9_FireFlow 技术原理

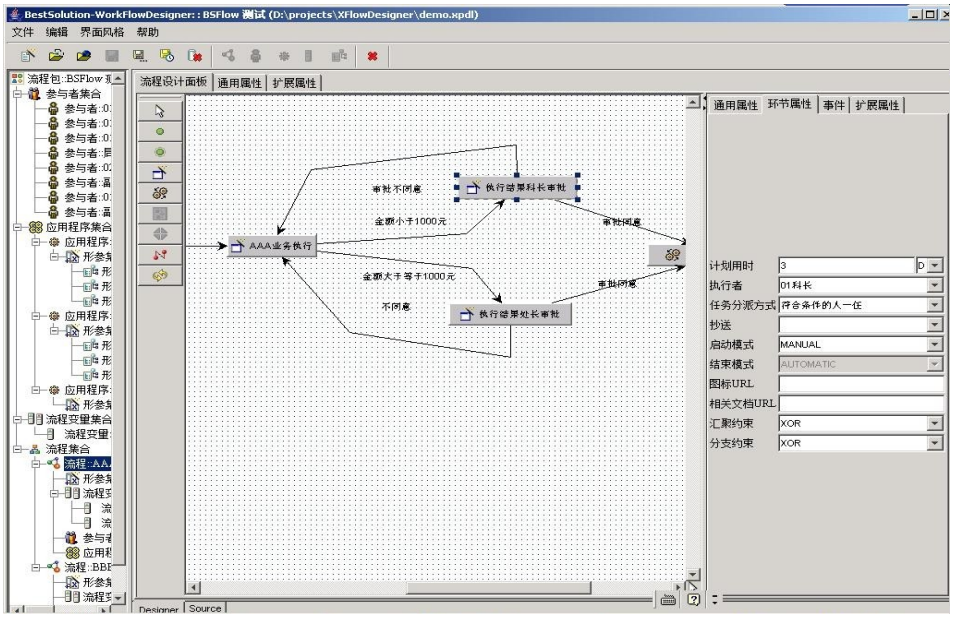
内容目录

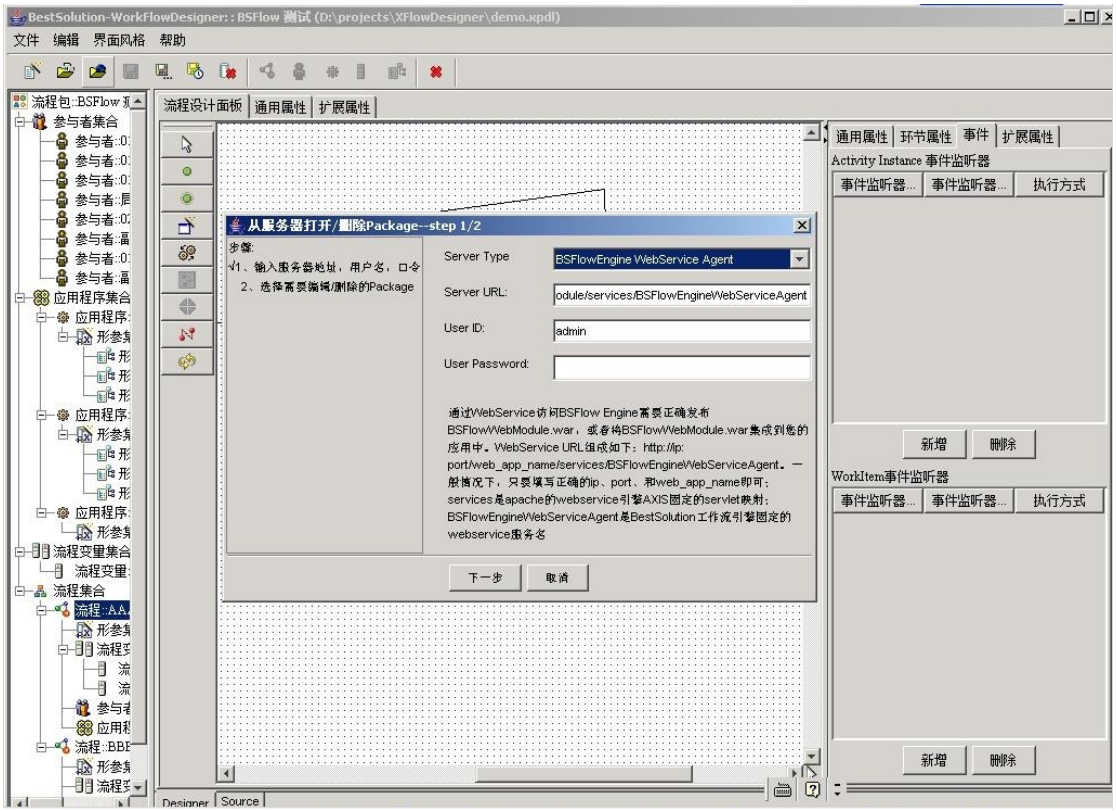
1、前言 2
2、预备知识 3
3、从流程建模说起 4
4、工作流逻辑网 7
4.1、相关定义 8
4.2、对 xpdl 的改造 9
5、工作流语义 10
6、FireFlow 工作流模型?? 11

1、前言

2003 年的时候，我曾经写过一个工作流产品，该产品叫 Bestsolution WorkFlow（Bestsolution？有点吹牛，哈哈），后来由于各种原因，没有继续写。后来，我在项目中用了 jboss 的 jbpm3.x，这个产品最灵活的之处就是他的事件机制和相关的 handler。在我当时开发的 Bestsolution WorkFlow 中也设计了一个非常不错的事件子系统。

在这里，我还是想把过去的“成就”show 一下 :) 。





和目前绝大多数 Workflow Engine 一样，Bestsolution Workflow 的最大毛病在于：**引擎逻辑不具备数学上的严密性**。

这个问题产生的原因首先是因为 xpdl 本身不具备数学上的严密性(个人观点^_^)。其次是因为本人对 petri net 的理论知识没有掌握。

这种不严密性导致的后果是，用 xpdl 建立的业务流程模型不能验证；engine 无法正确完成诸如“or 汇聚”、“and 汇聚”、router、子流程等复杂逻辑。

我在阅读 Jboss 的 jbpm 相关文档的时候，对他提出的 Graph Oriented Programming（面向图的编程）十分认同。但是 jbpm 的文档好像也没有论述引擎的数学原理。

今天，我重新写一个工作流引擎，主要是想弥补一下这一方面的缺陷。另外，我决定开源，让有兴趣的朋友都来研究 workflow。

那么为什么取 FireFlow 这个名字，而不沿用 Bestsolution Workflow 呢？因为“Bestsolution”相关的域名都被别人注册了；另外，既然是开源的，还是不用旧的名字为好。

与 Fire Flow 相关的所有代码、文档都在 http://code.google.com/p/fireflow/。

本人水平有限，错误之处，请大家不吝赐教。

2、预备知识

我认为，具备如下知识将会更好地理解 Fireflow 工作流。

1、WFMC 的工作流参考模型

2、XPDL （xml 流程描述语言）

3、离散数学相关知识

4、petri 网相关知识

在这里特别提示：fireflow 引擎的核(kenel)是根据袁崇义教授的著作《Petri 网原理与应用》中相关理论设计的。大家可以看看这本书。另一本很好的 petri 网著作是吴哲辉教授编写的《Petri 网导论》

3、从流程建模说起

我们以一个简单的案例讨论一下流程建模。

案例 3-1：某审批业务的业务流程如下，受理科接受用户的申请，并打印相关的回执；受理后的业务资料移送给审批科进行审批；若审批通过，则继续移交给制证科制作相关的文书和证件；最后业务材料由档案室归档。在这里暂不考虑审批不通过的情况，也不考虑文书和证件发放的工作。

通常，我们有如下方法对这个业务进行建模。

第一种方法：我们可以用 UML 的活动图描述这个业务，如下：

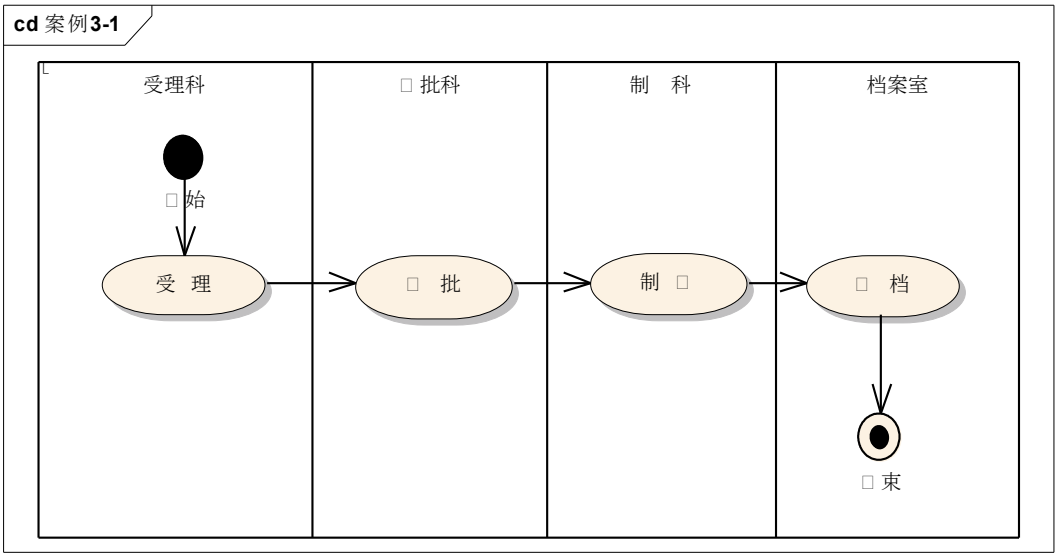


图 3-1

UML 活动图从业务层次精确描述了这个案例，但是，这种建模一般只用在需求分析阶段。因为现在好像还没有一个引擎可以去执行它，虽然他也可以转换为 xml 文件。

因此，活动图是概念层次的建模，离可执行的工作流建模有很大距离。

第二种方法：我们可以用工作流工具对这个业务进行建模。在这里，我用自己的 Bestsolution Workflow Designer 建立的流程模型如下图。

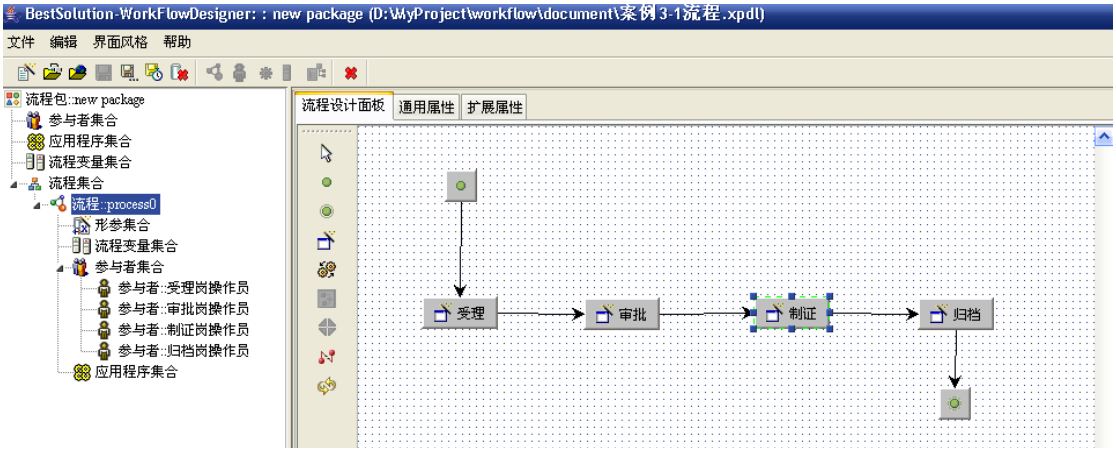


图 3-2

从图 3-2 可以看出，用工作流设计器建立的流程模型与 UML 活动图类似 。唯一的区别在于从图 3-2 导出的流程定义文件（xpdl1.0 格式）是从代码执行的角度描述业务的，从而比较方便开发出一个所谓的工作流引擎来执行这个流程定义文件。

然而，WFMC 定义的工作流建模语言，以及其他和多工作流建模语言都没有回答一个根本性的问题：

用这个建模语言定义的流程真的能执行吗？能正确无误地执行吗？

从我的经验看，xpdl 还缺乏逻辑严密的语义，在稍微复杂一点的情况下，它定义的流程很难正确执行。

如果我们把“案例 3-1”所描述的审批系统分为两部分：工作流逻辑子系统和业务逻辑子系统。那么“图 3-2”建立的工作流模型完全没有描述出工作流逻辑子系统需要完成哪些工作，业务逻辑子系统需要完成哪些工作；从这个意义上说，它和“图 3-1”的 UML 活动图没有什么本质区别，都是很笼统的一个业务流程而已，只是 xml 文件的 DTD 有点不同。

我个人认为，正是这种在两个子系统之间的职责的模糊与混乱导致工作流引擎无法进行正确的计算。

下面，我想用第三种方法对“案例 3-1”进行建模。这种建模在本章节还停留在图示阶段。目的是为了说明我所认为的工作流描述语言应该具备的特性。

我们把审批系统分成工作流逻辑子系统和业务逻辑子系统。工作流子系统的操作用圆圈表示，业务逻辑子系统的操作用方框表示。如下图 3-3。

系统的执行过程如下：

首先，工作流逻辑子系统启动一个新的业务流程实例，然后启动一个新的任务“受理”，并将控制权交给业务逻辑子系统，由业务逻辑子系统完成受理工作。

第二步、受理完成后，控制权交给工作流逻辑子系统。由该子系统决定下一步的业务操作。工作流逻辑子系统根据流程定义以及相关流程变量的计算和判断，得出下一步的工作是“审批”，于是启动之，并将控制权交给业务逻辑子系统。

第三步、第四步以及后续步骤与之类似，直至最后工作流逻辑子系统计算得出流程实例应该结束，于是结束该流程实例。

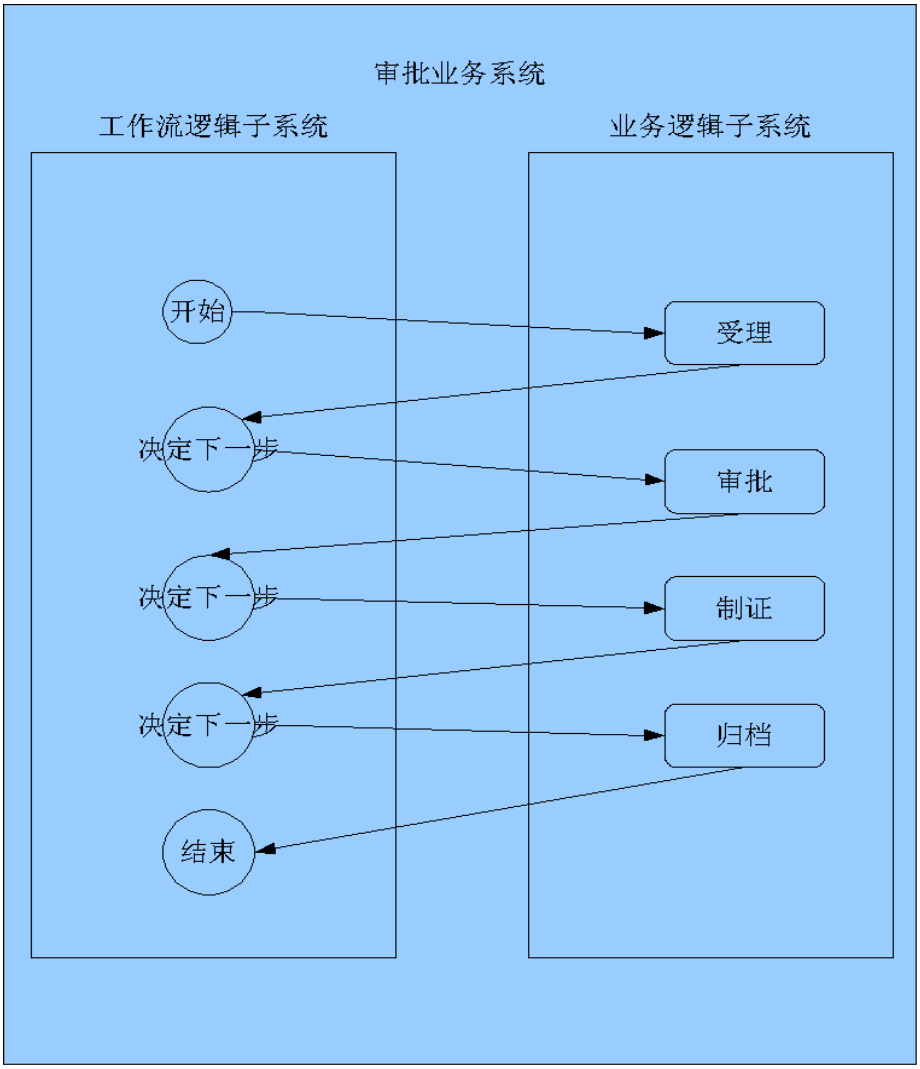


图 3-3

图 3-3 的建模似乎比图 3-2 没有什么优越性，反而有点复杂，至少图形变得复杂了。但是我们如果考虑下面这种情况，那么图 3-3 的建模的好处就非常明显了。在案例 3-1 中，我们没有考虑审批不通过的情况，现在把这种情况考虑进去，形成案例 3-2

案例 3-2：在案例 3-1 的基础上考虑审批不通过的情况。如果审批不通过，则需要告知申请人，然后结束业务流程。

如果我们用 xpdl 对案例 3-2 键模，则图形表示如下图 3-4

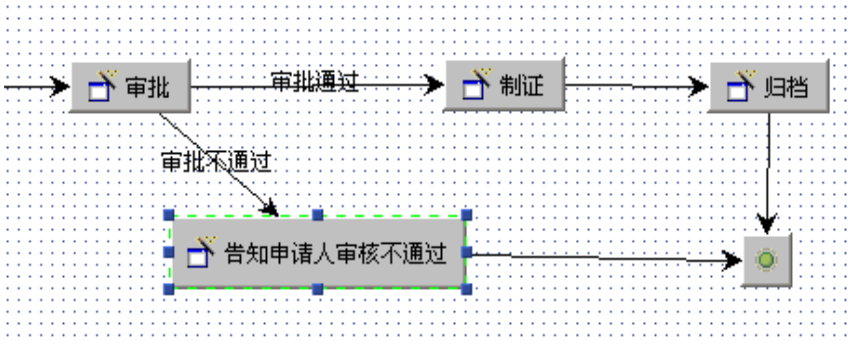


图 3-4

图 3-4 的问题在于，业务逻辑和工作流逻辑叠加在一起，没有精确地指出到底在什么时候，由谁来决定审批之后的操作。我们可以理解为由“审批”这个环节决定后续环节是“制证”还是“告知申请人审核不通过”，这种情况下审批环节既代表了业务操作，又代表了工作流逻辑操作。当然，还有其他的执行方式，例如：计算连接“审批”到“制证”之间的狐（xpdl 称之为转移）以及联结“审批”到“告知申请人审核不通过”之间的狐得值来决定下一步的操作。总而言之图 3-4 并没有把业务说清楚。

如果我们用第三种方法，也就是图 3-3 中的方法对案例 3-2 进行建模，那么这个模型的局部如图 3-5。

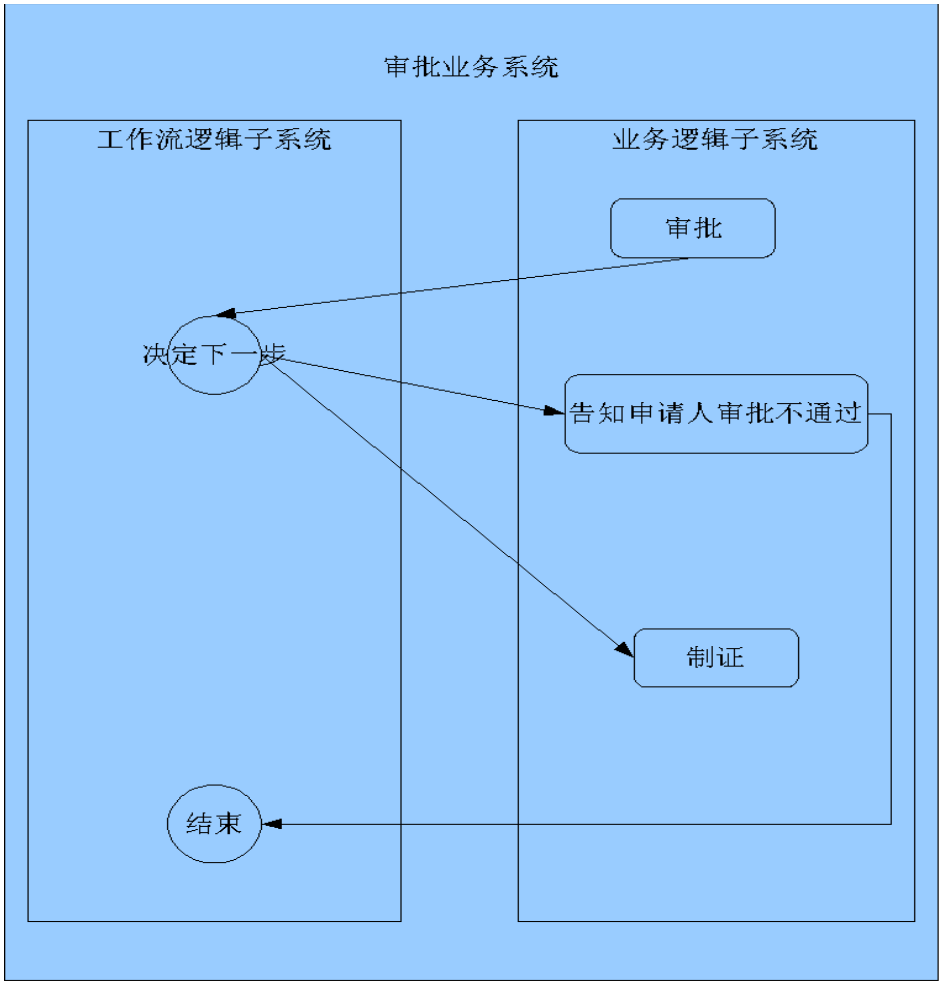


图 3-5

在图 3-5 中，工作流逻辑和业务逻辑分得非常清晰，审批之后执行哪个业务操作是由工作流逻辑子系统的一个“操作”决定的。业务逻辑子系统中的“审批”操作仅仅负责完成业务特定的逻辑，其他的与之无关。

我们讨论的第三种建模方法就是 Petri 网，下面我们讨论其数学定义，以求精确。

4、工作流逻辑网

4.1、相关定义

注：要深入理解该内容，请阅读袁教授的著作，我就不 copy 书中的内容了。另外，我这里写的是按照我的理解转换成通俗语言，如果有错误请大家及时指正。

定义 4-1：

工作流是对业务进程的形式化描述，包括描述任务之间依赖关系（因果依赖与规章依赖）的工作流逻辑和在此基础上增加显性内容的工作流语义。显性内容是指影响流程执行路径的业务数据、操作员的决定等等。

定义 4-2：

工作流逻辑(网)只关心任务之间的依赖关系，包括因果关系和规章依赖，不关心任务的具体操作内容。工作流逻辑网符合如下规则。

规则 1、工作流逻辑网是由业务任务集合 T（为了便于理解，可以把 T 当作 Task 的简写吧）和流程同步器集合 P 构成的加权有向图。即 Workflow Logic Net Σ=（P，T；F，K，W）。其中 F 是 T 到 P 或者 T 到 P 的所有的有向边的集合。K 是 P 的容量的集合，所谓容量，就是这个同步器能够容纳的 Token 的数量。W 表示边上的权。

流程同步器代表工作流子系统中决定下一步路由的逻辑，例如顺序、分发（split）、汇聚(join）等等。

在图示中，我们用矩形表示业务任务 t（t∈T），用小圆圈表示 p（p∈P），用箭头表示 f（f∈F），用箭头上的数字表示 w（w∈W），用圆圈中的数字表示 k（k∈K）。

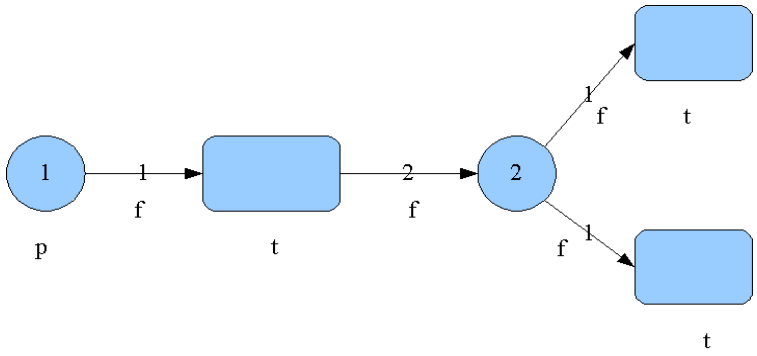


图 4-1

规则 2、任何 p 与 p′之间都没有边相连；任何 t 与 t′之间都没有边相连。这个规则可以理解为，整个系统的控制权只能在工作流子系统与业务子系统之间交互，不可以将控制权从一个业务任务直接转移到另一个业务任务，也不可以将控制权由一个同步器直接转移给另一个同步器。

规则 3、对于任何一个业务任务 t∈T，都有·t≠ø 且 t·≠ø。其中·t 表示任务 t 的输入集，t·表示任务 t 的输出集，由规则 1 知 t 的输入集和输出集都是{p|p∈P}。该规则通俗理解为所有业务操作的发生权都是由流程同步器授予，业务完成后交还给同步器。

规则 4、任何一个业务任务 t∈T，都有|·t|=1 且|t·|=1。这个规则表示，所有的任务只有一个输入且只有一个输出。如下图 4-2。如果|·t|>1 或者|t·|>1，则表现为图 4-3 的形式，则说明 t 除了完成业务操作之外，还要完成诸如“分发 split”和“汇聚 join”的操作，显然是越疽代庖，分发汇聚操作时流程同步器的职责。

图 4-2

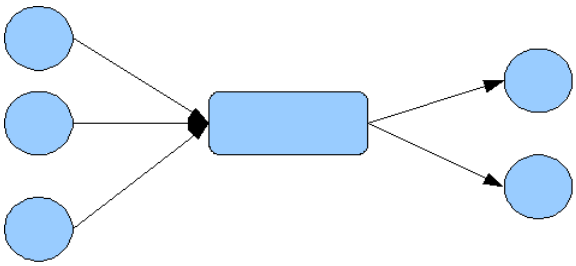


图 4-3

规则 5、对于任何 p∈P ，如果 ·p = ø ，则|·p|=1。我们称输入集为空的同步器为“**起始节点**”，|·p|=1 的意思是，整个网中，允许且只允许存在一个起始节点。在本文中，我们用下面的图形表示起始节点。

图 4-4

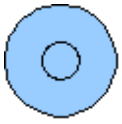


图 4-5

如果 p· = ø ，则称 p 为结束节点，整个网中可以出现多个结束节点。我们用图 4-5 表示结束节点。

规则六、工作流逻辑网是一个连通图；不允许出现环。至于实际业务中要求的“循环操作”，我们通过工作流语义解决，不通过模型解决。

？？工作流逻辑网的可达性如何定义？

规则 7、对于任何 p∈P ，p 的容量 K(p) = |·p| * |p·|。特别地，对于起始节点，其容量等于 |p·|；对于结束节点，其容量等于 |·p|

规则 8、对于任何 f∈F，如果 f∈{(t,p)|t∈∧p∈}，则 W(f) = K(p)/|p·|；如果 f∈{(p,t)|p∈∧t∈}，则 W(f)= K(p)/|·p|

定义 4-3：

工作流逻辑网的初始状 M_0：对于起始节点 M_0（p）=K(p)，其他任何同步器及中止节点 M_0（p）=0；

工作流逻辑网的中止状态 M_{end}：对于中止节点 M_{end}(p)=K(p)，其他任何同步器及起始节点 M_0（p）=0；

定义 4-4：

能够从起始状态开始运行，得到中止状态的工作流逻辑网是正确的。

4.2、对 xpdl 的改造

我认为 xpdl 比较好的方面是其大多数模型对象非常适合业务需求，缺点是工作流模型缺乏数学上的严密性。因此在 FireFlow 中保留其优点，改进其不足。

对 xpdl 中的元素，FireFlow 重新定义如下表。

序号	xpdl 元素	工作流逻辑网元素	备注
1	Process	Process	概念等同
2	Sub Process	Sub Process	概念等同
3	Activity	t	概念类似
4	Transition	f	意义完全不同，xpdl 中的 transition 可以有大量的属性。f 除了“权”外没有别的属性。
5	And-Join, Or-Join And-Split, Or-Split	p	
6	Iteration	无对应的结构	通过工作流语义解决。
7	Pre-Condition, Post-Condition	t 的执行条件	

注：主要变化，取消 Router Activity，增加 Synchronizer，重新定义 Transition

5、工作流语义

定义 5-1：

“工作流语义”： 工作流语义是指在工作流逻辑网的基础上增加的显性内容，如影响流程执行路径的业务数据、操作员的决定等等 。

定义 5-2：

“发生”：发生是指工作流逻辑网中的 p 或者 t，在没有加载业务逻辑的情况下的一次执行。任何 p 或者 t 是否具有发生权，是由 Petri Net 的相关规则来定义的，在这里不细说。

从 4.1 节相关的定义可知，一个正确的工作流逻辑网从初始状 M_0 执行到中止状态 M_{end} 后，对于任何 p 或者 t，他都发生过一次，其仅发生了一次。

显然，在实际业务中，同一个业务流程每个业务任务 t 并不是都要执行。在给定的案例中，有的流程分支上的任务要被执行，有的分支上的任务不被执行，有的还可以被执行多次（如：重做或者循环）。

定义 5-3：

实例化：如果 t 对应的实际业务逻辑被执行了，我们称 t 以及 ·t 被实例化；

下面我们讨论一下 t 和 p 可以被实例化的充分必要条件。

首先，对于任何一个 t，他对应的业务逻辑包含两部分：执行条件和执行体。以案例 3-1 种的制证业务为例，执行条件是“审批意见等于‘同意’”，执行体是“制证”。示例如下图 5-1：

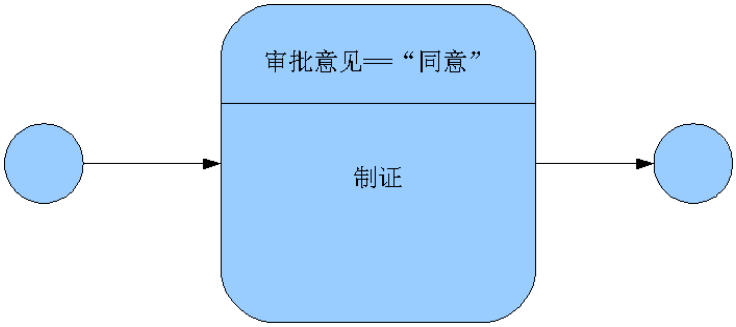


图 5-1

显然，t 的执行条件计算结果等于"true"是 t 可以被实例化的必要条件；t 能够被实例化的另一个必要条件是·t 已经实例化。所以有：

定义 5-3：

t 能够被实例化的充分必要条件：t 已经获得发生权，且·t 已经实例化，且 t 的执行条件计算结果等于"true"。

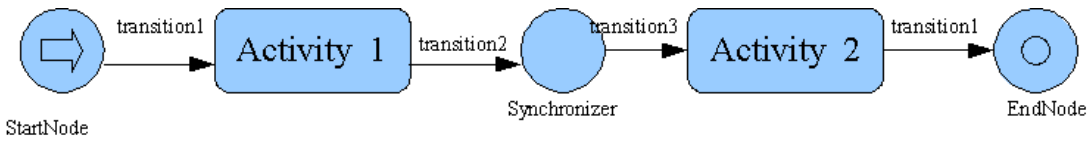
那么 p 能够被实例化的充分必要条件是什么呢？我们定义如下

定义 5-4：

p 能够被实例化的充分必要条件：p 已经获得发生权，且存在 t∈·p 已经实例化。

6、FireFlow 工作流模型??

一个最简的工作流如下图：



在此图中，以及以后的讨论中，我们不再使用第 4 章和第五章的**工作流逻辑网**的术语，而是采用我们通俗的 Fireflow 工作流模型术语。对照关系如下表

工作流逻辑网术语	Fireflow 工作流模型术语
PetriNet	WorkflowProcess（注意：WorkflowProcess 兼具工作流逻辑网和工作流语义双重身份）
p	Synchronizer(起始节点和结束节点是 Synchronizer 的特例)。
t	Activity
f	Transition

Fireflow 还引入了 Task 概念，一个 Activity 可以有多个 Task。Task 是语义层次的对象。

（文档还在整理中，有点七零八落，请大家参看《Fireflow 设计》第二章)