(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 103049264 A
(43)申请公布日 2013.04.17

(21)申请号 201210545707.1

(22)申请日 2012.12.17

(71)申请人 国电南京自动化股份有限公司
地址 210009 江苏省南京市鼓楼区新模范马路 38 号

(72)发明人 朱海东 李晓东 黄炳良 俞兴进

(74)专利代理机构 南京纵横知识产权代理有限公司 32224
代理人 董建林

(51)Int.Cl.
G06F 9/44(2006.01)

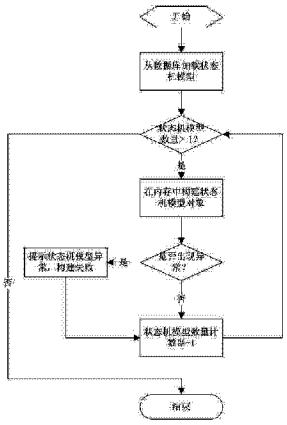
权利要求书 1 页 说明书 17 页 附图 4 页

(54)发明名称

一种通过状态机动态建模实现对业务系统控制的方法

(57)摘要

本发明公开了一种通过状态机动态建模实现对业务系统控制的方法，包括以下步骤：采用 XML 的方式，通过图形化工具，定义并生成业务对象的状态机模型文件；通过状态机模型驱动引擎，动态加载对应的业务对象的状态机模型文件；状态机模型驱动引擎根据该业务对象的状态机模型，对业务对象的状态转换过程和动态行为进行控制。本发明的方法通过状态机动态建模实现对业务对象的行为控制，通过图形化配置来建立业务对象的可运行状态转换模型和动态行为模型，当业务动态行为发生变化时只需要通过状态机图形化定义工具定义新的状态节点或操作规则即可。大幅减少了业务对象的动态行为需求变化带来的代码撰写的工作量，提升开发效率。



CN 103049264 A

1. 一种通过状态机动态建模实现对业务系统控制的方法，其特征是，包括以下步骤：

1）采用 XML 的方式，通过图形化工具，定义并生成业务对象的状态机模型文件；

2）通过状态机模型驱动引擎，动态加载对应的业务对象的状态机模型文件；

3）状态机模型驱动引擎根据该业务对象的状态机模型，对业务对象的状态转换过程和动态行为进行控制。

2. 根据权利要求 1 所述的通过状态机动态建模实现对业务系统控制的方法，其特征是，状态机模型定义完成之后保存为一个 XML 文件保存在硬盘指定的文件夹中，同时将该文件中的 XML 字符串上传到数据库中。

3. 根据权利要求 1 所述的通过状态机动态建模实现对业务系统控制的方法，其特征是，状态机模型的上传和加载采用的是 webservice 的方式，客户端图形化状态机模型定义工具通过访问 webservice 的方式与服务器端进行通信，webservice 提供 SyncModel 接口实现客户端、服务器内存、数据库三者之间的状态机模型同步。

4. 根据权利要求 3 所述的通过状态机动态建模实现对业务系统控制的方法，其特征是，新的状态机模型发布之前，先进行正确性的验证，验证的方法是通过将状态机模型加载到应用服务器端的内容中，在内存建立状态机模型的缓存，这个过程中将对状态机模型的正确性进行验证。

5. 根据权利要求 2 所述的通过状态机动态建模实现对业务系统控制的方法，其特征是，当状态机模型开始使用前，由状态机模型驱动引擎完成状态机模型的加载，加载的过程为：首先从数据库中把正在使用的状态机模型文件查询出来，保存到一个列表中，然后判断列表中的状态机模型数量是否大于等于 1，如果大于等于 1 就将遍历其中的状态机模型，并加装到内存中，构建出状态机模型的内存对象。

6. 根据权利要求 1 所述的通过状态机动态建模实现对业务系统控制的方法，其特征是，将所述状态机模型内部的逻辑关系构建起一个业务对象的状态转换图，这个图是一个有向图，图的节点定义为业务对象的每一个过程状态，图的每一个出度和入度都定义为业务对象的一次转换、转换条件和转换过程中发生的事件；状态机模型驱动引擎按照业务对象的状态转换图对业务对象的动态行为进行控制。

7. 根据权利要求 5 所述的通过状态机动态建模实现对业务系统控制的方法，其特征是，状态机模型被状态机模型驱动引擎加载到内存中以对象的形式实例化，并在状态机模型驱动引擎的协助下进行业务对象实例的动态行为控制。

一种通过状态机动态建模实现对业务系统控制的方法

技术领域

[0001] 本发明涉及一种通过状态机动态建模实现对业务系统控制的方法，属于信息系统技术领域。

背景技术

[0002] 目前，国内外信息系统设计领域基本上采用的都是代码开发的方式来解决业务对象的需求变化问题，这种方法工作量大、见效慢，同时新的代码的加入也会带来系统的风险，增加了测试的工作量。

[0003] 现有的业务对象的状态变化过程的实现方法基本上是采用开发代码的方法，开发周期长，对程序员的开发水平依赖性强，在应对界面需求变化方面灵活性不够。

[0004] 业务对象是面向对象的信息系统中用来实现业务功能处理的核心，是业务逻辑规则在信息处理形态下的主要载体，也是最容易引起需求变化的关键因素之一，在业务开发过程中经常会出现业务对象的行为和状态发生改变的情况，业务对象在每一个状态中都表现出不同的动态行为，经常发生的问题是业务对象从一个状态到另一个状态的转换过程是不确定的，是会根据用户的要求发生频繁变化的，另一方面，在每个状态中业务对象能够表现的动态行为也会因为需求的变化而变化。现有的方法都是采用修改代码的方式来应对这些变化，但这种开发方法周期长、见效慢，并且需要很专业的界面开发程序员才能够胜任该项工作，一个完整的信息系统界面开发往往需要耗费大量的人力物力，并且对系统的稳定性可靠性影响会很大。

[0005] 当前在国内外信息化领域，面向对象的业务系统中业务对象通常分为属性和操作两个方面，这些特征基本上都是在设计过程中事先确定下来的，并开发过程中通过硬编码的方式实现。并且在设计之初就要考虑好有关权限控制等行为特征。当业务需求变化需要对业务对象的行为特征进行调整时，就需要进行二次开发、测试、部署等工作，导致系统的应变能力相对较弱，由此带来的稳定性和需求响应的及时性都较低。

[0006] 一旦用户对开发出来的业务功能不满意或者提出了新的改进要求，开发人员就会面临很繁重的重新开发代码工作，然后还需要对新的代码进行测试、修正和重新安装、部署上线。

[0007] 在一个信息系统建设的过程中，用户对业务功能的变更需求是经常发生的，这使得信息系统的建设中基于代码开发的业务对象动态行为设计变成了程序员的噩梦，因此系统需要解决的一个重要问题就是如何让业务对象需求的变化能够基于配置而不是基于二次代码的开发。

发明内容

[0008] 本发明所要解决的技术问题是提供。

[0009] 为解决上述技术问题，本发明提供一种通过状态机动态建模实现对业务系统控制的方法，其特征是，包括以下步骤：

[0010] 1）采用 XML 的方式，通过图形化工具，定义并生成业务对象的状态机模型文件；

[0011] 2）通过状态机模型驱动引擎，动态加载对应的业务对象的状态机模型文件；

[0012] 3）状态机模型驱动引擎根据该业务对象的状态机模型，对业务对象的状态转换过程和动态行为进行控制。

[0013] 状态机模型定义完成之后保存为一个 XML 文件保存在硬盘指定的文件夹中，同时将该文件中的 XML 字符串上传到数据库中。

[0014] 状态机模型的上传和加载采用的是 webservice 的方式，客户端图形化状态机模型定义工具通过访问 webservice 的方式与服务器端进行通信，webservice 提供 SyncModel 接口实现客户端、服务器内存、数据库三者之间的状态机模型同步。

[0015] 新的状态机模型发布之前，先进行正确性的验证，验证的方法是通过将状态机模型加载到应用服务器端的内容中，在内存建立状态机模型的缓存，这个过程中将对状态机模型的正确性进行验证。

[0016] 当状态机模型开始使用前，由状态机模型驱动引擎完成状态机模型的加载，加载的过程为：首先从数据库中把正在使用的状态机模型文件查询出来，保存到一个列表中，然后判断列表中的状态机模型数量是否大于等于 1，如果大于等于 1 就将遍历其中的状态机模型，并加装到内存中，构建出状态机模型的内存对象。

[0017] 将所述状态机模型内部的逻辑关系构建起一个业务对象的状态转换图，这个图是一个有向图，图的节点定义为业务对象的每一个过程状态，图的每一个出度和入度都定义为业务对象的一次转换、转换条件和转换过程中发生的事件；状态机模型驱动引擎按照业务对象的状态转换图对业务对象的动态行为进行控制。

[0018] 状态机模型被状态机模型驱动引擎加载到内存中以对象的形式实例化，并在状态机模型驱动引擎的协助下进行业务对象实例的动态行为控制。

[0019] 本发明所达到的有益效果：

[0020] 本发明的方法通过状态机动态建模实现对业务对象的行为控制，通过图形化配置来建立业务对象的可运行状态转换模型和动态行为模型，将业务对象的静态属性和业务对象的动态行为进行了分离，当业务动态行为发生变化时可以不用修改业务对象的实体代码，只需要通过状态机图形化定义工具定义新的状态节点或操作规则即可。当用户的业务功能需求发生变化时，只需要调整业务对象的状态机模型，就可以改变业务对象的状态转换方式，实现对业务对象的动态状态控制，通过插入式的脚本和代码注入来实现业务对象的动态行为控制。

[0021] 业务对象的需求变更只需要修改模型配置信息即可，并可以实现动态的模型加载和灵活的版本控制，并通过状态机模型驱动引擎实现对业务对象行为动作和业务状态变化的轨迹记录，使得业务过程可以审计和追溯。

[0022] 本发明大幅减少了业务对象的动态行为需求变化带来的代码撰写的工作量，提升软件的开发效率和软件系统的敏捷性，并降低因此而带来的软件系统风险。

附图说明

[0023] 图 1 为图形化工具定义的报销申请对象状态机模型；

[0024] 图 2 为状态机模型加载的流程图；

[0025] 图 3 为 StateMachineSchema 构建的基本流程；

[0026] 图 4 为状态机模型驱动引擎负责状态机实例的创建和驱动过程。

具体实施方式

[0027] 下面结合附图对本发明作进一步描述。以下实施例仅用于更加清楚地说明本发明的技术方案，而不能以此来限制本发明的保护范围。

[0028] 本发明采用模型驱动的思路，通过状态机动态建模实现对业务对象的行为控制，通过图形化配置来建立业务对象的可运行状态转换模型和动态行为模型，其特征在于：包括以下具体的解决步骤：

[0029] 1）采用 XML（可扩展标记语言）的方式，通过图形化工具，定义并生成特定业务对象的状态机模型文件；

[0030] 2）通过状态机模型驱动引擎，动态加载特定业务对象的状态机模型文件；

[0031] 3）状态机模型驱动引擎根据特定业务对象的状态机模型，对业务对象的状态转换过程和动态行为进行控制。

[0032] 1. 步骤一说明：

[0033] 步骤一的主要工作是定义状态机模型，为方便和直观的进行状态机模型的定义，采用 C# 开发语言开发了图形化的定义工具，创建基于 Xml 的状态机模型文件，图形化工具定义的报销申请对象状态机模型实例如图 1 所示：

[0034] 模型文件的具体形式如下：

[0035]

```
<statemachine>
  <model……>//模型节点
    ……
</model>
  <view……>//视图节点
    ……
</view>
</statemachine>
```

[0036] 整个状态机的模型包含在“statemachine”节点中，该节点包括两个子节点，分别是“model”节点和“view”节点。

[0037] model 节点是状态机模型信息所在的节点，其中包含了特定业务对象的状态转换方式；

[0038] view 节点是状态机的绘图信息所在的节点，其中包含了用户使用工具进行定义时，每种图形元素在屏幕上的显示坐标，view 这部分内容不是本文的重点，不展开描述。下面详细描述 model 节点的设计：

[0039] 按照可扩展标记语言（XML）的规定，flow 节点由属性和子节点构成。model 节点的属性如下：

[0040] 1.1.Model 节点的属性

[0041] -Id :状态机模型的唯一标示符,数字类型 ;

[0042] -Name :状态机模型的名字,用来帮助用户理解和识别,例如 :“报销对象的状态机”;

[0043] -description :状态机模型的详细描述信息,用来说明状态机的作用等信息,帮助用户更清晰的理解和识别状态机模型的具体作用 ;

[0044] -version :状态机模型的版本,由于一个业务需求的变化,同一个业务对象的状态机在过程中需要不断的进行调整,以满足用户提出的新的需求,因此需要通过版本来帮助状态机引擎控制和区别新旧状态机,保证业务对象的正常工作 ;

[0045] -starttime,endtime :状态机的有效时间,某种特殊情况下,需要临时创建一个状态机,只在特定的时间范围内生效,这两个属性的作用就在于此,starttime 定义了状态机的生效时间,endtime 定义了状态机的失效时间,当生效时间和失效时间都为空时,表示该状态机一直有效 ;定义了 startime 和 endtime 时间的状态机模型优先级高于没有定义这两个属性的状态机 ;

[0046] -orgid :状态机模型对应的组织机构编码,当状态机用于一个集团级的信息系统中时,用来区分统一业务对象在不同的组织机构下的不同状态机模型 ;

[0047] -bizid :状态机模型对应的业务对象类型 id,用来描述该状态机是用来控制哪一类业务对象的。

[0048] 1.2.Model 节点的子节点

[0049] 以上对 model 节点的属性进行了描述,下面对 model 节点的子节点进行描述,model 节点的子节点包括 :

[0050] 1.2.1.Lookups 节点

[0051] 该节点用来定义业务对象中枚举类型的属性如何转换成用户能够理解的文字,例如 :当计算机把性别“sex”属性值保存为“0”和“1”时 , 状态机应该将其转换成“女”和“男”显示给最终用户。Lookups 属性可以有多个“lookup”子节点,其结构如下 :

[0052]

```
<lookups>
    <lookup field="sex"……>
      <info code="0" name="女" />
      <info code="1" name="男" />
    </lookup>
    <lookup field="type"……>
      <info code="A" name="本科" />
      <info code="B" name="研究生" />
    </lookup>
</lookups>
```

[0053] Lookup 中的 field 属性表示业务对象中要进行转换的属性，code 表示计算机存储的值，name 表示用户实际看到的值，这样当用户需要增加“type（学历类型）”或者修改用户看到的文字的时候，只要修改状态机的模型定义，不需要重新编写代码。

[0054] 1.2.2. blocks 节点

[0055] blocks 节点用来定义业务对象的属性控制区域，业务对象有很多属性，通常这些属性并不是都能够为业务人员所访问或修改的，因此在状态机定义的时候需要定义在每个状态下业务人员能拥有的属性访问权限，比如财务人员只能修改报销申请对象的金额（price）属性，但是不能修改报销人等其他信息，当业务对象的属性特别多的时候，为了简化状态机的配置，减少定义的工作量，可以将业务对象的属性进行归类，具有相同访问权限的属性被定义在一个属性块（block）中，集中进行权限的访问控制配置。其特征结构如下：

[0056]

```
< blocks>
        < block id="subject" attrs="name,product,time……" />
        < block id="other" attrs="so,meno,text,ass……" />
    </ blocks >
```

[0057] block 中的 id 是属性控制块的唯一标示，attrs 是具体的属性列表。block 具体的使用方式在后面 bizright 节点中描述。

[0058] 1.2.3. Startconditions 节点

[0059] 状态机的启动条件节点，该节点定义了具体业务对象的状态机启动条件，比如：一个报销单对象填写完成后，提交审批申请的之前，只能报销当年发生的费用。该节点可以包含一组 conditions 条件节点，每个 conditions 节点中定义了 condition 条件节点，具体的特征如下：

[0060] A）conditions 节点中的 condition 节点之间是“或”的关系；

[0061] B）conditions 节点之间是“与”关系；

[0062] 举例如下：

[0063]

```
<startconditions>
    <conditions name="启动条件 1" >
      <condition type="field" field="userid" value="" symbol="!=" datatype="string" />
      <condition type="field" field="price" value="0" symbol=">" datatype="string" />
    </conditions>
    <conditions name="启动条件 2">
      <condition name="date" type="field" field="date" value="2012-01-01" symbol=">=" datatype="string" />
    </conditions>
</startconditions>
```

[0064] 说明：上述“启动条件 1”和“启动条件 2”之间是“与”的关系，“userid”和“price”之间是“或”的关系，condition 节点中 field 定义了业务对象的属性，type 定义了该属性的合法检测方式是“字段比较”，value 定义了比较值，symbol 定义了比较符号，datetype 定义了比较的字段应该按照何种数据类型进行比较。

[0065] condition 节点中的 type 属性有多个取值，分别是“field、expression、action、storeproc、service”，type 属性的具体特征值含义如下：

[0066] -field：表示该条件是按照业务对象的字段与模型中定义好的具体的值进行比较，得出“真假”；

[0067] -expression：表示该 condition 节点是要求进行表达式计算来完成结果“真假”的计算；

[0068] -action：表示需要通过调用某个具体的预定义类来得出“真假”；

[0069] -storeproc：表示需要通过调用某个存储过程来得出条件“真假”；

[0070] -service：表示需要通过调用某个 webservice 得出条件“真假”；

[0071] 1.2.4. states 节点

[0072] 状态机模型的状态模型节点集合，这个节点由一组 state 节点构成，是状态机模型的核心节点，每个 state 节点都定义了业务对象在运行过程中的一种中间状态，state 节点有三种类型：startstate 节点、state 节点、endstate 节点。

[0073] Startstate 节点是状态机的启动节点，即发起点；

[0074] State 节点是一般节点；

[0075] Endstate 节点是状态机的结束节点，即终止点，从图论的角度上说该节点只有入度，没有出度。

[0076] 上述三种节点从特征定义上并没有特殊区别，节点之间通过 transitions 节点属性来关联；state 节点同样包含属性和子节点，下面进行详细描述：

[0077] • state 节点的属性：

[0078] -Id：状态节点的唯一标示符；

[0079] -name：状态节点的中文名称；

[0080] -description：状态节点的功能描述；

[0081] -colorremind：是否为状态机控制的对象设置特殊的显示颜色，当该属性为 true 时，用户看到的业务数据将根据状态的不同呈现出不同的颜色，更加清晰的帮助用户找到特定的需处理业务；

[0082] -signaturetype：状态的人工审批类型，取值为 single（单一）或者 counter（会签）；

[0083] • state 节点的子节点

[0084] state 节点的基本结构如下：

[0085] <state……>

[0086] <assignees/>// 定义该状态的参与人员

[0087] <events/>// 定义该状态中涉及的事件

[0088] <filters/>// 定义该状态需要执行的过滤条件

[0089] <buttonsright/>// 定义该状态的增加、删除、修改按钮的访问权限

[0090] <recover/>// 定义该状态是否允许追回及追回的条件和事件

[0091] <bizright/>// 定义该状态的业务对象的属性访问权限

[0092] <transitions/>// 定义该状态的出度（“出度”图论术语）以及发生状态转换时触发的事件和需要激活的下一状态 id

[0093] </state>

[0094] 上述结构的详细描述：

[0095] A）assignees 子节点：定义该状态的参与人员，该节点的特征结构如下：

[0096] <assignees>

[0097] <assignee name="报销申请人"type="anonymous"andparentfilter="true"value="*">

[0098] <leaveevents/>// 定义当该参与者执行状态转换时需要触发的动作行为

[0099] <filters/>// 定义该参与者进行业务对象的操作时，系统的过滤条件

[0100] </assignee>

[0101] <assignee>// 另一个参与者或者另一种类型的参与者

[0102] ……

[0103] </assignee>

[0104] </assignees>

[0105] 一个 assignees 节点可以拥有多个 assignee 子节点，每一个 assignee 子节点定义了一个或者一种类型的状态参与者。上述结构中 assignee 节点特征结构如下：

[0106] type 属性：该属性定义了参与者的类型，该类型是一个枚举类型，取值范围如下：

[0107] -user：某个具体的用户；

[0108] -creator：状态机的创建者或者启动者；

[0109] -anonymous：匿名用户，即所有能够访问该状态机模型对应的业务对象的人都可

以操作；

[0110] -actor：属于某个角色的用户；

[0111] -leader：上一个状态参与者的上级领导，比如：申请人是工程部的职员，那么该状态的参与者就是工程部的经理；

[0112] -group：属于某个工作组的用户；

[0113] -assigneeAction：通过执行某个预定义的类来获得该步骤的参与人；

[0114] -subordinate：上一个状态参与者的下属职员；

[0115] -assigneeService：通过访问某个 web 服务来获得该步骤的参与人，比如访问某个 LDAP 目录服务获得参与者；

[0116] andparentfilter 属性：是否复合父条件，该属性是用在当参与者为某些特殊用户时，需要对业务对象的访问施加一些特殊的过滤条件，属性值为 true 时将把本节点的 filter 子节点中定义的过滤条件与父节点中的 filter 条件进行“与”合并；

[0117] value 属性：参与者的具体取值；

[0118] leaveevents 节点：定义当该参与者执行状态转换时需要触发的动作行为

[0119] filters 节点：定义该参与者进行业务对象的操作时，系统的过滤条件，后面将详细描述。

[0120] B）events 子节点：定义了当前 state 状态节点中涉及到的行为事件，包括 enterevents、leaveevents、humanevents 三种类型：

[0121] enterevents：业务对象进入到当前状态时触发的事件；

[0122] leaveevents：业务对象离开当前状态时触发的事件；

[0123] humanevents：业务对象在当前状态能够执行的动作事件；

[0124] 上述三种事件中每一种事件都可以触发多个动作，举例如下：

[0125]

```
<events>
  <enterevents>
<event name="事件 1" type="assignment" variable="status" value="总经理已接收" />
<event name="事件 2" type="assignment" variable="price" value="5000" />
</enterevents>
<leaveevents />
<humanevents>
<event name="操作 1" type="assignment" variable="status" value="com.sac.patent.statusAction" />
<event name="操作 2" type="assignment" variable="price" value="com.sac.patent.priceAction" />
</humanevents>
</events>
```

[0126] 上述事例中 enterevents（进入事件）节点中定义了两个动作，“事件 1”将业务对象的 status 属性值改成了“总经理已接收”，“事件 2”将业务对象的 price 属性值改成了 5000；humanevents（手工事件）节点中定义了两个动作，“操作 1”将调用 com.sac.patent.statusAction 类对业务对象的 status 属性进行赋值，“操作 2”将调用 com.sac.patent.priceAction 类对业务对象的 price 属性进行赋值；

[0127] 不同的是，enterevents 和 leaveevents 两个节点中定义的事件都是系统自动执行的，执行发生的时间是业务对象进入到当前状态和离开当前状态的时候，humanevents 中定义的每一个 event 事件将会作为一个按钮出现在用户界面的工具条中，上述事例中，当业务对象进入该状态之后，用户打开当前操作界面时的工具栏中会出现“操作 1”和“操作 2”两个按钮，离开该状态时这两个按钮将自动消失。

[0128] C）filters 子节点：定义参与者在业务对象处于当前状态时，系统的过滤条件，filters 节点的由若干个 filter 子节点构成，每个 filter 子节点完成一组过滤条件：

[0129]

```
<filters>
  <filter >
    <conditions>…… </conditions>
……
  </filter>
  <filter>…… </filter>
</filters>
```

[0130] filter 内部有 conditions 条件子节点构成，完成具体条件表达式的定义工作，conditions 条件节点的构成特征与 Startconditions 节点的特征相同。filters 节点可以定义业务对象在当前状态时，系统对业务对象检索的条件。最终用户在访问系统页面时，看到的业务对象将是按照 filter 定义的条件过滤之后的。

[0131] D）buttonsright 子节点：定义业务对象处于当前状态时的访问控制权限，包括增加、删除、修改按钮的访问权限。缺省情况下，对于一个业务对象的操作包括了 CRUD（增删改查）四种操作，state 节点中的 buttonsright 子节点对增删改三种操作进行了控制，其特征结构如下：

[0132]

```
<buttonsright>
  <right name="add" right="1"…… /> //控制新增权限
  <right name="edit" right="1"…… /> //控制修改权限
```

[0133]

```
  <right name="delete" right="1" ……/> //控制删除权限
</buttonsright>
```

[0134] right 子节点中的 right 属性控制当前操作的参与人员拥有的操作权限“right=1”表示拥有相应权限，“right=0”表示没有相应权限；

[0135] E）recover 子节点：定义该状态是否允许追回及追回的前提条件和追回过程中需要引发的事件，处于“待处理”状态的业务对象如果在 state 中定义了 recover 信息，并且业务对象满足了允许“追回”的条件时，上一个状态（state）的业务人员可以执行追回操作，将状态重新转换到上一个状态。例如：报销人填写了报销申请并提交给上级领导审批，突然发现填写的内容有问题，那么在上级领导没有审批之前，系统将允许其追回报销申请并重新处理。Recover 子节点的特征结构如下：

[0136] <recover allow="true">// 定义该状态时是否允许执行追回操作

[0137] <conditions/>// 如果 allow=true，则定义了允许追回的条件是什么，其结构和 startconditions 节点相同

[0138] <events/>// 执行追回操作时将触发的系统动作，其结构和 state 的 events 子节点相同

[0139] </recover>

[0140] F)bizright 子节点:定义该状态的业务对象的属性访问权限,即被授权的业务人员(assignees 中定义的参与者)在当前状态下访问业务对象时,能够访问的业务对象的属性权限,比如:报销申请对象在状态从“申请”->“批准”->“审核”这几个状态转换的过程中,“审核”环节的参与者可以修改报销对象的“price”属性,但是不能修改报销人、报销物品等其他属性,状态机通过在“审核”状态的配置信息中定义 bizright 来实现对业务对象 price 属性的控制,其特征如下:

[0141] <bizright default="readonly">// 设置缺省情况下业务对象在当前状态的所有属性都是只读属性

[0142] <info id="price"right="modify"isblock="false"/>// 表示 price 属性在当前状态可以进行修改操作

[0143] <infoid="subject"right="readonly"isblock="true"/>// 表示“subject”块中定义的所有属性都只读

[0144] </biright>

[0145] bizright 的 default 属性设置缺省情况下业务对象在当前状态的属性访问权限,readonly 表示所有属性都是只读、modify 表示所有属性都可以修改,

[0146] G)transitions 子节点:定义当前状态(state)的出度(“出度”图论术语)转换以及发生状态转换时触发的事件和需要激活的下一状态 id。其特征结构示意如下:

[0147]

```
<transitions>
  <transition name="不同意" icon=1 destination="3" silence="0" counter="1">
    <assignees />
    <showconditions />
  </transition>
  <transition name="同意" icon=2 destination="7" silence="0" counter="1">
    <assignees />
    <showconditions />
  </transition>
</transitions>
```

[0148] 上述 transition 中:

[0149] -name 属性:定义了最终出现在用户界面工具栏中的按钮名称;

[0150] -icon 属性:定义了按钮的图标 id(图标资源存放于系统的资源文件中,不赘述);

[0151] -destination 属性:定义了当执行该转换(transition)时,状态迁移的目标状态

id（在 state 节点的 id 属性中定义）

[0152] -silence 属性：定义了执行转换时是否需要参与者（assignees 中定义）录入相关备注信息，例如：审批意见等，“0”表示不需要，“1”表示需要。

[0153] -counter 属性：定义了执行该转换时是否需要会签的人数，“1”表示只要有一个人同意（或不同意）就可以转换状态，“*”表示所有符合 assignees 节点中定义的人都要同意（或不同意）才能转换状态，其他数字表示有具体的多少人同意（或不同意）就可以进行转换状态。

[0154] -showconditions 子节点：定义了在界面工具条上显示转换按钮的条件，满足条件的时候显示，否则不显示。比如某些人只能查看业务对象但是不能执行审批动作的时候。

[0155]

```
<showconditions>
  <conditions>
    <condition type="field" field="memo" value="" symbol="!=" datatype="string" />
    <condition type="expression" expression="date>2012-05-03 and goodname.equal(“apple 电脑”);" />
  </conditions>
  <conditions>
    <condition type="action" classname="com.sac.patent.testAction" />
  </conditions>
</showconditions>
```

[0156] -assignees 子节点：定义了能够执行该转换的参与者，这个参与者的优先级高于 state 节点中定义的参与者（state 中定义的 assignees），如果该子节点的内容为空，采用 state 中定义的参与者。

[0157] 2. 步骤二说明：

[0158] 状态机模型定义完成之后会保存为一个 XML 文件保存在硬盘指定的文件夹中（默认为图形化状态机模型定义工具安装目录下的“model”文件夹，模型文件的扩展名为“.sm”），同时会采用 blob 字段的方式将该文件中的 XML 字符串上传到数据库中，表结构如下：

[0159]

字段名	描述

[0160]

Id	状态机模型的 id
Bizid	状态机模型绑定的业务对象模型 id
Orgid	状态机模型适用的组织机构编码
Name	状态机模型的名称
Description	状态机模型功能描述
Author	作者
Timestamp	更新时间
Version	版本号
Status	状态机模型的状态，Active（活动中），inactive（非活动中）
Model	状态机模型数据

[0161] 状态机模型的上传和加载采用的是 webservice 的方式，客户端图形化状态机模型定义工具通过访问 webservice 的方式与服务器端进行通信，webservice 提供 SyncModel 接口实现客户端、服务器内存、数据库三者之间的状态机模型同步；

[0162] 新的状态机模型发布之前，需先进行正确性的验证，验证的方法是通过将状态机模型加载到应用服务器端的内容中，在内存建立状态机模型的缓存，这个过程中将对状态机模型的正确性进行验证。

[0163] 状态机模型加载的流程如图 2 所示。

[0164] 当状态机模型开始使用前，状态机模型驱动引擎会调用一个名为 loadAllStateMachines 的方法完成状态机模型的加载，加载的示意流程如上所示：首先从数据库中把正在使用的状态机模型文件（即：status=acitive）查询出来，保存到一个列表（List）中，然后判断列表中的状态机模型数量是否大于等于 1，如果大于等于 1 就将遍历其中的状态机模型，并加装到内存中，构建出状态机模型的内存对象。

[0165] 过程中如果出现异常，表明该状态机模型的定义信息有问题，在出错处理中会提示有问题的地方（比如：状态机模型中定义的组织机构不存在、业务对象不存在、用户不存在或者其中指定的预定义类不存在等等），从而实现状态机模型正确性的验证，并放弃该状态机模型内存对象的构建，进行下一个循环，构建其他的状态机模型内存对象，直到所有的状态机模型内存对象都构建完成。

[0166] 状态机模型的内存对象构建过程实际上就是解析 XML 文件的过程，由于状态机模型的 xml 文件比较复杂，因此在内存对象构建的过程中也相对复杂。本专利采用了多个类共同完成状态机模型内存对象构建的方法实现状态机模型的加载，具体类如下：

[0167]

类名	描述
ActionSchemaForAssignment	用来构建模型中的 event 信息，作用：直接赋值操作
ActionSchemaForClass	同上，作用：调用预定义的类完成对属性的赋值操作
ActionSchemaForExpression	同上，作用：将表达式计算结果赋值给对象属性
ActionSchemaForStoreproc	同上，作用：调用存储过程对对象属性赋值
ActionSchemaForService	同上，作用：调用第三方 webservice 完成对象赋值操作
AssigneeSchema	用来构建模型中的参与者（assignee 信息），构建构建参与者类
HumanActionSchemas	构建手工动作信息，作用加载 humanevents 节点中的 event 信息，每个 event 一个对象
ConditionSchemas	条件对象，加载模型中的 conditions 节点信息
ConditionForExpression	条件对象的子对象，构建表达式类型的条件对象

[0168]

ConditionForField	条件对象的子对象，构建字段比较类型的条件对象
ConditionForAction	条件对象的子对象，构建预定义类类型的条件对象
ConditionForStoreproc	条件对象的子对象，构建存储过程类型的条件对象
ConditionForService	条件对象的子对象，构建访问 webservice 类型的条件对象
BlocksSchema	构建 blocks 子节点的区域对象
StateMachineSchema	状态机模型对象，一个 StateMachineSchema 的实例就是一个完整的状态机模型内存对象，他是所有其他对象的容器
StateSchema	状态机模型的状态对象，每一个 state 子节点一个对象实例
TransitionSchema	定义状态转换的对象，用来解析 transition 节点的内容
Lookup	用来加载 lookup 节点的内容
RecoverSchema	用来加载回退节点 recover 中的内容
QueryFilter	用来加载过滤条件的内容
StateMachineVersion	用来加载状态机模型的版本信息的内容

[0169] 上述这些类功能完成了状态机模型的内存对象(StateMachineSchema)功能，最终构建起一组 StateMachineSchema 对象，每个 StateMachineSchema 对象代表一类业务对象的状态机模型，这组 StateMachineSchema 对象被保存在一个用 hashmap 结构定义的内存缓冲区对象 StateMachineSchemaBuffer 中，方便在业务系统运行过程中，快速的定位和操作。

[0170] 采用类的方式完成对 xml 文档的解析工作，对于系统的性能带来的好处是不言而喻的，有了 StateMachineSchema 对象，业务系统在运行中就可以采用面向对象的操作方式访问状态机模型，快速稳定。StateMachineSchema 对象之间使用 Id 属性进行区分，相同 name 属性的 StateMachineSchema 对象之间采用 StateMachineVersion 属性对象进行区分。

[0171] StateMachineSchema 构建的基本流程如图 3 所示：

[0172] 首先创建一个 org.dom4j.Document 对象（dom4j 是文档对象模型的 java 版本，用来在 java 中访问 XML 文档方式存储的对象结构），将数据库中得到的 XML 结构保存的状态机模型信息加载到 org.dom4j.Document 对象中，这样便于在内存中对 xml 文档信息的访问操作。

[0173] 然后，状态机模型驱动引擎将创建一个 StateMachineSchema 类型的对象实例，并调用实例的 loadXmlDefine 方法加载保存在 Document 对象中的模型数据。

[0174] loadXmlDefine 方法执行的过程：

[0175] - 加载 Model 节点的属性信息，包括 Id、version、name……；

[0176] - 创建 BlockSchema 对象列表（BlockList），循环 blocks 中的子节点，并为每个子节点创建一个 BlockSchema 子对象，完成对权限控制区域（blocks）信息的加载工作；

[0177] - 创建 Lookup 对象列表（LookupList），循环 Lookups 的子节点，并为每个子节点创建一个 Lookup 子对象，完成 lookups 节点信息的加载；

[0178] - 创建 Condition 对象列表（ConditionList），循环 Conditions（或 startconditions）的子节点，并为每个子节点创建一个 Condition 子对象，完成 Conditions（或 startconditions）节点信息的加载；

[0179] - 创建 state 对象列表（StateList），循环 states 的子节点，并为每个 state 子节点创建一个 stateschema 对象，完成 states 节点信息的加载；

[0180] 由于 stateschema 对象的内容相对复杂，加载过程中还需要创建 assigneeschema 对象列表，event 对象列表、transition 对象列表、condition 对象列表、filter 对象列表、bizright 对象列表、buttonright 对象列表，用来加载 state 节点中的对应配置信息；

[0181] 状态机模型文件完整加载到内存中并用 StateMachineSchema 完成文档对象（dom）到 java 对象构建之后，状态机模型的模型就实现了在内存中建立和缓存。

[0182] 一个状态机模型（StateMachineSchema）内部的逻辑关系最终构建起一个业务对象的状态转换图，这个图是一个有向图，图的节点定义了业务对象的每一个过程状态，图的每一个出度和入度都定义了业务对象的一次转换、转换条件和转换过程中发生的事件，从而完成业务对象的动态行为特征。最终状态机模型驱动引擎就是按照业务对象的状态转换图来完成对业务对象的动态行为控制工作。

[0183] 3. 步骤三说明：

[0184] 状态机模型被状态机模型驱动引擎加载到内存中并以对象的形式实例化之后，就会在状态机模型驱动引擎的协助下进行业务对象实例的动态行为控制。状态机模型驱动引擎负责状态机实例的创建和驱动过程，驱动方法如图 4 所示。

[0185] 状态机模型驱动引擎对业务对象实例的动态行为控制分为以下几个特征：

[0186] 业务对象在一个状态转换过程中，被定义为“正在进入”、“待处理”、“处理中”、“已处理”、“正在离开”五种微状态，即一个状态中包含了上述五个微状态：

[0187] - 正在进入（enter）：表示业务对象从一个状态转换到另一个状态的过程中的过渡状态，这个状态里状态机模型驱动引擎会对业务对象进行进入该状态前的一些初始化工作，具体的工作就定义在 enterevents 节点中。

[0188] - 待处理（wait）：表示业务对象已经经过了“正在进入”的过渡状态后处于的状

态，这种状态下业务对象被放置在任务队列中，处于等待业务人员进行处理的状态，处于该状态的业务对象，如果在 state 节点中定义了 recover（追回）信息，可以被追回到上一个 state 状态节点。一旦该业务对象被业务人员选中并从任务队列中检索出来，就将从“待处理”状态进入到“处理中”状态。

[0189] - 处理中(process)：表示业务对象离开了“待处理”状态，已经被业务人员(assignees 中定义的参与者)开始处理。该状态下业务人员可以在权限允许的范围内对该业务对象进行各种操作。处于“处理中”状态时，不能被执行 recover（追回）操作；

[0190] - 已处理(completed)：表示业务对象已经处理完毕，等待进入下一个状态。该状态下用户不能再对业务对象的信息进行修改。

[0191] - 正在离开(leave)：业务对象在离开当前状态时的过渡状态，该状态下降触发 leaveevents 节点中定义的事件。

[0192] 上述五个微状态是业务对象在运行过程中从一个状态迁移到另一个状态期间经历的。根据业务需求和业务操作行为的不同，对于每一个业务对象都有一个属于自己的状态机实例，即：StateMachineInstance。

[0193] 业务状态机模型(StateMachineSchema)描述了一类业务对象的动态行为特征，而状态机实例(StateMachineInstance)描述了具体某个业务对象的动态行为，所以说状态机实例是状态机模型的具体化产物。

[0194] 以上所述仅是本发明的优选实施方式，应当指出，对于本技术领域的普通技术人员来说，在不脱离本发明技术原理的前提下，还可以做出若干改进和变形，这些改进和变形也应视为本发明的保护范围。

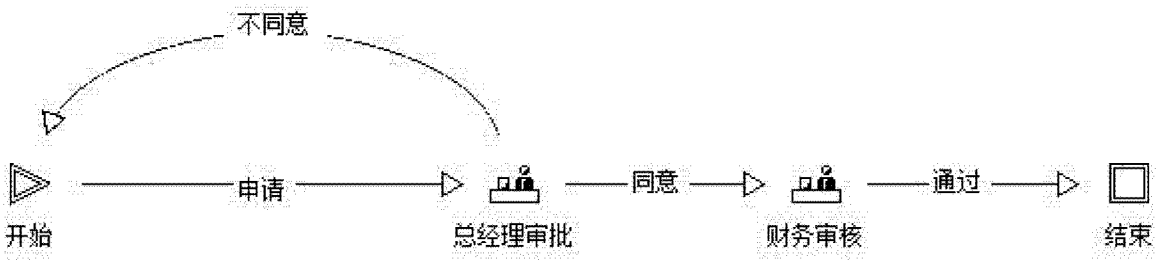


图 1

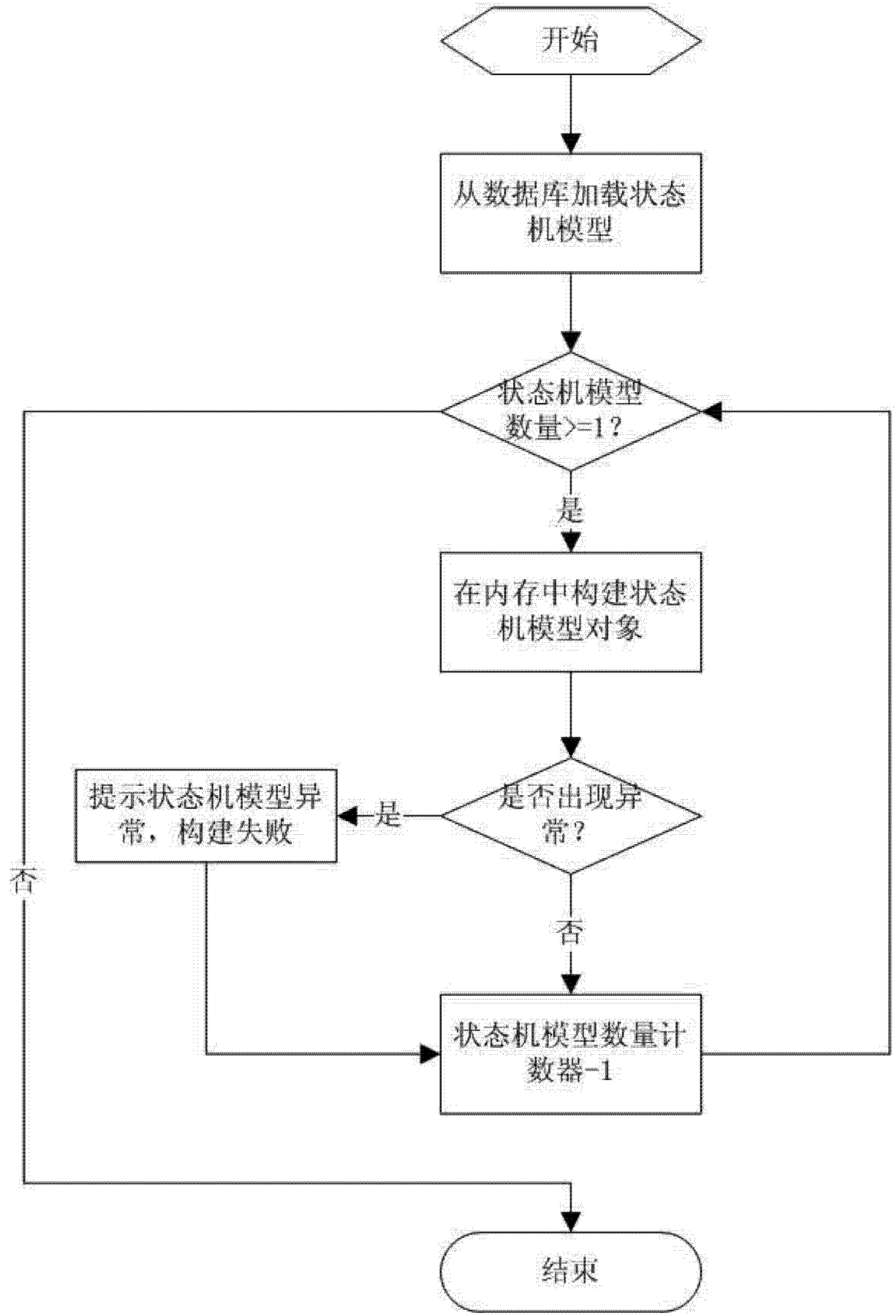

开始
从数据库加载状态机模型
状态机模型数量>=1？
是
在内存中构建状态机模型对象
是否出现异常？
是
提示状态机模型异常，构建失败
否
否
状态机模型数量计数器-1
结束

图 2

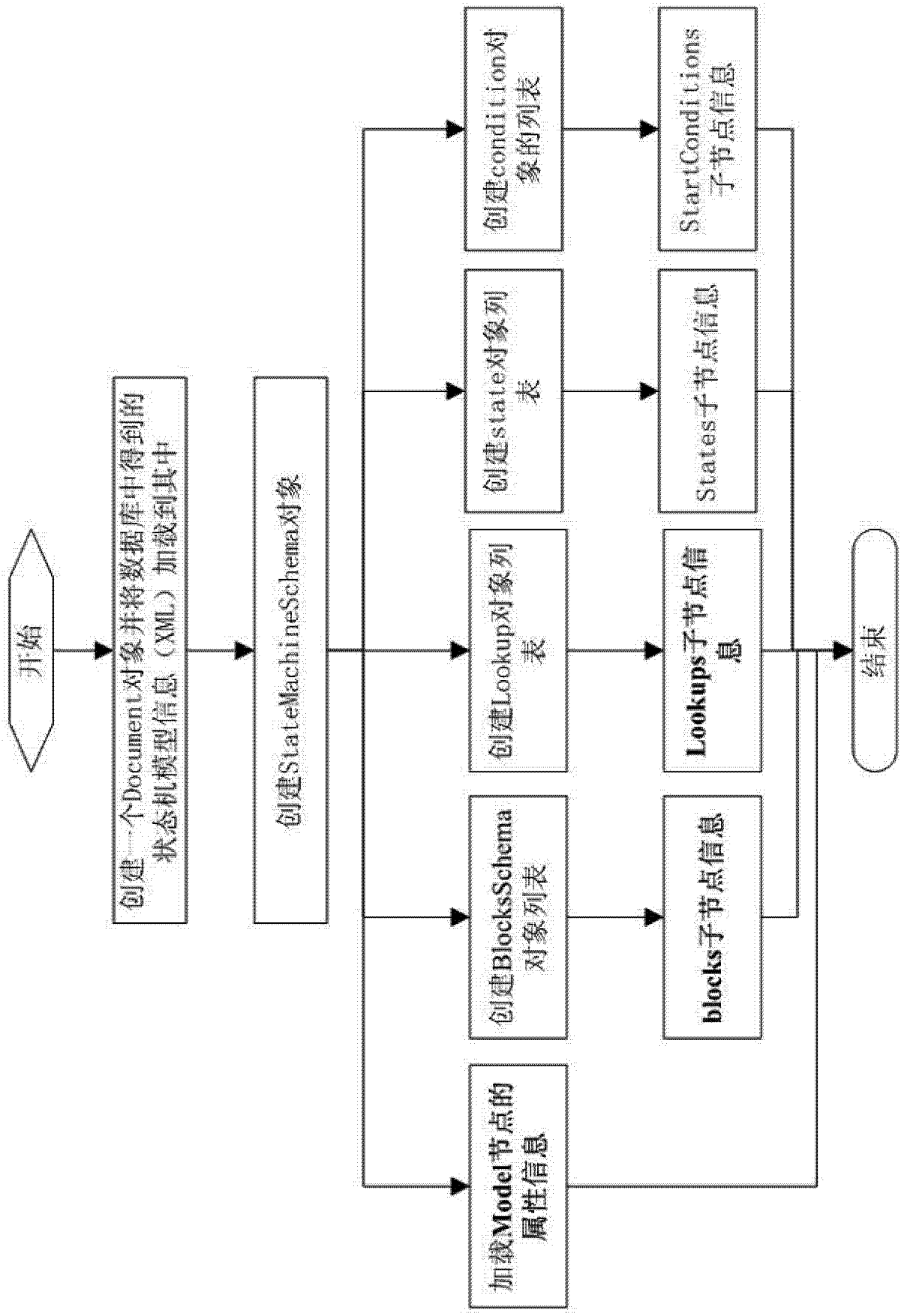

开始
创建一个DocumentL对象并将数据库中得到的状态机模型信息（XML）加载到其中
创建StateMachineSchema对象
创建condition对象的列表
创建state对象列表
创建Lookup对象列表
创建BlocksSchema对象列表
加载Model节点的属性信息
StartConditions子节点信息
States子节点信息
Lookups子节点信息
blocks子节点信息
结束

图 3

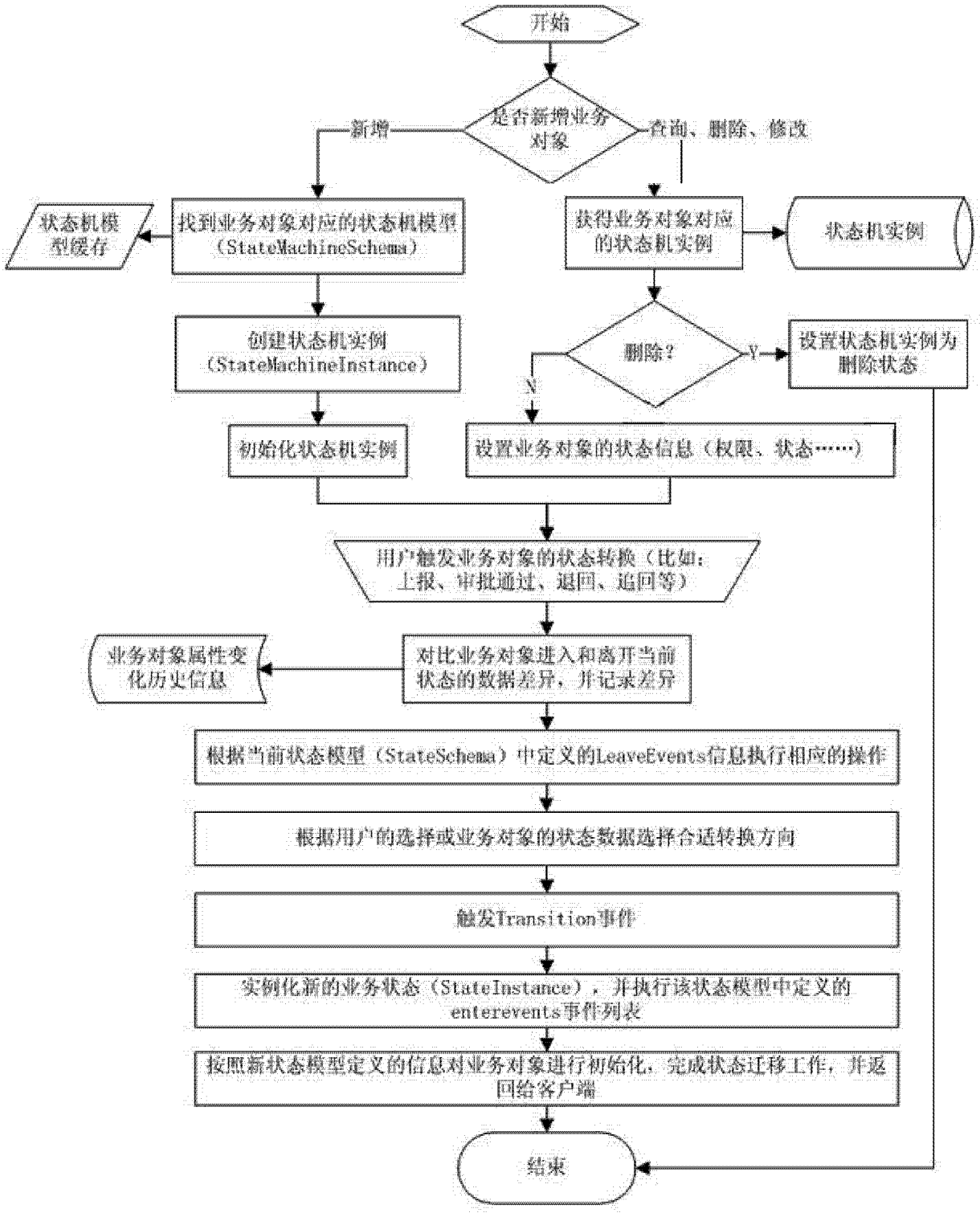

开始
是否新增业务对象
新增
查询、删除、修改
状态机模型缓存
找到业务对象对应的状态机模型（StateMachineSchema）
获得业务对象对应的状态机实例
状态机实例
创建状态机实例（StateMachineInstance）
删除？
Y
设置状态机实例为删除状态
N
初始化状态机实例
设置业务对象的状态信息（权限、状态……）
用户触发业务对象的状态转换（比如：上报、审批通过、退回、追回等）
业务对象属性变化历史信息
对比业务对象进入和离开当前状态的数据差异，并记录差异
根据当前状态模型（StateSchema）中定义的LeaveEvents信息执行相应的操作
根据用户的选择或业务对象的状态数据选择合适转换方向
触发Transition事件
实例化新的业务状态（StateInstance），并执行该状态模型中定义的enterevents事件列表
按照新状态模型定义的信息对业务对象进行初始化，完成状态迁移工作，并返回给客户端
结束

图 4