1. 前言

某公司的总部设在北京，在上海、广州、成都和西安有分支机构，全公司接近 700 名员工。鉴于业务和员工团队的快速发展，为了提升整体工作效率。该公司准备开发一套员工报帐系统，取代原来的人工处理方式。

2. 问题的说明、可行性论证

报帐系统将支持员工记录（或预见）日常业务活动的开销，并自动结算每个月应该返还员工的补偿金额，补偿金额会直接存入员工的工资帐户中。

报帐系统应具有基于先进技术的图形化界面，员工可以输入业务活动的种类和简短描述，活动开销的类别，选择不同的支付方式，并可以生成灵活的报表。

报帐系统应该有能力根据员工提供的信息和要求返还补偿额，同时保存全部员工的报帐信息。员工可以通过他们自己的电脑来使用报帐系统。由于牵涉到财务信息，报帐系统必须提供可信的安全机制。

该公司现有一套基于微软 SQL SERVER 的人事管理数据库系统，记录员工的基本信息和团队的组织结构。报帐系统将和现有人事管理数据库系统协同工作，需要引入人事管理数据库系统中的部分信息，但不会更新其内容。

通过报帐系统，员工能够在出差前（提前两天）按照规定的额度向公司申请借款，相关的经理人员能够通过报帐系统批复或拒绝。报帐系统应在相关负责人批复之后通知该员工提取现金或确认相应款项已经划入指定信用卡（根据员工的要求）；员工可以通过报帐系统报销合理的业务活动费用。

财务部门将指定一位报帐系统管理员监管拟建系统中的信息，负责初始设置和维护特定的分类额度准则，并能够定期或随机地向各部门负责人提交报帐情况的统计报告。

报帐系统在每月的 25 日对通过审批的报帐申请自动作一次结算，并以电子邮件的方式通知应该得到补偿的员工，同时生成一份统计报告传送给财务部门的系统监管人员。

拟建系统的“USE CASE 图”以及“风险与 USE CASE 的关系”如图 1、2 所示。

根据风险与 USE CASE 的关系不难发现，使用当前的技术完全可以实现这个系统，而且 USE CASE“提交报销申请”和“结算当月报销费用”几乎覆盖了全部风险，并且它们只占 USE CASE 的很小一部分，所以，将这两个 USE CASE 选定为“分析局部”，

其次，由于公司内部的电脑网络和人事管理数据库系统已经比较完善，拟建系统完全能够利用各个部门现有的系统，这样可以大大降低开发成本和开发周期。

因此，实现该系统不存在任何复杂的技术，而且利用现有的软硬件环境可以进一不提高开发的效率，降低开发成本，投入产出比较高，值得开发。

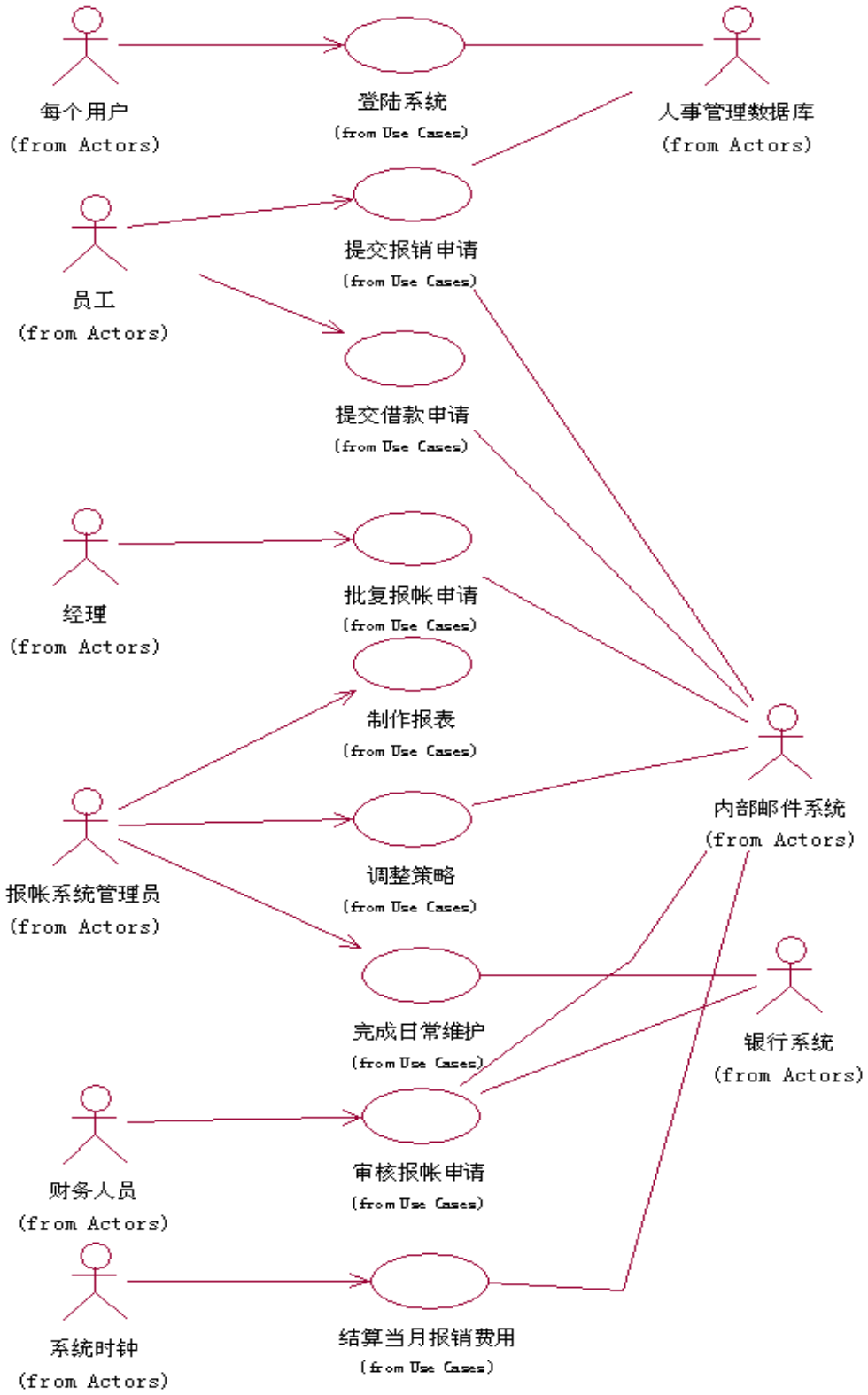

图 1 拟建系统的 USE CASE 图

USE CASE / 风险	调整策略	结算当月报销费用	批复报销申请	审核报销申请	提交报销申请	提交借款申请	完成日常维护	制作报表	登陆系统
数据库响应速度			√	√	√	√			
数据库的容量					√	√			
与内部邮件系统联接	√	√	√	√	√	√			
与银行系统联接		√		√					
与人事管理数据库联接					√	√			√
内部网络有足够带宽			√	√	√	√			√
数据备份							√		

图 2 风险与 USE CASE 的关系

3. USE CASE 实现报告

3.1 简介

“提交报销申请”的 USE CASE 图如图 3 所示。

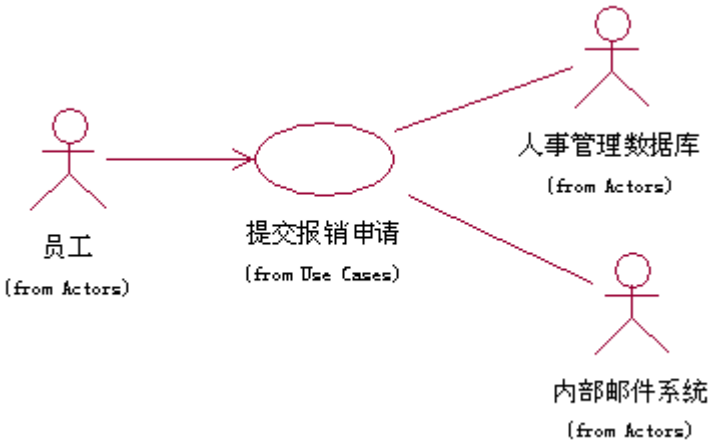

图 3 “提交报销申请”USE CASE 图

员工通过报帐系统填写报销申请，输入相关活动产生的费用，在一次或者多次填写后提交，经过验证之后，以电子邮件的方式通知经理批复。

3.2 事件流

3.2.1 基本事件序列

1、 打开报销序列

[员工]：员工选择进入“报销申请”功能。

[系统]：如果该员工当月报销单存在，系统将取出相应信息并展示给员工；如果该员工

的当月报销单不存在，则转至 A1 备选事件序列。
2、 添加报销记录
[员工]：员工要求添加一条报销记录。
[系统]；系统显示一条空白的报销记录。
3、 填写报销记录单
[员工]：员工开始填写报销记录，每条报销记录包括的信息有：业务活动发生的时间、地点、客户名称（可选）、原因以及费用金额和种类（交通、餐饮、会议、通信和杂项）。
[系统]：系统显示并记录员工输入的信息。为了让员工方便而准确地准确输入相关信息，除了客户名称、业务活动原因和金额之外，其他信息域提供相应的下拉式选择列表。
4、 验证报销单
[员工]：员工填写完毕所有报销记录之后，要求系统验证这些记录的合理性。
[系统]：报销记录的初始状态为“未验证”，每当一条报销记录被验证为合理，系统将该报销记录的状态设置为“已验证”，系统在验证所有报销记录（为“已验证”）之后提示用户可以提交本月的报销单。验证为合理的记录必须满足几种条件：第一，不同种类的费用不超过相应的限额；第二，报销费用的类型要和员工的职能匹配。对于未通过验证的报销记录，转至 A5 备选事件序列。
5、 提交报销单
[员工]：所有报销记录经过验证之后，员工提交当月的报销单。
[系统]：系统保存这张报销单，将报销单的状态设置为“已提交”并记录提交日期，同时这张报销单被设为“只读”。系统要从人事管理数据库中获知该员工及其经理（负担该员工当月开销者）的电子邮件地址。如果此时人事管理数据库不可用，转至 A6 备选事件序列。
为了及时通知相关人员，系统将自动生成一份以当前报销单为内容的电子邮件发送到该员工及其经理的信箱中。当邮件成功发送后，员工得到一个确认信息。如果此时邮件系统未能将邮件及时发出，转至备选事件序列 A7。

基本事件的序列图如图 4 所示，基本事件序列相应的协作图如图 5 所示。

考虑到“A1 备选事件序列”（建立 Claim_report）的内容过于简单，可以并入基本事件序列。“A3 备选事件序列”（更新 Claim_record）与基本事件序列“填写报销记录单”的唯一区别是针对不同的报销记录（基本事件序列中针对新创建的报销记录，A3 备选事件序列中针对任意选定的报销记录），但这种差异并不影响反映本质内容的“消息”传递，两部分对应的后续设计内容没有区别，而且维护这两部分完全相同的图非常困难，必须持续确保内容一致，否则将招致诸多麻烦。因此将 A3 备选事件序列合并到基本事件序列中。

基本事件序列中使用的“留存”机制和“分布处理”机制的序列图参见图 6、7、8、9、10。

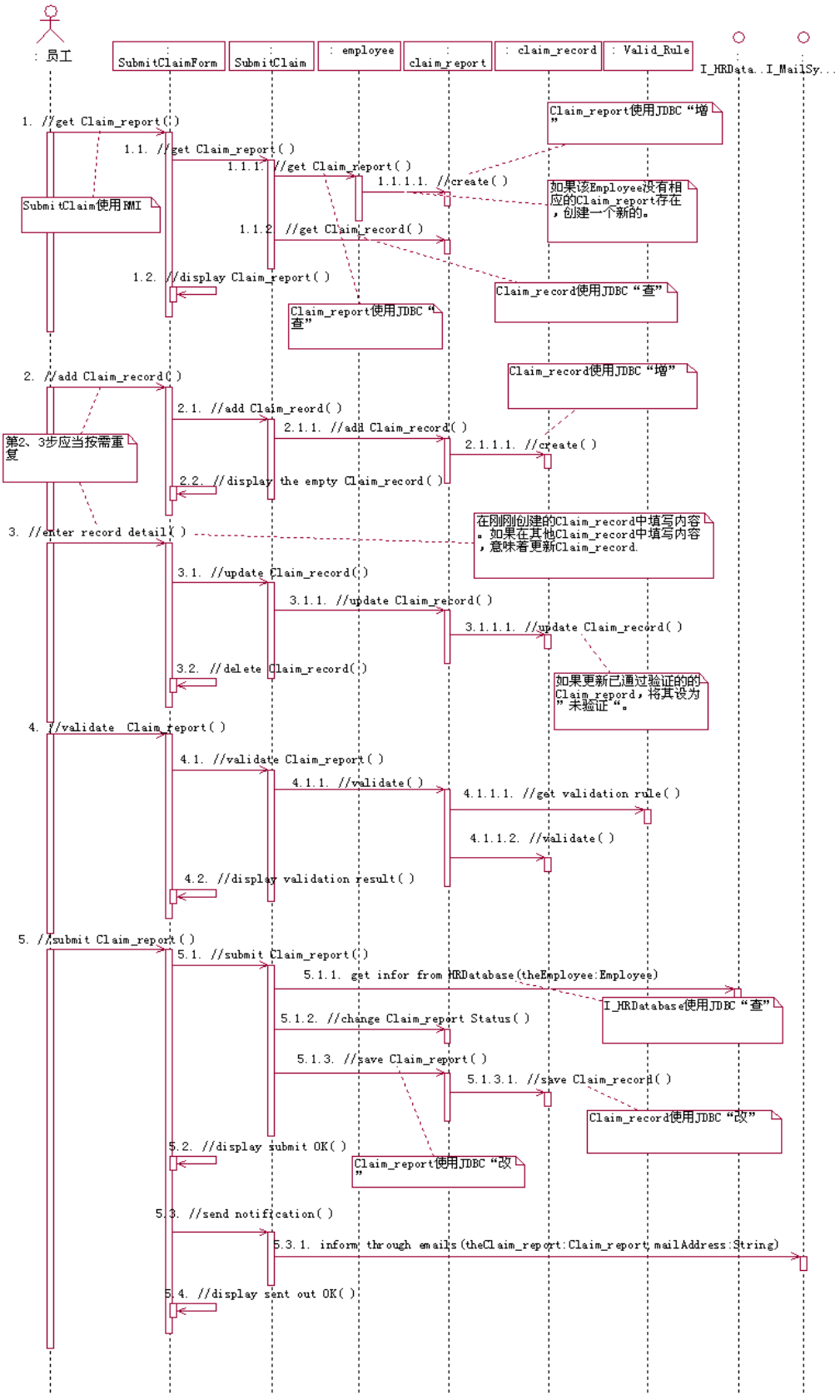

图 4 基本事件序列(包括 A1 和 A3,包含引入机制的注释信息)

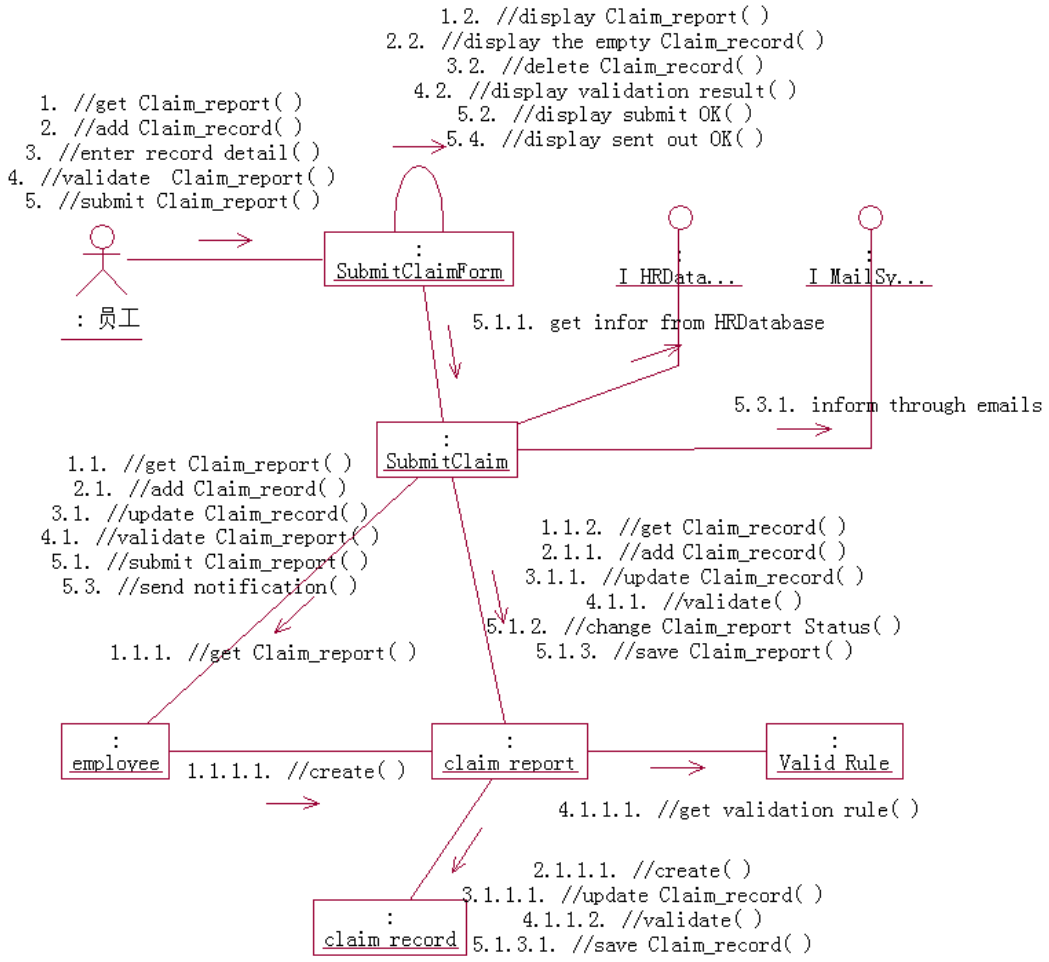

图 5 基本事件序列相应的协作图

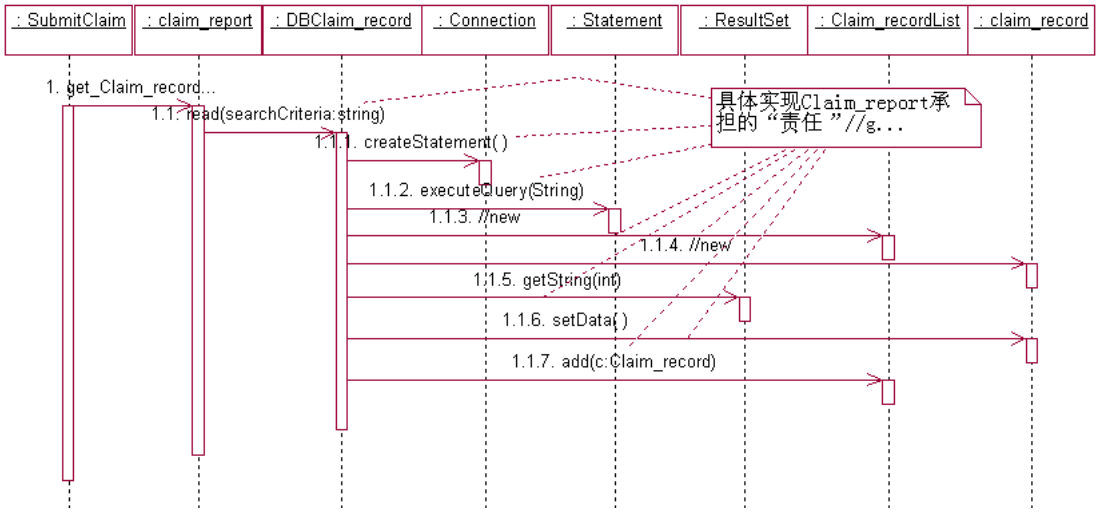

图 6 基本事件序列片段(Claim_record 用 JDBC"查")

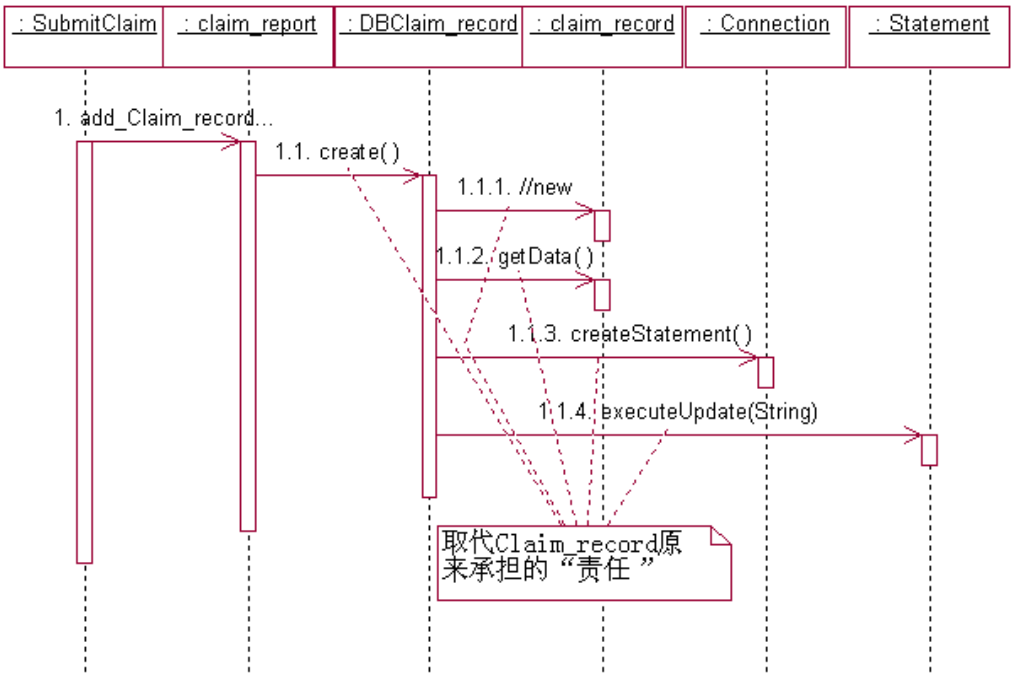

图 7 基本事件序列片段(Claim_record 用 JDBC"增")

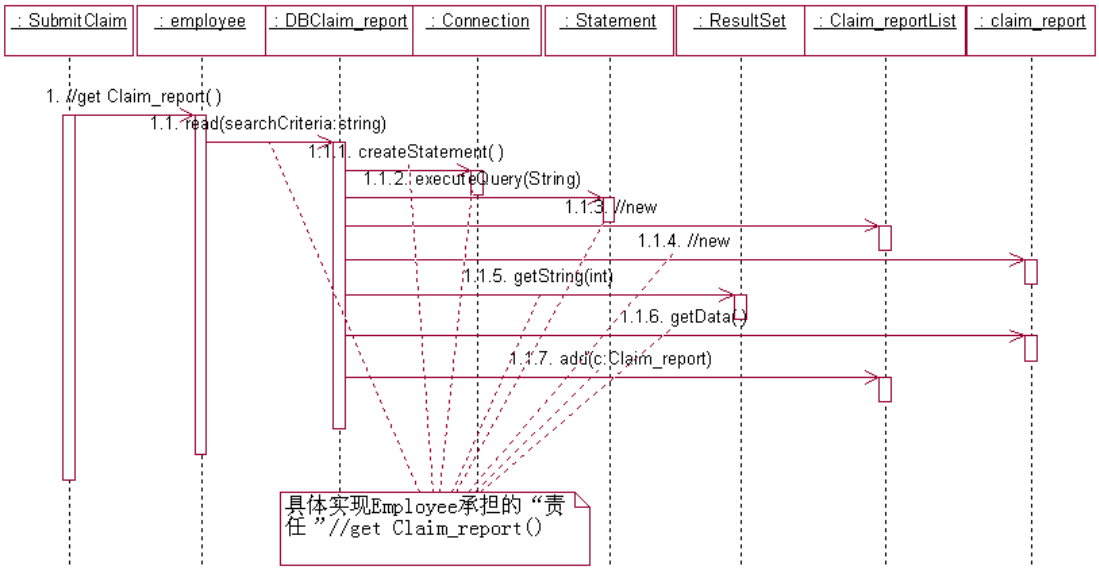

图 8 基本事件序列片段(Claim_report 用 JDBC"查")

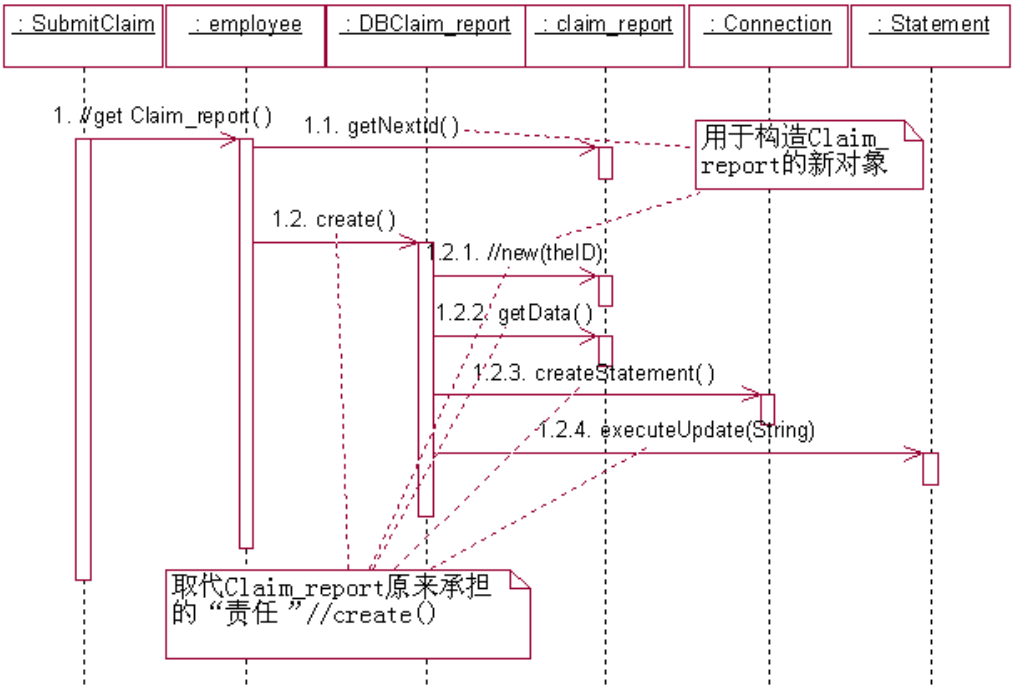

图 9 基本事件序列片段(Claim_report 用 JDBC"增")

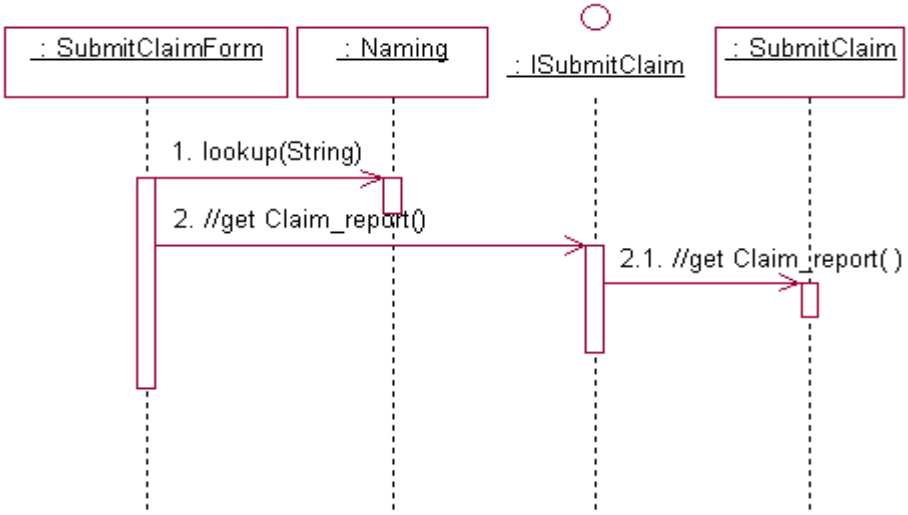

图 10 基本事件序列片段(SubmitClaim 用 RMI-JAVA

3.2.2 备选事件序列组

A1 创建当月报销单
[起始位置]：基本事件序列中，员工进入报销申请程序并准备打开当月报销单。
[触发条件]：系统没有发现和该员工对应的当月报销单。
[具体内容]：系统为该员工创建一张当月报销单。
[返回位置]：基本事件序列中的“打开报销单”步骤。
A2 删除报销记录
[起始位置]：在提交报销单之前任意时间点。

[触发条件]：员工希望删除某一条报销记录。
[具体内容]：系统删除由员工指定的某一条报销记录。
[返回位置]：同“起始位置”。

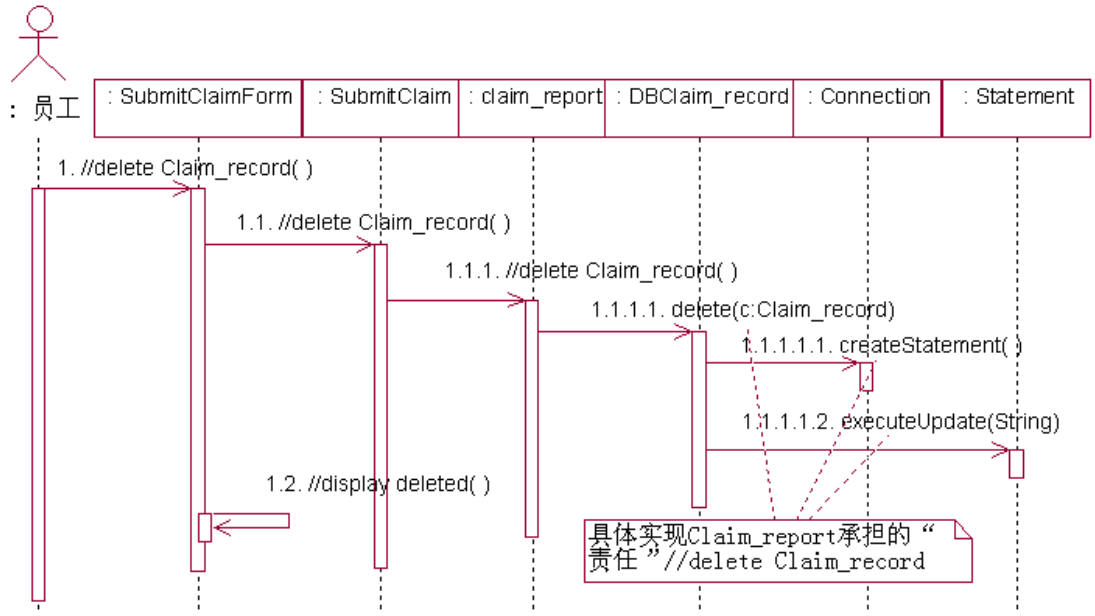

图 11 A2.备选事件序列(删除 Claim_record)

A3 更新报销记录
[起始位置]：在提交报销单之前任意时间点。
[触发条件]：员工希望更新某一条报销记录。
[具体内容]：系统根据员工重新输入的内容更新相应的一条报销记录。将该报销记录状态设置为“未验证”。
[返回位置]：同“起始位置”。
A4 保存当月报销单
[起始位置]：该 USE CASE 允许员工在事件流中的任意时间点保存当月的报销单。
[触发条件]：员工希望将已经录入的报销记录保存在报帐系统中。
[具体内容]：系统保存该员工的当月报销单，并给出确认信息。员工可以在保存当月报销单之后直接退出系统。
[返回位置]：同“起始位置”。

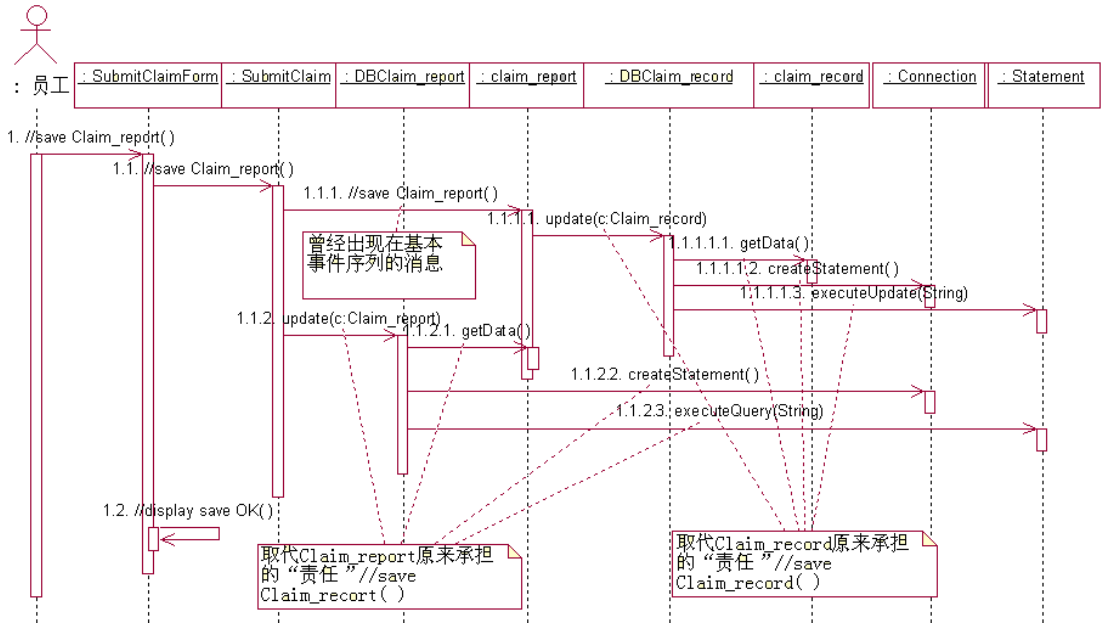

图 12　A4.备选事件序列(保存 Claim_report)

A5 报销记录不合理

[起始位置]：基本事件序列中，“验证报销单”步骤中对每一条报销记录验证结束之后。

[触发条件]：报销记录不满足某一条适用的准则。有两种情形：第一，某报销记录的金额超出了其对应类型费用的上限，已知有三种：请客户用餐人均超过 300 元，出差时每天住宿费超过 800 元，移动电话费在无特殊说明情况下超时 800 元；第二，报销费用的类型和员工所处的部门及只能不匹配。已知的情形是业务部门的员工申请加班补助。

[具体内容]：告知员工不合理的报销记录编号，以及未通过验证的原因。

[返回位置]：基本事件序列中的“填写报销单”步骤，目的是更正有问题的报销记录。

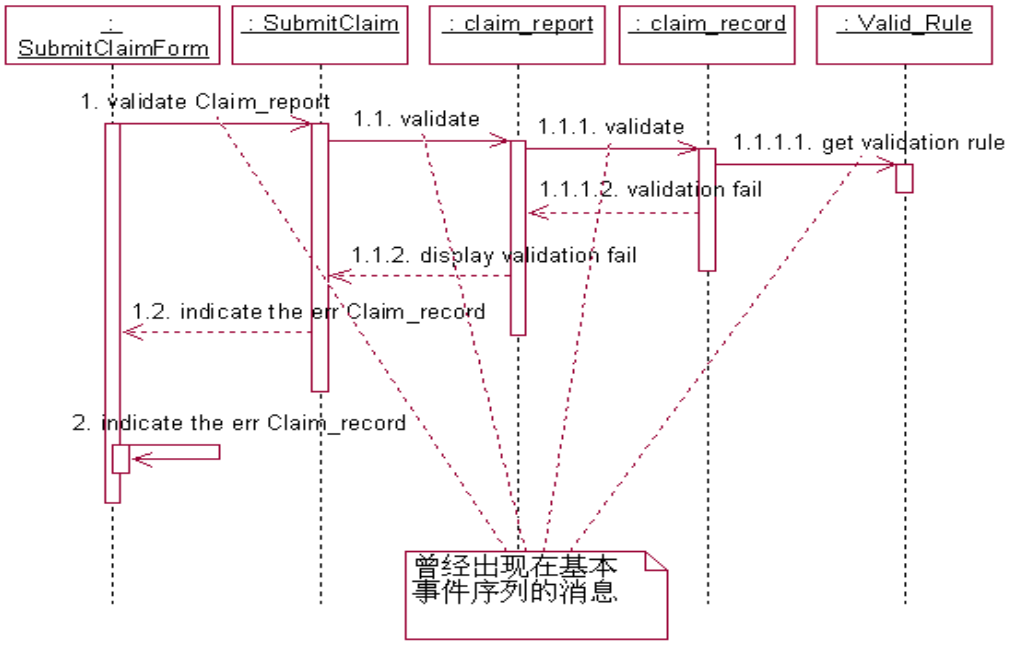

图 13　A5.备选事件序列(Claim_record 不合理)

A6 人事管理数据库不可用

[起始位置]：基本事件序列中，“提交报销单”步骤的结尾。

[触发条件]：当报帐系统向人事管理数据库索取信息而该数据库没有正常的响应。

[具体内容]：以对话框形式告知员工“人事管理数据库不可用，报帐单没有提交成功。”

[返回位置]：USE CASE 执行结束。

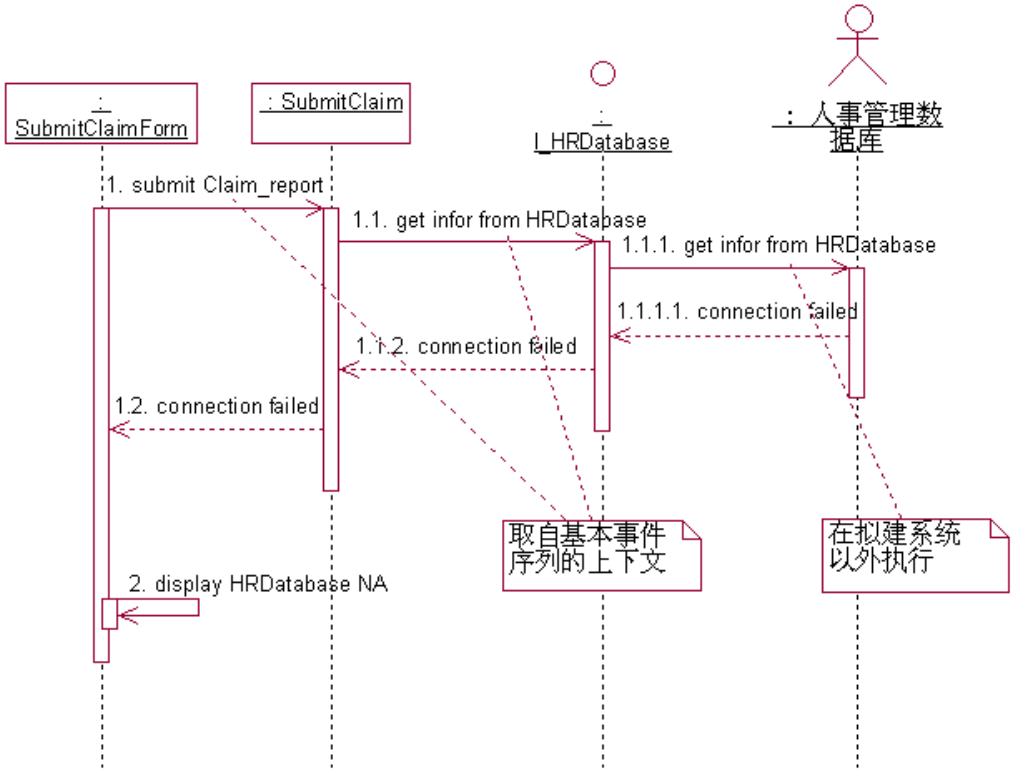

图 14　A6.备选事件序列(HRDatabase 不可用)

A7 邮件未及时发出

[起始位置]：基本事件序列中，“提交报销单”步骤的结尾，成功地从人事管理数据库获得相关信息之后。

[触发条件]：报帐系统要求发送相关邮件时邮件系统没有及时响应。

[具体内容]：系统将以提示信息的方式告知员工，“邮件没有及时发出，但是报销单在系统内已经提交成功，待邮件系统恢复后，相关邮件会自动发出。”

[返回位置]：USE CASE 执行结束。

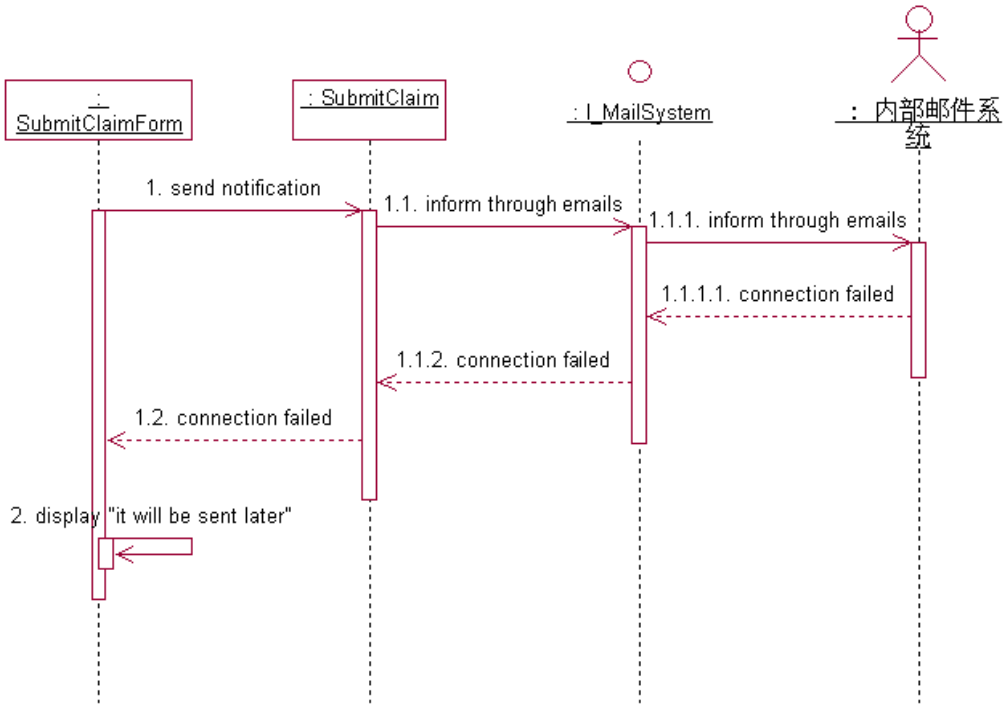

图 15 A7.备选事件序列(MailSystem 不可用)

3.3 参与类图

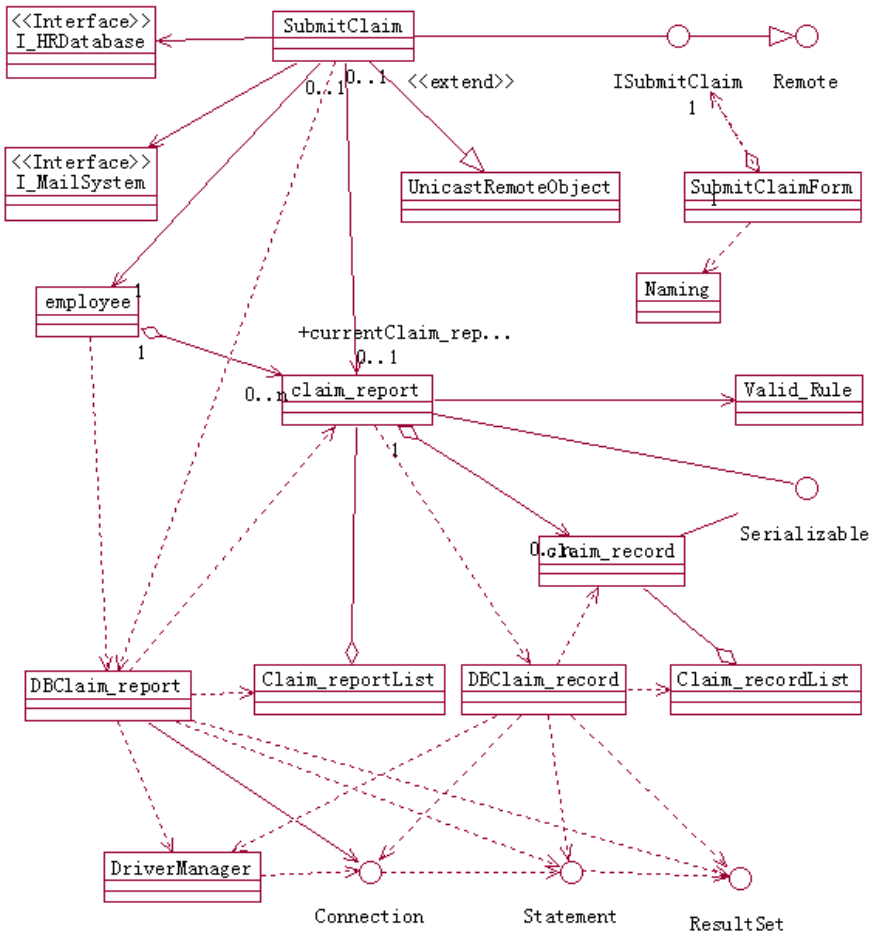

图 16　参与类图

3.3.1 设计类：

员工 employee。公司的正式雇员。

报销记录 Claim_record。与业务有关的某一项具体的花费，包括业务活动发生的时间、地点、客户名称（可选）、原因以及费用金额和种类（交通、餐饮、会议、通讯和杂项）。

报销单 Claim_report。员工在一个（自然）月内的所有报销记录的集合。

验证规则 valid_rule。验证为合理的记录必须满足几种条件：第一，不同种类的费用不超过相应的限额；第二，报销费用的类型要和员工的职能匹配。

提交报销申请 SubmitClaim。

提交报销申请用户界面 SubmitClaimForm。

3.3.2 子系统接口：

人事管理数据库 I_HRDatabase。该数据库中记录了有关人事管理的相关信息，与报帐

系统有关的是公司的组织机构（“员工”和“经理”的关系）。

内部邮件系统 I_MailSystem。该邮件系统负责收发与公司业务有关的电子邮件信息。

3.3.3 实现“留存”机制

DriverManage。用于建立数据库连接。

Connection。用于保持与数据露之间的连接。

Statement。用于直接和数据库进行交互。

ResultSet。用于记录 SQL 查询返回的结果。

Claim_record 使用 RDBMS-JDBC 的静态结构如图 17 所示。

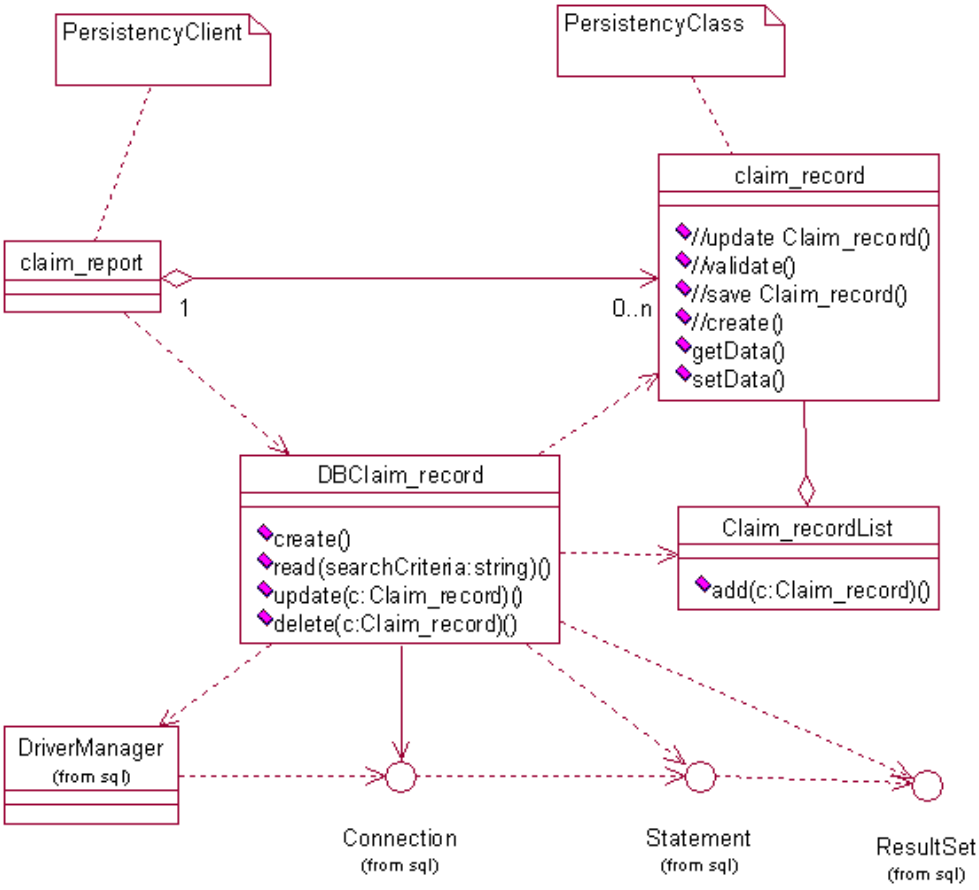

图 17 Claim_record 使用 RDBMS-JDBC 的静态结构

Claim_report 使用 RDBMS-JDBC 的静态结构如图 18 所示。

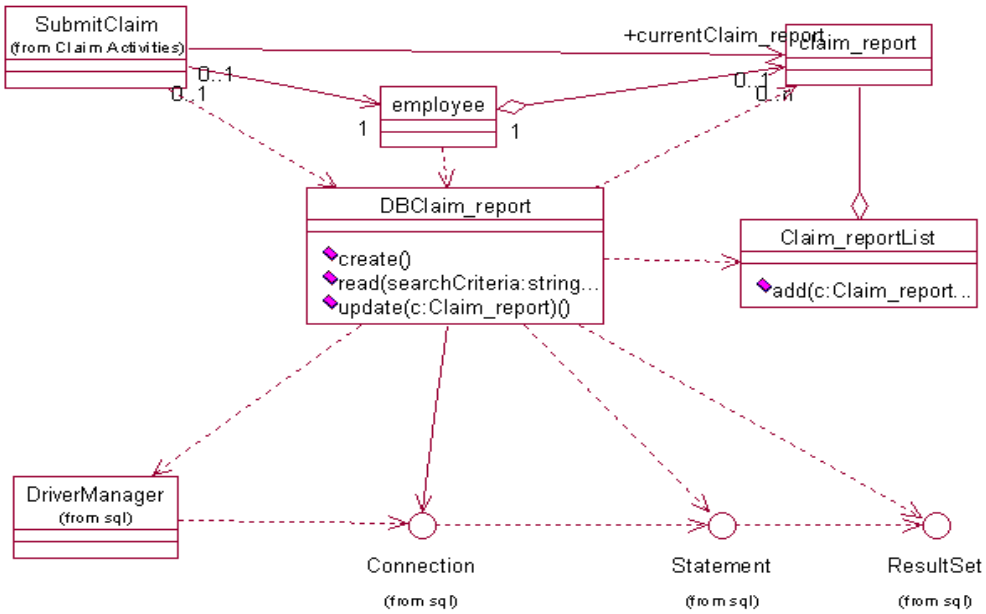

图 18　Claim_report 使用 RDBMS-JDBC 的静态结构

3.3.4 实现“分布处理”机制

采用 RMI 实现“分布处理”机制，需要使用 java.io 和 java.rmi 包中的类。

Naming。用于寻找分布在异地的对象，每一“地点”有一个此类的实例。

Serializable。一个 java 的 Interface。异地之间作为参数传送的对象必须实现(Implement)这个 java 的 Interface。

Remote。一个 java 的 Interface。被分布到异地的类必须（直接或间接）实现(Implement)这个 java 的 Interface。对于实现 Remote Interface 的类，环境会建立相应的 remote stub 和 remote skeleton，由它们具体解决实际运行中的分布处理通讯。

UnicastRemoteObject。支撑创建和导出被分布到异地的对象。

SubmitClaim 使用 RMI-JAVA 的静态结构如图 19 所示。

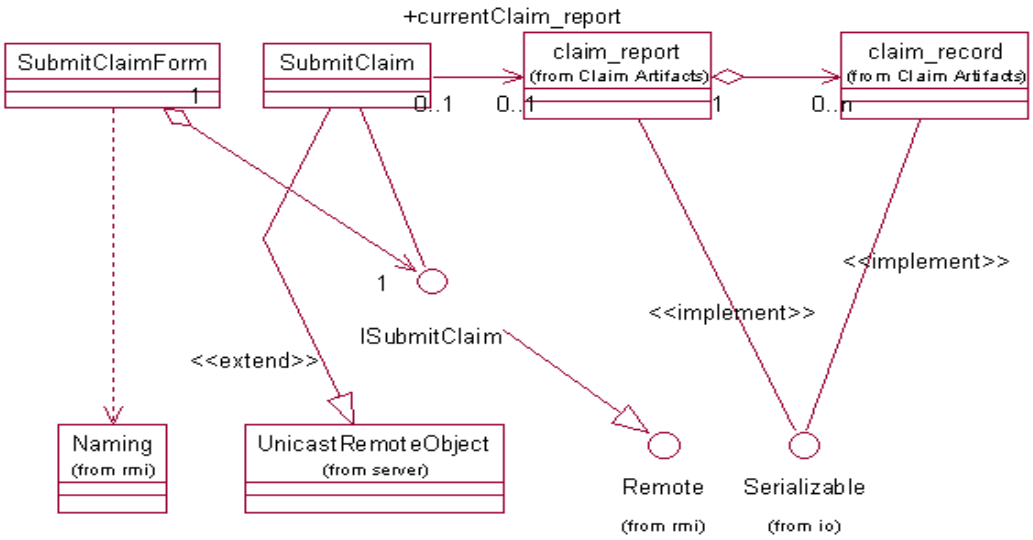

图 19 SubmitClaim 使用 RMI-JAVA 的静态结构

3.4 启动条件

员工成功登陆系统，通过身份验证。被系统提示进入“报销申请”或者“借款申请”功能。

3.5 结束状态

如果该 USE CASE 顺利执行，员工的报销申请记录将被建立，更新，保存或者保存并提交；否则，系统的状态应该保持和该 USE CASE 执行之前相同。

3.7 补充规约

用 JAVA 实现对关系型数据库的访问和分布式处理。

4. 设计模型纵览报告

- “USE CASE 实现”。反映软件需求对设计内容的驱动。
- 层次架构。将分析和设计的结果按照特殊到一般的等级分组，层次架构中的内容是后续开发活动的直接依据。
- 架构机制。描述可复用的设计经验。

逻辑上，层次架构依赖于“USE CASE 实现”和架构机制。如图 20 所示。层次架构中的内容代表了分析和设计活动的实质结果，处于设计模型的核心地位。

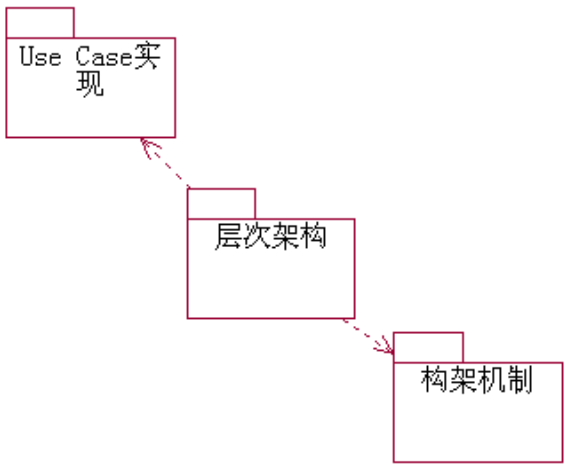

图 20 设计模型总体

图 21 给出了层次架构一种典型的划分方式。

- 特定应用层。包括那些仅仅与当前应用逻辑相关的设计要素及组合。
- 一般应用层。包括那些不仅在当前应用中有价值，在其他相关应用中可能具有重复利用价值，并且不属于纯粹软件技术范畴的“设计元素”及组合。
- 通用服务层。包括那些和应用领域无必然联系、属于软件技术范畴的“设计元素”及相关组合。

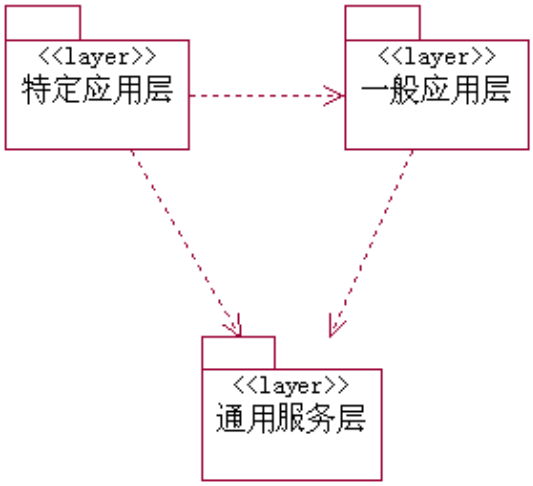

图 21 层次架构中的层次关联

在“特定应用层”内建立 Claim Activities 包，将类 SubmitClaim 和 SubmitClaimForm 放在一起，如图 22 所示。在该包内用一张类图描述“包内部的关系”，参见图 23；

在“一般应用层”中建立 Claim Artifacts 包，将关系紧密的类 Claim_record 和 Claim_report 放在一起。考虑到类 Valid_rule 与类 Claim_report 的关系比较紧密，同时这个类很可能参与其他 USE CASE 实现的协作，将类 Valid_rule 也放在此处，如图 22 所示。在该包内用一张类图描述“包内部的关系”，参见图 24；

在“一般应用层”中建立 External Interfaces 包，将表述与外部系统交互的“子系统接口”以及表述它们的实现和依赖关系的类图放在此处，参见图 22

图 22 层次架构

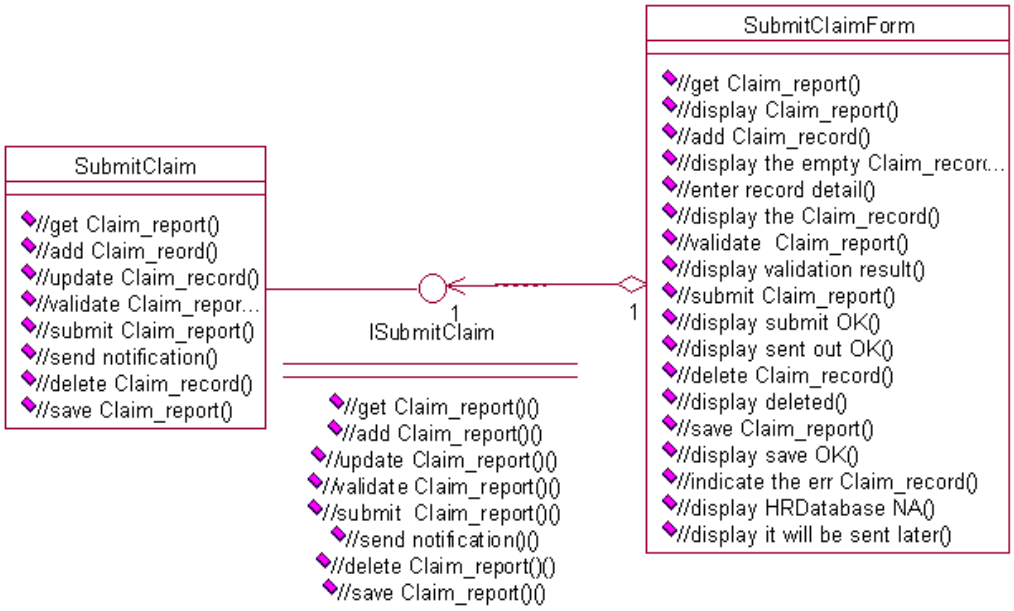

图 23 “特定应用层”中 Claim Activities 包内部的关系

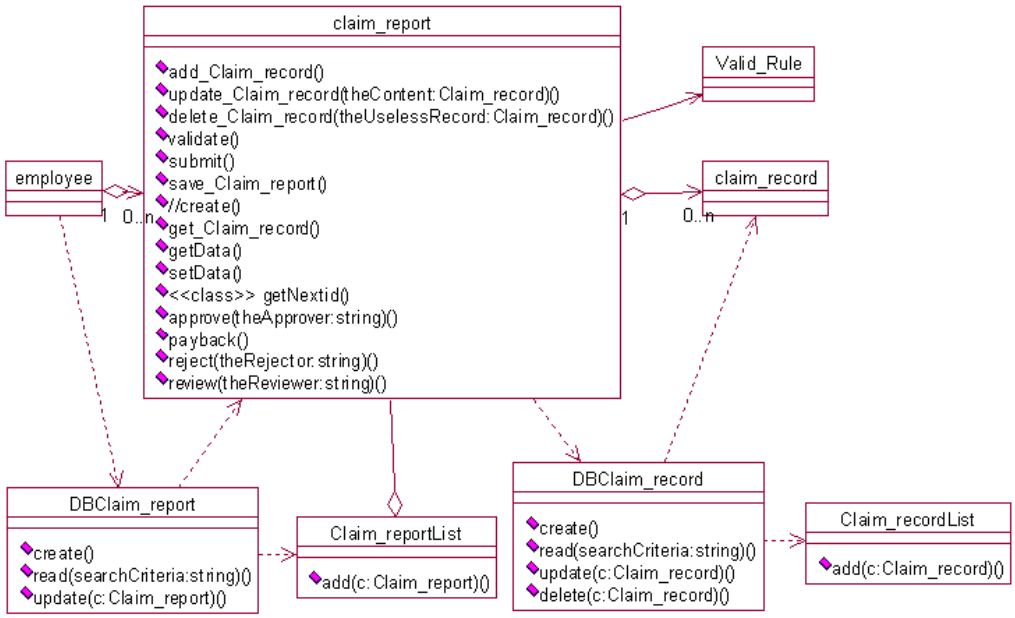

图 24 “一般应用层”中 Claim Artifacts 包内部的关系

根据“子系统接口”I_HRDatabase 对类 Employee 的依赖关系，以及“子系统接口”I_MailSystem 对类 Claim_report 的依赖关系，在 External Interfaces 包和 Claim Artifacts 包间建立依赖关系，参见图 25。

类 SubmitClaim 所属包与支撑“分布处理”机制的“基础设计元素”所属包之间建立依赖关系，即包 Claim Activities 对包 java.rmi 的依赖关系，参见图 25。

类 Claim_report 和 Claim_record 所属包与支撑“分布处理”机制的“基础设计元素”所属包之间建立依赖关系，即包 Claim Artifacts 对包 java.io 的依赖关系，参见图 25。

类 Claim_report、Claim_record 和 HRDatabase 所属包与支撑“留存”机制的“基础设计元素”所属包之间建立依赖关系，即包 Claim Artifacts 对包 java.sql 的依赖关系，子系统

HRDatabase 对包 java.sql 的依赖关系，参见图 25。

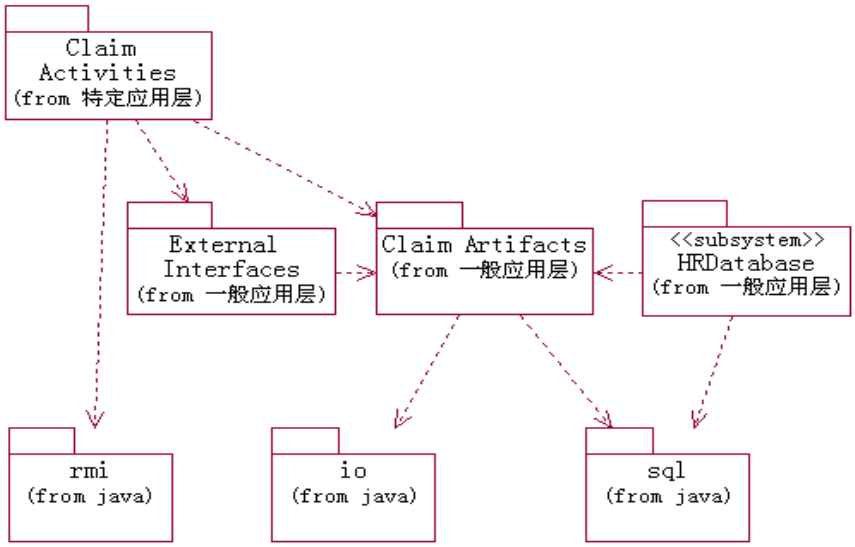

图 25 层次架构中跨越层次的包之间的依赖关系

5. 设计类报告

Claim_report。员工在一个（自然）月内的所有报销记录的集合。该类的属性和操作，如图 26 所示。

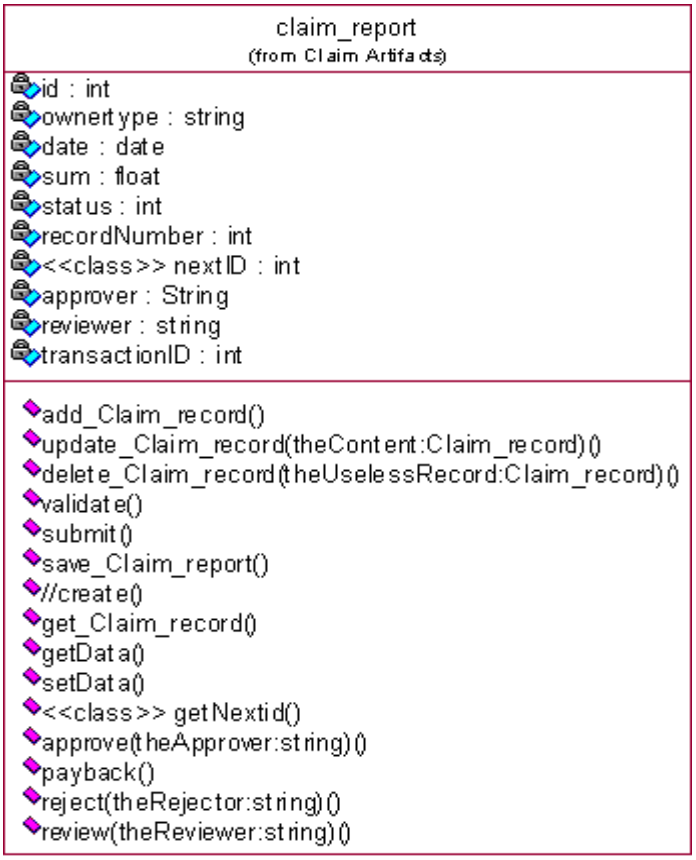

图 26 类 Claim_report

SubmitClaim 与 Claim_report 处于不同的两个包中，SubmitClaim 到 Claim_report 之间存在单向的关联关系。SubmitClaim 的实例能够访问 Claim_report 的实例，因此类 Claim_report 的可见度为“公开”。

类 Claim_report 的状态转换图如图 27 所示。

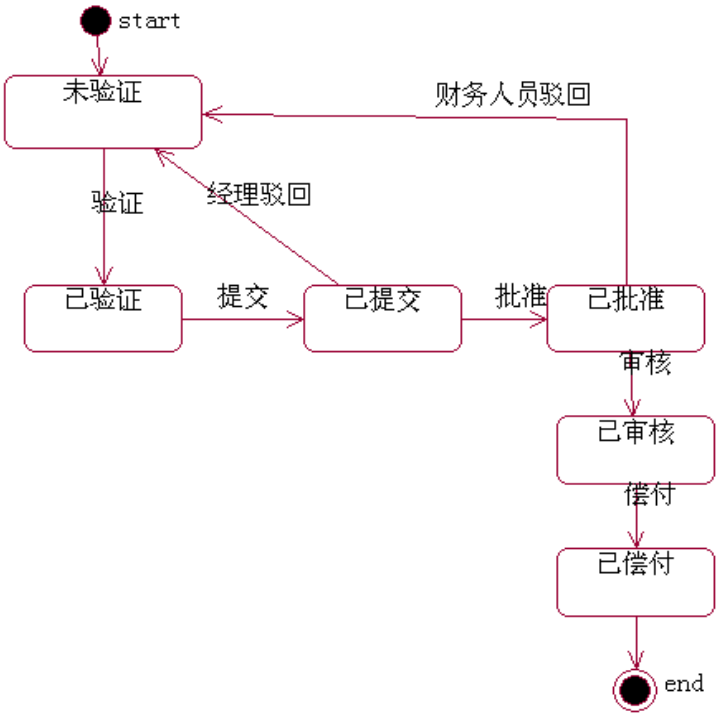

图 27 类 Claim_report 的状态转换图

类 Claim_report 各个属性、操作与类 Claim_report 各状态、转移的对应关系如表 1、2 所示。

表 1 类 Claim_report 各状态相应的属性的取值

状态（State）	相关属性的取值
未验证	Status=0
已验证	Status=1
已提交	Status=2
已批准	Status=3、approver 不为空
已审核	Status=4、reviewer 不为空
已偿付	Status=5、transactionID 不为空

表 2 类 Claim_report 各转移进行的操作

转移（Transition）	相关操作
验证	Validate()
提交	Submit()
批准	Approve()
审核	Review()
偿付	Payback()
财务人员驳回	Reject()
经理驳回	Reject()

操作 save_Claim_report()的具体含义是保存与 Claim_report 相应的全部 Claim_record。鉴于引入“留存”机制，Claim_report 自身信息的保存由 DBClaim_report 代办。

GetData()和 SetData()是在应用“留存”机制时引入操作。

GetNextID()是“属于类”的操作，其返回值在建立新的 Claim_report 时被使用。

NextID 是“属于类”的属性，用于记录下一个新增 Claim_report 的序号。

属性 recordNumber 用于记录某一 Claim_report 当前包含的 Claim_record 的个数。当创建新的 Claim_record 时，该属性用作相应 Claim_record 的序号。

6. 学习及设计体会

模型可以帮助我们在简约繁复的基础上，捕捉现实问题的本质。勾勒软件方案的雏形。模型有助于在问题到方案的过渡过程中更好地认知、理解和沟通。

按照纯粹的“瀑布式”开发思想，将软件需求向设计方案作一次性的映射，通常不能避免三方面的问题。

- 众多非重点问题分散对核心问题的注意力。
- 重复解决类似的问题，降低设计方案取得实质性进展的效率。
- 某一局部的设计被验证有问题之后，其他相关局部的工作成为无用功。

采用迭代化的开发思想，能够较好地避免上述问题。一次迭代做一部分，然后像滚雪球一样推进。而且迭代化方法中通常不作过多的假设，尽量降低对以往工作结果进行大面积否定的可能。在现实生活中，前期活动中的假设往往会导致后续工作不得不将错就错，表面上还能满足要求，但暗中牺牲了整体的质量隔和持续发展的能力。

面向对象应用建模的实践过程有三个目标：

- 有步骤、分层次地演进系统构架。
- 将软件需求逐渐转变为软件的设计方案。
- 保障软件的设计方案能够适应实施环境。