第 33 卷 第 6 期 计 算 机 工 程 2007 年 3 月
Vol.33 No.6 Computer Engineering March 2007
·软件技术与数据库· 文章编号：1000—3428(2007)06—0062—03 文献标识码：A 中图分类号：TP311

应用 AOP 设计的可动态重构工作流引擎架构

孙中轶，何 牧，蔡鸿明，姜丽红
(上海交通大学软件学院，上海 200240)

摘 要：鉴于现有的 AOP 技术并不能适应工作流系统的需要，该文应用 AOP 的基本概念和方法，提出了一种可动态重构的工作流引擎架构，解决了工作流引擎基本结构的动态扩展以及工作流定义的动态修改问题。在应用此架构开发的一个符合 WFMC 标准的工作流引擎上，验证了此架构的可行性及动态重构的便利性，分析了架构的优缺点。

关键词：工作流；工作流引擎；面向方面的程序设计；动态重构

Dynamic Refactoring Workflow Engine Architecture's Design with AOP

SUN Zhongyi, HE Mu, CAI Hongming, JIANG Lihong
(School of Software, Shanghai Jiaotong University, Shanghai 200240)

【Abstract】The existing AOP technology is incapable to meet workflow engine's requirements. Based on basic concept and methodology of AOP, this paper states dynamic refactoring workflow engine architecture. This architecture intends to resolve the dynamic extending of basic engine structure and modification of workflow definition at runtime. It verifies the feasibility and convenience on a WFMC compliance workflow engine developed using this architecture. It provides the advantages and disadvantages of this architecture.

【Key words】Workflow; Workflow engine; Aspect-oriented programming(AOP); Dynamic refactoring

面向方面的程序设计(aspect-oriented programming，AOP)最早在文献[1]中提出，它是一种程序设计范型。该范型以称为方面(aspect)的语言构造为基础。方面是一种新的模块化机制，用来描述分散在对象、类或函数中的横切关注点。

工作流最早是被用来做与时序或逻辑相关的业务流程的部分或全部自动化工作，但随着它在各种领域的广泛应用，支持动态改变的能力越来越重要。文献[6,7]等应用工作流实例迁移的方法解决了工作流定义的动态变化问题。而本文中的结构可以在不侵入原有结构的前提下，对工作流作临时或永久的动态变化，不仅在工作流的执行方面，而且在工作流引擎结构的扩展方面，应用 AOP 的方法学为支持动态改变提供了新的思路。

应用 AOP 来支持灵活的工作流系统的可能性最早在文献[2]中提出。在文献[3]中又进一步从理论上讨论了在工作流实例中动态应用 AOP 的情况。这二者中都没有对其理论做出验证。本文基于 AOP 的基本概念和方法提出了一个可动态重构的工作流引擎架构并予以实现，同时讨论为何现有的 AOP 技术不适用于工作流领域。

1 工作流引擎架构

应用了 AOP 方法学所构建的工作流引擎将拥有与传统模型不同的开发和部署方式。主要区别就在于引擎所具有的各种功能将被分离成各种方面，分别独立开发，而在部署阶段利用配置信息(WF Config)被结合在一起，从而在运行时动态组装为最后的执行序列。

图 1 为应用 AOP 方法学的工作流引擎的开发和部署方式。

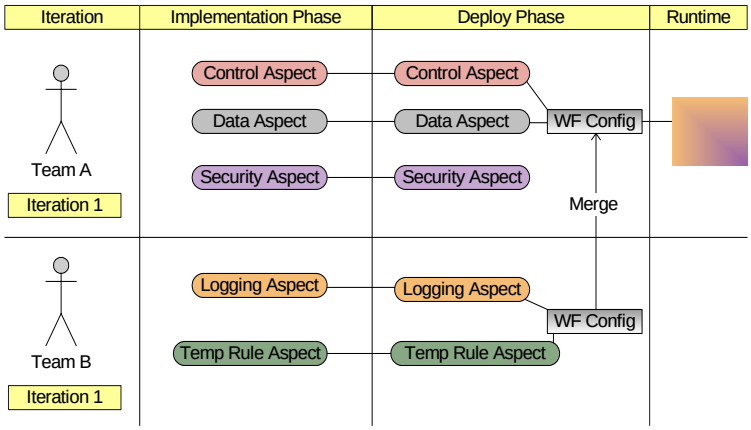

图 1 应用了 AOP 方法学的工作流引擎的开发和部署方式

1.1 方面

在工作流引擎架构中的方面从有效范围角度可分为全局方面和局部方面两种。全局方面即对系统运行的全部实例均有效的方面，局部方面即对特定实例有效的方面。从有效时

基金项目：国家自然科学基金资助项目(70471024)；部级开放实验室基金资助项目“铁路信息科学与工程”
作者简介：孙中轶(1983－)，男，硕士，主研方向：工作流，企业信息系统，系统架构；何 牧，硕士；蔡鸿明，博士；姜丽红，副教授
收稿日期：2006-03-25 **E-mail：**silverelf@sjtu.com

间角度可分为正常方面和临时方面两种，后者即在特定时间内有效的方面。而从结构角度可分为核心方面和扩展方面。

在本文所设计的架构中，核心方面只有一个控制方面，其他均为在控制方面之上的扩展方面，如：资源，日志，安全，负载平衡，备份。

核心的控制方面是整个架构的主要关注点，它实现了引擎中的判断路由以及执行活动等核心功能并只关注于此，包括资源的统一访问以及日志、安全等在内的其它功能则由扩展方面来专门完成。

1.2 织入(Weave)

不同于现有的 AOP 产品所采用的可针对任意函数进行织入的策略，本文中的引擎架构采用的是固定织入点的策略，原因如下：

AOP 可以看作一种二级语言，以逻辑(1)来加入要运行的代码：

$$W \ni \text{workflow instances} \wedge A \ni \text{Aspect} \wedge \text{A.weaveMode (W)} \wedge \forall \text{A.dependant} \ni \text{Local Aspect} \bullet \text{IsFinished (dependant)} \Rightarrow \text{execute (A.advice)} \quad (1)$$

其中，advice 表示建议代码；dependant 代表此方面所依赖的其他方面；isFinished 返回此方面实例是否运行完毕；weaveMode(wf: workflow Instance)判断对当前工作流实例是否织入建议代码。

传统逻辑系统由公理+规则组成，是对现实系统的规范，但不能修改现实系统的任何成份。而为实现 AOP 的这个二级语言的“执行建议代码”，需要对原来的工作流运行进行织入，则原来工作流运行的逻辑规范很可能不再被满足，导致原先的真理或公理不再为真。假设允许向工作流运行中的任何点动态插入建议代码，则可能导致原先对工作流运行的断言(如赋值不变性)失效，从而导致整个工作流系统运行的不一致性和错误的结果，因此必须限定织入点来保证结构的可靠性。

在结构可靠的前提下，织入设计如下：

织入点：在引擎中，一个完整的工作流执行过程如下：(1)工作流实例初始化→(2)解析模型→(3)判断路由→(4)执行活动→(5)工作流实例结束。其中(2)~(4)将根据具体流程以及工作流参数的不同执行不确定的有限次，因此得出可织入的地点为：(1)工作流实例初始化前后；(2)解析模型前后；(3)判断路由前后；(4)执行活动前后；(5)工作流实例结束前后。

织入方式：由于所织入的方面所具有的逻辑复杂性，并不是在任何上下文条件下均对所有的流程实例织入逻辑。因此，本架构的织入方式分为两种：

(1)通配：$A \ni \text{Aspect} \wedge \forall W \ni \text{WF instances} \Rightarrow \text{weave (A.advice)}$

(2)条件：$A \ni \text{Aspect} \wedge \forall W \ni \text{WF instances} \wedge \text{A.weaveMode (W)} \Rightarrow \text{weave (A.advice)}$

1.3 配置(Configuration)

由于方面本身以及织入的动态性，配置本身需要使用 XML 的格式并存储在特定的文件目录下。具体分为两种：

(1)全局：$\forall W \ni \text{workflow instances} \wedge A \ni \text{Aspect} \wedge \text{A.weaveMode (W)} \wedge \forall \text{A.dependant} \ni \text{Local Aspect} \bullet \text{IsFinished (dependant)} \Rightarrow \text{execute (A.advice)}$

(2)特殊：$\forall W \ni \text{workflow instances} \bullet \text{W.BelongsTo(specific definition)} \wedge A \ni \text{Aspect} \wedge \text{A.weaveMode(W)} \wedge \forall \text{A.dependant} \ni \text{Local} \bullet \text{IsFinished (dependant)} \Rightarrow \text{execute(A.advice)}$

其中，BelongsTo(workflow definition)判断此工作流实例是否由特定的工作流定义实例化而来。

1.4 执行优先级(Execution priority)

在有多个方面同时存在的情况下，不可避免地存在并发执行的情况。而在某些情况下不同的方面之间会存在依赖关系，包括时序依赖和数据依赖。而这些依赖关系是由部署人员在相应的特殊配置文件中指定的。因此必须存在一个执行策略以解决并发问题，具体策略：首先执行全局配置文件中的方面，然后执行当前工作流实例对应的特殊配置文件中的方面。而在特殊配置文件中，根据同一文件内的依赖关系进行反转排序，保证前置方面执行完毕后再执行依赖方面。由此在运行时保证数据以及时序的正确关系。

1.5 核心结构

依据上面提到的架构因素以及WFMC参考模型[5]设计出的核心结构如图 2 所示。

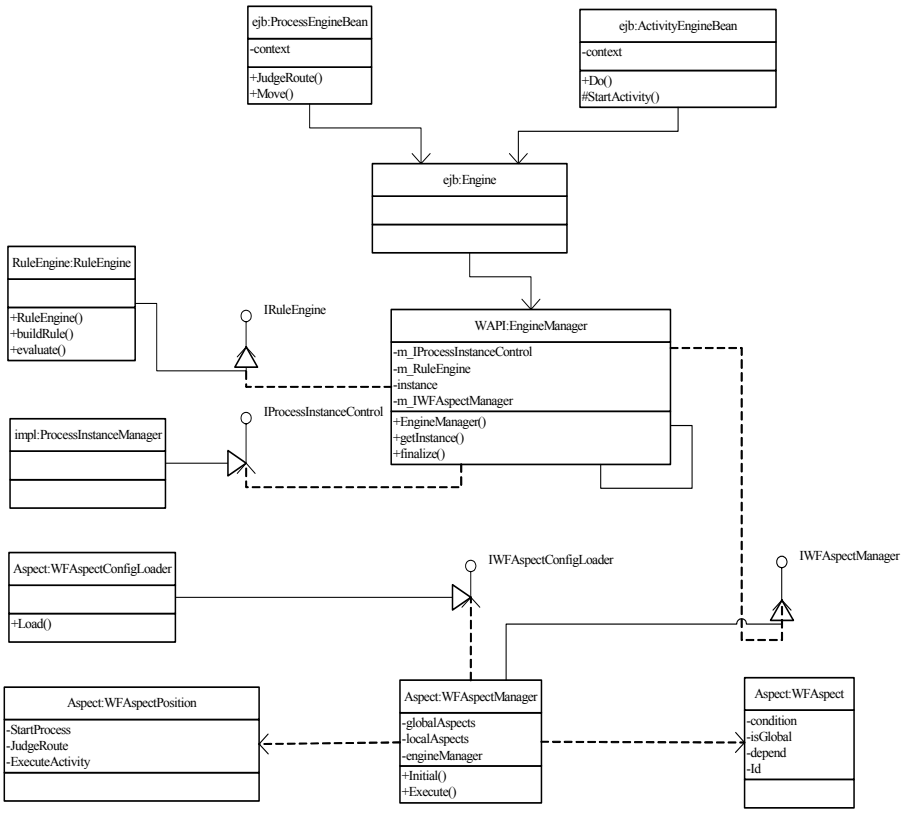

图 2 核心结构

其中核心类是 EngineManager，集中处理了各个部件的初始化过程。ProcessEngine 类的功能为判断路由，

ActivityEngine 类的功能为具体执行活动。RuleEngine 类的作用就是判断流程以及方面织入条件是否成立。WFAspectManager 类封装了对所有的全局方面以及当前的流程定义所涉及的特殊方面的管理及执行功能。

本架构之所以选择自己实现 AOP 的功能，原因是现有的 AOP 技术在以下方面不能满足工作流引擎的需求。

(1)AspectJ：最流行的 AOP 实现，但不支持动态织入，因此不能满足工作流引擎的动态重构。

(2)Jboss AOP：可动态织入，但与应用服务器绑定，不符合工作流引擎的跨应用平台需求，并且无法限制织入点。

(3)AspectWerkz：可动态织入，不与应用服务器绑定，但无法限制织入点且没有条件织入的功能。

执行过程举例：判断路由这个织入点前的方面的具体过程如下：

当 ProcessEngine 开始运行的时候，通过 EngineManager 向 WFAspectManager 发出请求，执行与当前工作流定义相关的方面。WFAspectManager 取出全局方面，然后通过 WFAspectConfigLoader 载入于当前工作流定义相关的局部方面，而后按照 1.4 节(执行优先级)中所说明的策略执行。最后将运行结果返回至工作流上下文，继续 ProcessEngine 的运行。

2 验证与分析

2.1 验证

依照上文中提出的架构，本文实现了一个符合 WFMC 标准的工作流引擎，并与其他模块结合构成了一个完整的工作流管理系统 Elune Workflow。目前实现的方面有控制、资源、安全和日志。下面是一个特殊配置文件的例子(其中有些不重要的信息省略)来体现具体的流程动态变更。

```
<aspects>
<aspect id="SpecialLog" Mode="every" valid="" WFID="
SpecialWF ">
<pointcut mode="*" position="*">
<advices><advice source=" " action="LOG"/></advices>
      </pointcut>
</aspect>
<aspect id="PostSpecialLog" Mode="SpecialLog.Success ==
true" valid="" WFID=" SpecialWF " depend=”SpecialLog”>
<pointcut mode="*" position="*">
<advices><advice source=" " action="PostLOG" /></advices>
</pointcut>
</aspect>
</aspects>
```

在这个配置文件中定义了两个方面，分别是 SpecialLog 和 PostSpecialLog。二者均针对同一工作流定义 SpecialWF 并在 valid 属性所指定的日期后过期。SpecialLog 的织入方式为通配，而 PostSpecialLog 只有当 SpecialLog 执行成功的时候才会执行，并且依赖于 SpecialLog。这两个方面的织入点以及织入方式均为通配。

这两个方面的行为方式如下：

SpecialLog：

∀ W ∋ instance • W. BelongsTo (SpecialWF)
^ A ∋ SpecialLog
^ CurrentTime < A. Valid
⇒ Execute (LOG)

PostSpecialLog：

∀ W ∋ WF instance • W.BelongsTo (SpecialWF) ^ Current Time < A. Valid
^ A ∋ PostSpecialLog
^ IsFinished (SpecialLog)
⇒ Execute (PostLOG)

2.2 分析

优点：本文中的工作流引擎架构解决了工作流引擎的基本结构动态扩展以及工作流定义的动态修改问题，使引擎核心以及扩展功能的开发和部署更为简单，并可实现运行时动态添加、修改以及删除功能。该架构还可以使不同组的开发人员完全并行工作，减少知识同步，最后达到分布式并行开发的结果。同时它还可实现对流程定义的动态修改，以使企业计算环境中因不可预见的暂时性因素所造成的流程影响可以及时地反应到引擎执行中，并在因素消失时自动过期，因此可模拟现实环境的变化。

缺点：分离出的 Aspects 层以及配置文件的处理会导致该引擎可能比实现同样功能的传统工作流引擎在性能上有所下降。关注点的分离，以及代码结构本身的分布性，会导致当前代码运行的不确定性。这需要开发人员对自己负责的模块做好完整的单元测试，特别是边界条件等约束。

3 结论及未来的工作

本文研究了应用 AOP 方法学设计工作流引擎架构以支持动态重构进行，证明了该架构的可行性及其在动态变化的环境中表现出的优越性，同时也讨论了在现有 AOP 技术在实现工作流引擎上的缺陷。

还有一些研究点需要进一步的工作，例如：(1)需要在被织入的功能中开放工作流的何种属性以供操作。现阶段为流程实例、定义模型、当前活动、目标活动集以及工作流参数。探索其他可被开放的属性，研究现有的开放属性是否会造成安全或性能方面的漏洞。(2)方面执行的优先级问题。本文中的优先级模型还比较简单，可进一步设计更为复杂的结构来为多变的方面间关系进行建模。(3)开放的织入方式。引入现有 AOP 技术的声明织入点技术以动态扩展织入点，但这涉及到工作流引擎本身的结构可靠性问题，因此还需要研究扩展限制的方法。

参考文献

1 Gregor K, John L, Anurag M. Aspect-oriented Programming[C]//Proc. of European Conference on Object-oriented Programming. 1997.

2 Schmidt R, Assmann U. Extending Aspect-oriented-programming in Order to Flexibly Support Workflows[C]//Proceedings of Aspect-Oriented Programming Workshop at ICSE’98. 1998.

3 Bachmendo B, Unland R. Aspect-based Workflow Evolution[C]// Proceedings of the Workshop on Aspect-oriented Programming and Separation of Concerns, Lancaster, UK. 2001.

4 Workflow Management Coalition: Terminology&Glossary, Document Number WFMCTC-1011[Z]. 1999. http://www.wfmc.org.

5 Workflow Management Coalition: Workflow Reference Model, Document Number WMFC TC00-1003[Z]. 1995. http: //www.wfmc.org.

6 Zhou Mingtian, Wang Minyi, Yao Shaowen. A Workflow Instance Migration Approach Based on the Extended-task-structures[J]. Journal of Software, 2003, 14(4): 757-763.

7 Cui Lizhen, Wang Haiyang. Dynamic Workflow Schema Evolution Based on Workflow Type Versioning and Workflow Migration[J]. Journal of Computer Science, 2004, 31 (9).