返回总目录

目　　录

第 11 章　图书馆信息系统 UML 实例

本章将通过一个实例来说明在一个应用中如何使用 UML。通过前面的讨论，首先在分析模型中用用例和域分析来描述应用。然后，将分析模型扩展成设计模型，描述技术上的解决方案；最后，用 Java 语言编程，具体实现可以运行的应用。有一点需要说明的是，本章中讨论的例子，并不包括所有的模型和图。

本章讨论的案例是一个图书馆信息系统，主要处理书和杂志的借阅和保存。虽然它算不上是一个大的应用，但可以对它作许多扩展。本章的案例研究的目的主要有三个：

演示在一个完整的应用中如何使用 UML：从分析到设计模型到真正的代码和可运行的应用。

以 Rational Rose 为例，说明用 UML 建模时使用的工具。

有兴趣的读者，可以根据本章中讨论的方法对模型进行扩展，从而达到提高应用 UML 的水平。

注意，本章给出的仅仅是一个可能的解决方案。可能还有许多其它的方案，不存在一个适用于所有环境的正确的解决方案。如果读者想对初始的模型作些改变，尽管去做。目的只有一个，产生满足需求且工作正常的系统，而不是产生在所有细节上都很完美的图。当然，某些解决方案将被证明比其它的好，但是只有各界经验和努力工作才能获得知识。

11.1　理 解 需 求

下面是一份典型的文本需求说明，它是图书馆应用程序的第一版的需求说明，是为系统的终端用户或客户而写的：

它是图书馆的支持系统。

- 图书馆将书和杂志借给读者，读者和书、杂志一样，必须在系统中注册。
- 图书馆负责购买图书。对于流行的书一般要多买几本。如果旧书或杂志过期了或很破烂则可以从图书馆中删除。
- 图书馆管理员是图书馆的雇员，负责与客户(借书者)打交道，他们的工作要得到系统的支持。
- 借书者可以预订目前借不到的书或杂志，一旦预订的书被返还给图书馆或图书馆新购买书到达，就立即通知预订者。
- 图书馆可以方便地产生、更新和删除系统中与书目、借书者、借书(loan)和预订的有关信息。
- 系统能够在所有流行的技术环境下运行(UNIX, Windows, OS/2 等等)，还应该有一个非常好的图形用户界面(GUI)。
- 系统应该具有很好的可扩展性。

系统的第一个版本不需处理当读者预订的书到达时通知预订者的消息，也不必检查一个书目是否过期。有关更高的版本的其它需求可在本章的练习中了解到。

11.2 分　析

分析就是描述系统的需求，通过定义系统中的关键域类来建立模型。分析的根本目的是在开发者和提出需求的人(用户/客户)之间建立一种理解和勾通的机制。因此，典型情况下，分析是开发人员同用户或客户一起来完成的。

分析不受技术方案或细节的限制。在分析阶段，开发人员不应该考虑代码或程序的问题。它是迈向真正理解需求和所要设计的系统的第一步。

11.2.1　需求分析

分析的第一步是定义用例，即描述图书馆系统的功能：确定系统的功能需求。用例分析主要涉及阅读和分析规格说明，和系统的潜在用户讨论。

图书馆中的角色为图书管理员和借书者。图书管理员是系统的用户，而借书者是客户，虽然偶尔图书馆管理员或另一个图书馆也可能是一个借书者。借书者的目的不是直接同系统交互，借书者的功能由图书管理员来实现。

图书馆信息系统中的用例如下所示：

- 借出书目(Lend Item)
- 返回书目(Return Item)
- 预订(Make Reservation)
- 删除预订(Remove Reservation)
- 增加标题(Add Title)
- 更新或删除标题(Update or Remove Title)
- 增加书目(Add Item)
- 删除书目(Remove Item)
- 增加借书者(Add Borrower)
- 更新或删除借者书(Update or Remove Borrower)

上面所列的用例中没有维护，维护是一个使用其它用例的更一般的用例。同时，还应注意到上述用例中出现的两个概念：“标题(Title)”和“书目(Item)”。因为在一个图书馆中，一个流行的标题可能有好几本，因此系统必须将标题(可能是书名或书的作者)同其它的书目(代表一个指定标题的物理副本)区分开来。从图书馆借的是书目。在图书馆拥有一本书的副本(书目)之前，加一个标题到系统中是可能的，这样做的目的是让借书者可以预订。

图书馆信息系统的分析可以用 UML 的用例图来描述，如图 11-1 所示。每个用例以文本的方式来描述，描述的内容包括用例以及用例与角色交互的更详细的信息。文本的内容是通过与用户/客户讨论后确定的。用例“借出书目”的描述如下：

（1）如果借书者没有预订：

a. 标记标题。

b. 标记可用的该标题下的书目。

c. 标记借书者。

d. 图书馆借出标记的书目。

e. 增加一条新的借书记录。

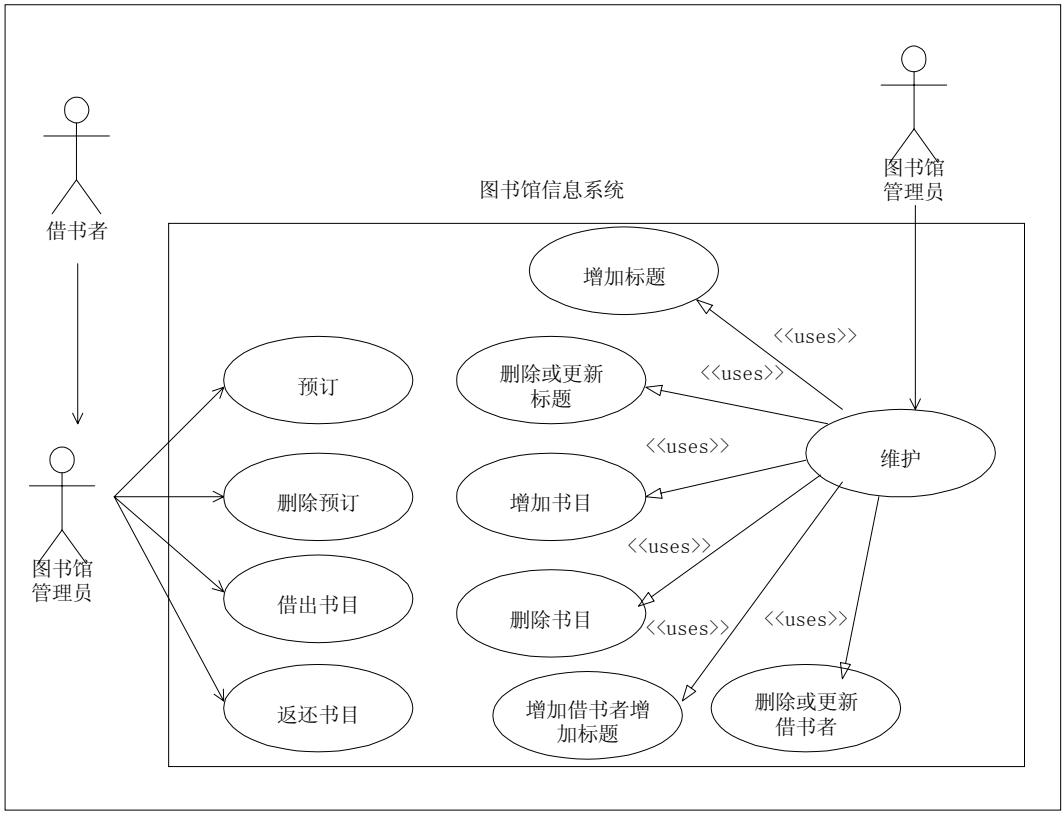

图 11-1 图书馆信息系统的用例图

（2） 如果借书者已经预订：

a. 标记借书者。

b. 标记标题。

c. 标记可用的该标题下的书目。

d. 图书馆借出标记的书目。

e. 增加一条新的借书记录。

f. 删除预订记录。

读者可以照此法描述其它的用例。在整个系统开发过程中，用例描述系统的功能需求。在分析阶段，利用它们来检查某一域类是否已定义。在设计阶段，可以用来证实技术方案是否能够处理要求的功能。可以在序列图中可视化用例。

11.2.2 领域分析

分析也将系统中的领域和关键类条理化。为了进行领域分析，需要阅读规格说明和用例，了解系统要处理的概念。或将用户、领域专家组织在一起开一个讨论会，设法确定所有必须处理的概念以及概念间的关系。

图书馆信息系统中的域类主要有：借书者(在这里将其取名为 BorrowerInformation，以便同用例图中的角色 Borrower 区分开)，标题，书的标题，杂志标题，书目，预订和借

书(loan)。在类图中将这些域类以及它们之间的关系表示出来，如图 11-2 所示。通过版类“Business Object”来定义域类。“Business Object”是一个用户定义的版类，用来表示类的对象，是关键域的一部分，应该永久地保存在系统中。

有一点要强调的是：在本阶段，域类还是处于“草图”状态。定义的操作和属性不是最后的版本，只是在现阶段看来这些操作和属性是比较合适的。一些操作是在序列图的草图中而不是在用例中定义的(有关情况将在本章后面讨论)。

某些类用 UML 状态图来显示类的对象的不同的状态以及改变状态的事件。有状态图的类有：书目和标题。标题类的状态图如图 11-3 所示。

为了描述域类的动态行为，任何动态 UML 图都可以使用：序列图，协作图或活动图。因为 Rational Rose 4.0 不支持活动图，对协作图也只提供有限的支持，因此我们选用序列图。序列图的基础是用例。在序列图中，说明域类如何协作来操作系统中的用例。很自然地，当建立这些序列图时，将会发现新的操作，并将它们加到类中(图 11-2 中的类图显示了建立的序列图模型)。另外，操作仅仅是草案，同样要用说明来详细描述。分析的目的是同用户/客户勾通以对要建立的系统有更好的了解，而不是一个详细的设计方案。

用例“借出书目”(没有预订)的序列图如图 11-4 所示。

当用序列图建模时，很显然需要窗口或对话窗作为到角色的接口。在图 11-4 中，借出书目的窗口是存在的。在分析时，意识到需要窗口来标识基本的接口就可以了。借出、预订和返还书目都需要窗口，维护窗口也是必要的。此时，还没有定义详细的用户接口。另外，关于用户接口中包含些什么仅仅还是个草案。注意，无论怎样，也有一些过程或方法提倡在本阶段设计或原型化一个详细的用户接口。但是，在本例中，因为应用比较简单，所以不采用这些技术。详细的用户接口是设计阶段的一部分。

在分析阶段，为了将域类同窗口类分开，将窗口类组装成一个 GUI 包(称为“GUI 包”)，将域类组装成业务包(Business Package)。此外，也给应用程序取一个名字，称为“通用图书馆应用(Unified Library Application)”。

11.3 设　计

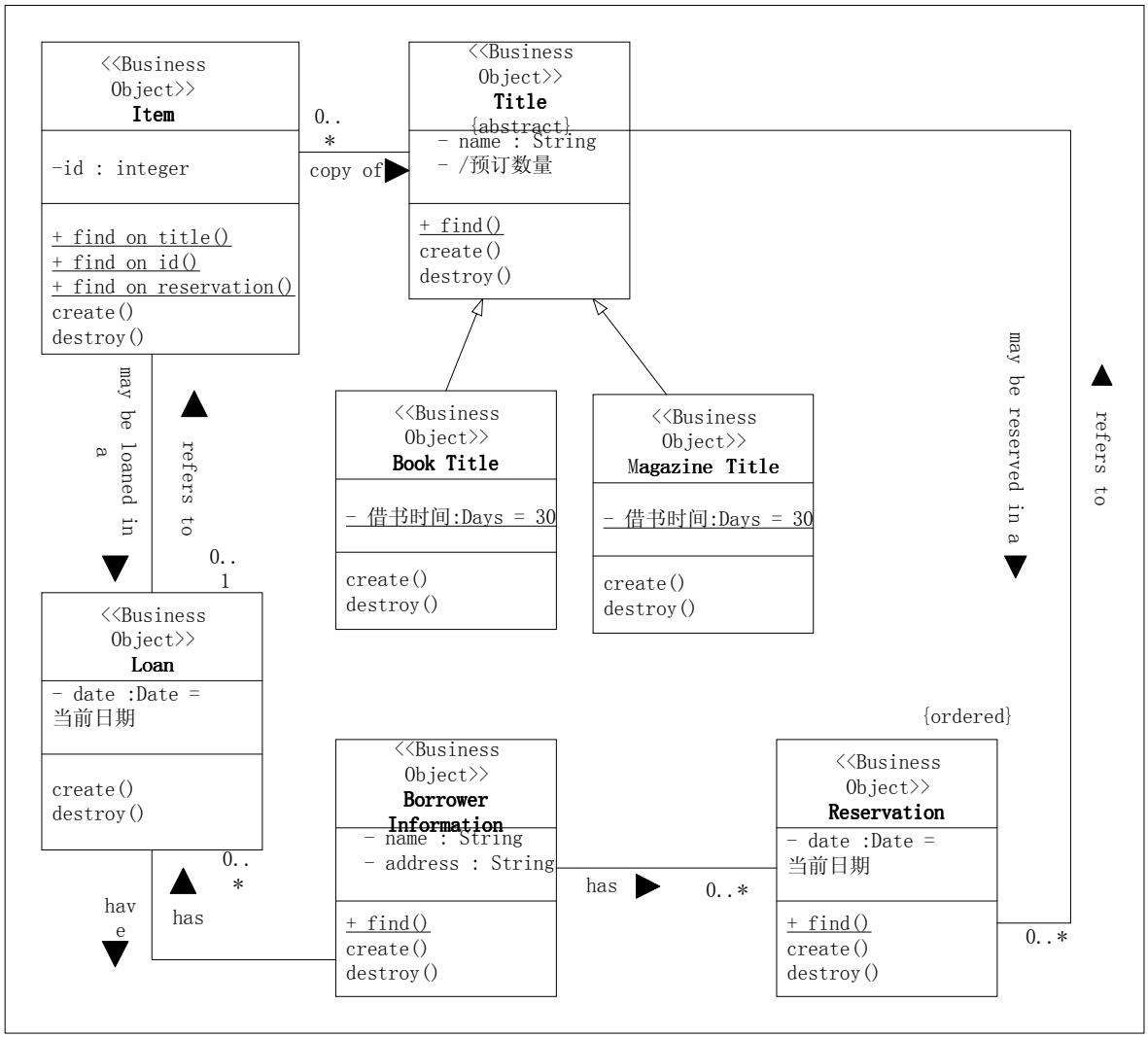

图 11-2　图书馆信息系统的域类结构

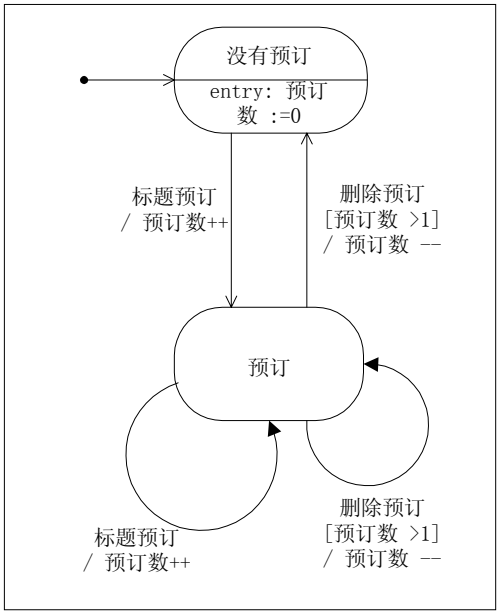

图 11-3 标题类的状态图

设计阶段和最后的 UML 模型是将设计阶段的模型进行扩展和细化，主要考虑所有的技术问题和限制。设计的目的是产生一个可用的解决方案，并且能够比较容易地将方案转换成程序代码。在分析阶段定义的类被进一步细化，定义新的类来处理技术方面的问题，如数据库、用户接口、通信、设备，等等。

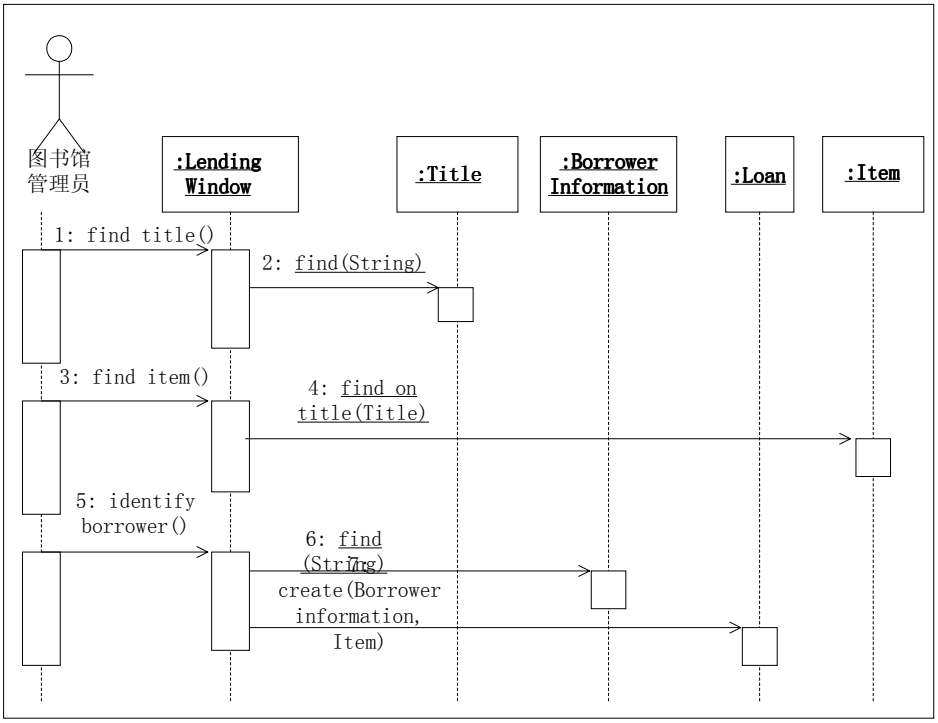

图 11-4 用例“借出书目”的序列图(没有预订的情况)

一般将设计分为两个部分：

- 架构设计：这是高级设计。在架构设计中，需用定义包(子系统)，包间的相关性和基本的通信机制。一个很自然的要求是，得到清晰而简单的架构，即在架构中，相关性要尽可能少，双方相关性要尽可能地避免。
- 详细设计：本部分将包的内容细化，即尽可能详细地描述每一个类，使得编程人员根据它们很容易地编码。UML 中的动态模型被用来显示类的对象在指定的情况下如何动作。

11.3.1 架构设计

一个设计良好的架构是系统可扩展和可改变的基础。包关心的是某一指定功能域或技术域的处理。将应用逻辑(域类)和技术逻辑分开是很关键的，从而使得任何一个改变不至于对其它部分有太多的影响。在定义架构时，需要描述的关键事情是：标识和建立包间相关性规则使得包间不存在双方相关性(避免包紧耦合在一起)；明确必须的标准库和发现要使用的库。今天市场上可买到的库主要涉及某些技术领域，如用户接口、数据库，或通信，但期望有更多的与应用有关的库出现。

本例中的包或子系统或层有如下几个：

- 用户接口包(User Interface Package)：通过用户接口类，用户可以浏览系统中的数据，输入新的数据。这些用户接口类都是基于 Java 的 AWT 包。Java 的 AWT 包是 Java 中用来写用户接口应用的标准库。该包同包含存储数据的类的业务包协作来完成任务。用户接口包调用业务包中的操作来检索和插入数据。
- 业务对象包(Business Object Package)：业务对象包包含分析模型中的域类，如 BorrowerInformation, Title, Item, Loan,等等。这些类的所有细节都已有明确定义，所以类中的操作都已定义好了，并支持加入持续性属性。业务对象包同数据包协作完成任务，因为所有的业务对象类必须从数据包中的持续性类(Persistent class)中继承下来。
- 数据库包(Database Package)：数据库包提供服务给业务对象包中的类，所以可以永久地保存它们。在当前版本中，持续性类将它的子类的对象存放在文件系统中的文件。
- 应用包(Utility Package)：应用包提供服务给系统中其它种类的包。目前，包中只有一个类，ObjId。它被用户接口包、业务对象包数据库包用于在系统范围内引用持续性对象。

这四类包的内部设计如图 11-5 所示。

11.3.2 细节设计

细节设计的目的是描述用户接口和数据库包中的类，扩展和细化业务对象类的描述(在分析阶段已有初步描述)。细节设计的方法通常是产生新的类图、状态图和动态图(如，序列图、协作图和活动图)。这些图与分析阶段中的图是一样的，但是，在此处，这些图的定义更详细，涉及更多的技术细节。分析阶段的用例描述被用来验证用例在设计中的处理；序列图被用来说明技术上如何在系统中实现每一个用例。

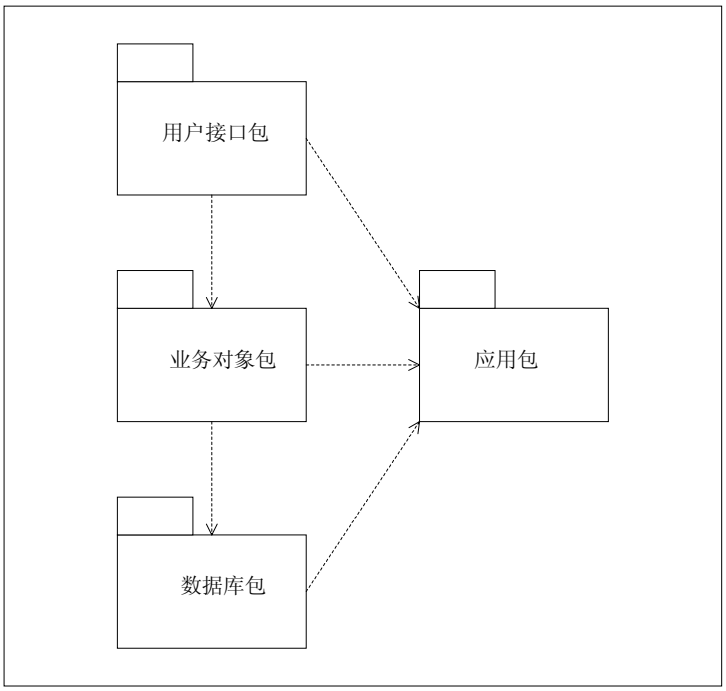

图 11-5 显示应用包及包间相关性的类图

1． 数据库包

应用必须永久保存一些对象，因此必须用数据库层来提供这种服务。对于一个全新的应用而言，最自然的解决方案就是使用商用数据库，要么选用一个真正的面向对象的数据库或选用一个传统的关系数据库，但要增加对象与表之间的转换层。但是，因为我们的例子应用的目的是要可移植的，并不要求某一供货商的许可证，所以我们选择了一个比较简单的解决方案。这种方案就是，将对象简单地存放在磁盘上的文件中。尽管如此，存储的细节对应用而言是透明的，它只需调用对象上的一些公共操作即可，如 store(), update(), delete(), find()等等。与永久存储有关的实现均由类 Persistent 来完成。所有需要永久存储的对象的类均需继承 Persistent 类。Persistent 类是抽象的，要求子类具体实现读写操作：write()和 read()。子类也可以有选择地实现在类范围内查找对象的操作，这样的实现需要调用 Persistent 超类中更多的操作。

持续性处理中的一个很重要的因素是 ObjId 类，它的对象被用来引用系统中的任何持续性对象(不管被引用的对象是在磁盘上还是已经在应用的内存中)。ObjId(Object Identity)，是一种著名的处理应用中对象引用问题的技术。通过使用对象标志符，将对象 ID 传给 Persistent.getObject()操作就可以从持续性存储器中找到该对象并返回。通常情况下，这一工作不是直接完成的，每一个持续性类中的 getObject 操作还要进行类型检测和转换。可以方便地将对象标志符作为参数在操作间传递(例如，查找一个指定对象的搜索窗口可以将它的搜索结果通过一个对象 ID 传给另一个窗口)。

ObjId 是系统中的一个通用类，系统中的所有包(用户接口、业务对象、数据库)都要使用它，所以将 ObjId 放在应用包中(目前情况下，它是应用包中的唯一的应用)。事实上，它与数据库包的关系最密切，但是如果将它放在数据库包中，就意味着用户接口包与

数据库包之间有直接的相关性，而这种相关性应该是尽量避免的(用户接口应该只与业务对象包有直接的相关性)。

还可以对目前的 Persistent 类的实现作些改进。当更新一个对象时，只需将对象的当前的记录标记为删除，并将对象的新的版本加在文件的结尾。不重写老版本的原因是对象可以已经改变了它的长度(如果对象有一个用来存储一对多或多对多关联的不受长度限制的矢量时)。另外，搜索当前的对象是串行进行的，方法是从文件头开始搜索(忽略删除的记录)。同样可以对这种搜索方法作些改进，方法是重新申请文件中被删除的存储空间，以建立一个索引文件来优化对象的搜索。

Persistent 类的接口被定义后，改变持续性存储的实现就比较容易了。可以将对象存放在关系数据库中或在一个面向对象的数据库中，或使用 Java 1.1 中的持续性对象支持的方式来存储。在当前的设计中，没有设置不允许这么做的任何限制，只需对 Persistent 类的内部实现作些改变。

2．业务对象包

设计阶段的业务对象包是基于分析阶段的相对应的包：域类(domain class)。保留域类中的类、类间的关系和行为，只是将类进一步细化，包括如何实现类间的关系和行为。在这些实现说明中，所有业务对象类继承数据库包中的 Persistent 类，实现必须的读写操作。

将分析阶段的操作细化意味着，将一些操作转换成设计模型中的几个操作，将一些操作改名。这样做是很正常的，因为分析阶段得出的是每一个类的能力的草案，而设计是系统的详细描述。因此，设计模型中的所有操作必须有明确的说明和返回值(因为空间的限制，在图 11-6 中没有将它们显示出来)。注意设计与分析之间的下列变化：

- 系统的当前版本并不检查一本书是否已按时返回，也不处理预订的顺序。因此，Loan 和 Reservation 类的日期属性还没有实现。
- 对杂志和书的标题的处理是一样的，除了最大借出日期，但对日期并不处理。因而，可以认为分析模型中的子类 Magazine Title 和 Book Title 是不必要的，仅仅 Title 类中的类型属性可以用来说明标题指的是杂志还是书。

如果认为在此应用程序的将来版本中需要，这些简化可以很容易地删除。

在设计阶段，分析阶段产生的状态图也被细化，即描述如何表示状态，在工作的系统中如何处理状态。Title 类的设计状态图如图 11-6 所示。这些状态用一个称为“预订(reservation)”的矢量来实现，该矢量中包含相关的预订对象的对象标志符(参考类图中的 Title 与 Reservation 之间的关联)。当矢量中没有元素时(也就是说，为空)，Title 对象处于没有预订(Not Reserved)状态。当矢量中有一个或多个元素时，状态为“已预订(Reserved)”。其它对象可以调用 addReservation()和 removeReservation()来改变 Title 对象的状态，如图所示。将分析阶段的状态图 11-3 和设计阶段的状态图 11-7 进行比较，可以看出如何将分析阶段的状态图转换成设计阶段的状态图。

3．用户接口包

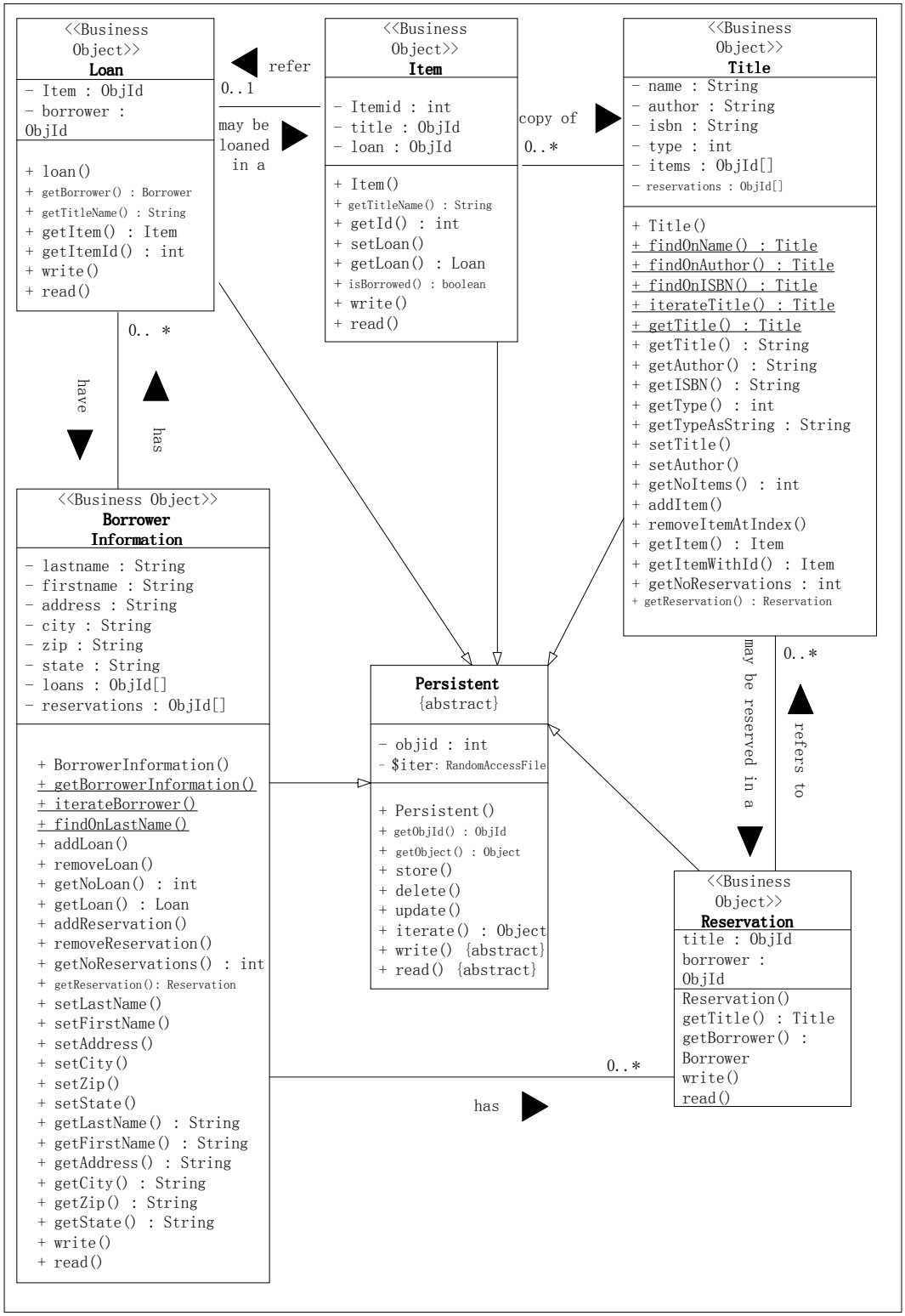

图 11-6　设计模型中的业务对象

用户接口包处理其它包的“顶部”。它将系统中的服务和信息显示给用户。如前所

述，用户接口包是基于标准的 Java AWT(Abstract Window Toolkit)类库，该类库是用来写 Java 应用中的用户接口，且能在所有 Java 平台上运行。在 AWT 库中的所有类没有在设计模型中显示出来，虽然这些类会在它们自己的包中显示(在 Rational Rose 的 Java 版中，读和显示 Java 类结构将其简化了)。在 AWT 类库中，提供不同类型的窗口类(如，框架(frame)、对话框(dialog)等等)和不同类型的接口组件类，如表(table)、按钮(button)、编辑条(edit field)、列表箱(list box)，等等。AWT 类库还负责管理用户产生的事件的处理，如单击鼠标或按键。

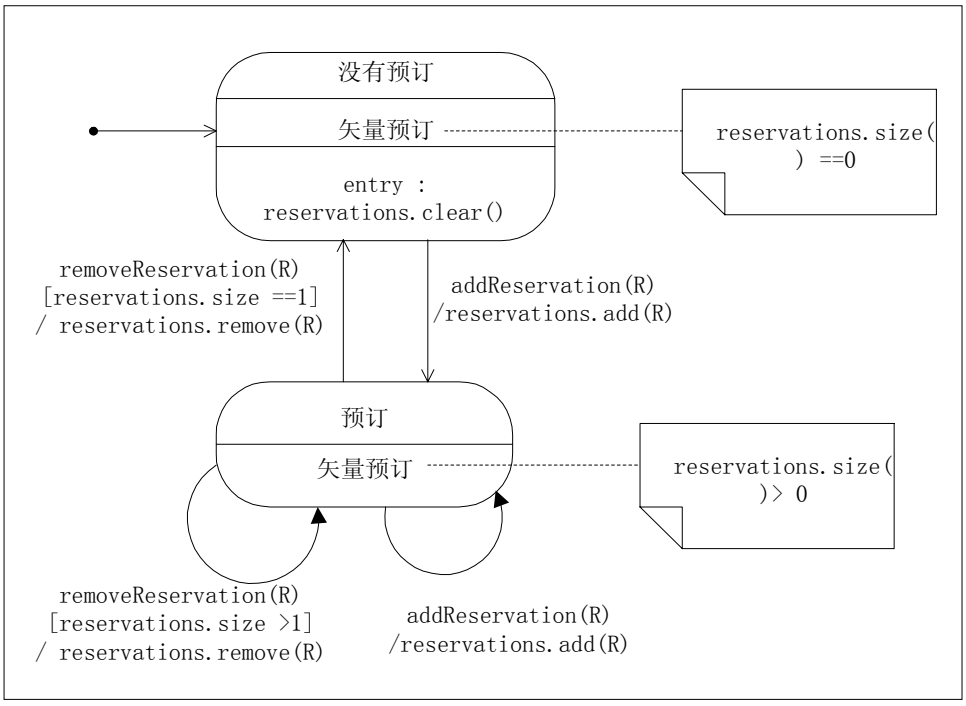

图 11-7　显示 Title 对象实现的设计状态图

设计模型中的动态模型被指派给 GUI 包，因为所有的与用户的交互是通过用户接口来初始化的。再者，序列图已经被用来显示动态模型，虽然序列图中的一些内容已经被转换到协作图中。序列图的基础是用例的分析，除了用例的实现是在设计模型中详细描述外(包括类中的实际的操作，交互分析更多是一种概念)。当序列图出现在此处或模型中时，不需画出来，但是在交互时需要画出来，因而最终的设计是慢慢产生的。而且，序列图中的其它的修改是由实现阶段发现有问题时引起的。设计模型中的序列图是这些交互的结果。图 11-8 显示 Add Title 的设计序列图。操作和说明同代码中的一样。

可以用协作图来代替序列图，如图 11-9 所示。注意，无论怎样，Rational Rose 并不完全支持 UML 有关协作图的全部概念。在其它的方面，消息编号机制则比较简单，给每一条消息一个序列号：1, 2, 3, 等等。因此，Rational Rose 设计模型中的协作图是序列图的简化的版本，它并没有显示 UML 协作图的所有特点。很难将一个复杂的序列图转换成 Rational Rose 4.0 中的协作图，而一个 UML 协作图覆盖同样的序列图是没有任何问题的。因此，还没有出现过将设计模型中的更复杂的序列图转换成 Rational Rose 模型中的协作图。

11.3.3 用户接口设计

在设计阶段进行的一项特殊活动是产生用户接口，定义用户接口的“外观和感觉”。这项活动是在分析阶段初始化，且与其它活动分开来做，但同其它的工作同步进行(如何设计成功的用户接口超出了本书的范围，可参考其它的有关文献)。

基于用例的图书馆应用中的用户接口被分成四部分，每一部分在主窗口菜单中有一个独立的菜单包，如下所示：

- 功能(Functions)：系统中的基本功能窗口，也就是说，借书、还书和预订。
- 信息(Information)：浏览系统中的信息窗口，有关标题和借阅者的信息。
- 维护(Maintenance)：维护系统的窗口，也就是说，增加、更新、删除标题，借阅者和书目。

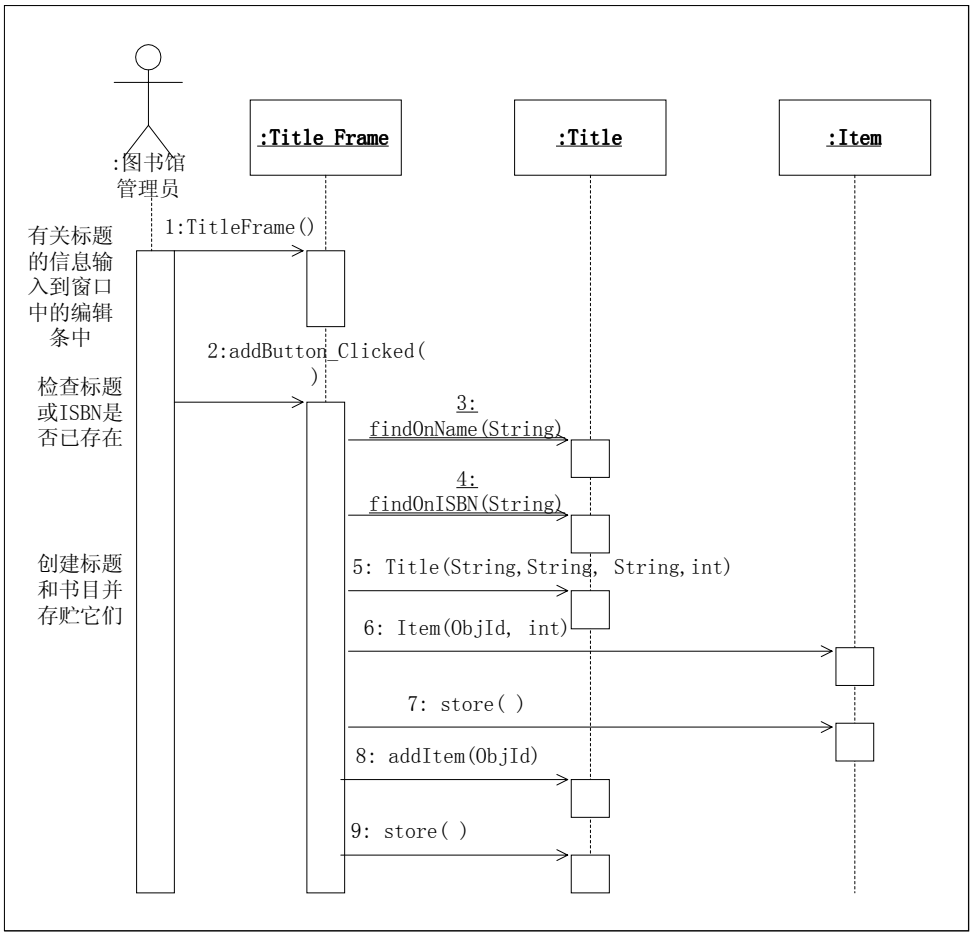

图 11-8　增加标题(Add Title) 用例的序列图

上述所有窗口均在 Symantec Visual Cafe(SVC)中画出。在 SVC 中，可视化组件，如按钮、编辑条，可以很方便地放入窗口中。然后，SVC 自动产生创建窗口所需的代码，包括添加放入窗口中的组件的属性，如，给 AWT Button 类添加一个属性 ok-Button，指示该按钮为 OK 按钮。属性是自动是产生的，所以它们的缺省名字类似于：label1, label2, textField1, 等等。

在 SVC 中，为用户产生的事件，如单击鼠标、选择菜单等，指定事件处理器是很方便的。具体做法是，选择一个指定组件，如 okButton，然后选择组件上需要处理的事件，如 Clicked 事件。随后，SVC 就自动产生一个方法 okButton_Clicked，当 OK 按钮被单击时，调用该方法。当研究用户接口包中的类时，有一条原则非常重要，如，当用户选择窗口中的 findTitleButton 时，方法 findTitleButton_Clicked 被调用，当列表项 itemList 中的某一项被选中时，方法 itemList_Selected 被调用,当菜单项 Lending Item 被选中时，方法 LendingItem_Action 被调用。图 11-10 显示了用户接口包中的类图中的一个例子，在图中可以发现这类事件处理器。按钮、标签、编辑条的属性在图中没有显示出来。

最终的用户接口由一个带菜单包和一个图形画面的主窗口组成，从主窗口可以访问到其它所有窗口。一般来说，每一个其它的窗口代表系统的某一服务，被映射到对应的初始用例(虽然并不是所有的用户接口都必须从用例中映射而来)。

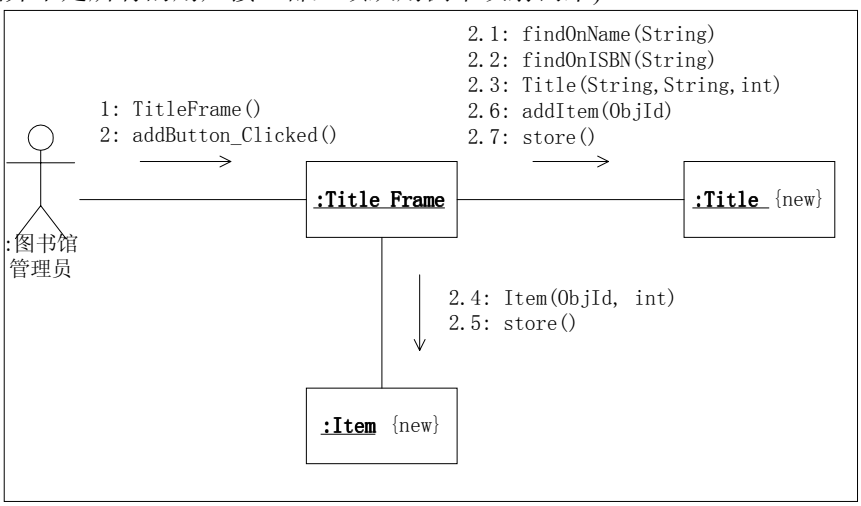

图 11-9 用正确的 UML 语法表示的 Add Title 用例的协作图

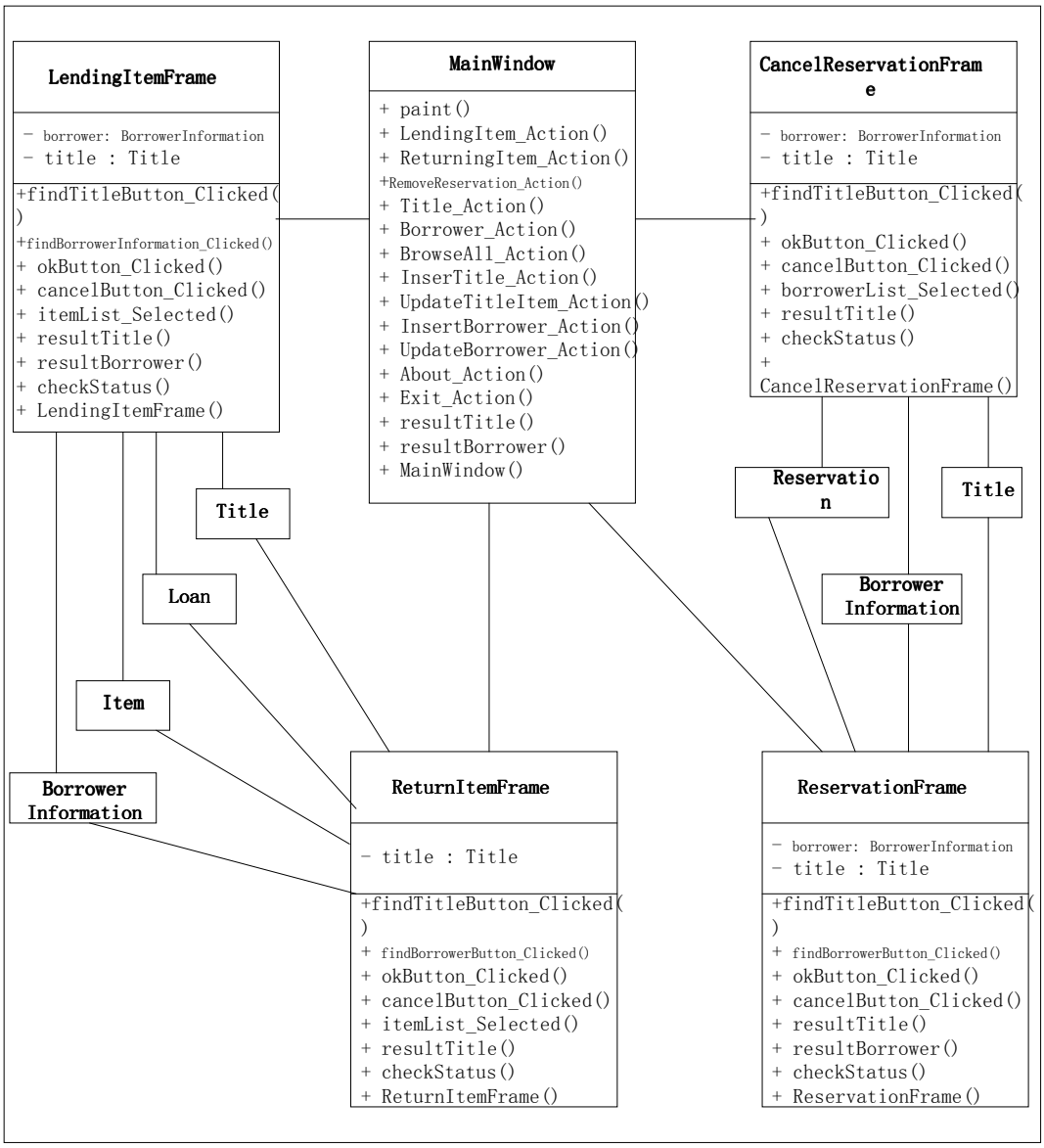

图 11-10 Function 菜单中的用户接口类(在模型中的 Function 类图中显示)。典型地，所有关联是一对一，且所有关联都表明在某一点的关联窗口类被创建或在某一点关联的业务对象类被访问。下面不将关联进一步细化

11.4 实　现

构造或实现阶段是指编程实现类。在系统需求中，要求系统可以运行在许多不同的处理器和操作系统上，所以选择 Java 来实现系统。但是，有一点需要说明，在这个应用中，Java 是作为一般的编程语言来使用的，没有使用 Java 的 Applet，因为 Java Applet 不能访问客户端的文件系统。Java 很容易将逻辑类映射到代码组件，因为这种映射是类和 Java 代码文件之间的一对一的映射(和一对一地映射到一个可执行的 .class 文件)。Java 也

要求文件名应该同它包含的类的类名一样。

图 11-11 说明设计模型中的组件图包含(在本例中)一个从逻辑的类到组件的简单映射。逻辑包也被映射到对应的组件包，所以组件包中的 UI 包包含逻辑包中的 UI 包。每个组件包含一条到逻辑类的描述的链接，使得可以方便地在逻辑视图与组件视图间切换(即使在本例中，使用的仅仅是文件名)。组件间相关性不在组件图中表示(除了业务对象包)，因为相关性可以从逻辑类图中得到。

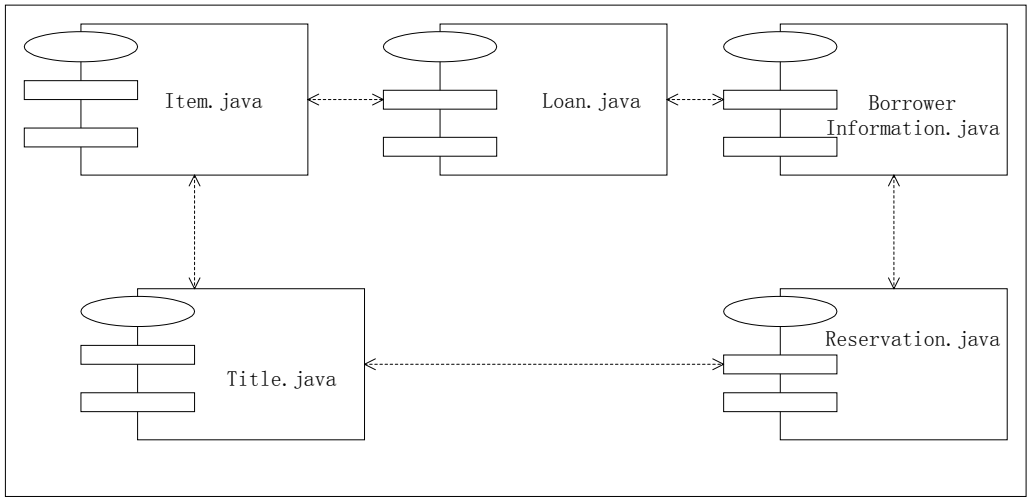

图 11-11 标志实现域类的组件的组件图。类间的关联是双向的

对于编码，从下列设计模型中的图获得规格说明：

类说明(Class Specifications)：每一个类的规格说明，详细显示必须有的属性和操作。

类图(Class Diagrams)：类图显示类的静态结构和类间的关系。

状态图：类的状态图，显示类的所有可能到达的状态，以及需要处理的状态转移(以及触发状态转移的操作)。

动态图(序列图、协作图、活动图)：显示类中方法的实现，以及其它对象如何使用类的对象。

用例图和说明：当开发人员需要了解更多有关如何使用系统的信息时(当开发人员感到他正从细节中迷失时)，可以通过该图来了解使用系统的结果。

很自然地，在编码阶段，设计模型中的缺陷可能会体现出来。因此，可能增加新的操作或修改已有的操作，也就是说，开发人员不得不改变设计模型。所有的工程都可能会碰到这种情况。重要的是使设计模型与编码同步起来，使得模型可以作为系统的最终文档。

这里给出的 Java 代码例子是有关 Loan 类和 TitleFrame 类。当研究这些代码时，脑海中想着 UML 模型，并试图了解如何将 UML 结构转移成代码。考虑下面的内容：

Java 包说明就是指明类属于哪一个包，是组件包还是逻辑视图。

私有属性对应于模型中的指定的属性，同样，Java 方法对应于模型中的操作。

ObjId 类(对象标志符)被用来实现关联，意思是通常情况下，关联是同类保存在一起的(因为 ObjId 类具有持续性)。

第一个代码例子是有关 Loan 类，它是一个业务对象类，用来存储有关借书的信息。实现是比较直接的，代码也比较简单，因为该类主要是用来存储信息。大部分功能是从数

据库包中的 Persistent 类中继承来的。类中的唯一属性是其关联类 Item 和 BorrowerInformation 类的对象标志符。这些关联属性同时也保存在 write()和 read()操作中。

```
   //
   // Loan.java; represents a loan. The loan refers to one title and one
borrower.

   package bo;
   import util.ObjId;
   import db.*;
   import java.io.*;
   import java.util.*;

   public class loan extends Persistent
   {
       private ObjId item;
       private ObjId borrower;
       public Loan()
       {
        }
       public Loan(ObjId it, ObjId b)
       {
           item  =  it;
           borrower = b;
       }
       public BorrowerInformation getBorrower()
       {
          BorrowerInformation       ret       =       (BorrowerInformation)
Persistent.getObject(borrower);
            return ret;
       }
       public String getTitleName()
       {
           Item it= (Item) Persistent.getObject(item);
           return it.getTitleName();
       }
       public Item getItem()
       {
           Item it =  (Item) Persistent.getObject(item);
           return it;
       }
       public int getItemId()
       {
```

```
            Item it = (Item) Persistent.getObj(item);
            return it.getId();
        }
        public void write(RandomAccessFile out)
        throws IOException
           {
              Item.write(out);
              borrower.write(out);
           }
        public void read(RandomAccessFile  in)
        throws IOException
           {
              item = new ObjId();
              item.read(in);
              borrower = new ObjId();
              borrower.read(in);
           }
    }
```

TitleFrame 是实现窗口框架的用户接口类，可以通过窗口框架将新的标题加到系统中。该类是以 Java AWT 库为基础，也包含 SVC 中自动产生的有关窗口布局和外观的 Java 代码。正如前面提到的，SVC 允许用户添加和定位窗口中的按钮或编辑条，同时它还自动产生运行时创建窗口的 Java 代码。自动产生的代码出现在类的尾部，包括控件的声明：如按钮、标签、编辑条等；一些标准的操作，如 show()和 handleEvent()；在构造类时控件的创建和初始化。有一点一定要记住，不能手工修改自动生成的代码，它不是类的设计模型中的一部分。仅仅“直正的”操作和属性才被模型化，它们出现在类的开始。

读者可以检查前面讨论的 Add Title 序列图中的 addButton_Clicked()操作的实现代码，它也是实现序列图的代码。一边读代码，一边看序列图，看看它是不是图中描述的同一协作的另一种更详细的描述。

```
//
// TitleFrame.java
//
package ui;
import bo.*;
import util.*;
import java.awt.*;

public class TitleFrame extends Frame {
      private Title current;

void addButton_Clicked(Event event)
{
        if (Title.findOnName(titleField.getText()) != null)
```

```
        {
          new MessageBox(this, "A Title with that name already exists!");
            return;
         }
         if (Title.findOnISBN(isbnFiled.getText()) != null)
        {
            new MessageBox(this, "A title with the same isbn/nr field
already exists!");
            return ;
         }
         int type = 0;
         if (bookButton.getState() = =true) type = Title.TYPE_BOOK;
            else if (magazineButton.getState() == true) type =
Title.TYPE_MAGAZINE;
                   else {
                           new MessageBox(this, "Please specify type of
title!");
                           return;
                         }
         current = new Title(titleField.getText(), authorField.getText(),
isbnField.getText(), type);
         int itemno;
         if (itemsField.getText().equals("")) itemno = 0;
             else itemno = Integer.valueOf(itemsField.getText()).intValue();
         if (itemno > 25)
            {
               new MessageBox(this, "Maximum number of items is 25!");
               return;
            }
         for (int i =0; i < itemno; i++)
         {
               Item  it = new Item(current.getObjId(), i +1)
               it.store();
               current.addItem(it.getObjId());
         }
         current.store();
         titleField.setText("");
         authorField.setText("");
         isbnField.setText("");
         itemsField.setText("");
         bookButton.SetState(false);
         magazineButton.setState(false);
      }
```

```
    void cancelButton_Clicked(Event event)
    {
      dispose();
    }

  public TitleFrame()
  {
     //{{INIT_CONTROLS
         setLayout(null);
         addNotify();
         resize(insets().left + insets().right + 430, insets().top +
insets().bottom +229);
         titleLabel = new java.awt.Label("Title Name");
         titleLabel.reshape(insets().left +12, insets().top +24, 84, 24);
         add(titleLabel);
         titleField = new java.awt.TextField();
         titleField.reshape(insets().left + 132, insets().top +24, 183,
24);
         add(titleField);
         authorField = new java.awt.TextField();
         authorField.reshape(insets().left + 132, insets().top + 60, 183,
24);
         add(authorField);
         isbnField = new java.awt.TextField();
         isbnField.reshape(insets().left + 132, insets().top + 96, 183,
24);
         add(isbnField);
         label1 = new java.awt.Label("Author");
         label1.reshape(insets().left + 12, insets.top + 96, 84, 24);
         add(label1);
         label2 = new java.awt.Label("Insert");
         label2.reshape(insets().left + 12, insets.top + 60, 84, 24);
         add(label2);
         addButton = new java.awt.Button("Insert");
         addButton.reshape(insets().left + 348, insets.top + 24, 60, 24);
         add(addButton);
         cancelButton = new java.awt.Button("Close");
         ancelButton.reshape(insets().left + 348, insets.top + 192, 60,
24);
         add(cancelButton);
         label3 = new java.awt.Label("Items available");
         label3.reshape(insets().left + 12, insets.top + 192, 108, 24);
```

```
          add(label3);
          itemsField = new java.awt.TextField();
          itemsField.reshape(insets().left + 132, insets.top + 192, 36,
23);
          add(itemsField);
          Group1 = new CheckboxGroup();
          bookButton = new java.awt.Checkbox("Book", Group1, false);
          bookButton.reshape(insets().left +132, insets().top +132, 108,
24);
          add(bookButton);
          magazineButton = new java.awt.Checkbox("Magazine", Group1,
false);
          magazineButton.reshape(insets().left +132, insets().top + 156,
108, 24);
          add(magazineButton);
          label4 = nw java.awt.Label("Type");
          label4.reshape(insets().left + 12, insets().top + 132, 108, 24);
          add(label4);
          setTitle("Insert Title Window");
      //}
          bookButton.setState(true);
          titleField.requestFocus();

         //{{INIT_MENUS
         //}}
   }

   public TitleFrame(String title)
   {
        this();
          setTitle(title);
   }

   public synchronized void show()
   {
          move(50, 50);
          super.show();
   }

   public boolean handleEvent(Event event)
   {
        if (event.id = = Event.WINDOW_DESTROY)
   {
```

```
            dispose();
            return true;
          }
         if (event.target = = addButton && event.id = = Event.ACTION_EVENT)
    {
            addButton_Clicked(event);
            return true;
          }

        if   (event.target   =   =   cancelButton   &&   event.id   =   =
Event.ACTION_EVENT)
    {
            cancelButton_Clicked(event);
            return true;
          }
          return super.handleEvent(event);
        }

            //{{DECLARE_CONTROLS
          java.awt.Label titleLabel;
          java.awt.TextField titleField;
        java.awt.TextField authorField;
          java.awt.TextField isbnField;
          java.awt.Label label1;
          java.awt.Label label2;
          java.awt.Button addButton;
          java.awt.Button cancelButton;
          java.awt.Label label3;
          java.awt.TextField itemsField;
          java.awt.Checkbox bookButton;
          CheckboxGroup Group1;
          java.awt.Checkbox magazineButton;
          java.awt.Label label4;
     //}}

          //{{DECLARE_MENUS
          //}}
     }
```

11.5 测试和配置

当然，必须对应用进行测试。对这个例子而言，首先用最初的用例来测试，检查应用是否支持这些用例，以及应用是否按照用例中所描述的那样运行。还需将应用交给用户进

行测试。一个大规模的应用则要求更形式化的描述和错误报告。

系统配置是指将系统提交给用户，包括所有的文档。在一个实际的工程中，用户手册和市场描述一般也是文档的一部分。还应该提交一个系统的物理配置图，如图 11-12 所示。本例中的系统可以安装在任何支持 Java 的计算机上，注意，根据需要，可能还需要一台打印机。

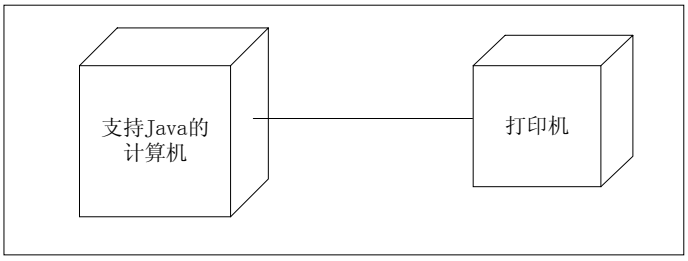

图 11-12 图书馆信息系统的配置图。所有组件都安装在计算机上，因为在本例中它们都在同一台计算机上执行，所以没有将它们在图中详细表示出来

11.6 小　结

本章通过一个具体的例子来说明分析模型的产生，并将分析模型扩展和细化成设计模型，最终用 Java 来实现系统。例子的不同部分由一组人员来共同实现，如同一个真正的工程一样。虽然不同的阶段和活动看起来好像是分离的以及严格地按一定顺序进行的，但是这项工作在实践中却是可重复的。根据设计中的经验和教训来更改分析模型，根据实现中发现的问题来指导对设计模型的修改。这就是最常见的面向对象系统的开发方式。