也是必须和最重要的步骤。如果使用文中提供的决策表方法可以比较方便地分离业务逻辑和商业规则，比较清晰准确地描叙规则，并且具有与基于规则引擎的企业服务模型结合比较紧密的特点，这一特性使其可以对后续开发步骤中问题的解决提供有力支持。

2.2 分析和解决规则冲突

在分离和提取出规则之后，必须考虑规则之间的冲突，这里提到的冲突主要是指由于规则之间同一条件的值域相交而引起的判定歧义。如果规则之间存在

冲突而没有得到解决将造成判定结果的不确定,使规则引擎的处理不能正确进行。可以通过使用决策网格来侦测规则之间的冲突。具体的做法是将各条规则依次置于网格的行首和列首,网格中的每一个单元表示对应规则的交叉点,用以记录相交的规则或者规则集合为真时,反馈值是否发生歧义。判断规则是否冲突的依据是相交规则所包含条件值域的交集以及各个判定的反馈值,具体流程如图 2 所示。

以 2.1 中得到的规则为例,使用决策网格分析对于判定 amortization 的反馈,各条规则之间的冲突情况,结果见表 3。

表 3　决策网格使用样例

	rule1	rule2	rule3	rule4	rule5	rule6
rule1	NC	Y[1,2]	N	N	N	N
rule2	Y[1,2]	NC	Y[2]	N	N	N
rule3	N	Y[2]	NC	N	N	N
rule4	N	N	N	NC	Y[1]	N
rule5	N	N	N	Y[1]	NC	Y[1,2]
rule6	N	N	N	N	Y[1,2]	NC

表 3 清晰地描述了各条规则之间的冲突情况,其中 Y 表示规则可能同时为真,并且有冲突,跟随其后的"[]"中指出具体发生冲突的判定名、判定序号或判定的描述;N 表示规则没有冲突或者不可能同时发生。

在明确了规则之间的冲突关系之后,自然必须考虑如何解决冲突。文中提出三种解决规则冲突的方案:优先级模式;队列模式;常用模式。

* 优先级模式:设定规则的优先级顺序,在规则发生冲突时采纳较高优先级规则的判定反馈。有利于保持业务处理结果的长期稳定性,适用于业务处理规则比较明确的企业应用。

* 队列模式:先进先出原则,即先使用原则。在一个时间段内集中批处理业务时,若发生规则冲突,则采纳先使用(前一次或第一次处理已使用)规则的判定反馈。有利于保证同一批次业务处理结果的一致性,适用于有业务高峰的企业应用。

* 常用模式:记录和参考业务处理中规则的使用频率(规则为真的频率),在发生规则冲突时采纳使用频率较高或较低的规则反馈。有利于根据业务开展情况灵活适用规则,适用于业务开展情况多变的企业应用。

在一个企业应用服务中,以上提出的规则冲突解决方案可以单一使用也可以结合使用,以构成企业应用的规则冲突解决机制。在没有采取规则冲突解决方案的情况下,一般规则引擎会采用 LIFO (last in, first out)原则解决规则冲突,但是对于一个成功的基于规则引擎开发的企业应用,一套规则冲突解决机制是必不可少的。

2.3　形式化描述规则

规则集的冲突问题得到解决之后,基于规则引擎的企业服务开发模式的下一步是要形式化地描述规则,使其具有可代码化和可运算化的性质。当前的规则引擎一般都是基于 Dr. Charles L. Forgy 于 1979 年提出的 Rete 算法[5](网络算法)。Rete 算法的基本思想是组织一个有效的辨别网络,通过数据在网络中的传播来过滤数据。Rete 算法的具体做法是首先建立一个根结点(root node)作为数据对象进入辨别网络的入口,根据规则所包含的条件建立测试结点构成辨别网络,在网络的最底层构建若干终结点(terminal node)描述相应的规则。数据对象进入辨别网络之后经过途径结点,最终到达某个终结点,激活这个终结点所描述的规则。根据 Rete 算法的特性,要求将规则分割为 LHS(left hand side)和 RHS(right hand side)[4],LHS 由规则的条件部分组成,决定辨别网络中测试结点的生成,RHS 由规则的判定部分组成,决定辨别网络中终结点的生成。

以 2.1 中得到的规则为例,可以采用在提取规则步骤中得到的决策表来解决问题,对于各个规则,其所在的列条件值构成它的 LHS,判定反馈值则构成它的 RHS。由此,每一条规则都可以被形式化地描述为:

(achieve Target = $input) ∩ (saleVolume = $input) ⇒ (warn = $feedback) ∪ (amortization = $feedback)

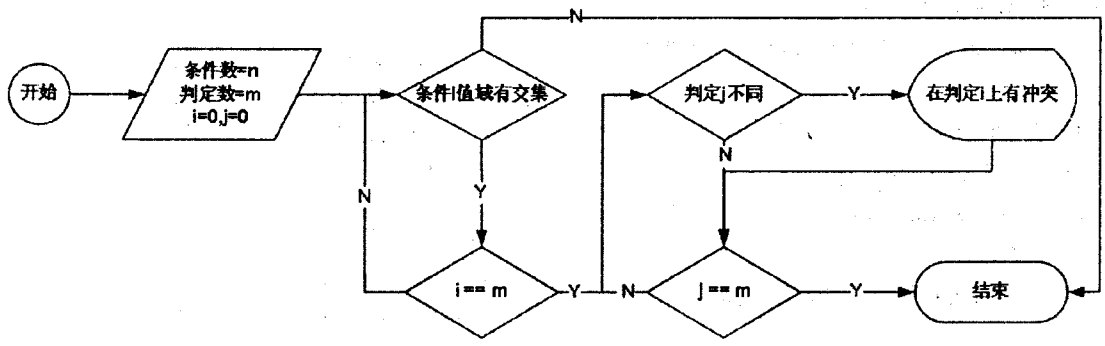

图 2　冲突解决算法流程

形式化的描述规则为基于规则引擎的企业服务开发模式中后续地将规则代码化过程做了铺垫。

2.4 设计业务流程

与通常的设计模式[3]相同,基于规则引擎的企业服务开发模式也包含和关注企业服务的业务流程设计。由于实现了业务流程与业务规则的分离,使得业务流程的设计得到很大的简化,业务流程中不再需要设计繁琐和庞大的条件判断,大大减轻了业务程序的负担。业务流程的设计原则应遵从图 1 描述的服务模型,设计方法则采用与以往设计模式相同的 UML 方法,文中不作详细的讨论,只以 2.1 中提出的例子为例,给出相应的业务流程活动图[3]来直观说明分离业务流程与业务规则之后使得业务流程设计得到简化的程度,如图 3 所示。

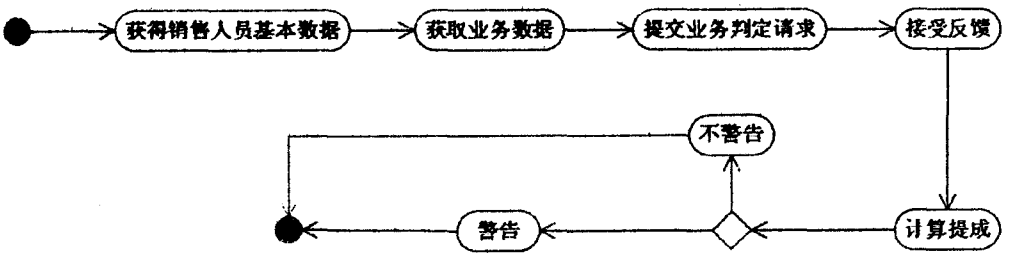

图 3 业务流程图样例

从图 3 中可以清晰地看到,业务流程已不包含业务逻辑,当业务逻辑改变时,不需要对业务流程作出改变。

2.5 业务规则代码化和业务流程代码化

代码化的步骤既是企业服务具体得到实现的过程,简而言之就是程序代码实现的过程。在这个过程中,需要遵从前序步骤的设计和结果,并且围绕图 1 描述的服务模型展开。在此以 2.1 中提出的例子为例,选用 Java 语言和"Drools"规则引擎来实现代码化。文中只给出需要实现的类及其简述,而不给出具体实现代码。

(1)应用程序部分的实现。

* SellApplication. class:捕获数据、提交业务判定请求、接收反馈、实现业务流程。

* SalesPerson. class:记录销售人员相关数据的数据对象。

* FeedBack. class:记录判定和反馈数据。

(2)支持服务部分的实现。

* AchieveTargetService. class:负责给出销售人员完成销售指标的情况。

* SaleVolume. class:负责提供销售人员月和上月的销售额。

(3)业务服务部分的实现。

AmortizationService. class:接受业务判定请求、调用规则引擎服务、反馈判定。

(4)规则的代码化。

以表 1 中的 rule1 为例,其对应"Drools"规则引擎的代码如下:

```
rule "rule1"
  salience -100 //优先级设定
  when //LHS
    SalesPerson(achieveTarget == false, SaleVolume <= 100000)
    $sa: SellApplication()
    $fb: FeedBack()
  then //RHS
    fb.setWarn(true);
    fb.setAmortization("0%");
    sa.setFeedBack(fb);
end
```

3 结束语

文中提出的基于规则引擎构架的企业服务开发模式紧密地与规则引擎思想相结合,提出的步骤和方法能够很好地帮助开发人员在开发企业应用服务过程中分离业务流程和业务规则,明确业务规则,并且使得这些业务规则具有可描述化、可用化的性质,最终得以在应用服务中实现作用。虽然规则引擎的应用还不广泛、不深入,但是其概念和理论已经比较成熟,针对于企业服务应用,尤其在动态商业模式下的企业应用有着明显的优势。在实际的企业应用服务开发过程中,引入基于规则引擎构架的企业服务开发模式将使得企业服务应用具有更高的可维护性和更高的灵活性,具有推广使用的价值。

参考文献:

[1] Chisholm M. How to Build a Business Rules Engine[M]. [s. l.]:Morgan Kaufmann,2003.

[2] Graham I. Business Rules Management and Service Oriented Architecture[M]. [s. l.]: Wiley,2007.

[3] Gamma E, Helm R, Johnson R, et al. 设计模式——可复用面向对象软件的基础[M]. 李英军 译. 北京:机械工业出版社,2005.

[4] Java Community. Java Rule Engine API Specification[EB/OL]. 2002 - 09. http://jcp. org/aboutJava/communityprocess/review/jsr094/.

[5] Temlpe University CIS587. The RETE Algorithm[D/OL]. 2004 - 05. http://www. cis. temple. edu/~ingargio/cis587/readings/rete. html.