北京航空航天大學
BEIHANG UNIVERSITY

UML 建模之旅：
“旅游业务申请”系统分析与设计建模
案例使用说明书

编写单位：北京航空航天大学软件学院

编 写 人：谭火彬，林广艳

编写时间：2018 年 10 月

目　录

1. 案例说明

本案例完整地展示如何利用 UML 开展系统分析和设计。借助于 UML 所提供的各种模型，可以有效地处理系统分析和设计中的各类问题。目前，该案例主要用于“面向对象分析与设计”课程教学，贯穿课程教学的各个阶段。该案例可以用于课程教学阶段，也可用于学生实践。

该案例总共包括 3 个组成部分，分别是需求建模、系统分析和系统设计；这三部分是软件系统编码前的三个核心过程，也是软件工程专业学生必备的专业技能。本案例通过利用 UML 完成三部分的工作，通过带领学生完成 UML 建模之旅，从而向学生全面展示了如何利用 UML 建模技术来构建系统的需求、分析和设计模型。教师可根据理论授课的进度，逐步完成案例教学内容。

2. 案例教学目标

本案例适用于软件工程专业的高年级本科生和研究生，其的目标是就是针对前面提出的三个方面的问题，引入 UML 建模技术，引导学生通过 UML 建模完成需求定义、需求分析和系统设计这三个软件系统开发。具体的教学内容包括以下三个方面的建模工作：

（1）基于 UML 用例模型的需求定义方法。通过利用 UML 用例图、用例文档等技术，引导学生构建目标系统的需求模型，以完成需求定义工作。

（2）基于 UML 的用例分析方法。系统地介绍 UML 包图、顺序图和类图等技术，引导学生构建目标系统的分析模型，完成用例分析工作。

（3）基于 UML 的系统架构设计和对象设计方法。借助于 UML 包图和相应的包设计原则，完成系统的架构设计方案；并利用 UML 类图和交互图完成对象设计。

3. 案例准备

本案例是配合面向对象系统分析与设计课程教学使用案例，在教学的不同阶段使用案例的不同部分。下一章案例教学要点详细介绍了各个部分的实践内容。学生在在开始学习和实践整个案例之前应具备的基本知识主要包括：

（1）学生了解了面向对象的基本概念和方法，掌握一门面向对象的编程语言。

（2）学生会使用基本的 UML 建模工具绘制各类 UML 模型。

对于每个案例教学要点，学生在开始实践前，应已完成相应的知识点的学习，每部分对应的知识点分别为基于用例的需求建模、面向对象的分析和面向对象的设计。

课前，教师可以下发“CASE01-LY01-案例描述文档”，让学生首先属性案例相关的业务背景。此外，本案例已经整合到作者出版的教材中，选用本案例的教师，可以考虑同步选用配套教材《UML2 面向对象分析与设计》（清华大学出版社）。

4. 案例教学要点

配合三个教学目标，本案例教学分为三个阶段，教师可以配合理论教学内容的开展，使用不同阶段的内容开展案例教学。

4.1 需求建模

需求是项目开发的基础，描述系统必须满足的条件或具备的能力，而 UML 用例建模技术则是

一种非常有效的需求定义手段。通过本部分的学习能够了解需求的基础概念，并掌握利用用例技术进行需求建模的方法和实践过程，并能够动手完成某一给定系统的用例模型。

4.1.1 识别参与者

构建用例模型的第一步就是识别模型中的参与者。参与者代表了以某种方式与系统交互的人或事；他是在系统之外，透过系统边界与系统进行有意义交互的任何事物。

运用课堂教学中所提到的识别参与者技巧，从案例描述中获取目标系统等参与者。当然，在实际系统开发时，这个过程是在与用户沟通过程中完成的，通过与用户沟通，记录那些系统的各种外部因素（包括系统用户、外部系统和外部激励等），再从在系统的职责入手分析这些外部因素从而来确定是否作为参与者，如果是参与者，则从系统角色的角度给出合适的名字。为了使该过程更具操作性，可以采用表 1 的方式来逐步分析和确定参与者。

表 1. 获取系统参与者

抽取角度	外部事物种类	主要日常工作	使用目标系统职责	参与者	典型代表
相关用户	前台招待顾客的员工	洽谈客户事宜并为客户办理各种申请和取消手续、完成费用支付	办理申请手续以及相关的取消、支付等后续业务	前台服务员	具体用户代表
	负责催款的员工	打印和邮寄旅游确认书和交款单	打印旅游确认书和交款单	收款员工	…
	旅行社内的会计人员	财务记帐	不使用本系统，不是参与者		…
	宣传和路线管理员工	制作宣传资料、定期维护旅游路线和活动	维护旅游路线和旅游活动	路线管理员	…
其他外部事物	财务系统	记帐等财务操作	接收本系统中与现金相关的财务信息	财务系统	...
	外部激励	关注或影响系统的运行	定期自动导出财务信息	时间	

从表中可以看出，该系统存在五个参与者，根据他们在系统中的职责给出了合适的名字。而其中旅行社内的会计人员并不能成为参与者，因为他并不直接使用该系统，而是使用财务系统进行日常工作。而财务系统与本系统存在信息交互，因此也作为一个参与者存在。另外，由于系统每天晚上需要定期运行，因此还需要定义一个时间参与者。获得了这些参与者之后，将这些参与者在建模工具中绘制出来，如图 1 所示。

图 1 旅游申请系统参与者视图

4.1.2 识别用例

有了这些参与者之后，就可以从参与者使用系统的职责入手来定义用例；参与者使用系统所要达成的一个目标（或者说所要完成的一项工作）就作为一个用例而存在；当然如果这项工作不通过系统来完成就作为用例存在。这个过程也可以使用表 2 的方式来完成

表 2 从参与者的角度获取用例

参与者	主要工作	是否使用系统	用例
前台服务员	向申请人介绍申请情况	否	
	为申请人办理申请手续	是	办理申请手续
	对申请参加人的增、删、改、查等日常维护	是	管理参加者
	记录申请人支付信息	是	完成支付
	为申请人取消申请	是	取消申请
收款员工	打印旅游确认书和余额交款单	是	打印旅游确认书和余额交款单
	邮寄旅游确认书和余额交款单	否	
路线管理员	制作宣传资料	否	
	设计旅游路线	是	管理路线
	设计旅游团（活动）	是	管理旅游团
	调整旅游团价格	是	设定价格
财务系统	记账等财务操作	否	
	接收与现金相关的财务信息	是	导出财务信息（被动）
时间	定期导出财务信息	是	导出财务信息
其它 辅助用例	系统要区分各种不同的用户身份，并提供不同的功能	是	登录

这里主要存在的问题是“办理申请手续”用例的粒度问题，案例描述中有很大一段篇幅介绍了办理申请手续的过程，即员工首先要查询申请信息，之后录入申请责任人和其它相关信息，最后还要录入订金信息并打印收据。系统分析师有可能会把这里的几个步骤分成几个用例来表示，但仔细考虑前台服务员所完成的这些步骤实际上只有一个目的，即为申请人办理完成相关的申请手续，这些步骤是紧密联系在一起的。因此作为只需要定义一个“办理申请手续”用例即可描述用户目标，至于这些细节将在下一阶段作进一步描述。当然，其中的某些步骤可能可以独立存在，如“查询申请信息”，而且前台服务员也可能需要这样一个独立的业务，此时可以把这样独立存在的步骤独立形成一个子用例，这部分内容将会在重构用例模型中完成。但由于目前只是用例建模的早期，对于每个用例的内部细节还没有描述，因此现阶段也没有必要进行这样的分解。

4.1.3 构造用例图

识别这些参与者和用例之后，就可以采用用例图把它们表示出来，图 2 就是该系统的初始用例模型。注意参与者和用例之间的关联关系的定义，以及关联方向的含义。此外，在画用例图时注意图形的美观，采用合理的布局保持图形整洁、清晰，尽量不要出现一些线和图形的交叉等容易混淆的情况。

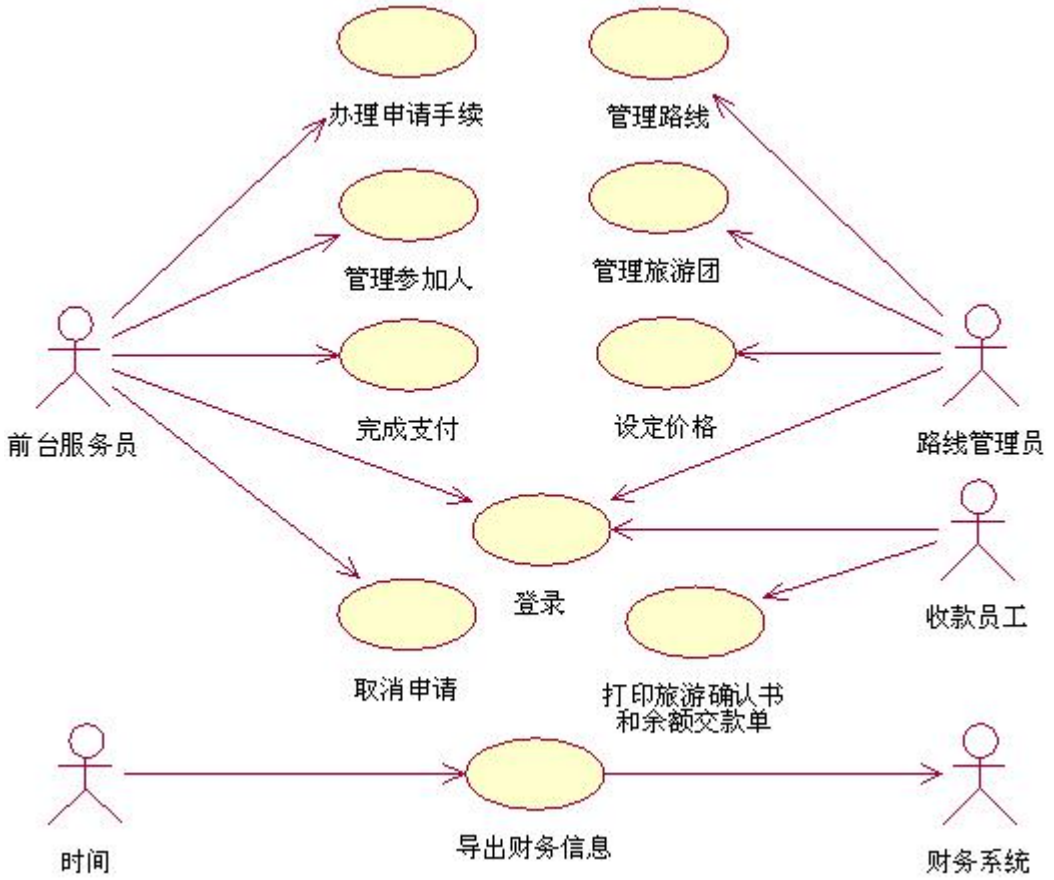

图 2. 旅游申请系统初始用例图

4.1.4 编写用例文档

绘制出系统的初始用例图后，整个用例建模过程其实才刚刚开始；更主要的工作就是要描述用例内部的处理细节，这是很多用例初学者容易忽略的问题。事实上，需求作为开发方和用户所达成的契约，必须要定义得非常具体而且准确；仅仅通过一幅图形显然是无法满足要求的。而且，正如前面所说，一个用例是用户所需要达到的目标，为了达到这个目标用户需要和系统进行频繁的交互，这些交互过程也无法在用例图中体现出来。为此，在完成用例图后，需要对图中的每一个用例进行详细的描述，这个描述的过程就是编写用例文档。通过文档的方式将用户与系统交互的过程一一记录下来，从而为以后的分析和设计提供基础。可以这样来描述用例图和用例文档的关系：用例图是整个需求的骨架，而用例文档则是需求的肉。也就是说，通过用例图建立了需求模型的基本结构，而需求的内容则需要通过用例文档来表示。此外，用例文档是与单个用例关联的，需要为每个用例编写一份独立的文档。

节给出了旅游申请系统的两个核心用例的文档，注意其中一些细节的编写方法。表 3 给出了旅游申请系统中“办理申请手续”用例的文档。

表 3 “办理申请手续”用例文档

用例名	办理申请手续
简要描述	前台服务员通过该用例为申请人办理申请旅游团的手续
参与者	前台服务员
涉众	申请人、前台服务员
相关用例	暂无
前置条件	前台服务员登录到系统

<table>
<tr><td>后置条件</td><td>申请信息被正确保存，相关旅游团可申请人数减少</td></tr>
<tr><td colspan="2">基本事件流
1. 该用例起始于旅客需要办理申请手续；
2. 前台服务员录入要申请的旅游团旅行路线代码和出发日期；
3. 系统查询要申请的旅游团信息（A-1）；
4. 系统显示查询到的旅游团和相关路线信息（D-1）（A-2、A-3）；
5. 前台服务员录入本次申请信息（D-2）；
6. 系统显示旅行费用的总额和申请订金金额；
7. 前台服务员提交该申请信息；
8. 系统保存该申请信息（A-4），用例结束。</td></tr>
<tr><td colspan="2">备选事件流
A-* 前台服务员在提交该申请前，随时都可能中止该申请
1. 系统显示中止确认的消息；
2. 前台服务员可以结束该用例，也可以选择继续录入下一个申请。
A-1 无法查询到所需的旅游团信息
1. 系统显示录入的旅游线路代码或者出发日期有误信息；
2. 前台服务员再次录入旅游路线代码和出发日期，也可以结束用例。
A-2 旅行已超过申请截止日期
1. 系统提示已超过申请截止日期，不能申请；
2. 前台服务员重新输入旅游线路代码和新的出发日期，也可以结束用例。
A-3 可以申请的人数为 0 人
1. 系统提示旅游团人数已满；
2. 前台服务员重新输入新的旅游线路代码和出发日期，也可以结束用例。
A-4 保存信息失败
1. 系统显示保存失败，并提示用户需要再次提交；
2. 前台服务员可以重新提交该申请，也可以结束用例。</td></tr>
<tr><td colspan="2">补充约束-数据需求
D-1 显示的旅游团和路线信息包括：旅游路线代码、旅游路线名称、出发日期、天数、申请截止日、可申请人数、大人的单价和小孩的单价等。
D-2 录入申请信息包括：申请责任人的姓名、电话号码、参加的大人人数、小孩人数
补充约束-业务规则
B-1 所申请旅游团的截止日期在申请日期之前
B-2 所申请旅游团的人数限额未满
B-3 申请订金的计算规则如下表所示：
<table>
<tr><th>距出发日期的天数</th><th>订金比例</th></tr>
<tr><td>≥2 个月</td><td>10%</td></tr>
<tr><td>≥1 个月，且<2 个月</td><td>20%</td></tr>
<tr><td><1 个月</td><td>全款</td></tr>
</table></td></tr>
<tr><td colspan="2">待解决问题
（暂无）</td></tr>
<tr><td colspan="2">相关图
（暂无）</td></tr>
</table>

表 4 给出了“管理参加人”用例文档，与前一个用例文档不同的是，该用例文档的基本事件流被划分成三个子流程，通过子流程的方式可以使该用例文档的结构更加清晰。

表 4 “管理参加人”用例文档

用例名	管理参加人
简要描述	前台服务员通过该用例为对申请参加人的信息进行维护
参与者	前台服务员
涉众	申请人、申请参加人
相关用例	暂无
前置条件	前台服务员登录到系统
后置条件	申请参加人的信息被正确的录入到系统中

基本事件流

1. 用例起始于前台服务员需要对申请的参加人信息进行维护；
2. 前台服务员输入查询条件（D-1），查询申请信息；
3. 系统查询该申请（A-1），并显示申请详细信息（D-2）；
4. 前台服务员选择所要进行的操作；
5. 系统根据前台服务员选择的操作，执行以下的子流程：
   选择“增加参加人”操作时，开始“增加参加人”子流程（S-1）；
   选择“修改参加人”操作时，开始“修改参加人”子流程（S-2）；
   选择“删除参加人”操作时，开始“删除参加人”子流程（S-3）；
6. 子流程完成后，用例结束。

子流程 S-1：增加参加人

1. 系统显示申请责任人的姓名和电话号码；
2. 前台服务员录入申请责任人信息（D-3）；
3. 前台服务员录入申请责任人旅行途中的联络人信息（D-4）；
4. 前台服务员继续录入其它参加人的信息；
5. 前台服务员录入参加人信息（D-3）；
6. 前台服务员录入参加人有关旅行途中的联络人信息（D-4）；
7. 重复步骤 5.6，录入所有的参加人（A-2）；
8. 前台服务员提交本次录入信息（A-3）；
9. 系统保存参加者信息（A-4），结束该子流程。

子流程 S-2：修改参加人

1. 系统显示全部参加人的姓名；
2. 前台服务员选出要修改的参加人；
3. 系统显示要变更的参加者信息（D-3）和联络人信息（D-4）；
4. 前台服务员修改相关的信息；
5. 前台服务员提交本次修改（A-2）；
6. 系统保存参加人信息，结束该子流程。

子流程 S-3：删除参加人

1. 系统显示全部参加人的姓名；
2. 前台服务员选出删除的参加人；
3. 系统显示取消手续费用和返还金额；
4. 前台服务员确认删除；
5. 系统保存本次删除信息；
6. 若删除的参加人就是申请责任人，为了选择新的申请责任人，系统会显示所有参加人的姓名；
7. 前台服务员选择新的申请责任人；

8. 系统录入新的申请责任人（A-4），结束该子流程。

备选事件流

A-* 前台服务员在操作提交之前，随时都能够结束子流程

1. 系统显示确认中止的消息；
2. 前台服务员可以结束子流程，也可选择继续其它操作。

A-1 没有找到申请信息

1. 系统提示未找到该申请信息
2. 前台服务员输入查询条件进行查询，也可以结束用例。

A-2 必填项有未输入项目

1. 系统提示有未输入项目；
2. 前台服务员再次输入未输入项。

A-3 尚未录入所有参加者的信息

1. 系统提示有未录入的参加者信息；
2. 前台服务员可以继续录入参加者的信息，也可登录目前已录入的参加者的信息，结束子流程。

A-4 系统保存失败

1. 系统提示保存失败；
2. 前台服务员可以再次提交，也可结束该用例

补充约束-数据需求

D-1 查询条件包括：旅游线路代码、出发日期、申请责任人姓名

D-2 显示的申请信息包括：旅游线路代码、旅游团名称、出发日期、申请日期、申请责任人姓名、支付情况。

D-3 参加人信息包括：性别、出生年月、现在住所、邮政编码、A-Mail 地址等

D-4 联络人信息包括：姓名、与本人关系、住址、邮政编码、电话号码等；

补充约束-业务规则

B-1 取消手续费如下表所示：

距出发日期的天数	取消手续费用
1 个月以上	无
1 个月到 10 天	20%
10 天到 1 天（前一天）	50%
0 天（出发当天）	全款
距出发日期的天数	取消手续费用

待解决问题

（暂无）

相关图

（暂无）

4.1.5 重构用例模型

有了基本的用例模型和相应的用例文档，就完全可以对各类应用的需求进行定义。然而，对于大规模的复杂应用而言，仅使用这些可能还是不够的。在这些大规模系统中，可能会存在十几个甚至几十个用例，过多的用例也使得用例图变得过于复杂；而且每个用例的规模也存在很大的差别，有的比较简单（如登录），有的特别复杂，其文档可能就有十几页甚至几十页；此外，每个用例的重要程度也不一样。针对这些问题，需要采用更多的用例建模技巧，来对初始的用例模型进行重构，以便构造规模适中、结构合理的用例视图。在本案例中主要采用用例关系来重构用例模型。通过用

例关系将复杂的用例进行适当的分解，以便于提高需求的复用性和可扩展性等，从而使用例模型的结构更合理。

在编写旅游申请系统的用例文档时，可以发现“管理参加人”和“完成支付”中都首先需要查询已有的申请信息，并显示申请的情况；虽然有些细节的差异，如管理参加人时并不需要显示有关余款的信息，而完成支付时还可以按照交款单编号来查询。但这些细微的差别可以通过适当地调整用例文档来解决（如管理参加人虽然不需要支付信息，但查询到这些信息也没什么影响，只是不需要处理而已）。因此，从这两个用例中就可以抽取出公共行为“查询申请信息”，而这个公共行为即可构成一个单独的被包含用例“查询申请信息”。这三个用例之间的关系如图 3 所示。注意图中包含关系的方向，由基用例指向被包含用例。

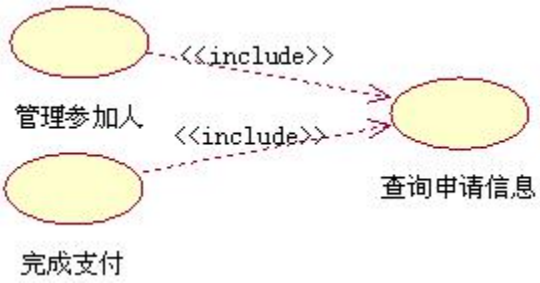

图 3 用例间的包含关系

通过包含关系对用例模型进行重构后，相应的用例文档也会受到影响。原有的用例中已经抽取出了的部分就不再出现在用例文档中，而是通过引用被包含用例来表示。当然，还需要为新增加的“查询申请信息”用例编写相应的用例文档。

在编写旅游申请系统的用例文档时，可以发现“导出财务信息”用例中存在一些复杂的异常处理行为（即 A-2 备选流），这部分行为在当前用例中并没有展开论述；而且这些异常处理方式本身和当前导出财务信息的业务没有太大的关系。因此，完全可以把此部分内容分离出去，构成扩展用例，修改后的用例图如图 4 所示。

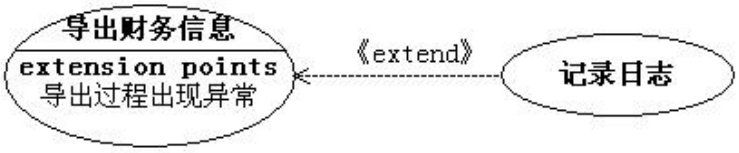

图 4 用例间的扩展关系

从图中可以清楚地看出基用例“导出财务信息”定义了一个扩展点“导出过程出现异常”，而扩展用例“记录日志”则是基于该用例进行扩展的。需要说明的是，目前很多工具并不将扩展点在用例图中显示出来，但不管是否显示出来，只要存在扩展关系，基用例中就存在相应的扩展点；而且有关扩展点和扩展关系也还需要在用例文档中体现。

4.2 系统分析

通过用例模型对系统的需求进行了完整地规格说明，这个规格说明将作为后续分析和设计的依据。而如何继续后续的分析过程也并没有一个统一的标准，甚至可以回到老路采用传统的结构化方法进行分析和设计；当然更好的选择显然还是继续采用面向对象方法。UML 用例分析技术则是一种典型的利用 UML 进行面向对象分析的方法，其主要思想来源于 RUP 分析设计工作流中的分析阶段，并适当的借鉴其它一些方法中的成功经验。通过本部分的学习使学生能够掌握利用用例分析方法进行面向对象分析的基本过程和实践技能，并能够动手完成某一给定系统的分析模型。

4.2.1 架构分析

架构（Architecture）在面向对象的系统中扮演着越来越重要的角色，从某种意义上来说，面向对象的分析和设计都是以架构为中心进行的。在分析和设计的不同阶段，软件系统的架构被一步步细化和完善，最终形成一个规范的、稳定的、符合设计要求的架构模型。架构分析的过程就是定义系统高层组织结构和核心架构机制的过程。在早期迭代中，架构分析的主要目标是建立系统的备选架构，以用于组织后续的用例分析所获得的分析模型，该备选架构通过架构设计进行细化和调整，从而获得软件系统的基础架构。

本案例的早期架构选型时，采用流行的分层结构，其架构如图 5 所示。

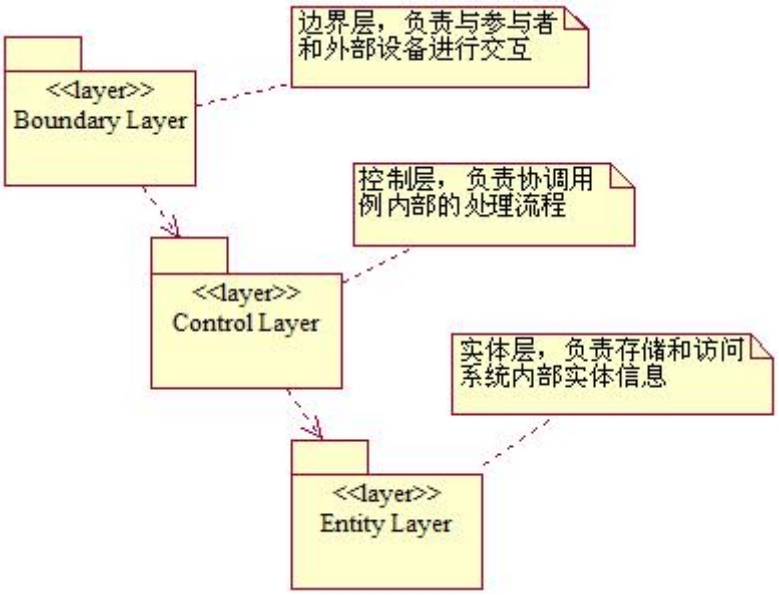

图 5 系统备选架构

在 B-C-E 三层架构的定义过程中，采用包图进行架构建模。包是一种通用的模型分组机制，为了有效地表示分层的概念需要引入构造型<<layer>>，并在层和层之间建立了依赖关系。通过 B-C-E 这三层划分系统三类处理逻辑，其中：

- 边界层(Boundary Layer)负责系统与参与者之间的交互；
- 控制层(Control Layer)处理系统的控制逻辑；
- 实体层(Entity Layer)管理系统使用的信息。

目前只是给出了这三个层次的基本定义，而各层的内部还没有任何元素，后续过程将对架构中的各个层次进行填充。

4.2.2 识别分析类

在对象系统中，系统的所有功能都是通过相应的类来实现的；因此首先需要从用例文档中找出这些可用的类，之后再将其所描述的系统行为分配到这些类中。面向对象的分析是对这个过程的第一次尝试，这是一个从“无”到“有”的跨越，也是整个分析过程中最难的一部分任务之一。而分析阶段所定义的类称之为分析类。

分析类代表了系统中具备职责和行为的事物的早期概念模型，这些概念模型最终会演化为设计模型中的类或子系统。分析类关注系统的核心功能需求，用来建模问题域对象。此外，分析类主要用来表现“系统应该提供那些对象来满足需求”，而不关注具体的软硬件实现的细节。

架构分析中所定义的 B-C-E 三层备选架构为识别分析类提供了很好的思路，按照该备选架构，系统中的类相应地对应三个层次，即边界类、控制类和实体类。识别分析类的过程就是从用例文档中来定义这三类分析类的过程。

旅游申请系统中分析类较多，根据备选架构识别了三种分析类。图 6~图 8 分别给出了该系统的边界类、控制类和实体类的主视图，从图中可以看出该系统首次迭代所识别的全部分析类，这些分析类完全是按照前面所介绍的方法识别出来的。

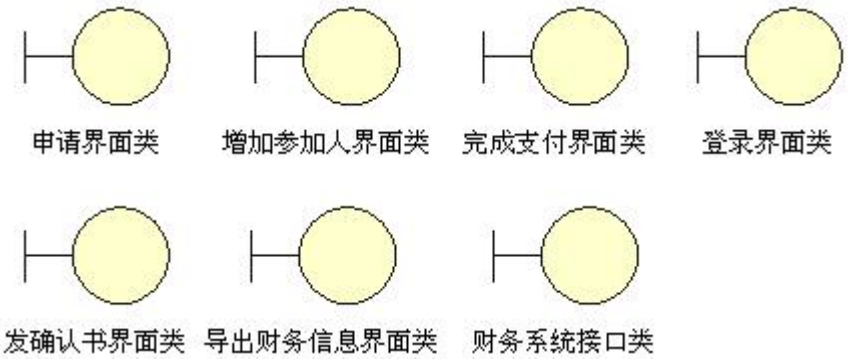

图 6 旅游申请系统初始边界类

本系统首次迭代共定义了 7 个边界类，其中包括 6 个界面类和 1 个系统接口类。由于首次迭代针对“管理参加人”用例只实现“增加参加人”子流程，因此其界面类也定义为“增加参加人界面类”；而“登录界面类”则同时处理前台服务员和收款员工两类用户的登录；此外，虽然时间参与者可能并不需要操作界面来启动系统（一般情况下是系统后台自动运行的业务），但为了后续分析的一致性，目前也定义一个“导出财务信息界面类”。

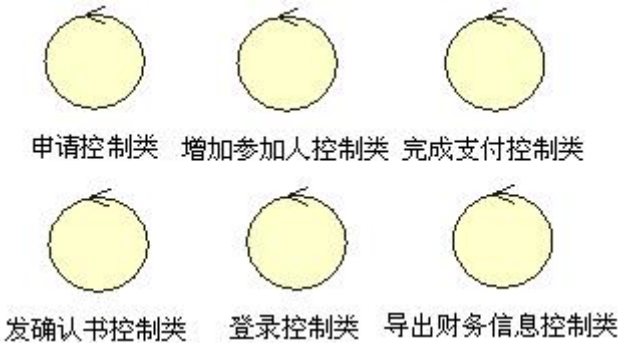

图 7 旅游申请系统初识控制类

控制类的定义比较简单，对应于首次迭代的 6 个用例，定义了 6 个控制类来封装相应用例的业务流程和逻辑规则。

图 8 旅游申请系统初始实体类

本系统目前初步定义了 7 个实体类，这些实体类与前面的关键抽象基本一致；所多出来的“用户”类是从登录用例中提取出来的，该类记录了系统的用户（包括前台服务员、收款员工和路线管理员等不同角色）信息，如用户名、密码等。关键抽象的提取更多的是凭借着对业务的理解和相关项目的经验；而此阶段对实体类的提取还可以按照前面所提到的名词筛选法来进一步明确并获得更多的实体类。

4.2.3 构造用例实现

目前所识别的分析类都是静态的描述，而为了确认所识别的分析类是否达成用例实现的目标，

必须分析由这些类所产生的对象的动态行为，这就是构造用例实现的过程。在 UML 模型中，通过交互图来表示对象间的交互，它由一组对象和它们之间的消息传递组成。下面将利用构造用例实现的相关技术，详细讨论“旅游申请系统”中“办理申请手续”用例实现的构造过程。

图 9 给出了“办理申请手续-用例实现”的基本场景顺序图。该顺序图描述的基本流程如下：

前台服务员在申请界面上首先录入路线代码和出发日期（消息 1），界面类根据这些信息向控制类查询所要申请的旅游团和路线信息（消息 1.1），控制类执行查询请求，根据查询结果生成相应的旅游团对象（创建消息 1.1.1）和路线对象（创建消息 1.1.2），并将这两个对象关联起来（消息 1.1.3，即设定该旅游团对应的路线），最后返回旅游团对象（返回消息 1.1.4）；界面对象接收到返回结果后，进行刷新，从而显示所查询到的旅游团和路线信息。基本事件流的 1~4 步完成。

之后，用户确认需要申请该旅游团，前台服务员向界面录入用户的申请信息（消息 2），界面类将申请内容提交给控制类（消息 2.1），控制类针对申请信息的不同方面交由相应的实体类进行处理：首先生成一个申请对象（消息 2.1.1），并与旅游团对象关联（消息 2.1.2），表明该申请所对应的旅游团；之后，生成一个参加人对象（消息 2.1.3），来存储申请的责任人信息，并在申请对象中关联责任人信息（消息 2.1.4）；最后，控制类要求申请对象计算本次申请有关的支付信息（消息 2.1.5），申请对象根据自身的信息（包括申请的大人人数、小孩人数、申请日期等）和所关联的旅游团信息（包括旅游团的价格、出发日期等）来计算费用和订金等支付信息（消息 2.1.5.1），并生成支付明细对象来保存相应的结果（消息 2.1.5.2），并将结果返回给控制类（返回消息 2.1.5.3）。控制对象将本次申请的明细返回给界面后（省略了该返回消息），界面类进行刷新显示（消息 2.2）。

最后，用户根据系统计算出来的订金进行支付，服务员通过界面类将用户的支付信息录入到系统（消息 3），界面将支付结果提交给控制类（消息 3.1），控制类根据支付结果更新申请对象的状态（消息 3.1.1），同时申请对象也会把支付情况记录到支付明细对象中（消息 3.1.1.1）。

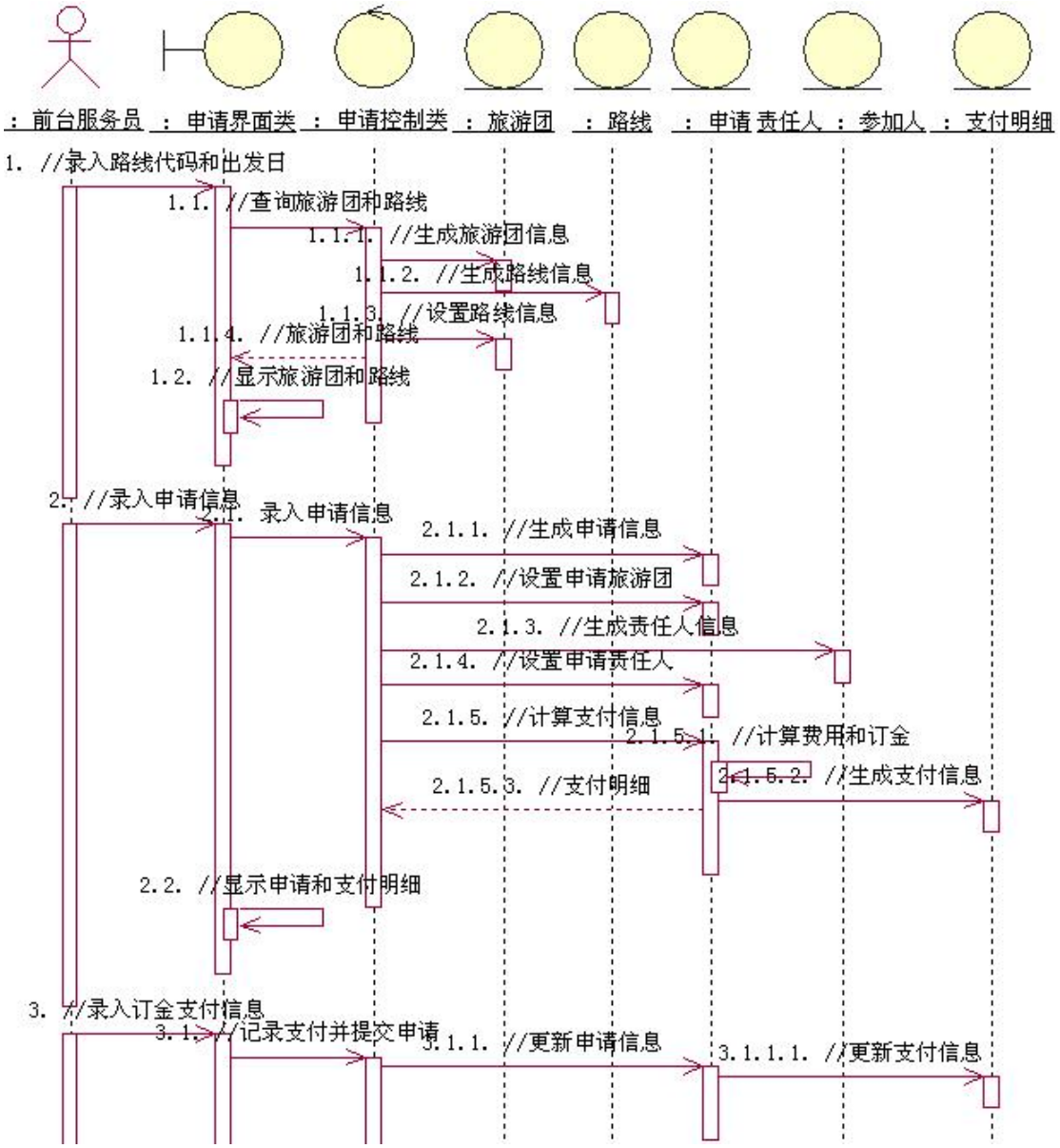

图 9 办理申请手续-用例实现的基本场景顺序图

针对该顺序图的绘制，还存在三个方面的问题需要进一步说明：

① 命名对象。正如前面所展示的顺序图，图中大部分对象都是匿名对象，并不需要指定特定的对象名称，只需要明确所属的类类型，从而泛指所有通过该类所构造的对象。但在有些特殊场合下，为了特指该类某个特殊的对象，需要为该对象取特定的名字。在该图中，为“参加人”类定义了一个“责任人”对象，从而区分不同参加人的身份。这意味着，在当前办理申请手续-用例实现中，只需要记录那个作为责任人的参加人信息，而不需要维护其他普通参加人的信息。

② 设置对象间的关联。在该顺序图中，有诸如 1.1.3、2.1.2、2.1.4 等“设置某某信息”的职责，这类职责实际上是建立两个对象之间的关系。比如 1.1.3 职责是为旅游团设置其路线信息：通过控制类查询到了旅客所要申请的旅游团和路线，并分别存储到两个类中，但对于某个旅游团对象而言，应该要指定该旅游团是针对哪个路线的，即要建立旅游团类和路线类之间的关联关系，这个操作就是 1.1.3 所要达到的目标。同样的道理，2.1.2 建立申请和旅游团之间的关系，而 2.1.4 建立申请和责任人之间的关系。

③ “计算费用和订金”职责的处理。对于此类与业务逻辑相关的、涉及到多个对象的职责很

多时候都是由控制类来处理，但该顺序图中将该职责分配给申请类。而这种分配策略在此处是非常合适的，因为计算费用和订金所需的申请人数信息和旅游团费用信息都与申请类之间存在关系，即通过申请类可以明确的获得这些内容，因此按照专家模式，这是一个很合理的方案。

4.2.4 构造分析类图

通过构造用例实现验证了需求的可实现性：即可以利用识别出的分析类和它们之间的交互来达到用例的目标；这是整个分析过程最核心的工作。然而，对于面向对象的系统来说，所描述的这些交互最终都应在相应的类中来定义并由类的对象来实现。虽然在构造用例实现的过程中已经获得了类的基本定义，但那是在一个个用例实现的基础上完成的，主要关注的是用例事件流的交互过程，而对单个类自身的特征和行为缺少统一的考虑。因此，下一阶段就需要将注意力集中到每一个分析类本身，在关注类自身定义的基础上再重新评估每个用例实现的需要。此外，一个类及其对象常常参与多个用例实现，此时更需要从类整体角度去协调用例的行为。

构造分析类图的最终目标就是从系统的角度明确说明每一个分析类的职责和属性以及类之间的关系；并根据这些视图来描述和理解目标系统，从而为后续的设计提供基本的素材。

图 10 展示了“旅游申请系统”实体类类图（为保持图形的清晰，图中有些类的属性和操作并没有完全显示出来，如参加人类的操作等），该图中涉及到类、类的职责、属性和关系等内容都全部展示出来了。

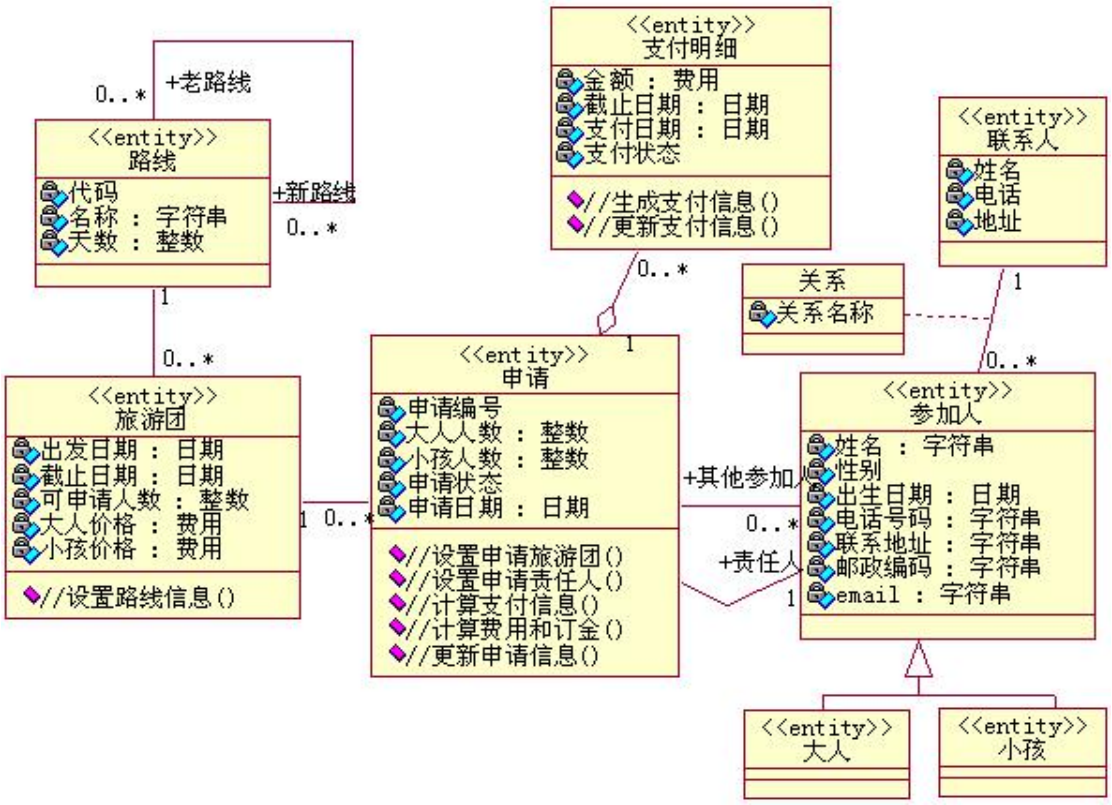

图 10 旅游申请系统实体类类图

需要说明的是，事实上整个分析阶段的重点就在于找出体现系统核心业务所需数据的实体类，而界面和业务逻辑细节分别由边界类和控制类隐藏；因此图 10 所展示的实体类类图就可以很好地反映“旅游申请系统”的分析成果。在有些面向对象分析方法中，分析阶段的工作就是找到这些实体类，由这些实体类即构成了系统概念模型这一最主要的分析成果。

4.3 系统设计

设计是把分析模型转变为设计模型的过程，这个过程可分解为两个相对独立的阶段，即架构设计和构件设计。在架构设计中，架构设计师根据项目的设计目标和相关的设计原则，对系统进行合理的分解，形成不同的系统层次和各类构件，并对其中的核心元素和架构机制进行定义。而在构件设计中，构件设计师利用架构设计提供的设计元素和架构机制，利用特定的实现技术来完成各类构件的详细设计方案，从而为实现提供输入。

4.3.1 架构设计

对于小规模软件系统来说，通过类可以很方便的组织整个应用系统。然而，随着系统规模和复杂度的增加，类的数量会越来越多，仅仅使用类很难有效地组织和规划系统开发活动。因此，需要更大粒度的组织单元对系统进行组织，这就是“包”；而在“包”这一层的设计活动就是架构设计。

架构是一个系统的组织结构，包括系统分解成的各个部分、它们的连接性、交互机制和指导系统设计的相关规则。具有合理架构的系统，将使得对其的理解、测试、维护和扩展都很容易。在当今以构件化、复用技术为主流的系统开发中，架构的作用更加重要。

架构设计的活动在分析阶段就已经开始，然而分析阶段主要关注基础架构的选型和并确定核心的分析机制。而在设计阶段，则需要针对分析阶段的备选架构的各个方面进行详细的定义，以设计出符合特定系统的架构。

对于“旅游申请系统”，首先考虑的是要消除备选架构中的循环依赖。分析产生循环依赖的原因是“导出财务信息控制类”需要访问“财务系统接口类”，为此，简单的解决方案就是将“财务系统接口类”从边界层中拿出来，单独放在另一个包中，命名为外部接口包（External Interfaces）。

其次，考虑边界类中剩余的界面类仍然保留为单独的界面包（User Interfaces）；而有关控制类主要实现各类申请业务，可以将这些控制类与申请业务相关的实体类（如申请类、支付明细类、日志类等）打包为申请业务包（Application Services），负责与前端进行交互，并处理与申请相关的业务。最后，与申请参与者相关业务可以考虑和其它系统的复用（如与客户关系管理系统），与路线管理相关的业务也存在一定的复用性，这些均可放在单独的包中，作为旅行社的基础业务组件单元（Tour Artifacts，在该包的内部可以将这两类业务再单独分为两个子包：客户关系包 Customer Relationships 和旅游资源包 Tour Resources）。而至于数据库访问以及其它与实现相关的内容将在后面的步骤逐步加入。按照这些分包策略，得到系统的初始架构图如 11 所示。

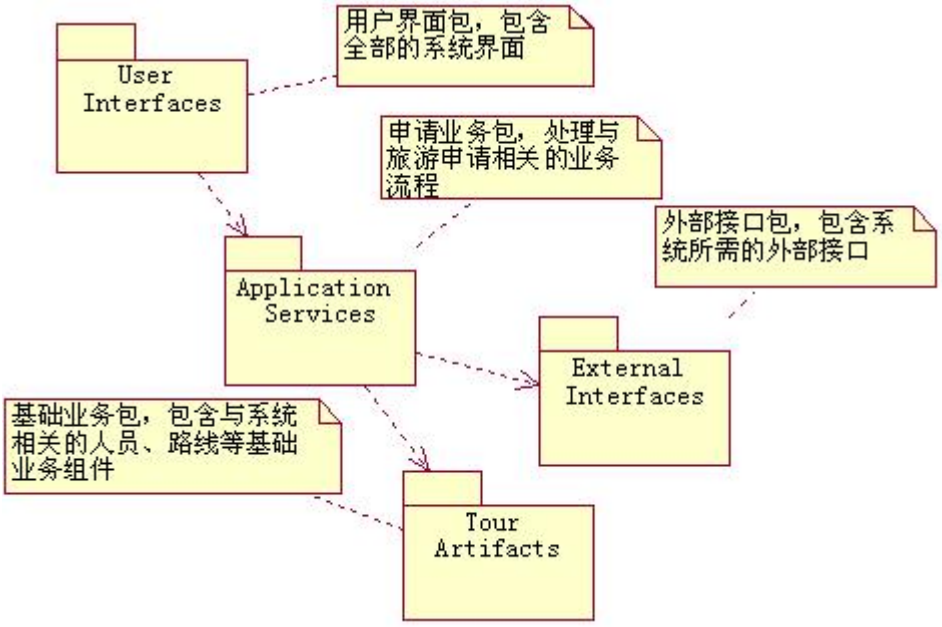

图 11 旅游申请系统初始架构

4.3.2 构件设计

架构建立了系统的核心结构和关键要素，而系统内部各组成成分的实现细节则需进一步细化，包括设计机制的实际应用、设计元素的细化等内容，这个过程就是构件设计。构件设计过程从用例设计开始，根据用例实现流程封装相关的子系统和接口、细化设计类的实现细节，以保证设计类可以到直接实现的程度。

图 12 是旅游业务申请系统中“办理申请手续”用例实现的部分设计场景，描述了前台服务员在录入申请信息后保存申请信息的过程，对应于用例分析中的第二步（参见图 9）。由于是设计模型，所以对应的类都采用英文名字，以便映射到代码实现中。

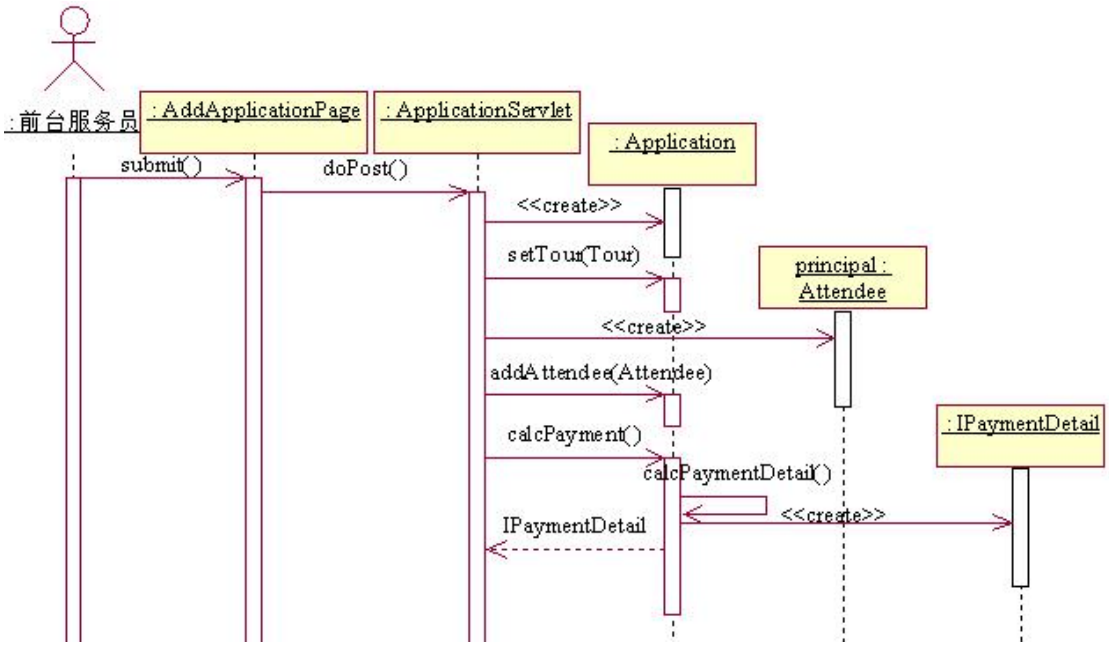

图 12 录入申请信息用例设计

从图中可以看出，前台服务员在添加申请信息的页面（AddApplicationPage）录入信息后提交

（submit），该页面通过 Post 方法向控制类（ApplicationServlet）提交表单，控制类创建相应的实体类存储这些信息。根据该交互图，可以获得申请类（Application）的部分操作。

为了获得 Application 类的操作，需要观察它的生命线，在生命线上所接收到的消息就是该类需要提供的操作。从图中可以看出，它接收到的第一个消息是创建消息（由于 rose 中不直接支持创建消息，因此图中使用带 create 构造型表示），该消息对应 Application 类的构造函数。由于构造函数比较特殊，不同的语言可能有不同的实现，所以需要结合实现语言的语法定义构造函数的签名。而之后的 setTour、addAttendee、calcPayment、calcPaymentDetail 等它接收到的消息均对应该类的操作，根据消息的签名和具体的实现需求，可以对各个操作的签名做进一步的定义。以 setTour 消息为例，该消息的作用是设置某申请对应的旅游团，因此它需要一个 Tour 类型的参数；可以采用 bool 类型的返回值，说明是否设置成功；而操作的可见性应为 public，以便控制对象调用。为此，该消息对应操作的签名可以做如下的定义（采用 Java 语法），图 12 给出了从该顺序图中获得的 Applicant 类的操作签名。

```
public bool setTour( tour: Tour);
```

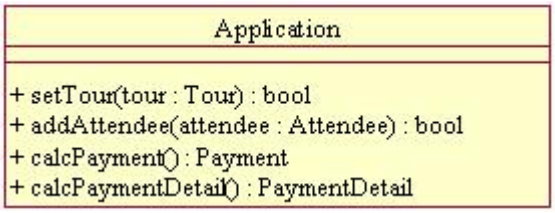

图 12 定义操作

除了通过交互图的消息获得的操作外，可能还需要从类自身的业务或实现需求的角度去完善和补充对应的操作。自身实现方面的操作可能包括构造、析构的操作；类拷贝的需要（如判断类对象是否相等、创建对象副本等）以及其它操作机制的需要（如垃圾收集、测试等方面）。这些操作将随着设计的深入不断地补充和完善，甚至在实现期间还可能根据需要进行适当地调整。此外，类可能还存在一些内部的私有操作，这些操作可以在设计期间定义，也可以在实现期间补充完善。

操作的实现往往会受到对象的状态影响。当对象处在不同的状态时，其方法实现可能会有所不同。如在图书馆管理系统中，当一个图书对象处在被借阅的状态时，就不允许再执行借阅操作。因此，在借阅方法实现时需要判断对象是否处在可借阅的状态。由此可见，针对此类状态受控的对象，为了充分保证其方法实现的正确性，还需要对该对象的状态细节进行建模，以分析对象内部状态变化以及相关事件的发生情况。这就需要用到 UML 状态机图完成状态建模。

图 13 给出了系统中“申请”对象的状态模型。图中 Initial 状态为办理申请手续时初次生成的对象状态；提交新申请（save 事件）后转入 Confirmed 状态，在该状态下可以发生添加（addAttendee）或删除（removeAttendee）参加人等内部转移（图中的内部事件），删除时如果大人参加人数（adult_nums）为 0，则转移到 Cancelled 状态。当用户支付余额（savePayment）后转入 Committed 状态，该状态下可以管理参加人。当旅游团结束（finish）后转入 Finished 状态。在 Initial、Confirmed 和 Committed 状态下都可以直接取消（cancel）从而转入 Cancelled 状态。Cancelled 和 Finished 状态最后都转移到终态，表明该对象处理结束。

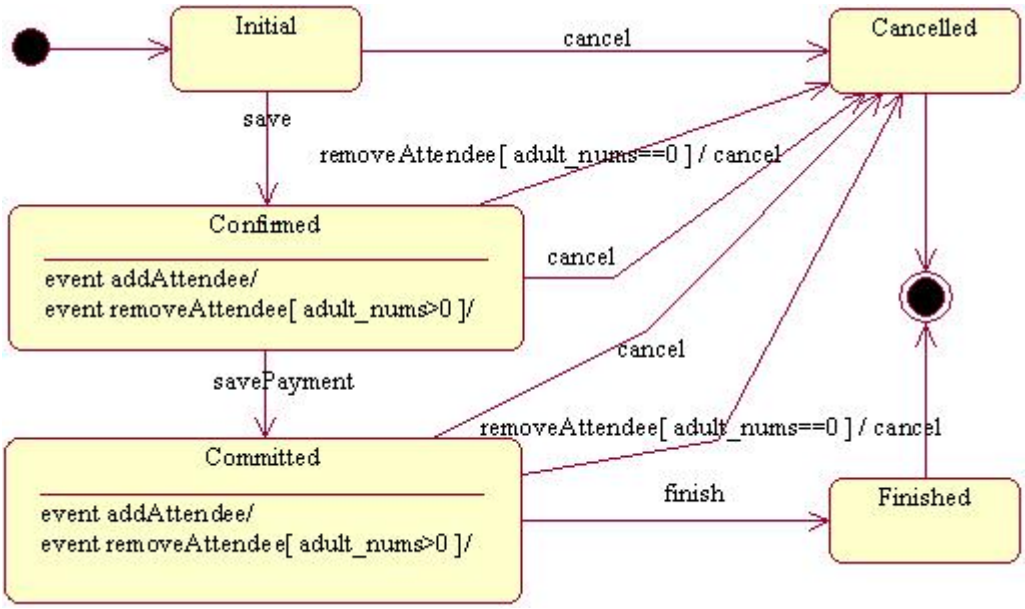

图 13 申请对象状态模型

状态建模完成后，下一步需要将状态模型中的信息映射到其它的模型中，主要包括类、交互等模型。

- 转移事件触发器一般来自对对象操作的调用；
- 守卫条件中的变量一般来自对象属性的引用；
- 对象的状态也可以通过单独的属性记录；
- 交互模型中可以使用状态不变量来约束操作的调用。

图 14 给出了状态建模后申请对象新的结构。与之前相比，当前对象所添加了 removeAttendee、save、cancel、finish 操作均来自状态图中的事件；adult_nums 属性则来自于守卫条件“[adult_nums==0]”；state 属性用于记录当前所处的状态，采用字符串的形式记录。

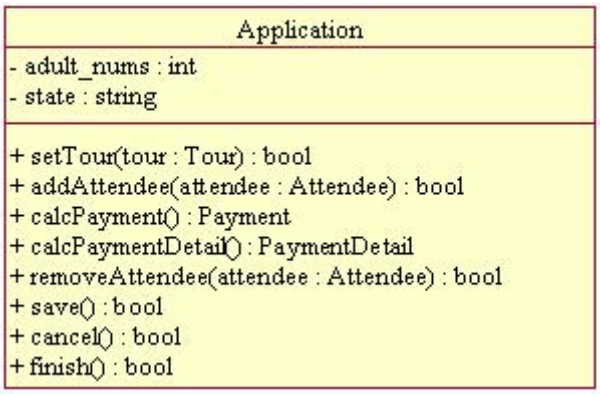

图 14 从状态模型中获得新的操作和属性

5. 案例教学组织方式

作为一个适用于课程教学多个阶段的完整案例，本案例的内容配合课程教学进度逐步开放给学生，基本的教学安排如表 1 所示。

案例主要包括需求建模、系统分析和系统设计三个知识点，分别对应软件开发中的需求、分析和设计三个阶段；考虑设计中架构设计和对象设计的差异性较大，为此可以划分为两个子知识点。同时每个知识点将重点考核不同的 UML 模型的使用，使学生能够在案例实践中理解和掌握各类

UML 模型。各部分建议实践学时可以根据学生的实际情况和课程的总体安排灵活调整。

案例教学方式主要分为两种，一类是小组讨论，安排同学根据课程材料进行头脑风暴，提出问题；另一类是建模实践，经过讨论后，学生可以自己利用 UML 建模工具开展具体的实践，此部分建议在小组讨论的基础上个人独立完成。

表 1. 案例教学组织

案例教学要点	理论知识点	涉及的 UML 图	建议实践课时	教学方式
需求建模	软件需求	用例图；活动图	4 学时	小组讨论 建模实践
系统分析	面向对象分析	包图；顺序图；类图	4 学时	建模实践
系统设计：架构设计	架构设计	包图；构件图；部署图	2 学时	小组讨论 建模实践
系统设计：对象设计	面向对象设计	类图；顺序图；状态图	4 学时	建模实践

6. 案例小结

本案例使用说明书，给出了各个阶段的主要要点，教师可以结合案例描述文档开展案例教学。此外，与之配套的，还提供了这些 UML 案例的原始设计模型。设计模型采用业界流程的 UML 建模工具分别采用 Rational Rose、Enterprise Architect 两种不同的工具绘制，学生也可以在老师的引导下，自己手动完成各类模型的构建工作，真正能够动手实践，从而达到在实践中学习的目标。

面向对象的分析和设计是一门实践性很强的课程，通过 UML 建模语言，可以将与之相关的理论转化为系统建模实践，为此构建一个完整的贯穿整个过程的案例非常重要。旅游业务申请系统案例是在多年的教学和实践经历上总结出来的，案例背景来自于作者在企业培训期间接触到的业务场景，是一个非常典型的软件系统。作者在原始业务场景的基础上，进行了适当的删减和调整，使之尽可能地覆盖课程知识点，并适合于课程案例教学需求。