第 32 卷　第 21 期　　计　算　机　工　程　　2006 年 11 月
Vol.32　No.21　　Computer Engineering　　November 2006
·工程应用技术与实现·　文章编号：1000—3428(2006)21—0229—04　文献标识码：A　中图分类号：TP316

一个面向电信级系统的嵌入式工作流引擎

高　鹏
(卓望数码技术有限公司，深圳 518057)

摘　要：在一个为国内某著名电信运营商提供的数据业务运营管理系统的建设中，引入了工作流技术，自主研发了一个轻量级嵌入式工作流引擎；分析探讨了该引擎的技术架构和关键特性。目前对工作流技术的应用研究主要集中在企业级应用系统，该文通过对一个成功案例的分析，为电信级软件系统引入工作流技术提供了一个参考实践。

关键词：工作流引擎；电信级；嵌入式；XPDL

An Embedded Workflow Engine Designed for Telecom-level System

GAO Peng
(Aspire Technologies Limited, Shenzhen 518057)

【Abstract】 In a data service information management system, which is served for a famous telecom operator in China, this paper introduces the workflow technologies and develops the light embedded workflow engine. This thesis gives an introduction to the engine. As to the workflow application, most researches are focused on the enterprise-level application. The workflow engine gives an actual example on how to apply workflow on telecom-level system successfully.

【Key words】 Workflow engine; Telecom-level; Embedded; XPDL

本项目来源于一个国内某著名电信运营商的数据业务运营管理系统。该系统的主要职责是对运营商的数据业务合作伙伴(SP)进行管理，包括一系列的评估、考核等电子流程，最终目的是通过这些电子流程，运营商可以对合作伙伴进行等级划分，为不同等级的合作伙伴提供不同的资源和营销服务等。

该系统一方面要求能够实现灵活的流程定制，以适应不同省公司的个性化管理需求，另一方面要求达到电信级的稳定性和可靠性，以保证在系统全国商用后，能够有效支撑业务运营。

在该系统建设中，引入了工作流技术，自主研发了一个工作流引擎产品。目前，工作流引擎已经在该运营商数据业务运营管理系统中成功担当起了流程控制中心的角色。本文将对该工作流引擎系统的设计及实现技术进行分析。

1 系统架构

设计上，考虑到所针对系统的特点，我们采用了嵌入式模型，一方面方便和业务系统集成，另一方面可利用数据库技术来保证事务的严格 ACID 特性，满足业务系统高可靠要求。架构上，引擎参照工作流管理联盟(WfMC)的工作流参考模型实现，并遵循其 XPDL 工作流过程定义语言规范。

1.1 外部视图

基于工作流引擎的业务系统的整体架构如图 1 所示。

系统架构简要说明如下：

(1)整个业务系统运行于 J2EE 平台之上，业务系统分成 2 部分：业务系统外壳和业务系统内核。内核系统负责业务核心流程、逻辑的执行；外壳系统负责业务界面的展示、用户交互。外壳和内核间采用 EJB 接口，目的是使一个内核可以服务于多个外壳系统。

(2)工作流引擎内嵌在业务系统内核中，作为控制中心运行，引擎由 2 大部分组成：核心控制系统及管理控制台。核心控制系统负责对流程的解析、调度、执行，调用工作流活动相关的外部应用，存储工作流控制数据/相关数据及对外提供交互接口；管理控制台提供对引擎的管理控制界面，供引擎管理员控制业务流程的执行以及监控引擎的运行状况。

(3)引擎对外提供 3 类接口：

1)I1 接口，它是引擎对外提供的 Workflow API 接口。主要是提供操作工作项的相关方法；

2)I2 接口，它是引擎调用外部应用的 API 接口；

3)I3 接口，它是引擎为引用外部资源定义的回调接口，业务系统实现这些接口后，引擎就能够使用业务系统提供的各种资源。

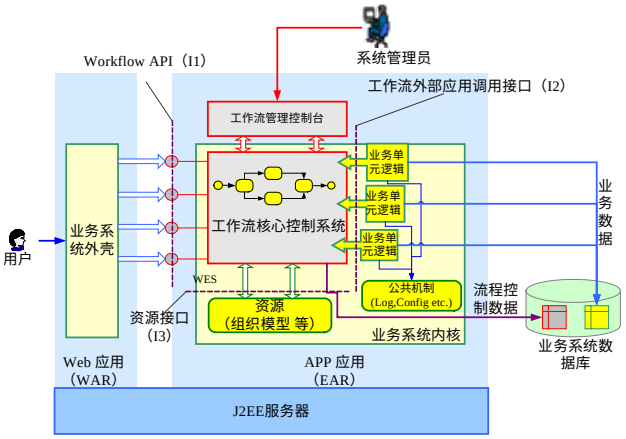


图 1　总体架构

作者简介：高　鹏(1975－)，男，硕士、系统架构师，主研方向：无线数据业务领域相关技术，工作流技术

收稿日期：2005-11-30　**E-mail：**gaopeng@aspire-tech.com

1.2 内部结构

引擎的内部结构如图 2 所示。

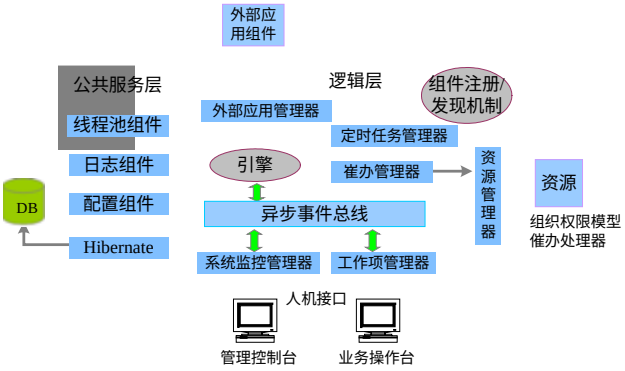


图 2 内部结构

总体上，引擎分为公共服务层和逻辑层，公共服务层提供日志、数据库访问等基础功能；逻辑层完成引擎的逻辑功能，包括几大组件：

(1)调度核心 负责解析 XPDL 过程定义，管理过程实例生命周期并调度执行，同时向其它组件指派工作，相当于引擎系统的 CPU；

(2)异步事件总线 引擎系统中的所有异步通讯都通过该总线以事件的方式进行；

(3)系统监控&管理器、工作项管理器 这 2 个组件构成了引擎的管理控制台，管理员通过和这 2 个组件交互，实现对引擎的监控和管理；

(4)外部应用管理器 负责代理引擎系统调用业务应用逻辑，并负责控制事务边界；

(5)资源管理器 管理外部资源，代理引擎系统使用外部资源；

(6)定时任务、催办任务管理器 提供时间任务支持机制，如定时完成、超时催办等；

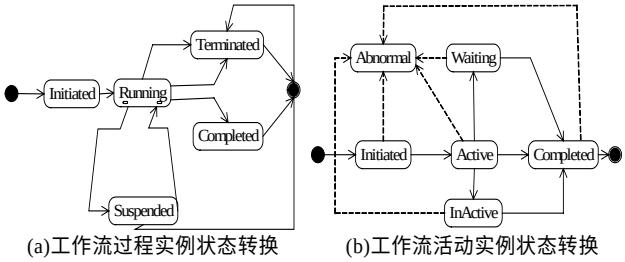
(7)容器 引擎系统提供了一个管理组件的容器，提供组件注册/发现机制，管理上述组件。任何对上述组件的使用均须从容器入口。

以下将对本引擎系统的关键技术和系统特色进行介绍。

2 关键技术

2.1 系统架构模式

总体架构设计上，引擎系统采用了“事件驱动的状态模式”。引擎为每个工作流过程和活动分别定义了一个状态机，状态机的状态变迁由事件触发，实例在不同的状态下，对到达的事件有不同的响应行为。外部系统或人与引擎交互时，将产生各种事件，触发状态机运转。过程/活动的状态转换如图 3 所示。



(a)工作流过程实例状态转换　　(b)工作流活动实例状态转换

图 3 状态机

120 当一个业务流程启动时，引擎系统将根据流程定义创建一个流程实例，该流程实例将基于上述过程实例状态机走完整个生命周期。

(1)过程实例状态说明

1)Initiated 过程实例已经创建，初始活动已经创建并处于 Initiated 状态。等待调度运行；

2)Running 过程实例处于运行中，当前活动实例可以处于任何状态。只有在本状态下才能运行新活动实例；

3)Suspended 过程实例挂起，当前活动实例可以处于 InActive/Waiting/Initiated/Abnormal 状态，可恢复到 Running 状态。本状态下不能运行新的活动实例，也不允许有 Completed 状态的当前活动实例存在，Completed 状态的活动在过程实例挂起前必须被处理完成：要么后继活动已经创建(处于 Initiated 状态)，要么已经是最后一个活动，否则本过程实例自动转换到完成状态(Completed)；

4)Completed 过程实例完成，当前活动实例是结束点活动并且该实例处于 Completed 状态(只能有 1 个结束点)。通常在该状态下，过程的所有活动实例均处于完成状态，过程实例可以从内存中清除；

5)Terminated 过程实例终止，当前活动实例可以处于 InActive/Completed/Waiting/Abnormal 状态，过程实例及其所有活动实例从内存中删除；

6)Locking 过程实例死锁，没有当前活动时，过程进入本状态。进入本状态说明过程定义存在纰漏，导致过程运行时，所有后继分支均无法满足，从而进入流程死锁状态，这是一个异常状态。

(2)过程状态转换说明

1)只有过程实例在 Running 状态时，可以接受流程管理员转换到 Suspended 和 Terminated 状态的指令。管理员通常在希望暂停一个流程实例或者手工结束一个流程实例时会使用这些指令；

2)所有新创建活动实例的运行都必须在过程实例的 Running 状态下；

3)过程实例进入 Suspended 或者 Termiated 状态时，将发送状态改变事件到工作项管理器，工作项管理器对当前活动的相关工作项作相应的处理；

4)Running/Suspended 状态下，当前活动完成而找不到相应的后继活动时，过程进入 Locking 状态。

过程实例运行过程中，引擎系统会根据过程定义不断地创建和销毁活动实例，来完成过程的逻辑功能。活动实例也是基于状态机走完其生命周期。

(3)活动实例状态说明

1)Initiated 活动实例已经创建，执行条件满足；等待系统调度执行(通常是等待线程)；

2)Active 活动实例处于运行中，可以是正在执行自动任务、执行活动内部操作、调用外部应用、正在创建子过程等；

3)Waiting 活动实例处于等待状态，等待子过程返回；

4)InActive 活动实例处于等待状态，等待工作项返回；

5)Completed 活动实例完成，本活动终止；

6)Abnormal 活动实例异常，当发生系统级异常以至于无法判断流程下一个转向时，活动进入本状态。本状态下活动僵死，需要人工干预处理。子过程 Terminal 时，父活动应转换到本状态，以区分子过程正常完成。

(4)活动状态转换说明

1)过程实例在 Running 状态收到活动创建成功事件时，将活动状态从 Initiated 状态跃迁为 Active；

2)活动状态转换不受外部命令(挂起、终止)的直接影响，外部命令通过影响工作项状态或子过程的状态而间接影响活动状态。如挂起命令可能导致活动长期处于 InActive 状态(因工作项被挂起)或者 Waiting 状态(因子过程被挂起)；

3)活动创建子流程时异常进入 Abnormal 状态后，子流程应抛弃，即子流程执行结果不能影响活动状态。

过程和活动实例的状态跃迁均由事件触发，状态跃迁的结果可能又产生新的事情，推动新的跃迁，或者产生人工项。引擎系统定义的主要事件如：

(1)工作项创建事件 活动进入 InActive 状态时将发送本事件，要求工作项管理器创建对应的工作项；

(2)工作项完成事件 用户完成工作项时，工作项管理器将发送本事件，触发 InActive 的活动状态跃迁；

(3)活动完成事件 活动实例结束时发送本事件，指示引擎推动流程流转，创建下一个活动实例；

(4)请求暂停过程事件 管理员通过管理控制台发送本事件要求暂停特定的过程实例运行；

(5)请求终止过程事件 管理员通过管理控制台发送本事件要求终止特定的过程实例运行；

(6)请求激活过程事件 管理员通过管理控制台发送本事件要求激活被暂停过程实例运行；

(7)子过程完成事件 子过程运行结束发送本事件通知父活动。

工作流引擎是一个内部逻辑非常复杂的系统，不良的设计很容易造成今后修改维护的困难，而该架构模式使得本引擎系统内部组件职责分明，高度松耦合，同时又确保逻辑清晰明了。事实证明，在后继不断为引擎追加新特性的过程中，该架构模式使得系统始终高效、可控。

2.2 高性能线程池技术

为提高引擎系统的处理性能，应付大并发访问，我们引入了线程池技术。线程池组件包含 3 个逻辑实体：任务队列，空闲线程池，忙线程池。线程池组件定义了任务接口，任何实现了任务接口的对象均可被作为任务实体，添加到任务队列；对队列中的每个任务，线程池组件从空闲线程池中抓取一个线程处理，正在处理任务的线程进入忙线程池，处理完成后又回到空闲池，整个线程池大小可配置，并会在运行时根据系统忙闲状态作自动智能调整。组件的使用者只需要定义一系列的任务对象，并加入到任务队列中，就可自动由线程池组件执行并行处理。

此外，为保证关键任务的执行，采用了多线程池实例方案，分离关键任务的执行线程池和普通任务的执行线程池，确保关键任务的线程资源不受普通任务影响。

2.3 循环竞选模型支持技术

2.3.1 问题背景

工作流管理联盟(WFMC)发布 XPDL 语言提供了基本的控制结构：顺序(Sequence)，分支(Split)，会聚(Join)。但没有提供对循环结构的支持。这使得在使用这些基本结构定义具有循环特性的复杂过程时，可能发生逻辑悖论。悖论发生在以下场景：

(1)循环入口点活动同时也是其它或会聚(Xor Join)点，如图 4；

(2)当循环体中，包含“ 与分支(And Split)～或会聚(Xor Join)”配对的模型，称之为“ 竞选模型 ”，如图 5 所示。

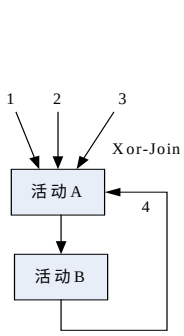


图 4 多汇集模型

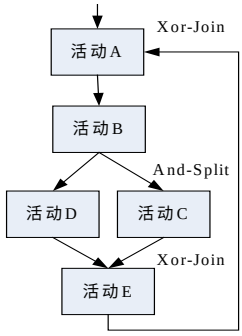


图 5 循环带抢答模型

在图 4 中，活动 A 是或会聚(Xor Join)活动，要求 1、2、3 任何一个流转到达时，活动 A 应该被创建并执行，同时应保证活动 A 不能被重复创建；即如果 1 到达触发活动 A 被创建执行，则随后如果还有 2、3 流转到达，那么活动 A 不应该再被创建执行。即对 1、2、3 到达而言，A 不可重入。同时活动 A 又是循环入口点，对到达 4 而言，活动 A 应可重入，否则循环无法继续。

在图 5 中，活动 A、活动 E 是或会聚(Xor Join)活动，活动 B 是与分支(And Split)活动。从活动 B 到活动 E 形成“ 竞选模型 ”：活动 B 执行完成后同时产生两个活动 C 和 D，活动 C 和 D 同时执行，先执行完成的将触发活动 E 创建并执行，谓之竞选成功；后执行完成的将不会再次触发活动 E 创建并执行，谓之竞选失败。

存在的问题有以下 2 点：

(1)XPDL 规范没有提供标志指示或会聚活动是否可重入。对图 5 而言，显然要求活动 A 是可重入的，否则循环无法继续；另一方面，要求活动 E 是不可重入的，否则失去竞选的意义。无法区分可重入活动和不可重入活动必然导致该过程无法正确执行；

(2)即使可以识别或会聚活动可重入性，上述过程仍然存在逻辑悖论。对图 5，假定标识活动 A 可重入，活动 E 不可重入。在过程生命周期内，循环体可能被多次执行，但活动 E 不可重入必然导致过程在第 2 轮循环执行时，被堵塞在活动 E 上，造成过程死锁，而如果标识活动 E 可重入又违反“ 竞选模型 ”规则。对图 4，标识活动 A 可重入保证循环能执行，但无法约束 1、2、3 到达仅启动活动 A1 次；反之可以保证 1、2、3 到达仅启动活动 A1 次，但堵塞循环。

本系统定义以下术语：

(1)如果一个活动在单个过程实例生命周期内可以多次创建和执行，则该活动称为可重入；

(2)如果一个活动在单个过程实例生命周期内最多可以创建和执行 1 次，则该活动称为不可重入；

(3)通常一个过程中的活动都是可重入的，但对于或会聚活动，因存在竞选模型，某些情况下会要求其不可重入，区分或会聚活动的可重入性是本系统引入“ 可重入性 ”定义的目的。

在当前工作流技术的相关研究中，较少提及对复杂循环模型的解析，对上述问题，还没有发现通用的解决方案。

2.3.2 解决方案

本方案的目的就是针对上述问题进行的，通过模型转换和属性扩展，提供了一个完备的普适方案，使得复杂循环过程可以完全正确地由 XPDL 基本控制结构表达。方案包括两部分内容：

(1)活动可重入性识别 解决上述问题 1；

(2)模型转换解决上述问题 2。

方案通过引入或会聚活动可重入性定义，以过程实例生命周期内活动可以被创建和执行的次数定义活动可重入性，以区分循环入口会聚活动与竞选模型出口会聚活动。通过对 XPDL 活动定义增加扩展属性，标识活动的可重入性。在引入可重入性标志的基础上，进行模型分析转换。

对图 4 展示的逻辑问题，本质上属于对活动 A 有 2 种可重入性要求，通过模型分拆可解决本问题：将活动 A 分拆成 2 个活动，一个或会聚路由活动，具备不可重入特性，接受 1、2、3 到达；另一个或会聚活动，执行活动 A 的操作，具备可重入特性，接受上一个或会聚路由活动输出，以及作为循环入口。

对图 5 展示的逻辑问题，从竞选模型内部看，要求出口会聚活动是不可重入的；从竞选模型外部看，整个竞选模型应该是可重入的，本质上，这是一个可重入的作用域或者说范围问题。通过引入 XPDL 子过程，可以巧妙地解决活动可重入的范围问题：将竞选模型作为子过程定义，竞选模型入口与分支(And Split)活动作为子过程的首活动，出口或会聚(Xor Split)活动作为子过程的尾活动。定义为子过程后，竞选模型出口或会聚活动将仅在子过程生命周期内保证不可重入性，从而保证“竞选”有效；而整个竞选模型是可重入的，当循环时，每重入一次，就创建一个新的子过程实例，子过程实例间无相关性。

该方案具备如下特点：

(1)遵循 XPDL 定义。任何支持 XPDL 的工作流产品均可使用本方案实现对循环模型的完备支持；

(2)基于模型转换，实现简单。仅需要识别活动可重入性，即可使产品具备支持复杂循环过程的功能；

(3)强制结构化流程。由于过程具备单入口单出口特点，因此以子过程定义竞选模型强制过程定义者须保证各种复杂竞选模型的单入单出特性，这符合结构化流程的特点，避免不合理的分支跳转，其作用类似于对程序语言禁用 goto 语句。

2.4 基于状态机的自恢复技术

工作流系统负责业务流程的自动运行，运行期将产生重要的过程相关数据和工作流状态数据，同时过程运行期间，已经执行完毕的活动对外部系统产生的影响基本上不可回滚。因此必须具备系统一旦崩溃后的自动恢复能力，保证系统能恢复到最近正确状态，否则业务系统和工作流系统的数据完整性和一致性将被破坏。系统意外情况包括 2 种：系统整体崩溃(包括宕机重启、网络异常等)，单个流程实例意外异常(如过程死锁、活动僵死等)。

本引擎系统基于状态机自触发机制来实现系统恢复：当恢复过程被触发时(系统重启/界面指令触发)，系统会根据各实例当前存储的上下文信息和状态做出智能决策，自动触发相关事件的发送，推动状态机继续运转，系统的上下文信息、状态信息等数据通过数据库提供可靠存储。

2.4.1 自恢复原理

过程实例运行过程中，在某些状态点上，外部数据和引擎数据将刚好达到一致(比如上一个活动刚刚完成，下一个活动尚未开始)。在这些状态点上，引擎将过程实例相关数据写入持久层(数据库)，如果系统宕机，则重启后可以从数据库将实例恢复到上述最近的状态点上，由于这些状态点上的数据是一致的，因此实例恢复出来是正确的。这些状态点称之为“可恢复点”。

基于上述原理，引擎的恢复方案将包括 2 部分：

(1)运行时，在可恢复点持久化实例，以备恢复；

(2)恢复时，从持久层将实例恢复到最近的可恢复点，并触发它继续执行。

恢复过程的重点，在于如何触发恢复出来的实例继续执行。由于引擎是事件驱动的，因此触发操作就对应到发送特定的事件。而发送事件时，要考虑以下因素：

(1)已经生成事件的情况。系统宕机时，实例的相关事件可能已经发送，但尚未处理；

(2)成事件的情况。宕机时，实例相关事件尚未发送，或者已经被后继操作处理，但后继事件尚未生成。

系统恢复时，要优先处理完已经生成的事件，触发对应实例执行；然后再对没有触发事件的待恢复实例做自动恢复：重新发送相关事件。

系统定义了以下可恢复点：

(1)过程实例处于 Running 状态，所有当前活动实例为非 Active 状态；

(2)过程实例处于 Suspended 状态，所有当前活动实例处于非 Active 状态；

(3)过程实例处于 Completed / Terminated / Locking 状态。

2.4.2 总体恢复过程

系统恢复过程包括恢复引擎线程池、引擎事件队列、引擎过程实例、完成事件订阅等。过程实例恢复时，为防止循环依赖，遵循父子过程互不强关联的原则：父子过程无引用耦合，完全通过事件交互。

2.5 基于活动集的高级事务支持

为支持工作流跨活动事务，我们引入了 XPDL 活动集概念。引擎为定义在一个活动集中的活动序列提供事务支持，保证 ACID 特性。

此外，为解决通常工作流产品所回避的事务一致性问题：如何保证业务逻辑事务和工作流本身控制数据事务的整体 ACID 特性？采用了嵌入式工作流设计，确保工作流引擎系统和业务系统可以共享同一个数据库连接，以数据库技术来实现事务性支持。实现上，将工作流引擎平台化，业务系统使用工作流引擎时，将从引擎系统获取数据库连接，而不允许直接连接数据库，这样确保引擎系统可以对数据库连接进行控制，从而实现业务数据和引擎控制数据的同一事务保证。

对业务系统而言，实现活动逻辑时，只需要关注业务逻辑，事务边界由引擎系统自动控制，业务开发更加容易。

3 结束语

面向电信级的工作流系统和面向企业级的工作流系统关注的重点是不一样的，企业由于业务流程变化快等因素，更多的是希望系统能够灵活、易变，因此对工作流系统的要求更多的是功能性方面；而电信级系统通常是服务于广泛的大众，一旦发生宕机或者数据丢失，影响范围将非常广泛，甚至会升级成政治性事件，因此电信级系统首要关注的是系统的稳定、可靠等非功能性因素，在这样的系统中引入工作流技术，需要从技术架构、方法论等诸多方面进行探索和创新，本文仅仅是一个开始，进一步的研究有待进行。

参考文献

1 WFMC. Workflow Process Definition Interface——XML Process Definition Language[EB/OL]. http://www.wfmc.org/, 2002-10.

2 WFMC. The Workflow Reference Model[EB/OL]. http://www.wfmc.org/, 1995-01.