需要建模物理容积的情况下，比较适合使用合成。这意味着在进行合成的各个部分之间，整体与部分间的耦合更为稳固，部分不能独立于整体而存在。也就是说，部分参与了整体的生命周期，它随着整体的出现而出现，随着整体的消亡而消亡。

在使用实现语言如 C++时，与聚合和合成所对应的代码也不尽相同。例如，聚合中没有引用，而合成中不考虑值。但是，这种区别并不适用于 Java。因此，即使用不同的方法模仿它们，以沟通设计意图和单独实现的重要元素，聚合和合成所对应的代码还是相同的。

除了在菱形已被填充的情况下以外，合成与聚合的用法都是一样的。

7. 自反关系

类可以有一个自身的关联。例如，一个人雇用了另一个人，Person 类就会有一个与自身有关的关联，该关联中带有雇主和雇员的角色名称。这样的关系就称为自反关系。

这种标准化的标记法可被看成建模的一种简写形式。无须用两个类图标，只要用一个就可以表示出这种关系。如图 8-29 所示，用它完全可以表示带有这种关系的两个相互独立的 Persons 类图标。不过，这样做会占用图中的一些空间。

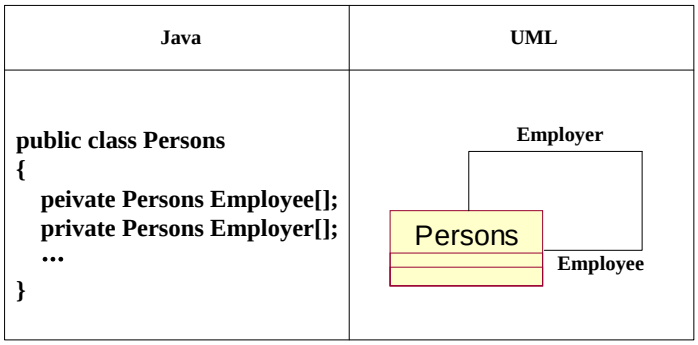

图 8-29 自反关系的对应关系

以上主要介绍了与类图相关的主要概念，这些概念如下：

- 类、属性和操作，以及在 Java 中它们之间的关系。
- 用作分组工具的包，以及其与 Java 的关系
- 类间各种不同的关系，以及在什么情况下使用下面这些关系：
  关联
  聚合
  合成
- 在 UML 中表示继承。
- UML 中角色的实现，以及它是如何与 Java 实现语言中的 extends 相联系的。

建模并不是件容易的工作。与任何其他基于技术的工作相比，需要花费更多的精力才能掌握 UML 和建模技术。在后面几章中，将探讨一下在 J2EE 开发环境中包含了这些概念的应用程序。

8.4 统一建模语言的综合应用

本节通过自动柜员机(Automatic Teller Machine，ATM)系统软件的设计过程来讨论 UML 的综合应用技术，共分为静态设计、永久对象设计、动态模型设计、通用模块接口设计、系统体系结构等几个方面。首先对系统做一概括性描述，再进行面向对象的分析和设计。在设计中，将从一个核心设计模式出发，逐步形成整体的设计。

8.4.1 项目概述

本项目的目的是建立一个自动柜员机系统软件(ATM System Software)。整个ATM系统包括自动柜员机(ATM)、中央服务器(Central Server)、自动柜员机系统软件及有关的界面软件。以下是ATM系统软件的系统概述。

ATM系统软件的工作，是处理系统中自动柜员机和银行计算机之间的事务(Transaction)，如查账、存款、提款等。与系统合作的银行有多家，它们各有自己的计算机，处理自己的账户和事务。所有自动柜员机都通过中央服务器与每个银行的计算机通信。

每家银行计算机处理的事务及其中运行的软件，乃至该银行的内部事情与本开发项目无关。但是，处于银行计算机和ATM系统软件之间的界面软件则属于本项目。

银行的顾客持有提款卡(Cash Card)，可在任何一个自动柜员机查账、提款、取收据等。提款卡插入自动柜员机后，卡上资料被读出，并提示顾客输入其个人密码。之后，ATM系统检验密码的正确性，若一切没问题，便把顾客要做的事务传送到中央服务器进一步处理。

对ATM系统软件的要求有多项，现排列如下：

(1)对所有使用ATM系统的事务，都要留有记录，且能每天作报告。

(2)系统要有适当的安全措施。

(3)一个账户必须能够同时从多处接入作查询或其他账务。

(4)必须能兼顾未来的扩展，如增加出纳点、因特网客户机等。

在设计系统时，必须把以下限制考虑在内：

(1)ATM系统由银行协会拥有。

(2)ATM系统软件在中央服务器里运行。

(3)各银行自己发行提款卡，并保管有关资料。一家银行发行的卡，前6位数字相同。

(4)所有自动柜员机有同样的程序接口。

(5)每一家银行的计算机有其程序界面，不一定与其他银行的相同。

8.4.2 静态分析和设计

模型对象的分析和设计有两方面:静态和动态。静态分析包括认出对象类及其静态关系。动态分析则集中于对象间的相互作用和对外接口。以上列出的四条要求中，(1)和(4)直接影响着系统的静态设计。(2)和(3)则和动态设计相关。这里要注意，通常涉及系统安全的，都和静态设计有关。但现在因为用户密码存放在银行计算机，ATM系统软件只需作中继站，把有关数据传到银行计算机，故(2)对静态设计影响不大。

作静态分析和设计时，可以参考本书第6章中提供的方法。在此，先找出一个核心设计样式，然后再增加其他对象。从以上项目概述得出一些候选对象，包括事务、提款卡、银行、自动柜员机等。为了进一步找出合适的对象，我们来看系统的要求。

从对数据存储要求，可以找到有关的候选对象。例如，要求(1)记录所有使用ATM系统的事务是最重要的。这些记录或永久性数据，常常引出系统中的核心对象。为简化起见，先把在一次里使用柜员机的所有事务，归纳为单一的事务。

什么是事务呢？事务其实就是持提款卡的顾客，在柜员机所做事情的记录。所以，它是提款卡与柜员机之间的一个连接。也就是说，可以把事务看成二元关联类，并作为设计的核心(见图8-30)。

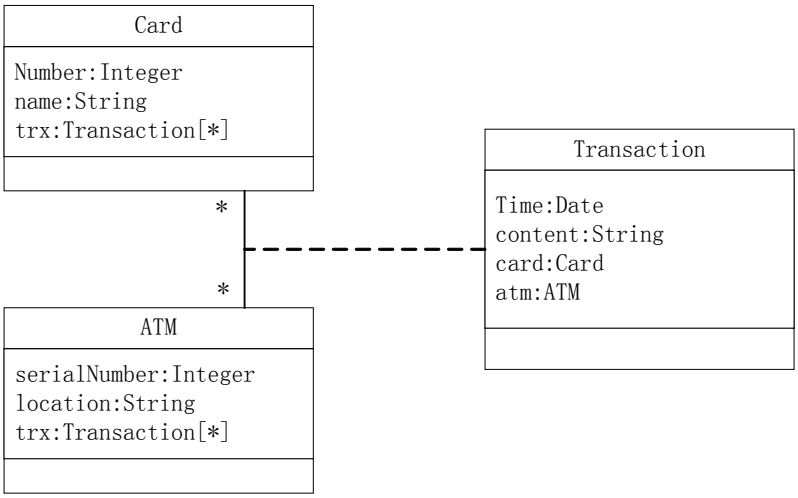

图 8-30 事务(transaction)是 ATM 系统软件的核心设计

图 8-30 的事务是一个二元关联类，把提款卡和自动柜员机联系起来。其中的属性有 number(号码)、name(名字)、trx(代表一系列事务)、serial Number(序号)、location(地点)、time(时间)、content(内容)等。日期(Date)类型包括时分秒的时间。

接下来，由于 ATM 系统软件需要把提款卡和事务资料送到银行计算机，系统软件必须知道银行的存在。关于银行的资料，可存放在银行资料类(BankInfo)内。而且，每家银行发行自己的提款卡,所以可用银行资料类来管理自己银行的卡。类似地,另加一个 ATM 管理器(ATM Manager)管理所有银行资料和事务。这就得到图 8-31 所示的局部设计。

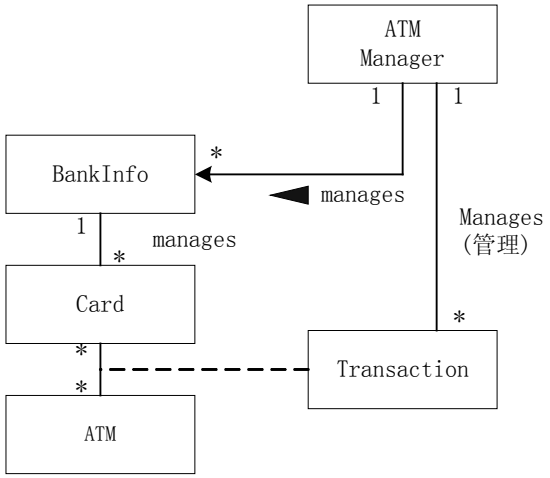

图 8-31 ATM 系统软件的局部设计图

图 8-31 是图 8-30ATM 系统软件核心设计的扩充。其中 ATM 管理器(ATM Manager)管理所有银行资料(BankInfo)和事务(Transaction)，银行资料则管理自己银行的提款卡(Card)。图中用到关联名“管理”(manages)。

最后，尝试改良设计，以照顾未来的需要，如增加出纳点和因特网客户机等。这样做起来比较容易，把 ATM 类换成抽象类“代理”(Agent)，并由后者衍生 3 个子类，得到如图 8-32 所示的 ATM 系统软件设计总图。注意，图中代理类也由 ATM 管理器管理。

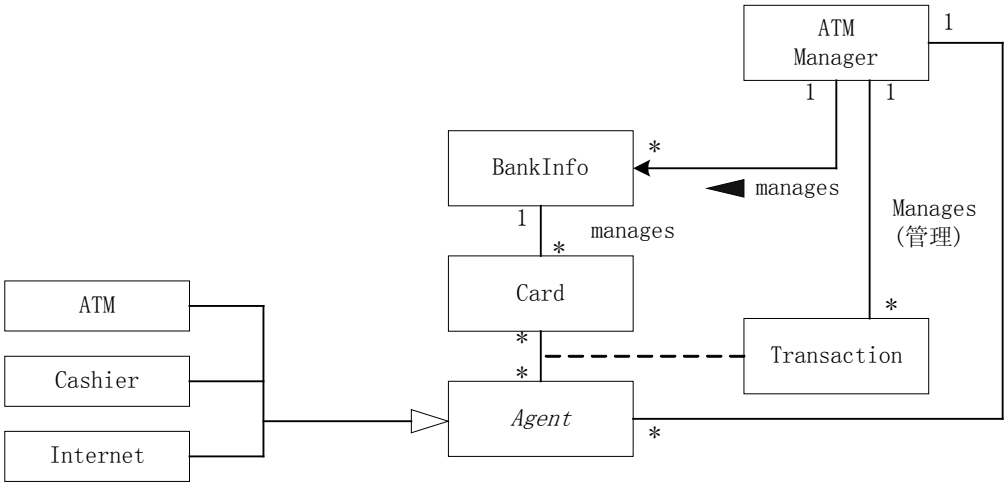

图 8-32 ATM 系统软件设计总图

图 8-32 是图 8-31 的进一步扩充，并以关联类和集合管理器为核心设计样式。它允许将来有不同的“代理”(Agent)，包括出纳点(Cashier)和因特网(Internet)客户机等。注意，用斜体标识的“代理”(Agent)是一个抽象类。

至此，完成了 ATM 系统软件的静态模型，其中以关联类和集合管理器为核心设计样式。

8.4.3 持久对象设计

在这一节，应把重点放在持久化上。从上面讨论的系统要求可知，图 8-32 中所有的类都要持久化。由于 ATM 管理器(ATM Manager)处于层次关系的顶端，就把它作为数据库的根(database root)。由这个根可以达到其他对象集合，包括银行资料(BankInfo)、事务(Transaction)和代理(Agent)。持久化后的设计在图 8-33 中显示。

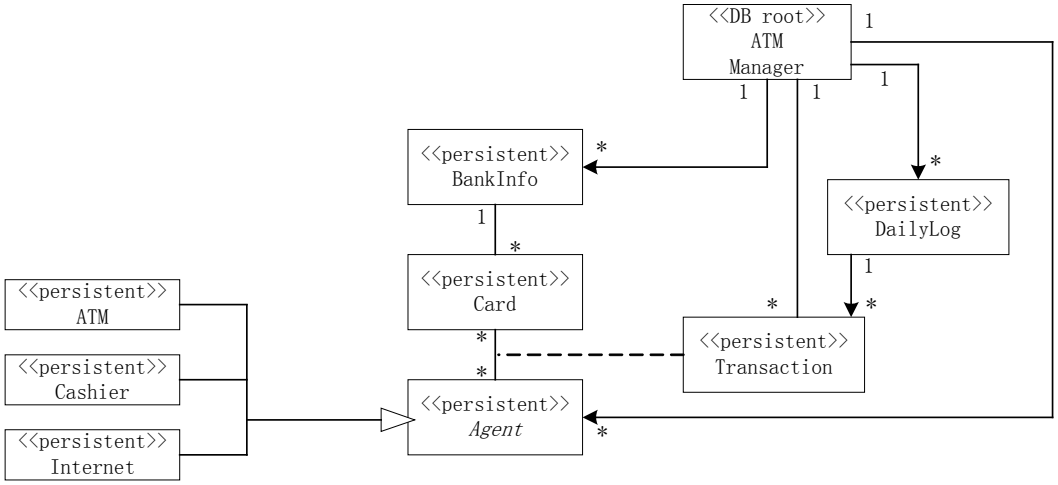

图 8-33 自动柜员机系统软件(ATM system software)的持久对象设计

这里的类有 ATM Manager(ATM 管理器)、BankInfo(银行资料)、Card(提款卡)、DailyLog(日志)、Transaction(事务)、代理(Agent)、ATM(自动柜员机)、Cashier(出纳点)、Internet(因特网客户机)。所有类都是持久类<<persistent>>，而 ATM 管理器则是数据库根<<DB root>>。

图 8-33 中也增加了一个新的分域，“日志”(DailyLog)类。它把同一天的事务归为一组，以加快每日总结报告的速度。

8.4.4 动态对象设计

现在，对自动柜员机系统软件 (ATM system software)做动态分析。

首先考虑的是持有提款卡的客户,使用自动柜员机(ATM)的用例。个案的行动者 (actor)就是持卡客户。对于客户来说，系统就是 ATM。发生在 ATM 背后的事情，包括 ATM 系统各个部件的相互作用等，暂时不必理会。用例详情在图 8-34 中给出。

使用个案：在自动柜员机(ATM)办理事项
行动者：　持有提款卡的银行客户
描述：　本个案描述客户用ATM办理事项的过程
典型事件过程：

行动者	系统
插入提款卡	询问口令
键入口令	根据卡的号码，查证口令。若正确,则询问事项种类(查资料，提款,存款，转账等)
选择事项种类	若是查资料，显示账户资料，否则询问金额
键入金额	处理并确认事项是否完成。若是提款，颁发现金
若是提款，提取现金	询问客户是否要办理另一事项
按“完毕”按钮	退回提款卡

图 8-34 持有提款卡的客户，使用自动柜员机的用例

根据概述中对 ATM 系统的讨论，可以得出 3 个暂时对象：ATM 管理器(ATM Manager)、ATM 客户机(ATM client)和银行(bank)。接着，、把图 8-34 的用例，利用序列图做更详细的描述，由此得出图 8-35 所示的 ATM 系统软件的序列图。

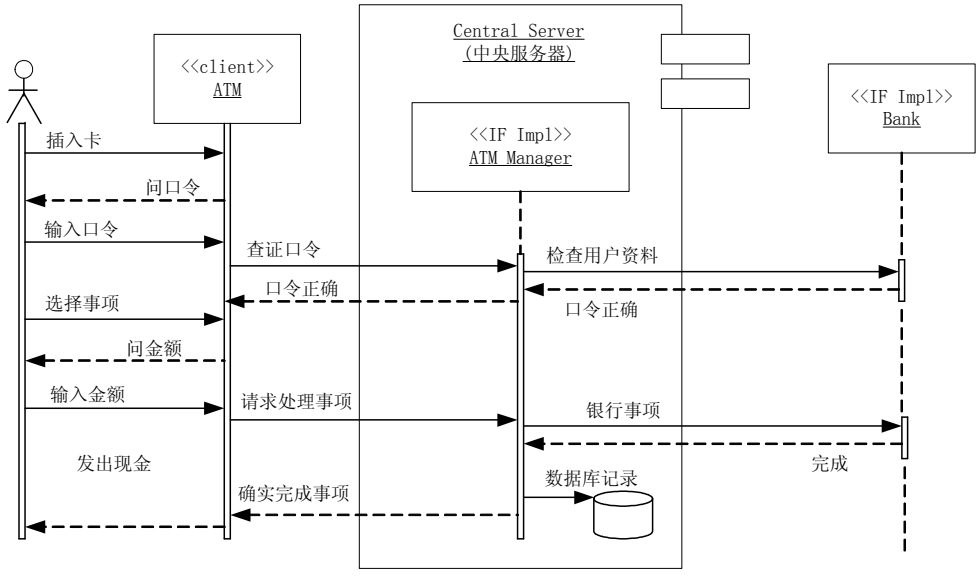

图 8-35 ATM 系统软件的序列图

图 8-35 显示的是一个提款过程，左上角的人形符号是持卡客户。带有类别<<client>>的是 ATM 客户机，也就是与客户直接接触的机器。中央服务器(Central Server)部件中，包

括 ATM Manager(ATM 管理器)和数据库(即图中的小圆柱体)。右边的 bank(银行),代表与 ATM 系统联网的某一家银行的服务器对象。类别<<IF Impl>>代表接口实施对象。

图中含有两对客户机 / 服务器,即 ATM 客户机和 ATM 管理器,以及 ATM 管理器和银行。注意,ATM 客户机和银行都在中央服务器之外。此外,ATM 客户机代表安装在自动柜员机里的一个软件部件,而不是图中的持久对象 ATM。

显然,图中的 ATM 管理器是中央服务器的主要对象。特别是,它可以作为中持久对象 ATM 管理器的外部接口,与外界对话。注意,图中的 ATM 管理器既是服务器,也是客户机。

序列图中的 3 个对象,在静态时是分离的。事实上,它们各存在于不同的计算机中,只在客户使用 ATM 时,才相互作用。

图中有两对客户机/服务器,提示我们要定义两个接口。一个 ATM 管理器接口,确定 ATM 客户机和 ATM 管理器之间的对话。另一个银行接口,指定 ATM 管理器怎样和银行通信。根据上面用例分析和序列图,得出两个接口的候选运算(见表 8-3)。

表 8-3 ATM 管理器(ATM Manager)和银行(Bank)接口的候选运算

接　口	候选运算

ATM Manager	《access operations》 //访问运算
	login(cardinfo: Cardinfo) //进入系统; 输入卡的资料
	logout() //退出系统
	《operations on an account》 //账户的运算
	getBalance(): Float //询问账户结余
	getHistory(): String //提取账户历史
	getAccountInfo(): AccountInfo //提取账户资料
	deposit(amount: Float): Float //存入输入金额
	withdraw(amount: Float): Float //提取输入金额
	transferTo(accountName: String, amount: Float): Float//转账到输入账户
	payBill(receiver: string,amount:Float): Float //付款给输入收款者

Bank	《access operations》 //访问运算
	login(accountInfo: AccountInfo) //进入系统; 输入卡的资料
	logout(account) //退出系统
	《account management》 // 账户管理
	open(accountInfo: AccountInfo。initialAmount: Float)
	// 开户; 输入账户资料和开户金额
	remove(accountInfo: AccountInfo)//除去输入账户
	showAccounts(): String //列出所有账户
	《operations on an account》 //账户的运算
	getBalance(): Float //询问账户结余
	getAccountInfo(): AccountInfo //提取账户资料
	getHistory(): String//提取账户历史
	deposit(amount: Float): Float//存入输入金额
	withdraw(amountt: Float): Float//提取输入金额
	transferTo(accountName:String,amount:Float):Float //转账到输入账户
	payBill(receiver:String,amount:Float):Float //付款给输入收款者

表 8-3 中出现的一些变量是 cardInfo(卡资料)、account(账户)、accountInfo(账户资料)、amount(数目或金额)、accountName(账户名)、receiver(接收者)。

表中出现的类型 CardInfo(卡资料)、AccountInfo(账户资料)是待定的数据结构。两个接口有相同的账户运算和访问运算(进入和退出系统)。主要的不同是,ATM 管理器以卡为中心,而银行则以账户为焦点。例如,银行接口中有账户管理的运算,包括开户,除去账户等。

表 8-3 并没有列出所有细节，以后在改良这两个接口时，会把接口及其运算更精确地定义。

8.4.5 通用接口设计

这一节继续介绍自动柜员机系统软件的案例，并做出详细的模块接口设计。先从银行模块接口开始，根据 8.4.4 节的讨论结果，银行的运算包括访问控制、账户管理和账户本身的运算等。银行需要管理多个账户，并控制对它们的访问。利用对象管理器样式是一个很好的模型。

在图 8-36 中，银行软件包(bank package)中有两个模块接口：AccountMgrIF(账户管理器)和 AccountIF(账户)及其实施类。这里“IF”结尾表示模块接口(interface)。

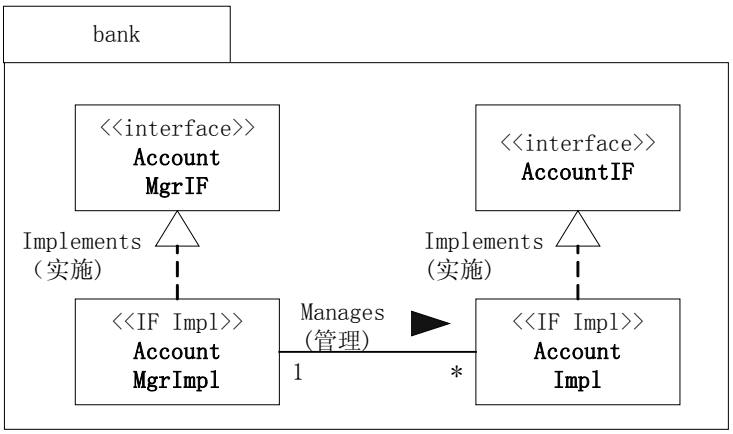

图 8-36 银行软件包中的两个接口

账户模块接口含有 3 个提取运算和 4 个账户事务运算，如图 8-37 所示。所有事务(transaction)都给回一个短实数或浮点实数(Float)，代表事务完成后账户的结余。账户资料(AccountInfo)是一个数据结构，含有账户名、号码、口令等。

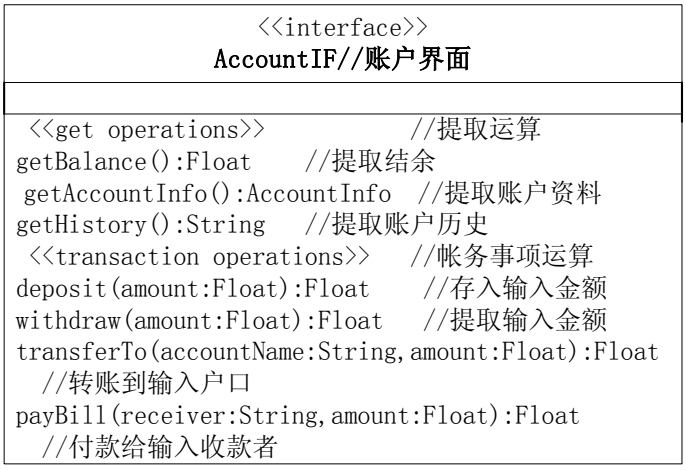

图 8-37 账户模块接口（AccountIF）的详细定义

图 8-38 的账户管理器模块接口的详细定义中，则含有访问运算和账户管理运算。例如，开新户(open)运算输入一个初始数目(initial Amount)，并给回新的账户对象。类似地，进入系统(login)运算先检查输入的账户资料，若没问题才给回一个现有账户的对象。

图 8-38 账户管理器模块接口（AccountMgrIF）的详细定义

接下来看 ATM 管理器(ATM Manager)模块接口。对象管理器样式仍然合用，并得到两个模块接口：SessionIF(使用期)和 SessionMFIF(使用期管理器)。图 8-39 的模块接口是给提款卡使用期定义的。由于这个模块接口只是把客户机的要求传递到银行的账户对象，所以它和图 8-37 的账户模块接口有同样的运算。

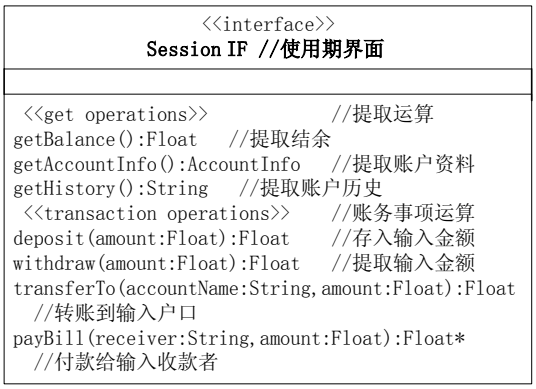

图 8-39 使用期模块接口（SessionIF）的详细定义

账户的实施类在图 8-40 中显示。注意其中的账务事务运算，都有“同步”(synchronized)指示词。这是由于一个账户可以有几张提款卡，所以在同一期间，可能有多于一个 ATM 客户机或使用期对象连接到同一个账户。为了保证多个客户机有秩序地更新账户资料，就要用“同步”的更新方法。也就是说，每个账务事务是一个整体，必须圆满结束后，才处理第二个。

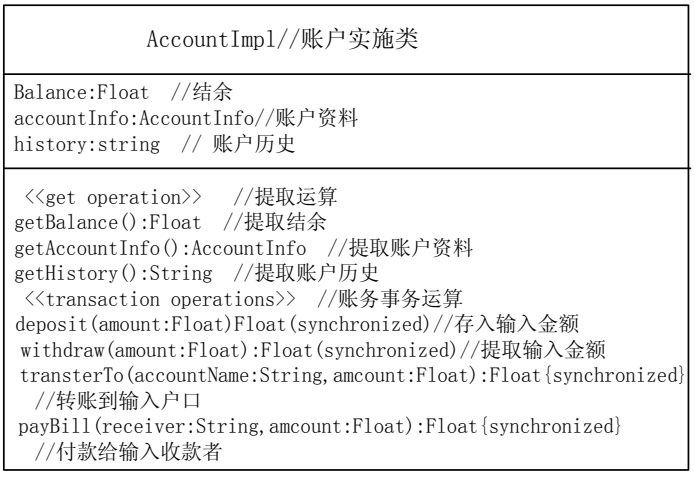

图 8-40 账户的实施类

UML 的“同步”指示词与 Java 语言中含义一样：在有多线程(multithread)运作时，某一个客户机线程在调用运算前，必须先把调用对象(这里是账户实施对象)“锁起来”，以防止其他客户机线程调用同一运算。等运算完毕后，再把锁开启，让其他客户机线程调用该对象。

在 ATM 系统中，由于每个银行有自己的程序模块接口，与其背后的数据库接触，账户模块接口的实施就可能在各个银行中有所不同。图 8-37 的账户模块接口保证了各银行的计算机都可以和 ATM 系统对话。这样做模块接口定义，是企业应用整合的常见策略。

最后，图 8-41 的使用期管理模块接口，含有访问控制运算(进入和退出系统)。它的实施类也可能有管理使用期对象的运算，但由于客户机无权管理使用期，这些运算也就不出现在图 8-41 的模块接口中。

图 8-41 使用期管理模块接口（SessionMgrIF）

使用期模块接口(SessionIF)的实施类中，可以取账户模块接口(AccountIF)作服务类，把账务事务运算委托给具体的账户对象。因此，使用期模块接口及其实施类 Sessionlmpl，就成了账户对象的适配器(adapter)，如图 8-42 所示。注意图中软件包 atmcs(代表 ATM 中央服务器)里的模块接口，是由图 8-42 的 ATM 管理器(ATM Manager)改良而来的。

图 8-42 软件包 atmcs 里包含的两个模块接口

图 8-42 中软件包 atmcs(代表 ATM 中央服务器)里包含两个模块接口：SessionIF(使用期)和 SessionMgrIF(使用期管理器)及其实施类。账户模块接口(AccountIF)作为服务类，把账务事务运算委托给具体的账户对象。这里“IF”结尾表示模块接口(interface)。

为了项目概述中开列的系统需求都被满足，表 8-4 就每一个需求，列出了对应的设计特点。这些特点总括了这个案例的几个部分，清楚地显示出所有需求的确都已满足。

表 8-4 ATM 系统软件设计的需求对应表

(1)对所有 ATM 使用和账务事务要保留记且能每天作报告	所有记录均由事务(Transaction)及其他持久对象提供（图 8-30）。报告则由具体程序提取持久对象资料得到
(2)系统要有适当的安全措施	图 8-41 的使用期管理模块接口，含有进入和退出系统运算，并以用户标识和口令作保护
(3)一个账户必须能同时从多处接入，即容许几个客户机同时连接到一个账户	账户的服务器对象(即实施类 AccountImpl)可以同步地处理多个客户机的多线程运作。
(4)允许将来有出纳点(Cashier)和因特网(internet)客户机等	这些新的客户机，可以作为抽象类“代理”(Agent)的子类而加进系统中(图 8-32 和图 8-33)

根据上面的两个软件包(bank 和 atmcs)，可以建立一个模拟的银行服务器，和一个模拟 ATM 中央服务器，其中只包含模块接口层的对象(即没有持久对象)，供测试和学习使用。

8.4.6 体系结构设计

这是本案例的最后一节，把前面的内容综合起来，并探讨自动柜员机系统软件 (ATM System Software)的体系结构。这一步其实不太难，因为前面的准备工作已经十分充分了。

在图 8-43 中，有 ATM 客户机(Client)、中央服务器(Central Server)和银行(Bank)。给每家银行加上自己的数据库，便得到图 8-44 的四级体系结构。

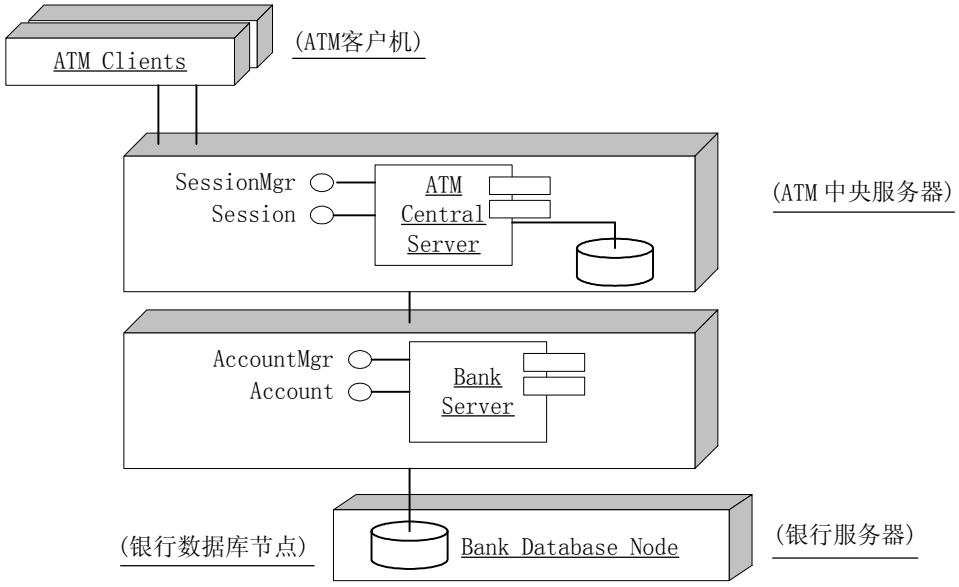

图 8-43 自动柜员机系统软件（ATM System Software）的四级(four-tier)体系结构

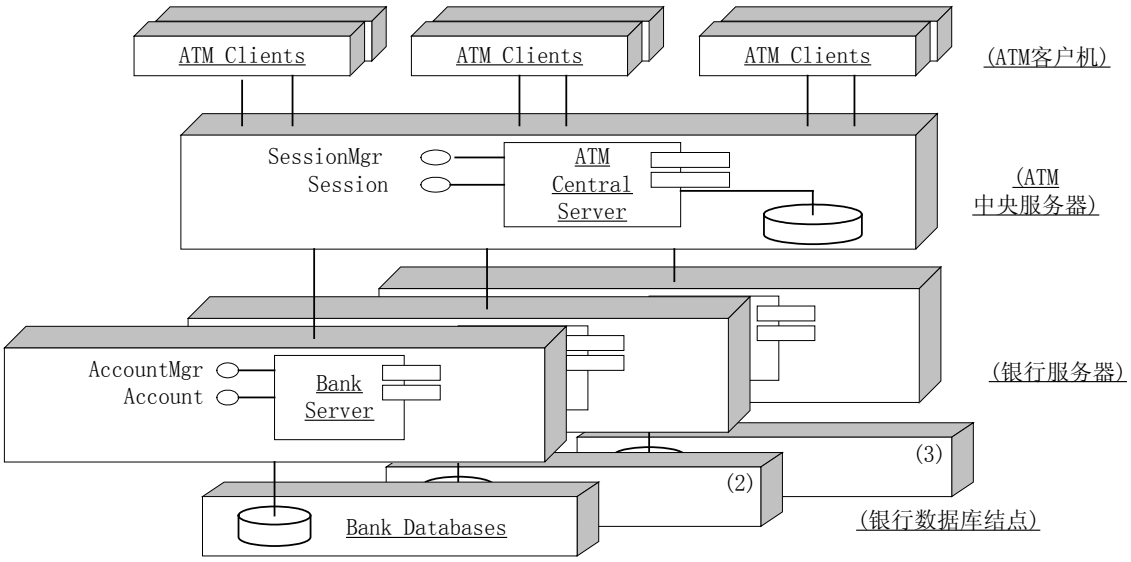

图 8-44 自动柜员机系统软件的四级体系结构

图 8-43 中的 ATM 客户机使用中央服务器的模块接口，而后者则用银行服务器的模块接口。

在这个体系结构中，ATM 客户机与中央服务器的模块接口对话。中央服务器的模块接口包括：使用期管理器(SessionMgr)和使用期(Session)，它们与图 8-42 中显示的模块接口一样。至于图 8-36 银行软件包中出现的模块接口，即账户管理器(AccountMgr)和账户(Account)，则由中央服务器中的使用期实施对象使用。

图 8-43 中的中央服务器有持久数据或数据库，而每间银行则有自己的数据库，存储账户的详细资料。

也可在中央服务器之上增加一个网服务器。网服务器作为中央服务器的客户机与之对话，并可处理大量轻便客户机的要求。

自动柜员机系统软件的四级体系结构，可以让大量的 ATM 客户机连接到多个银行服务器（如图 8-44 所示）。由此，中央服务器就成了各银行的统筹机制。当然，如果客户机数量很大，它也可能成为整个 ATM 系统运作的瓶颈。

图 8-44 中显示大量的 ATM 客户机与多个银行服务器连接，中央服务器(Central Server)是整个 ATM 系统的统筹。

有了系统的体系结构，进一步了解每一个节点的详细情况。图中的银行数据库可以假设是已知或给定的。在实践中这些数据库多半是关系数据库，它们决定了银行服务器部件的具体实施。中央服务器节点有两个模块接口，也有自己的持久数据。下面详细审查中央服务器中的层状结构。

中央服务器里的对象可以分成两层：模块接口层和持久层。以前在图 8-42 已经得出模块接口设计，现在就可以用它来进一步改进图 8-33 的持久对象设计。

比较上述两个设计图，发现它们并不完全吻合。图 8-42 的使用期管理器实施类(SessionMgrImpl)对应图 8-33 中的数据库根“ATM 管理器”(ATM Manager)。但是，使用期实施类(SessionImpl)则没有对应的持久对象。考虑到每一个客户机，都持有使用期实施类的对象引址或把柄，不妨在持久层中插入一个使用期(Session)对象，并用它来管理一个使用期内的所有事务（Transaction)。由此得到图 8-45 的持久层设计。

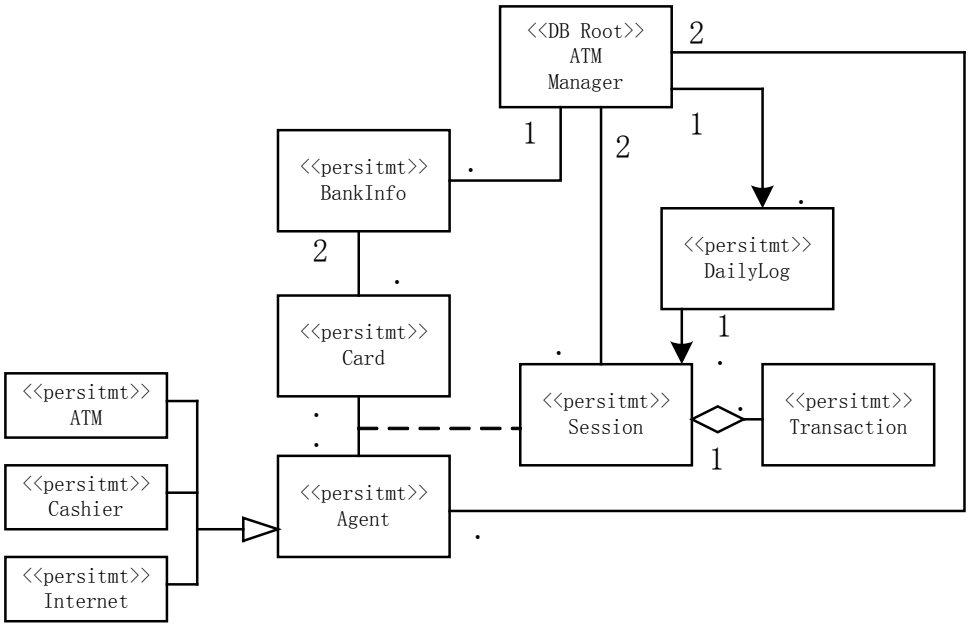

图 8-45 中央服务器里的持久层对象设计

图 8-45 中的类有：ATM Manager(ATM 管理器)、BankInfo(银行资料)、Card(提款卡)、DailyLog(日志)、Session(使用期)、Transaction(事务)、代理 (Agent)、ATM(自动柜员机)、Cashier(出纳点)、Internet(因特网客户机)。所有类都是持久类<<persistent>>，而 ATM 管理器则是数据库根<<DBroot>>。

接着把两层对象连接起来，结果在图 8-46 给出。注意图中只显示了两个持久对象。

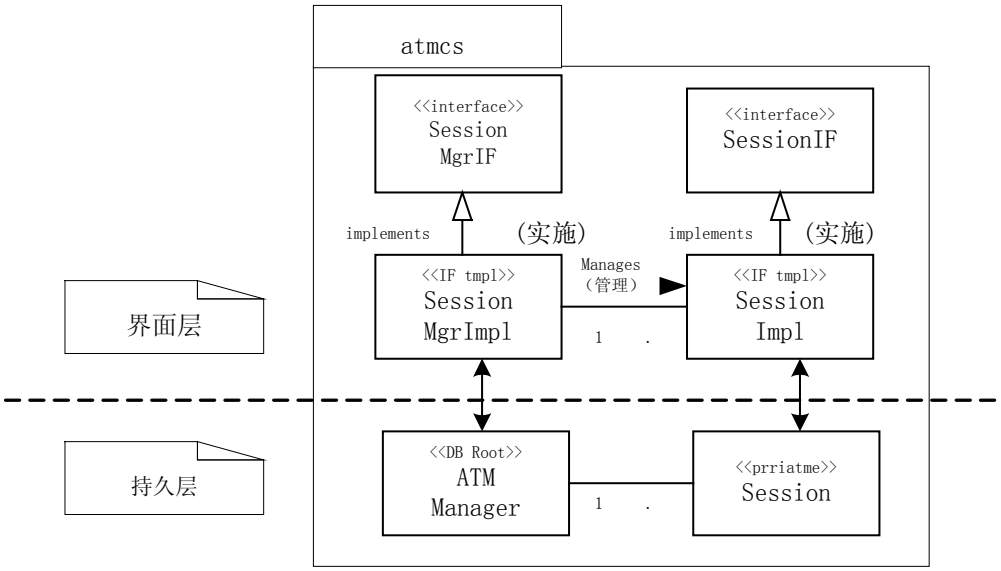

图 8-46 ATM 中央服务器里的模块接口层和持久层

软件包 atmcs 含有两层对象，包括 SessionMgr(使用期管理器)、Session(使用期)及其实施类的前面层，包括 ATM Manager(ATM 管理器)及使用期的持久层。注意，图中只显示两个持久对象。

把上面的层式对象设计考虑在内，就能修改图 8-35 的序列图，以显示两层对象之间的相互作用。在动态对象设计中的用例(见图 8-34 “在 ATM 办理事务”)将影响到下列的持久对象：ATM 管理器(ATM Manager)、使用 期(Session)、事务(Transaction)和日志(DailyLog)。

至于其他持久对象，如银行资料(BankInfo)和提款卡(Card)等，实际是由另外的管理过程来处理(包括构造和维护)。

系统的管理(Administration)通常有几个用例，由它们可引出详细的管理功能要求。尽管系统要求中可能没有明确地列出这些要求，一定的管理功能总是需要的。在实施方面，可以让程序直接操纵有关的持久对象，也可以通过一个独立的管理模块接口来做。把管理功能实施以后，ATM 中央服务器开发项目结束了。

小 结

统一建模语言 UML 实现了面向对象建模工具的统一，自 1997 年由 Booch、Jacobson 和 Rumbaugh 等提出到现在的十余年来，它已经成为面向对象方法中的一个标准。UML 作为一种建模语言，用图表示它的语法，用元模型表达它的语义，采用模型来描述系统的结构(或静态特征)以及行为(或动态特征)。它能从不同的视角为系统建模，形成了各种视图。每个视图代表系统完整描述中的一个抽象，显示了被开发系统中的一个特定方面。

本章介绍了 UML 中包含的视图和图，介绍了如何用例图、类图、对象图、构件图、部署图表示系统的静态模型；用状态图、时序图、协作图和活动图描述系统的动态模型；给出了 UML 相关图与 Java 表示之间的对应关系，有助于更具体地理解建模、图等抽象概念和表示；最后，通过一个实例，具体说明了如何利用 UML 进行分析与建模。

由于 UML 内容十分丰富，同时，还在不断地发展中。要完全掌握和运用自如，需要进行不断地项目开发实践。本章只对 UML 做了简要介绍，希望读者能够通过本章的学习，了解和掌握 UML 及建模的基本方法和相关理念，为尽快将相关知识应用于实际，提供一个导向和参考作用。

习 题

1. 建立分析和设计模型的一种重要方法是 UML，它是一种什么样的建模方法？它如何表示一个系统？

2. UML 的定义是什么？它的组成部分有哪些？

3. UML 的内容包括哪些成分？它的特点是什么？

4. UML 作为一种建模语言，它是如何表示其语法和语义的?

5. UML 中提供了哪几种图，试述每种图所描述的内容。

6. 对工资管理系统，用 UML 所提供的图形符号建立系统的静态模型。

7. 对飞机订票系统，用 UML 所提供的图形符号建立系统的动态模型。

8. 对图书馆管理系统，用 UML 所提供的图形符号建立系统的物理架构模型。