针对应用系统ER模型的MDA模型转换方法*)

姜 泉 赵建华 李宣东 郑国梁

(软件新技术国家重点实验室 南京大学计算机科学与技术系 南京 210093)

摘 要 MDA是由OMG提出的一种以模型为主要开发产品的软件开发方法。开发人员首先建立与具体技术平台的实现细节无关的高抽象程度的平台无关模型(Platform Independent Model,PIM)。然后,开发人员可以通过模型转换,将高层次的模型逐渐转换成为包含了实现细节的平台相关模型(Platform Specific Model,PSM),直到最终的代码。模型转换的有效性决定了MDA开发方法的效率。模型转换的难点不在于模型的语法映射,而是保证转换过程中模型语义的一致。本文采用EDOC作为平台无关模型的描述方法。EDOC是OMG制定的、和具体应用平台无关的、适用于企业分布式系统建模的标准。本文针对EDOC中的实体关系模型建立了从这些模型到J2EE平台模型以及目标代码的模型转换规则。我们还在转换规则中引入了多个设计模式。软件开发人员可以根据对软件的非功能性需求,应用这些规则来选择不同的设计模式,将平台无关的ER模型转换成为J2EE平台上的模型。

关键词 MDA,模型转换,EDOC,设计模式

MDA Model Transformation Methods for EDOC ER Models

JIANG Quan ZHAO Jian-Hua LI Xuan-Dong ZHENG Guo-Liang

(The State Key Laboratory of Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing 210093)

Abstract The Model Driven Architecture(MDA) proposed by OMG uses software models as the primary artifacts of software development. In MDA, application system models are first constructed independently of specific platforms and implementation technologies. These high abstract models can be automatically mapped into low-level models or declarative codes by model transformation technologies. The efficacy of Model Driven Architecture is mainly determined by the efficacy of model transformation tools. The main difficult problem of model transformation is to maintain semantic consistency between models of different abstract levels during transformation. We use the EDOC profile, a platform-independent standard adopted by OMG, to describe PIMs. This standard is designed for modeling distributed enterprise applications. In this paper, we focus on the problem of how to define transformation rules from EDOC entity and relationship models to concrete implementations on J2EE platform. Several design patterns are introduced into model transformation rules in order to improve transformation efficacy. Using these transformation rules, platform-independent models(PIMs) can be transformed to J2EE platform dependent models(PSMs) automatically.

Keywords MDA, Model transformation, EDOC, Design patterns

1 引言

模型驱动体系结构(Model Driven Architecture,MDA)[1]的目标是在体系结构的层次将系统的结构/功能与具体的平台实现细节相分离,提升软件开发的抽象层次。MDA将模型分为平台无关模型(Platform Independent Models,PIMs)和平台相关模型(Platform Specific Models,PSMs)两个大的抽象层次。其中抽象层次比较高的应用系统的平台无关模型是MDA软件开发的重心。而在其他的软件开发过程中,这样的抽象模型仅仅是开发人员据以编写代码的蓝图。MDA思想具有以下优点:①PIM提供了系统结构和功能的形式定义,抽离了技术实现的细节;②通过PIM可以映射到多个平台的PSM,实现了异构平台之间的互操作性,以及应用程序在不同平台之间的可移植性;③通过PIM到PSM的一致性转换,保证了设计和实现的一致和同步。

MDA是一个以模型为中心的软件开发过程,这就需要使建模者能够用抽象的平台无关模型精确、全面地表达他的真实意图。从问题域的角度,解决这个问题的最好方案不是由程序员将领域知识实现为低层次的代码,而是由建模者直接使用特定领域中熟悉的术语和概念来描述系统[3]。这就要求领域建模语言能够描述特定领域的知识。企业分布式对象计算规约(Enterprise Distributed Object Computing profile,简称EDOC规约)就是领域建模语言的一个例子,它是一套基于UML的适用于企业分布式计算的建模标准。

MDA另外一个重要的目标是实现不同层次的模型之间的自动转换,以加速系统开发过程。其中,PIM-PSM自动转换是实现MDA思想的关键。高层次PIM到低层次PSM的转换建立起应用系统的设计与实现之间的桥梁,并保持同一系统不同视点的模型之间的一致性。实现自动化的模型转换需要工具的支持,而工具的核心是描述如何从PIM转换得到

*)国家自然科学基金(批准号60203009,60233020)、江苏省自然科学基金(批准号BK2003408)和国家973项目(批准号2002CB312001)资助项目。姜 泉 研究生,主要研究领域为软件工程;赵建华 博士,副教授,研究领域为形式化方法、软件工程、程序设计语言;李宣东 教授,博士生导师,研究领域为形式化方法、模型检验;郑国梁 教授,博士生导师,研究领域为软件工程、软件开发环境。

PSM 的方法的模型转换规则。

目前，模型转换工作的难点在于如何设计转换规则，以正确实现领域建模语言所表达的意图。基于这个问题，Jonathan Sprinkle 在文[4]中指出：模型转换过程必须基于模型语义，而语法映射只是模型语义转换的部分输出。因此我们将根据领域建模语言的元模型语义来建立转换规则，以实现基于语义的模型转换。

我们在本文中将提出并实现一个针对 ER 模型的 MDA 模型转换方法。该方法根据 EDOC 实体-关系元模型的语义，并结合 J2EE 设计过程中常用的设计模式，建立 EDOC 平台无关模型到 J2EE 平台相关模型的转换规则。我们的目标是使开发人员可以使用模型转换工具由抽象的 EDOC 模型自动生成可执行的、实用的分布式计算系统。开发人员还可以根据应用系统的特定非功能性需求在模型转换中使用不同的设计模式。

2 MDA 框架中的模型转换

根据文[3,4,7]的工作，我们认为实施 MDA 模型转换的基础包括：

(1)建模所需的基本元素和建模规则，即模型的抽象语法(abstract syntax)；

(2)模型的具体表示法，即模型的具体语法(concrete syntax)，如图形化的 UML 模型；

(3)模型元素如何与现实世界的概念相关联，即模型元素的语义(semantics)；

(4)不同层次模型之间的映射，通过映射关系建立起模型之间的关系。

在 MDA 框架中，人们使用元模型来描述建模过程中可使用的合法元素，以及建模的规则[5]。本文除了使用元模型定义模型的抽象语法，还使用元模型定义模型的语义以及不同模型之间的映射关系。在 OMG 提出的 MOF(Model Object Facility)[5]的 4 层模型体系中，位于 M2 层的元模型定义了 M1 层模型的模型元素、模型语义和建模规范。作为标准化的建模语言，EDOC 规约在 MOF 体系中位于 M2 层，定义了具体的应用系统的 EDOC 模型(位于 M1 层)。本文将使用 EDOC 规约的实体-关系元模型作为 PIM 的元模型，提供分布式计算环境下 ER 模型的建模元素和模型语义。

我们选择已为工业界广泛应用的 EJB 元模型作为平台相关模型的元模型。我们将根据 EDOC 模型语义在 M2 层定义平台无关元模型(EDOC)和平台相关元模型(EJB)之间的转换规则。转换规则将保证 PIM 模型语义到平台相关模型的正确传递。模型转换工具根据转换规则建立 PIM 与 PSM 的映射关系，并根据开发者设定的 PIM 标记在转换中应用设计模式，使得转换所得 PSM 能够适应于不同的非功能性需求。我们将在下面章节简单介绍 EDOC 中实体关系模型的概念。

3 EDOC 实体-关系元模型

EDOC 规约是 ISO RM-ODP(Reference Model of Open Distributed Processing)[9]运用于企业规模的系统所得到的，目标是指导企业分布式计算系统的建模。在 EDOC 的建模框架中，实体元模型和关系元模型的组合从 ISO RM-ODP 定义的 3 个视点(viewpoint)[9]，包括企业视点、计算视点和信息视点，描述了分布式计算系统的静态结构。从企业视点来看，业务活动中的执行者、响应者和产生的结果均建模为实体模型，而这些参与者之间的关系则建模为关系模型，从而为描述业务活动提供了必要的词汇表；从信息视点来看，业务活动中信息的结构被建模为实体模型，信息之间的关联被建模为关系模型，从而表达了信息的静态语义；从计算视点来看，实体模型描述了可计算的实体组件，为在计算机上实现业务概念提供了面向对象和数据库的结构[10]。EDOC 实体-关系元模型如图 1 所示(参见文[10,11])。

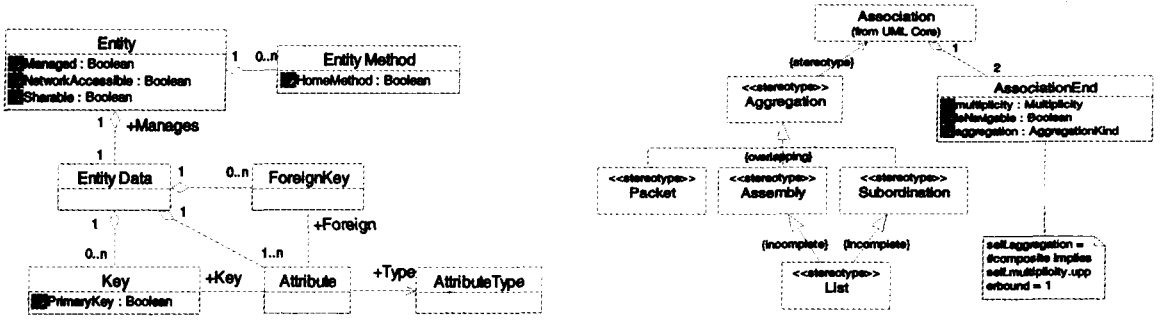

图 1 EDOC 实体-关系元模型

EDOC 为实体关系模型的建模定义了良好的语义，因此使用 EDOC 实体-关系元模型描述的系统模型具有精确和相对完备的语义。这将使我们的转换规则可以根据 PIM 模型的语义来生成 PSM 模型。下面将介绍 EDOC 实体-关系元模型的语义。

3.1 EDOC 实体元模型语义

实体(Entity)代表了业务领域中的概念，封装了表示客观世界状态的信息，并提供对它所包含的信息进行的业务逻辑操作。实体封装了实体数据(Entity Data)和基于实体数据的方法(Entity Method)，并提供了对实体数据的管理和访问接口。实体数据由属性(Attribute)及其类型(AttributeType)组成。

在实体元模型中，实体具有 Managed，NetworkAccessible 和 Sharable 3 项分布式计算属性。Managed 属性指示是否所有的实体实例均被某个实体管理。具有 Managed 属性的实体可以提供全局的管理操作，如新建、查找和删除实体实例的方法。NetworkAccessible 属性指示是否能够通过网络访问实体。具有 NetworkAccessible 属性的实体需要实现远程访问接口。Sharable 属性指示实体是否支持并发事务和多用户会话。具有 Sharable 属性的实体必须提供事务管理机制以支持序列化访问，而没有共享属性的实体不能实现持久化。

与面向对象技术中的普通类不同，每一个实体实例都由

一个唯一的关键字(key)所标识。关键字是能够唯一标识每个实体实例的最小属性组。关键字可能有多组,被设计者挑选出来在实体中唯一标志每个实体实例的关键字称为实体的主关键字(Primary Key)。如果一个实体的关键字出现在另一实体中,则称此关键字为另一实体的外键(Foreign Key)。更加详细的定义请参阅参考文[10]。

3.2 EDOC 关系元模型语义

关系(relationship)是实体之间的一组有意义的关联。关系具有两个关联端,分别指向参与关系的两个实体。关系的语义信息通常由 3 部分组成:实体是否可通过关系访问,实体参与关系的程度和关联实体之间的依存关系。关联端可溯性(isNavigable)指示了实体是否可以通过关系被另一端实体所访问,我们约定关系中至少有一端可溯。实体参与关系的程度由关联端多重性(multiplicity)决定。其中,多重性的下界决定了实体是否必须参与关系。但关联端的可溯性和多重性并不指示该实体是否可以决定另一方的生命期,因此我们需要使用聚合关系(Aggregation)显式定义实体之间的依存关系。

聚合关系是一种整体-部分关联,定义聚合属性(aggregation)的一端所指向的实体为聚合关系的整体(whole),而另一端所对应的实体为聚合关系的部分(part)。聚合关系的聚合属性可以取共享(shared)和组合(composite)。

EDOC 关系元模型的 4 种聚合子类 Packet, Assembly, Subordination 和 List[11]分别定义聚合关系中整体与部分的不同依存关系:Packet 聚合代表整体和部分无依赖;Assembly 聚合代表整体依赖部分;Subordination 聚合代表部分依赖整体;List 聚合多重继承了 Assembly 聚合和 Subordination 聚合,代表整体与部分互相依赖。可见,Assembly 聚合和 Subordination 聚合的语义定义存在交叉,它们的交集即为 List 聚合。

为了制定无歧义的转换规则,我们增加两种聚合子类:Assembly * 和 Subordination * 。Assembly * 是一种 Assembly 聚合,但不属于 List 聚合;同理,Subordination * 是一种 Subordination 聚合,但不属于 List 聚合。因此,Packet, Assembly * , Subordination * 和 List 4 种聚合子类的语义互不交叉,并完备描述了整体与部分的依存关系。我们将使用以上 4 种聚合子类定义关系模型中实体的依存关系。

另外,在我们的转换方法中将聚合属性看作附加于聚合子类之上的约束:聚合属性为 shared 时对聚合子类和多重性均没有约束;聚合属性取 composite 时,该聚合关系整体端多重性的上界必须为 1,且聚合子类必须为 Subordination * 或 List。这样做的原因是 composite 聚合描述了部分依赖于单一整体的聚合语义。

除了聚合关系,EDOC 关系元模型还定义了一种引用关系(Abstract Reference)。但引用关系仅适用于描述概念模型,不作为模型转换的实际输入[11]。因此本文只讨论针对聚合关系的模型转换。

4 基于模型语义的 EDOC-EJB 转换

实现自动模型转换需要两个前提[2]:①作为转换输入的模型具有足够明确的语义;②高质量的转换算法。EDOC 领域建模语言已经为 PIM 提供了准确和丰富的语义。下面我们将描述如何实现基于模型语义的从 EDOC 到 EJB 的模型转换,并描述如何使用设计模式提高模型转换的有效性。首先我们简单介绍一下 Enterprise JavaBeans(EJB)。

4.1 EJB 概述

J2EE(Java 2 Enterprise Edition)是由 Sun 公司推出的中间件平台。Sun 推出这个平台的目的在于简化和规范多层分布式企业应用系统的开发和部署。Enterprise JavaBeans (EJB)是 J2EE 分层体系结构的核心,为企业级应用程序开发屏蔽了系统的底层细节,提供了对基于组件的企业应用程序开发、组装和部署的全生命周期的支持。

EJB2.1 规范[12]提供了 3 种 EJB,即会话 Bean、实体 Bean 和消息驱动 Bean。会话 Bean 是一种非持久性的对象,它实现了在服务器上运行的业务逻辑,其实例不能被多客户共享。会话 Bean 可以分为两类:有状态(stateful)会话 Bean 和无状态(stateless)会话 Bean。实体 Bean 是一种持久性对象,它代表存储在相应数据库中的实体的一个对象视图。EJB 规范支持两种实体 Bean 的持久性策略,分别是 Bean 管理的持久性(Bean-Managed Persistence, BMP)和容器管理的持久性(Container-Managed Persistence, CMP)[12]。对于前者,开发者自己编写数据库持久化代码来保证实体 Bean 与数据库的交互;而对于后者,CMP 实体 Bean 使用说明性代码定义持久性的实现,由 EJB 容器负责自动生成具体的数据库操作。本文选择 CMP 方式实现实体 Bean 的持久化。

按照 EJB2.1 规范,会话 Bean 和实体 Bean 都可以包含 Home 接口、组件接口以及 Bean 的实现类。对于实体 Bean,还可以包含有主键类。实体 Bean 的 Home 接口和组件接口还具有远程和本地的区别。EJB 的各个构成要素以及其他特性则由组件描述信息指定。图 2 是参考文[12~14]给出的一个简化的 EJB 元模型。

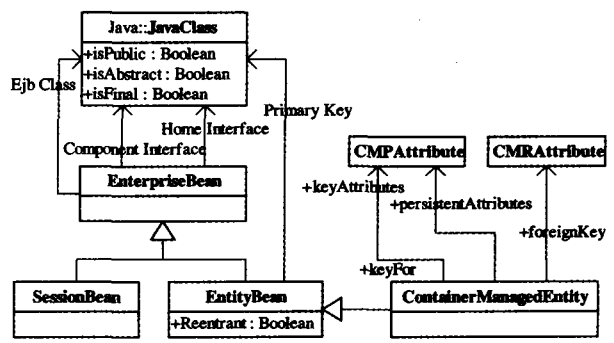

图 2 EJB 元模型

4.2 EDOC 元模型到 EJB 元模型的转换规则

基于语义的模型转换不要求 PSM 与 PIM 在语法上同构,但必须保证 PSM 与 PIM 在语义上一致。这要求转换所得的 PSM 即 EJB 是 EDOC PIM 的一个语义完备的实现。我们的解决方法是在元模型层次定义基于语义的 EDOC-EJB 转换规则,在执行转换时根据转换规则建立 PIM 和 PSM 的映射关系。例如,模型转换工具可以根据 PIM 中的一个实体模型是否具有多用户共享语义,将它映射为一个 EJB 实体 Bean 或者一个普通的 Java 类。

在我们的 EDOC-EJB 转换规则中,实体模型中的元素被映射为 EJB 实体 Bean 的组件结构及其属性。主要的规则如下:

①Sharable 属性为真且 Managed 属性为真的每一实体被转换为一个 CMP 实体 Bean,并生成相应的 Home 接口、create()和 findByPrimaryKey()方法。其它的实体转换为普通

的 Java 类。

②对于转换成为 Bean 的实体，如果其 NetworkAccessible 属性为真，转换工具将自动生成相应实体 Bean 的远程 Home 接口和远程组件接口；而 local 标记（开发者设定的 PIM 标记）为真的实体则生成本地 Home 接口和本地组件接口。

③PIM 的实体主键被转换为实体 Bean 的主键类。同时主键类的 hashCode()和 equals()方法也将由工具自动生成。

④PIM 的实体属性被转换为 Bean 实现类的 CMP 域。每一 CMP 域在实现类中对应一对抽象的 get/set 方法。

在我们的 EDOC-EJB 转换规则中，关系模型的元素被映射为 EJB2.0 的 Container Managed Relationship(CMR)和对应实体 Bean 的删除约束。同时，关系模型也会影响相关实体转换结果。这些转换规则主要包括：

①PIM 中每一关系对应 EJB 组件描述信息中 relationships 项的一个〈ejb-relation〉。

②若 PIM 关系的关联端的多重性上界大于 1，则将〈ejb-relation〉对应关联端的 multiplicity 置为 many，否则置为 one。

③关联端的可溯性转换为可溯端实体的属性：如果关系只有一端可溯，则在不可溯一端实体对应的 Bean 中生成可溯端的 CMR 域；如果关系两端可溯，则参与关系的两实体分别生成对方的 CMR 域。

④PIM 实体的依存关系转换为实体 Bean 的删除约束：(1)如果关系为 Assembly * 聚合且部分的多重性上界为 1，为对应于整体的实体 Bean 添加 cascade-delete 约束；(2)如果关系为 Subordination * 聚合且整体的多重性上界为 1，为对应于部分的实体 Bean 添加 cascade-delete 约束；(3)如果关系为 List 聚合且整体和部分的多重性上界均为 1，为双方对应的实体 Bean 均添加 cascade-delete 约束。有关 cascade-delete 约束的规则的依据是：EJB2.1 规范[12]中规定，只有关系中某实体 Bean 的多重性为 one，并且另一实体 Bean 的生命期依赖于该实体时，才能为另一实体 Bean 添加 cascade-delete 约束。

从上面的转换规则可以看出，我们的转换规则不是简单的字面上的替换。通过针对语义的转换规则，模型转换工具可以帮助开发者完成更多的开发工作。

5 设计模式在模型转换中的应用

与 PIM 相比，PSM 需要考虑更多与运行和性能相关的因素。为了提高转换结果的实用性和性能，在模型转换过程封装设计模式是一种有益的尝试。设计模式[15]是对常见的重复出现问题的最佳解决方案，体现了领域内集体的智慧和经验。使用设计模式可以避免重新设计和创造，并为设计工作提供可复用的结构。例如，为了将功能描述与具体实现分离，我们通过应用 EJB 组件结构隐含的 bridge 模式[17]为 PIM 实体生成 EJB 组件接口，从而隐藏了 EJB 实现类的细节。

在 MDA 框架中，开发者可以使用附加于 PIM 模型元素的 PIM 标记来指示针对该元素的不同转换方式[1,6]。PIM 标记并不是 PIM 的组成部分，也不增加 PIM 的语义。我们的转换工具将 PIM 标记处理为 PIM 元素的附加属性，它有自己的类型和对应的值。我们将增加 5 个 PIM 标记：transferObject，isLocalDTO，facade，delegate 和 locator。其中 facade 标记为字符型，其他标记均为布尔型。这些标记将在模型转换过程中被用来确定不同的设计模式的应用。

5.1 数据传输对象和会话外观模式

影响 EJB 实体 Bean 性能的关键是远程方法调用的粒度。粗粒度的实体难以复用，并增加了设计的复杂性；但对细粒度实体的频繁调用又会显著增加网络开销，降低系统性能。对于这个问题，数据传输对象和会话外观模式可以为细粒度的实体 Bean 提供粗粒度的网络访问机制[15,16]。因此我们在转换中允许开发者通过 transferObject 和 facade 标记来确定是否使用这两种模式来封装对实体数据的访问。

数据传输对象模式通过为实体 Bean 生成一个包含其所有属性的数据传送对象(Data Transfer Object，DTO)来完成对实体数据的批量传输[15]。转换工具为 transferObject 标记值为真的实体按照以下规则自动应用数据传输对象模式：

①为实体自动生成一个普通的可序列化的 Java 类，即附属于该实体的数据传输对象。该 Java 类封装实体所有的属性，并为实体每一个可访问属性提供 get 方法和每一个可修改属性提供 set 方法(实体的主键和外键不可修改)；

②在实体 Bean 远程接口自动生成 getDTO()和 setDTO()方法，使实体属性的远程访问和更新可以通过数据传输对象批量操作，有效提高实体组件的性能；

③如果实体的 local 标记和 isLocalDTO 标记的值均为 true，转换工具在实体 Bean 本地接口自动生成 getDTO()和 setDTO()方法；如果实体 local 标记值为 true 但 isLocalDTO 标记值为 false，转换工具在实体 Bean 本地接口为实体每一个可访问属性提供 get 方法和为每一个可修改属性提供 set 方法。增加 isLocalDTO 标记的原因是本地接口的数据传输使用了引用传递，因此是否需要本地的批量数据传输要由开发者进一步确认。

会话外观模式源于传统的 facade 模式[17]。这种模式是把会话 Bean 用作外观以封装参与工作流的实体 Bean，并提供了统一的粗粒度服务层，向客户端提供业务方法。通过会话外观模式的应用，设计者将紧耦合的实体组合为子系统，会话外观作为子系统的接口对外提供统一的实体管理和业务方法。使用会话 Bean 而不是普通 Java 对象作为外观的原因是普通 Java 对象不具备事务处理、远程访问、安全控制等 EJB 公用设施的支持。

会话外观模式应用的关键在于如何将 PIM 中的实体划分为不同的子系统。我们的转换方法允许开发者设定附加于实体的 facade 标记。转换工具在模型转换时将具有相同 facade 标记值的实体组织为子系统，提供一个统一的会话外观封装。会话外观模式主要应用于互相关联的本地实体，以隐藏本地关联实体之间交互，并整体对外提供访问接口以提高性能。因此我们规定：同一会话外观中的不同实体必须具有 local 标记并互相关联。

为了提高性能并隐藏实体 Bean 访问细节，会话外观需要以实体的数据传输对象为参数或返回值。因此转换工具在应用会话外观模式时，将强制为参与该模式的每一实体应用数据传输对象模式。主要的转换规则如下：

①如果具有相同 facade 标记值的实体均具有 local 标记并互相关联，则为这些实体生成一个无状态会话 Bean(必须包含远程接口)，作为整体的会话外观；

②对于 NetworkAccessible 属性为真的实体，在会话外观的远程接口生成其远程业务方法的包装。具体方法是复制该方法的接口，同时隐藏从实体 Bean 调用的细节；

③在会话外观的远程接口自动生成每一实体的 createXXX()，removeXXX()，updateXXX()，XXXfindByPrima-

ryKey()方法,实现对持久化对象的数据管理。以上方法均以实体的数据传输对象为参数或返回值,以隐藏实体 Bean 的结构并作为整体对外提供数据访问;

④在会话外观的远程接口中自动生成 Home 接口的其他 find 方法的包装。与 XXXfindByPrimaryKey()方法相同,find 方法在获取相应实体的组件接口或组件接口集合后,调用每一接口的 getDTO()方法,以生成数据传输对象或数据传输对象集合,并以此作为返回值;

⑤在会话外观的实现类中自动生成业务方法的实现:createXXX()方法解包传入的数据传输对象,并以此为参数,调用实体 Bean 的 create()方法,实现实体对象的创建;removeXXX(),updateXXX(),XXXfindByPrimaryKey()则根据传入的数据传输对象创建实体 Bean 的主键对象,并使用该主键对象实现实体 Bean 的删除和查找并调用实体 Bean 组件接口的 set 方法或 setDTO()方法实现实体数据的更新;

⑥自动生成会话外观的 setSessionContext()方法,目的是在会话外观初始化时为每一实体 Bean 缓存 Home 接口实例,有效避免频繁访问 JNDI 造成的性能损失。

5.2 业务代理和服务定位器模式

J2EE 是一种分层体系结构[18]。一个基于 J2EE 的系统可以划分为客户层(Client Tier)、表示层(Web Tier)、业务层(Business Tier)和数据层(EIS Tier)。如果表示层直接访问位于业务层的会话外观或实体 Bean,会使表示层依赖于 EJB 技术,造成表示层与业务层的紧耦合[15]。因此我们需要进一步考虑模型转换结果(在业务层和数据层)与表示层的分隔,即对表示层隐藏 EJB 的查找和访问细节。

业务代理(Business Delegate)模式[15]源于 proxy 模式[17],用来为表示层提供业务层的代理,从而降低表示层组件与业务层组件的耦合度,隐藏业务层的实现细节。转换工具为具有 delegate 标记的每一会话外观生成一个业务代理,并为业务代理生成对应于会话外观所有业务方法的接口。业务代理是一个普通的 Java 类,在使用时分发到表示层的各客户端,负责为表示层隐藏访问 EJB 的细节。

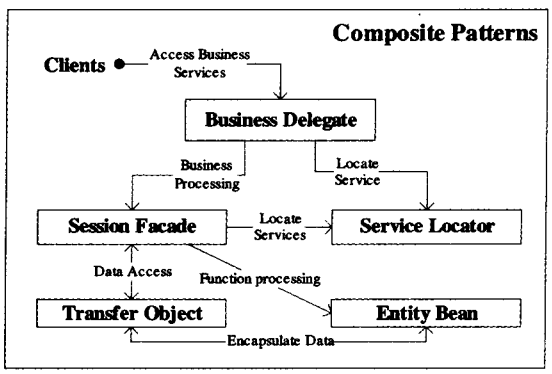

图 3 模式的组合

业务代理和会话外观均需要查找 EJB 组件,而 EJB 查找服务需要环境初始化、JNDI 目录检索和对象绑定等复杂的操作。为了防止转换结果中查找代码的大量繁殖,转换工具在转换时生成全局的服务定位器(Service Locator)。服务定位器帮助我们隐藏了查找复杂性、与服务容器相关的设置和业务对象的实例化过程等查找机制的细节[16]。客户端通过服务定位器可以简单地以名称获取组件资源。这样的做法使得更改转换所得代码的部署位置时不需要改动任何代码,只要重新配置服务定位器。

以上描述的 4 种设计模式很多时候是组合使用的。组合使用时,典型的各个模式之间的相互关系如图 3 所示。

6 一个实例

我们选取 Pet Store 的订单系统来演示我们的转换规则的作用。Pet Store 是 J2EE Blueprints[19]中的一个在线宠物商店演示系统。在转换工具建模环境下,Pet Store 订单系统的实体关系模型包括的 6 个实体,以及这些实体之间的关系如图 4 左侧所示。其中,联系方式和付款方式模块(包括 ContactInfo,CreditCard 和 Address 实体)可以被 Pet Store 其他系统复用,我们使用 facade 标记将其封装为 InfoFacade 子系统。同理,我们将订单模块(包括 Product,LineItem 和 PurchaseOrder 实体)封装为 OrderFacade 子系统。转换工具按照上文所述的转换规则将订单系统 PIM 转换为图 4 右侧所示的 PSM。

其中,转换工具将 Pet Store 订单系统中的实体转换为一一对应的实体 Bean,转换所得的每一实体 Bean 均具有图 4 右侧下方的组件结构:EJB 接口、实现、部署信息和数据传输对象。转换工具将 PIM 聚合关系转换为实体 Bean 之间的 CMR(如图中虚线所示)。同时,Pet Store 订单系统包含了全部 4 种聚合子类,分别转换为实体 Bean 部署信息中的不同删除约束。

转换工具将 PIM 的 OrderFacade 和 InfoFacade 子系统分别转换为 PSM 的 OrderFacade 和 InfoFacade 会话外观,提供了对 ProductBean, LineItemBean, PurchaseOrderBean 和 CreditCardBean,ContactInfoBean,AddressBean 整体的访问接口。在同一会话外观下,实体 Bean 之间的交互被隐藏起来。同时,转换工具使用业务代理模式分别为 OrderFacade 和 InfoFacade 会话外观生成表示层的代理类:OrderDelegate 和 InfoDelegate,针对表示层隐藏了 Pet Store 系统的 EJB 结构。代理类通过自动生成的服务定位器获取所需的会话外观服务。

从此实例可以看到,通过模型转换,EDOC 实体和关系分别转换为 EJB 实体 Bean 和实体 Bean 之间的关系。通过在转换中应用会话外观、数据传输对象、业务代理和服务定位器 4 种模式,转换所得到的系统可以拥有较好的非功能性特性。

结论以及和相关工作的比较 MDA 是软件开发方法学发展的一个重要方向,其目标是提高软件开发的自动化程度。实施 MDA 开发方法的关键在于:①如何实现自动化的模型转换;②保证转换过程模型语义的一致。本文提出了一个针对应用系统 ER 模型的 MDA 模型转换方法。该方法使用 EDOC 的实体-关系元模型建立适用于分布式计算系统的 ER 模型(PIM)。这个模型转换方法的核心是一套基于 EDOC 模型语义和设计模式的 ER 模型到 EJB 相关模型的转换规则。这些规则依据平台无关模型的语义来完成不同抽象层次的应用模型之间的转换,而不是仅仅进行语法级别的映射。同时,我们还尝试在转换中应用设计模式以提高模型转换的质量。开发者可以使用模型转换工具将抽象的 EDOC 模型转换为相应的 EJB 系统。通过选择使用不同的软件设计模式,生成的系统将具有更好的非功能性特性。

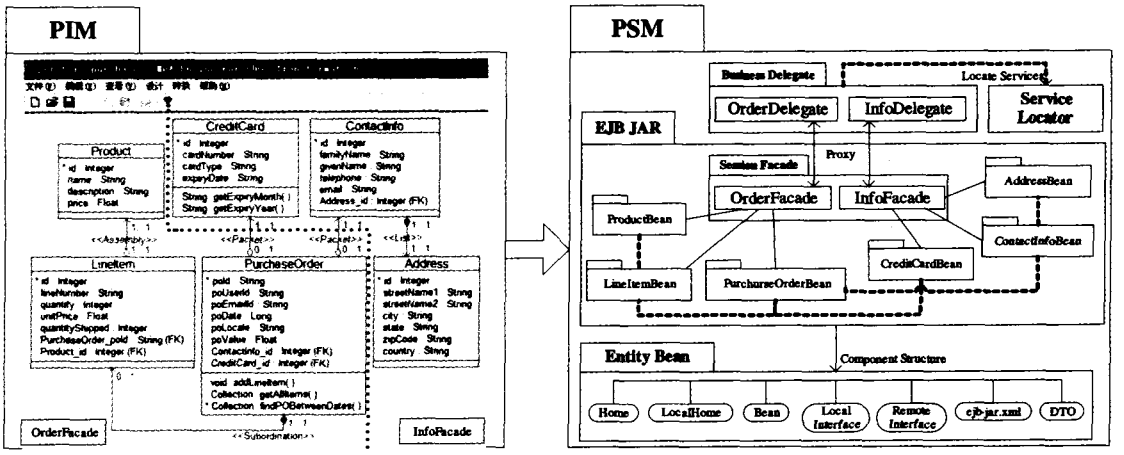

图 4　Pet Store 订单系统转换实例

近年来 MDA 模型转换方法的研究取得不少成果，也出现了一系列基于 UML 的模型映射工具。但这些工具往往着重于 UML 元素的语法映射，缺乏对基于模型语义的转换和领域建模的支持。AndroMDA[20] 是一个开源的 MDA 的模型转换框架，通过其模板机制(cartridges)定制转换规则，转换规则根据 UML 模型的构造型(stereotype)将其转换为相应的组件代码。AndroMDA 并不提供建模环境，其建模过程由传统的 CASE 工具完成。ArcStyler[21] 是 Interactive Objects Software GmbH 公司推出的一个商用 MDA 开发工具，提供了基于 UML Profile 的 PIM 建模环境，同时支持 J2EE 和. NET 作为目标平台。与 AndroMDA 类似，ArcStyler 在其"MDA Cartridge"模块中集成了转换规则的定义。但对于 ER 模型的转换，AndroMDA 和 ArcStyler 的模板机制只支持定义 PIM 到 PSM 的一对一映射，因此难以在转换中应用设计模式，也不能根据模型语义自动生成相应的约束。Compuware 公司的 OptimalJ[22] 将领域知识和商业规则组织为知识库，以此建立不同领域的 PIM 模型。OptimalJ 提供了一套复杂的基于模式的转换规则，在模型转换中实现了文[16]中所有 J2EE 模式的应用。但 OptimalJ 的领域建模基于 Compuware 公司独立开发的知识库，建模规则与 UML 标准并不兼容，难以实现 PIM 模型的复用。

与上述工作相比，本文的模型转换工作使用 EDOC 规约建立基于 UML 的 ER 模型，为分布式计算系统的建模提供了标准的方法，利于 PIM 模型在不同工具之间的复用。与其他基于 UML 的转换方法相比，本文定义的转换规则根据 ER 模型的语义信息实现 PIM 到 PSM 的不同映射，并将关联模型的语义转换为 J2EE 平台上相应模型之间的关联和约束。同时，本文提出的模型转换方法允许开发者通过 PIM 标记定制多种设计模式的应用。通过基于设计模式的转换规则，转换结果被整体组织为 J2EE 平台上可执行的，具有较好非功能性特性的应用系统，而不是作为孤立的模型交付给开发者。

参 考 文 献

1　Object Management Group. MDA Guide. Object Management Group, 2003. ww. omg. org

2　Object Management Group. Model Driven Architecture. Object Management Group, 2001. ww. omg. org

3　Karsai G, Agarwal A, Ledeczi A. A Metamodel-Driven MDA Process and Its Tools. In: UML'2003 Workshop in Software Model Engineering(WiSME 2003), 2003

4　Sprinkle J. Managing Intent: Propagation of Meaning During Model Transformations. In: UML' 2003 Workshop in Software Model Engineering(WiSME 2003), 2003

5　Object Management Group. Meta Object Facility(MOF)Specification, v1. 4. Object Management Group, 2002. ww. omg. org

6　Frankel D S. Model Driven Architecture: Applying MDA to Enterprise Computing. Wiley Publishing, 2003

7　Atkinson C. The Role of Meta-modeling in MDA. In: UML'2002 Workshop in Software Model Engineering(WiSME 2002), 2002

8　Object Management Group. Unified Modeling Language Specification, v1. 5. Object Management Group, 2003. ww. omg. org

9　ISO. ISO/IEC 10746-1: Information technology - Open Distributed Processing -Reference model. ISO, 1998

10　Object Management Group. Enterprise Collaboration Architecture (ECA)Specification, v1. 0. Object Management Group, 2004. ww. omg. org

11　Object Management Group. UML Profile for Relationships Specification, v1. 0. Object Management Group, 2004, ww. omg. org

12　Sun Microsystems. Enterprise JavaBeans Specification, Version 2. 1. Sun Microsystems. 2003. http://java. sun. com/j2ee/1. 4/docs/#specs

13　Object Management Group. Metamodel and UML Profile for Java and EJB Specification, v1. 0. Object Management Group, 2004, ww. omg. org

14　Java Community Process. JSR 26: UML/EJB Mapping Specification. Java Community Process, 2001. www. jcp. org

15　Marinescu F. EJB Design Patterns: Advanced Patterns, Processes, and Idioms. Wiley Publishing, 2002

16　Alur D, Crupi J, Malks D. Core J2EE(tm)Patterns: Best Practices and Design Strategies, Second Edition. Prentice Hall PTR, 2003

17　Gamma E, Helm R, Johnson R, et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994

18　Sun Microsystems. The J2EE 1. 4 Tutorial. Sun Microsystems, 2003. http://java. sun. com/j2ee/1. 4/download. html#platformspec

19　Sun Microsystems. Sample Application Design and Implementation, v1. 3. 1. Sun Microsystems, 2002. http://java. sun. com/j2ee/reference/blueprints/index. html

20　AndroMDA. http://www. andromda. org/

21　IO-software. ArcStyler. http://www. io-software. com

22　Compuware Cor. OptimalJ. http://www. compuware. com