ENTERPRISE ARCHITECT

用户指南系列

模型变换

Author: Sparx Systems

Date: 2022-08-30

Version: 16.0

创建于 ENTERPRISE ARCHITECT

目录

模型变换

创建模型的一大优势是能够操纵它们以产生输出，从而节省时间并减少出错的可能性。 Enterprise Architect使用灵活且完全可配置的模板系统实现模型驱动架构(MDA) 转换。模板充当机器的指令，机器将模型作为输入并将其转换为更解析的模型作为输出。输入可以是一个大而复杂的模型，也可以是一个单一的元素，一个输入模型可以转化为多种输出模型。

转换通常是单向的，采用平台模型(PIM) 并将其转换为一个或多个平台特定模型 (PSM)。这很有用的A很好的例子是系统必须在许多不同的关系数据库系统中实现。 A独立于平台的概念模型（PIM）可以转换为多个平台特定的模型，比如 Oracle、 MySQL和 SQLite。作为进一步的生产力提升，一旦生成输出模型，它们也可以转换为编程代码、数据库定义语言或模式。 Enterprise Architect自动创建可用于可视化输入模型中的元素如何转换为输出模型中的元素的可追溯性。

功能

功能	描述
变换元素 	了解如何在图表或浏览器窗口包中转换元素。
内置转换 	Enterprise Architect提供了许多支持多种目标语言的内置转换。每个都可以根据您的特定需求完全定制。
编辑变换模板 	了解如何调整转换模板以生成特定于您的系统的转换。
编写转换 	创建自己的转换所需的所有信息。

现成的转换

Enterprise Architect安装程序包括许多基本的内置转换，包括：

- PIM 至：
  - C #
  - C++
  - DDL表元素
  - EJB实体Bean
  - EJB 会话 Bean
  - Java
  - PHP
  - VB.Net

- XSD

- 数据模型到实体关系图 (ERD)
- 实体关系图 (ERD) 到数据模型
- 序列图到通讯图
- 通讯图到序列图
- Java模型to JUnit测试模型
- .NET模型转NUnit测试模型
- WSDL 接口模型到 WSDL

随着时间的推移，进一步的转换将变得可用，无论是内置的还是作为可从Sparx Systems网站下载的模块。

自定义转换

您可以使用Enterprise Architect的简单代码生成模板语言修改内置转换或定义自己的转换。这仅涉及编写模板以创建简单的中间源文件；系统读取源文件并将其绑定到新的 PSM。

变换关系

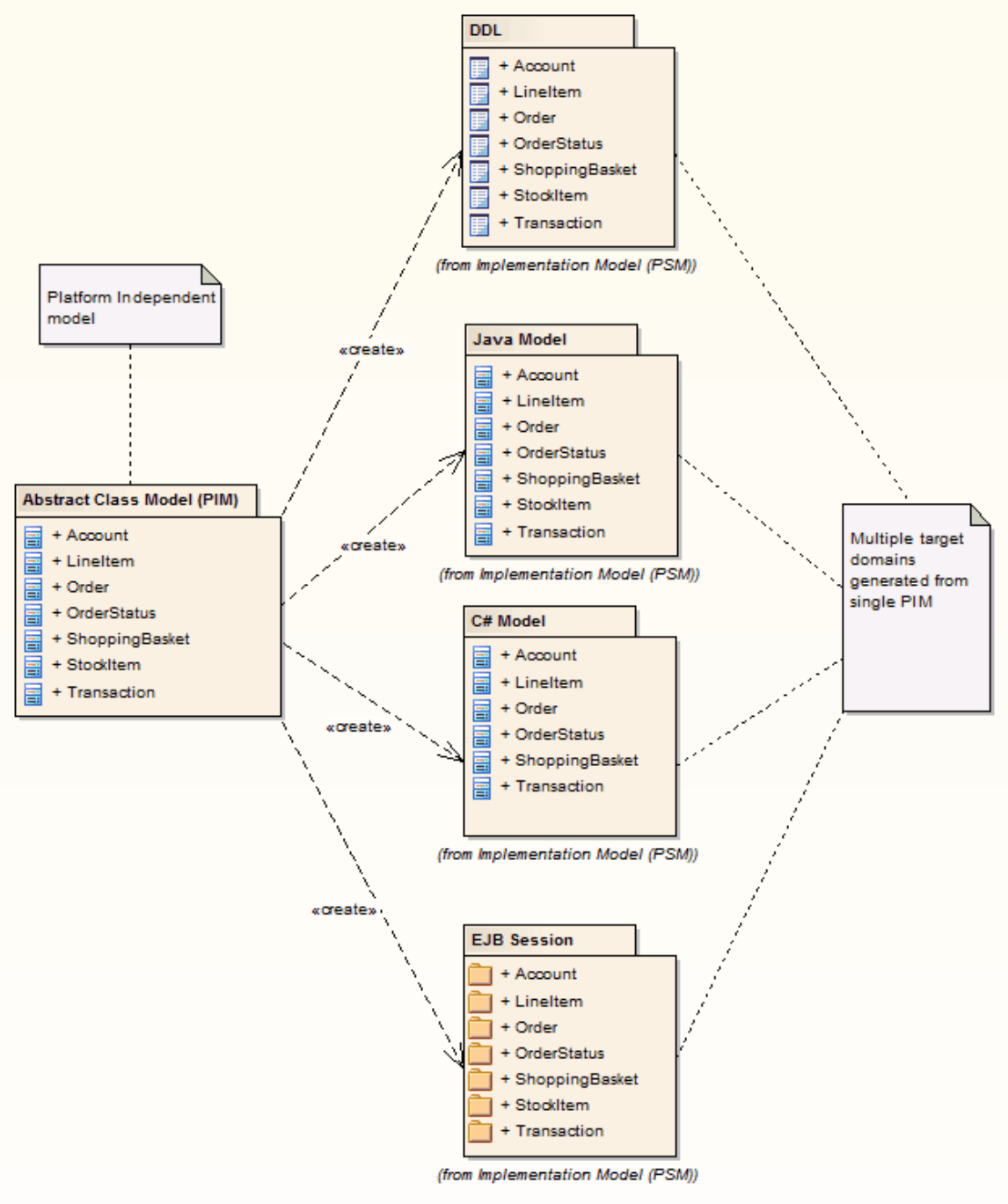
当您执行转换时，系统会在创建的每个 PSM 和原始 PIM 之间创建内部绑定（变换关系）。这是必不可少的，它提供了从 PIM 到 PSM 多次转发同步的能力，可以随时添加或删除特征；例如，向 PIM类添加新属性可以前向同步到数据模型中的新列。

您可以使用可修改可追溯性窗口观察包的变换依赖关系，检查对 PIM 的更改对每个生成的 PSM 中的相应元素的影响，或者验证 PSM 中所需的更改应该在PIM（也反映在其他 PSM 中）。变换依赖项是管理模型可追溯性的宝贵工具。

Enterprise Architect不会删除或覆盖最初不是由转换生成的任何元素特征；因此，您可以向元素添加新方法，而Enterprise Architect在正向生成过程中不会对它们进行操作。

一个示例的变换

此图突出显示了转换的工作原理以及它们如何显着提高您的生产力。



注记

- 如果您使用的是企业版、统一版或终极版，如果启用了安全性，您必须具有 变形包”访问权限才能对包的元素执行 MDA变换

变换元素

A转换是用户启动的函数，它启动将一个或多个平台模型模型(PIM) 元素转换为其相应的平台特定模型(PSM) 元素的过程。此过程根据变换模板中的规则进行。可以通过在浏览器窗口中选择一个包或在图表中选择一个元素来启动转换。

访问

功能区	设计>包>变换>变换选区
键盘快捷键	Ctrl+H（变换选定元素） Ctrl+Shift+H（变换选中包）

执行变换

选项	行动
元素	列出图表中选择或包含在包中的所有单个元素。任何一个： • 单击一个元素以将该元素包含在转换中 • 按住 Ctrl 并单击几个单独的元素以将它们包含在转换中，或者 • 按住 Shift 并单击块中的第一个和最后一个元素以将这些元素包含在转换中
全部	单击此按钮可选择列表中的所有元素以将它们包含在转换中。
没有任何	单击此按钮可取消选择列表中的所有元素。
包括子包	（如果您已选择转换包中的元素。） 选中此复选框以包括（在 元素”列表中并可能在转换中）来自所选包的子包的元素。
转型	选中要执行的每种转换类型的复选框。当您选择一个复选框时，将显示 浏览项目”对话框；定位并选择生成转换元素的目标包。 如果要更改选定的目标包，请单击包名称右侧的…按钮并从对话框中选择新包。
结果生成代码	选中此复选框以指定是否为以代码语言为目标的转换类自动生成代码。 如果你选择这个选项，你第一次转换到类时系统会提示你选择一个文件名来生成代码；随后的转换会自动生成该文件名的代码。
对结果执行转换	选中复选框以自动执行先前在目标类或类上完成的转换。
中介文件	如果要捕获中间语言文件（例如，调试它），请输入文件路径和名称，或单击…按钮并找到并选择文件。

总是写	选中此复选框以始终将中间文件写入磁盘。
现在写下来	单击此按钮可生成中间文件而不执行完整转换。
做变换	单击此按钮以执行转换。 当变换完成时， 模型变换”对话框关闭。
关	单击此按钮可关闭 模型变换”对话框而不执行变换。

注记

- 当对话框显示时，所有元素都被选中，并且之前从这些类中执行的所有转换都被选中
- 此过程不适用于序列图/通讯图转换，或通讯图/序列转换

链接转换

链接转换为执行转换提供了额外的灵活性和能力。例如，如果两个变换有一个共同的元素；您可以将这个元素分离成它自己的变换，然后从公共点执行原始变换。分离的变换甚至可以产生有用的模型本身。

您可以通过选择 模型变换"对话框中的 对结果执行变换"复选框来链接变换，以便已经对目标类执行过的变换在下次变换到该类时自动执行。

内置转换

Enterprise Architect提供了一组丰富的内置的、常用的转换。这些将证明对从领域建模到代码工程的各种学科都有用。转换模型的功能是一种实用的生产力工具，预计建模者将希望创建自己的自定义转换。内置的转换提供了有用的示例，是建模者的宝贵参考。

内置转换

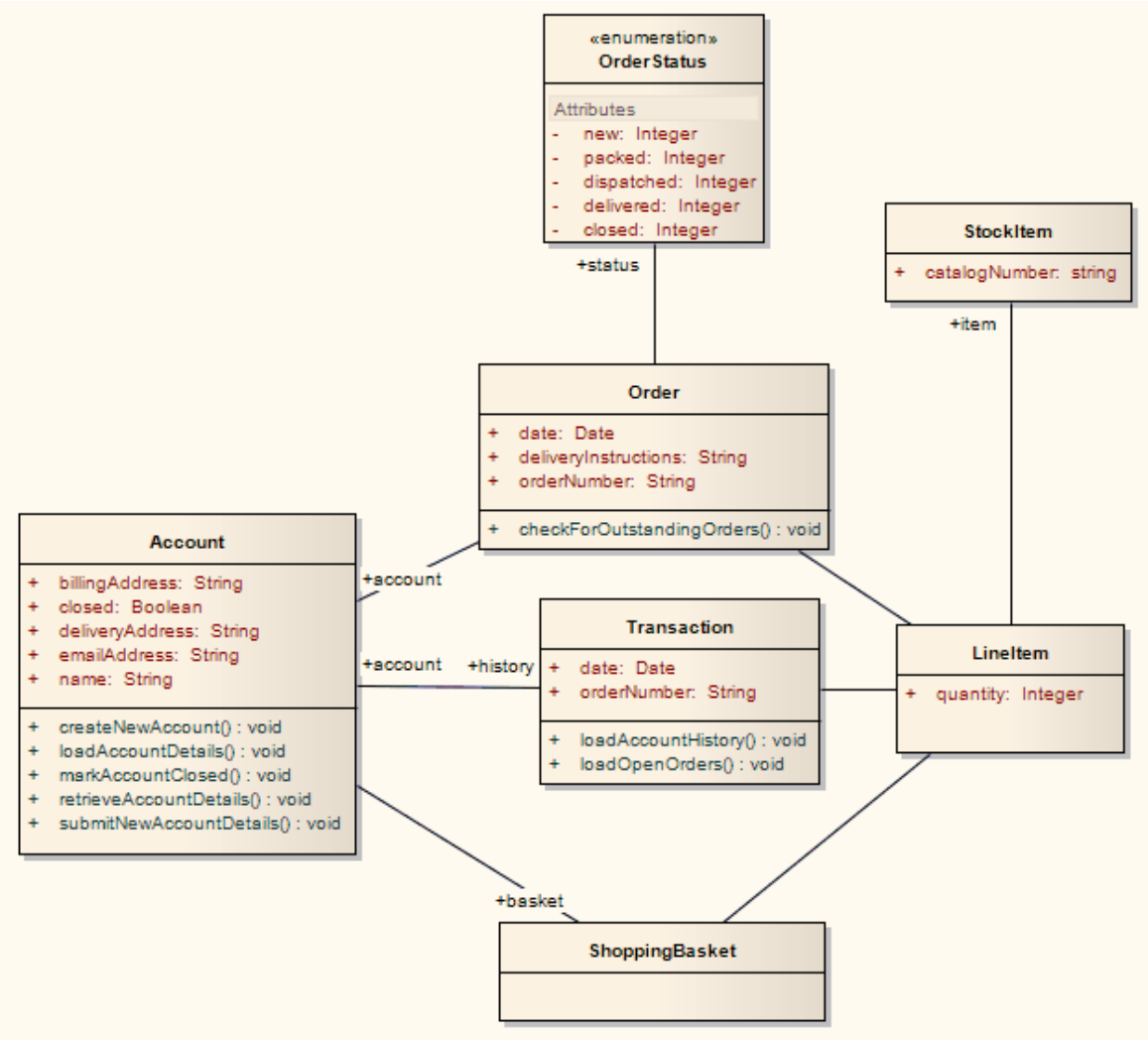
变换	转换
C #	平台模型（PIM）元素到特定语言的 C#类元素。
C++	PIM 元素到特定于语言的 C++类元素。
数据定义语言	A逻辑模型到一个以默认数据库类型为目标的数据模型，为 DDL 生成做好准备。
实体关系图表数据模型	将 ERD 逻辑模型转换为以默认数据库类型为目标的数据模型，为 DDL 生成做好准备。
数据模型实体关系图表	数据A到模型模型
EJB 会话 Bean	EJB 会话元素A单个类元素。
EJB实体Bean	EJB 实体的元素A单个类元素。
Java	PIM 元素到特定于语言的Java类元素。
JUnit	现有的具有公共方法的Java类元素到具有每个公共方法的测试方法的类，以及适当设置测试所需的方法。
单元	现有的.NET兼容类与公共方法转换为具有每个公共方法的测试方法的类，以及适当设置测试所需的方法。
PHP	PIM 元素到特定于语言的 PHP类元素。
序列/通讯图表	一个打开的序列图中的所有元素和消息都匹配到一个通讯图中的元素和消息中，反之亦然。
VB.Net	PIM 元素到特定于语言的 VB.Net类元素。
WSDL	WSDL接口A简单模型到扩展模型，适合生成。
XSD	PIM 元素到用于 XML 元素的UML配置文件，作为创建XML Schema的中间步骤。

C#变换

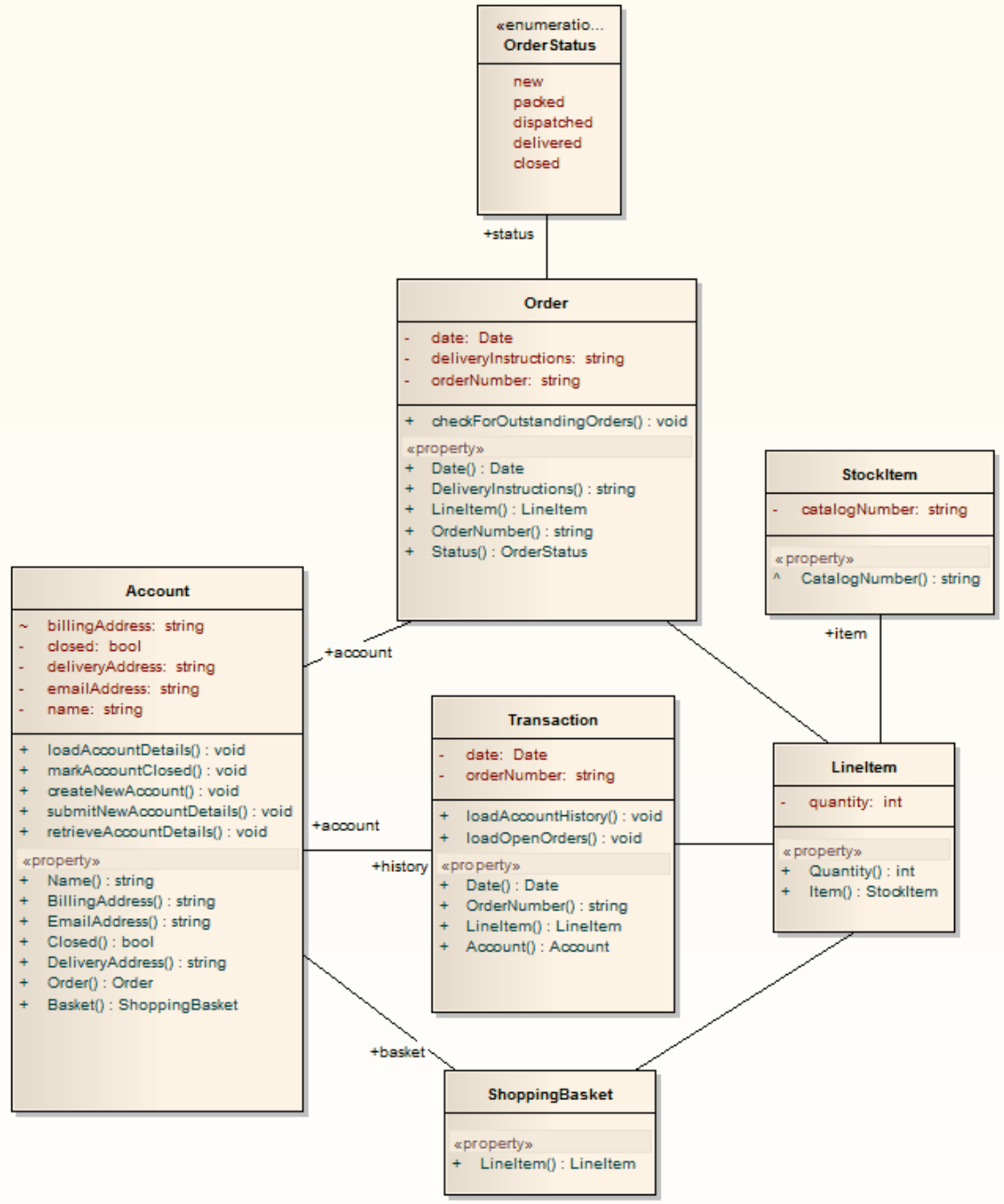
C# 转换将平台模型(PIM)元素类型转换为 C# 特定的属性类元素类型，并根据您设置的用于从 C# 属性创建的系统选项创建封装（在 首选项"的 C# 规范"页面上）'对话框）。

示例

PIM 元素



改造后成为PSM元素

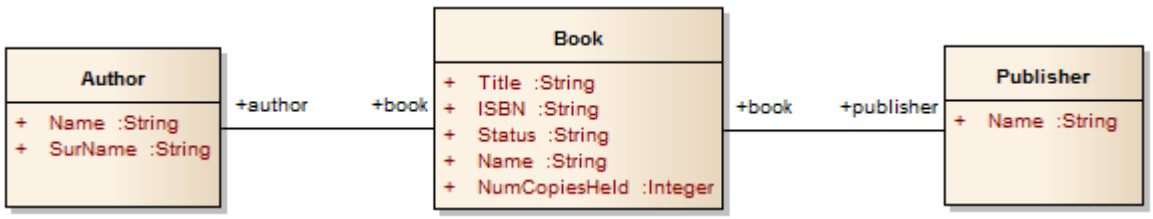

«enumeratio...
OrderStatus
new
packed
dispatched
delivered
closed
+status
Order
- date: Date
- deliveryInstructions: string
- orderNumber: string
+ checkForOutstandingOrders() : void
«property»
+ Date() : Date
+ DeliveryInstructions() : string
+ LineItem() : LineItem
+ OrderNumber() : string
+ Status() : OrderStatus
StockItem
- catalogNumber: string
«property»
^ CatalogNumber() : string
+item
Account
~ billingAddress: string
- closed: bool
- deliveryAddress: string
- emailAddress: string
- name: string
+ loadAccountDetails() : void
+ markAccountClosed() : void
+ createNewAccount() : void
+ submitNewAccountDetails() : void
+ retrieveAccountDetails() : void
«property»
+ Name() : string
+ BillingAddress() : string
+ EmailAddress() : string
+ Closed() : bool
+ DeliveryAddress() : string
+ Order() : Order
+ Basket() : ShoppingBasket
+account
+account
+history
Transaction
- date: Date
- orderNumber: string
+ loadAccountHistory() : void
+ loadOpenOrders() : void
«property»
+ Date() : Date
+ OrderNumber() : string
+ LineItem() : LineItem
+ Account() : Account
LineItem
- quantity: int
«property»
+ Quantity() : int
+ Item() : StockItem
+basket
ShoppingBasket
«property»
+ LineItem() : LineItem

C++变换

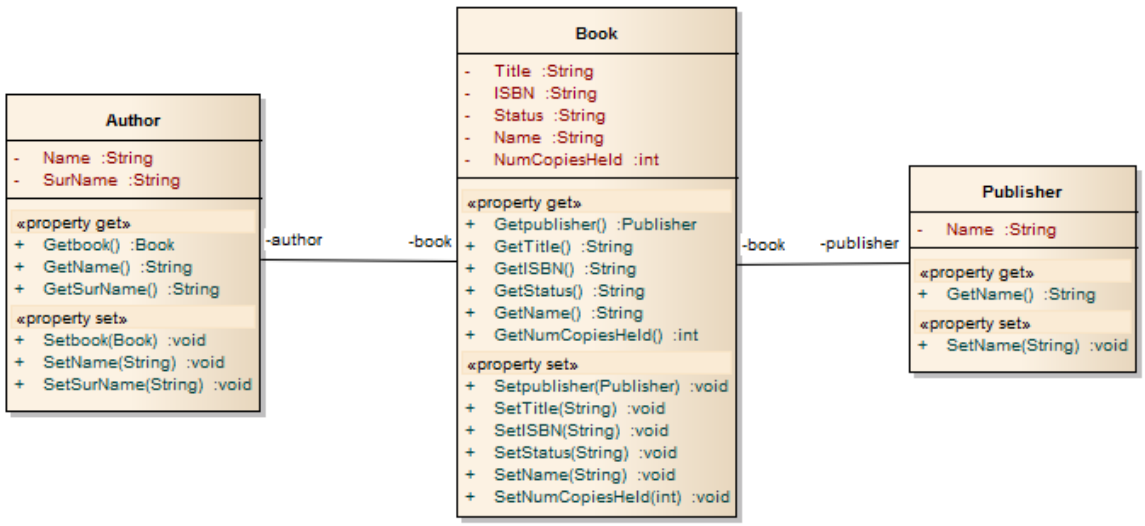
C++ 转换将平台模型(PIM)元素类型转换为 C++ 特定类元素类型，并根据您设置的用于从 C++ 属性创建属性的选项（在 C++ 规范”页面上）创建封装（生成 getter 和 setter） 'Preferences' 对话框）。注记PIM 中的公共属性被转换为 PSM 中的私有属性。接口上的所有操作都转换为等效类上的纯虚方法。

示例

PIM 元素



改造后成为PSM元素



数据模型到 ERD变换

数据模型到实体关系图 (ERD) 转换从数据模型模型它是 ERD 到模型转换的逆过程。此转换使用并演示了中间语言对许多特定于数据库的概念的支持。

支持的概念

概念	影响
实体	一对一映射到库表元素上。
属性	一对一映射到列。
首要的关键	派生自 PrimaryKey 类型的列。

注记

- 有时您可能想要限制菱形关系连接器的伸展；只需选择一个关系连接器，右键单击以显示上下文菜单，然后选择 Bend Line at Cursor″选项

DDL变换

DDL 转换将逻辑模型转换为结构化的数据模型，以符合支持的 DBMS 之一。目标数据库类型由模型中设置为默认数据库的 DBMS 确定（参见数据库数据类型帮助主题， 设置为默认值"选项）。然后可以使用数据模型自动生成运行语句以在系统支持的数据库产品之一中运行。

DDL 转换使用并演示了中间语言对许多特定于数据库的概念的支持。

概念

概念	影响
库表	将一对一映射到类元素上。 转换支持 多对多"关系，创建汇合库表。
柱子	一对一映射到属性。
首要的关键	列出所有涉及的列，以便它们存在于类中，并为它们创建主键方法。
外键	A特殊的连接器，其中源和目标部分列出了所有涉及的列，以便： ● 列存在 ● 目标类中A匹配的主键，并且 ● 转换创建适当的外键

MDG 技术自定义默认映射

针对新的、用户定义的 DBMS 的 DDL 转换需要MDG 技术来将 PIM 数据类型映射到新的目标 DBMS。

为此，创建一个名为 "UserDBMS Types.xml"的MDG 技术.xml 文件，将 UserDBMS 替换为添加的 DBMS 的名称。将文件放在 EA\MDGTechnologies 文件夹中。 MDG 技术文件的内容应具有以下结构：

< MDG .技术版本1 "1.0">

<Documentation id="UserdataTypes" name="Userdata Types" version=" 1 .0" notes="类型for UserDBMS"/>

<代码模块>

<CodeModule 语言="用户数据" 注释="">

<代码选项>

<CodeOption name="DBTypeMapping-bigint">BIGINT</CodeOption>

<CodeOption name="DBTypeMapping-blob">BLOB</CodeOption>

<CodeOption name="DBTypeMapping-boolean">TINYINT</CodeOption>

<CodeOption name="DBTypeMapping-text">CLOB</CodeOption>

...

</代码选项>

</代码模块>

</代码模块>

</ MDG .技术>

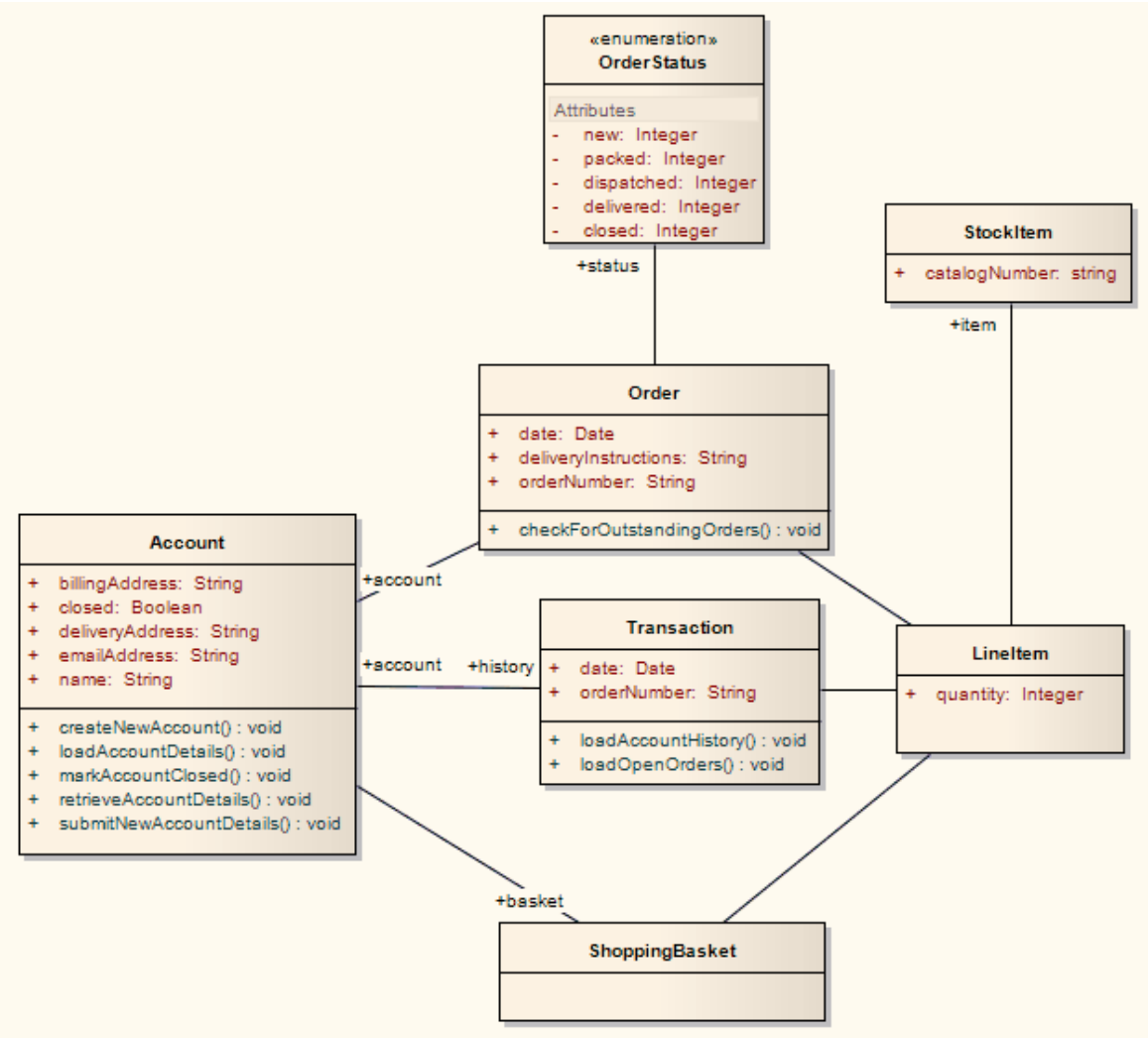
例如，“text”是映射到新 DBMS 的 “CLOB”数据类型的公共类型（如 数据库数据类型”对话框中所列）。

注记

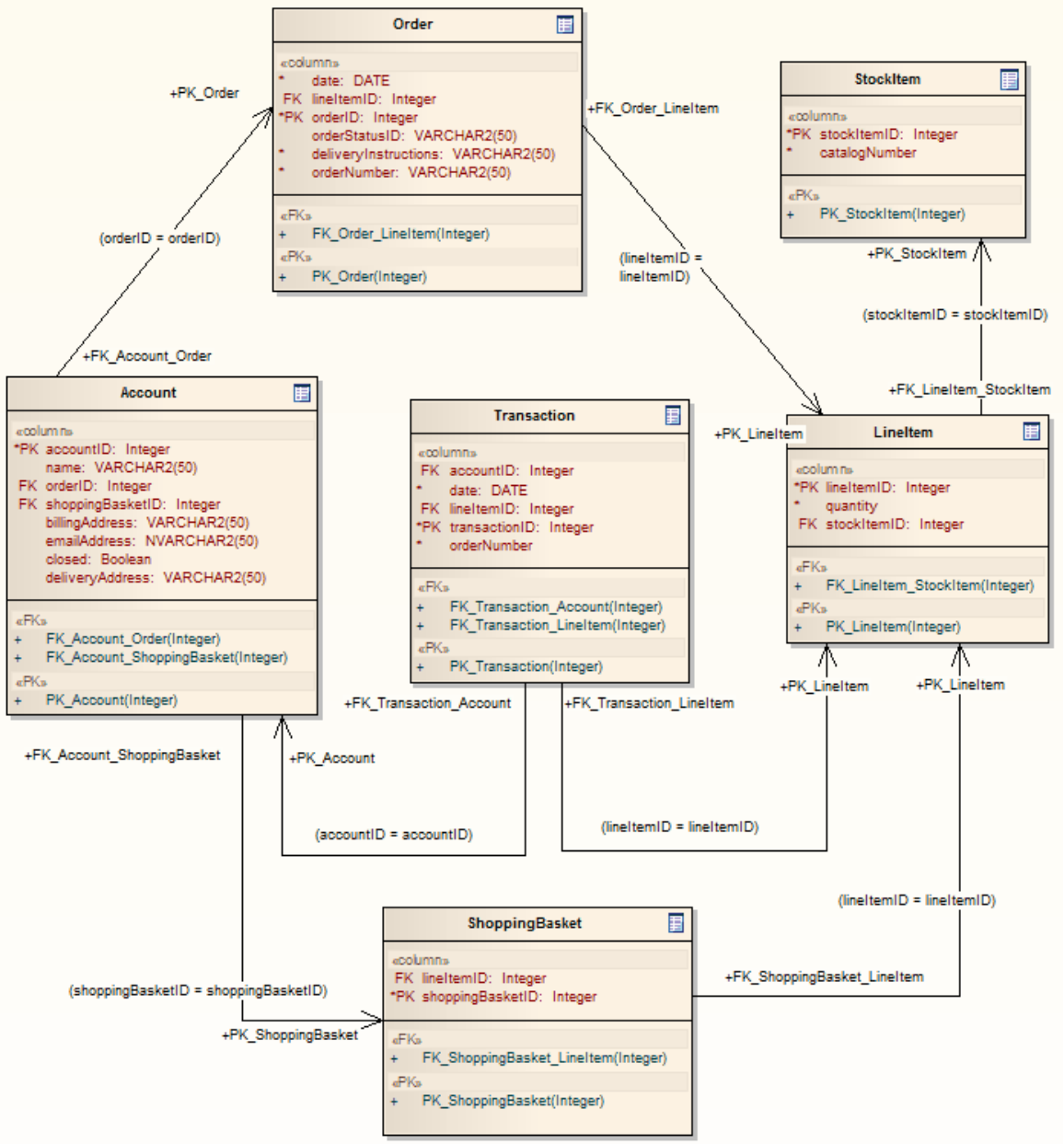
- 您可以在转换后定义逻辑模型中未描述的 DBMS 特定方面，例如存储过程、触发器、视图和选择约束；查看*Physical Data*模型帮助

示例

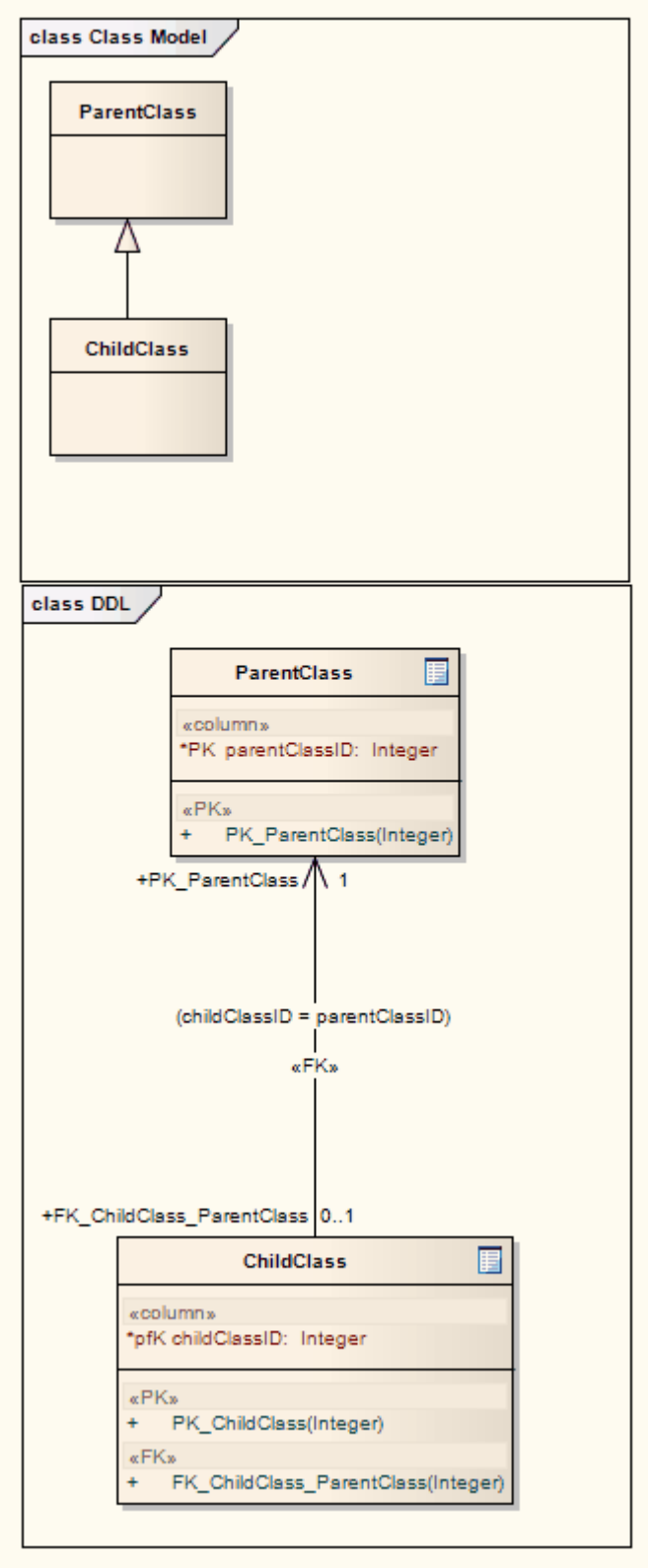
PIM 元素



改造后成为PSM元素



泛化是通过为子元素提供父元素的外键来处理的，如图所示。不支持向下复制继承。


class Class Model
ParentClass
ChildClass
class DDL
ParentClass
«column»
*PK parentClassID: Integer
«PK»
+ PK_ParentClass(Integer)
+PK_ParentClass
1
(childClassID = parentClassID)
«FK»
+FK_ChildClass_ParentClass
0..1
ChildClass
«column»
*pfK childClassID: Integer
«PK»
+ PK_ChildClass(Integer)
«FK»
+ FK_ChildClass_ParentClass(Integer)

EJB 转换

EJB 会话 Bean 和 EJB实体Bean 转换减少了生成企业Java Bean 内部所需的工作。因此，您可以聚焦于更高抽象级别的建模。

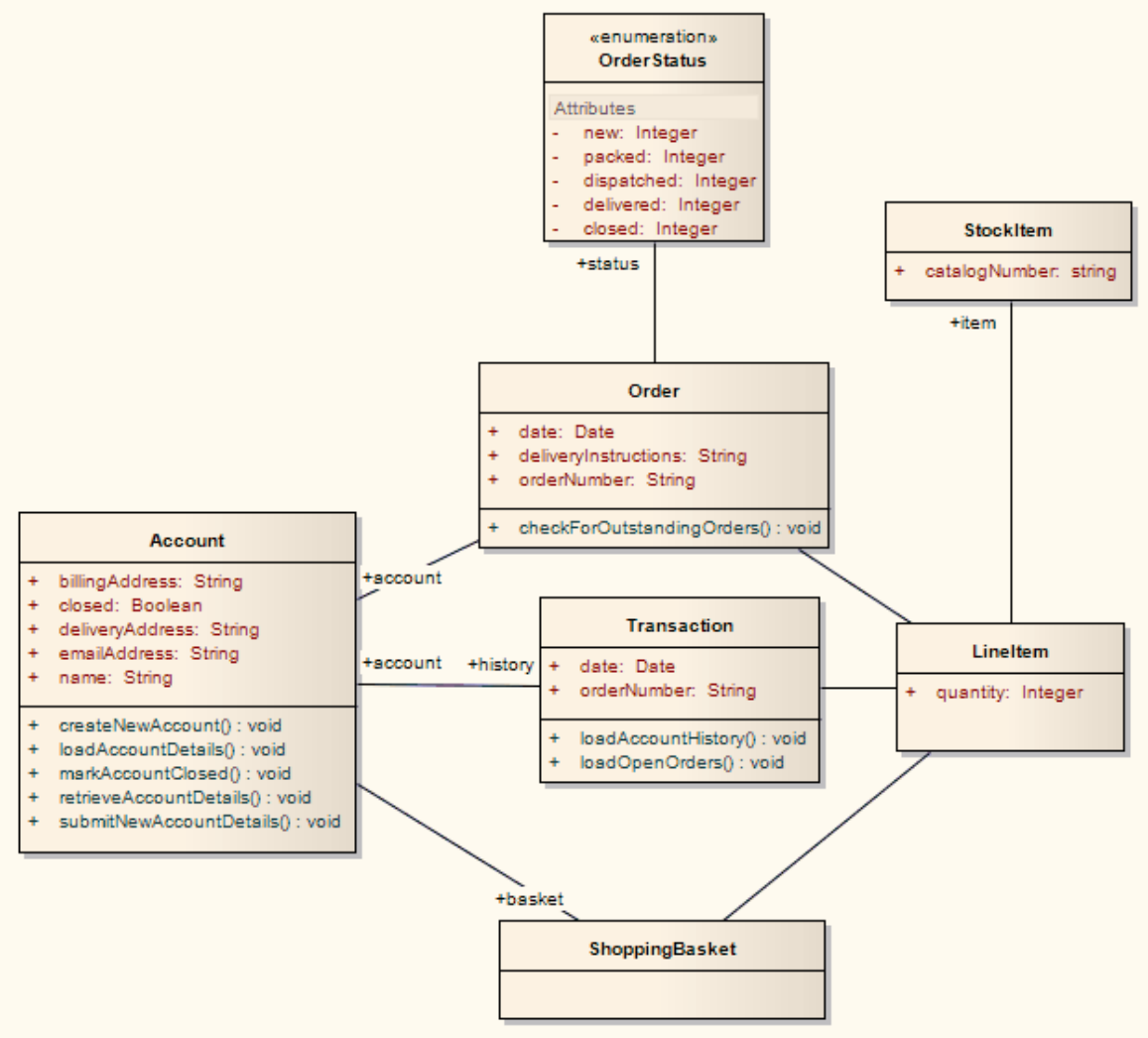
转型

两种转换还生成一个包含部署描述符元素的 META-INF包。

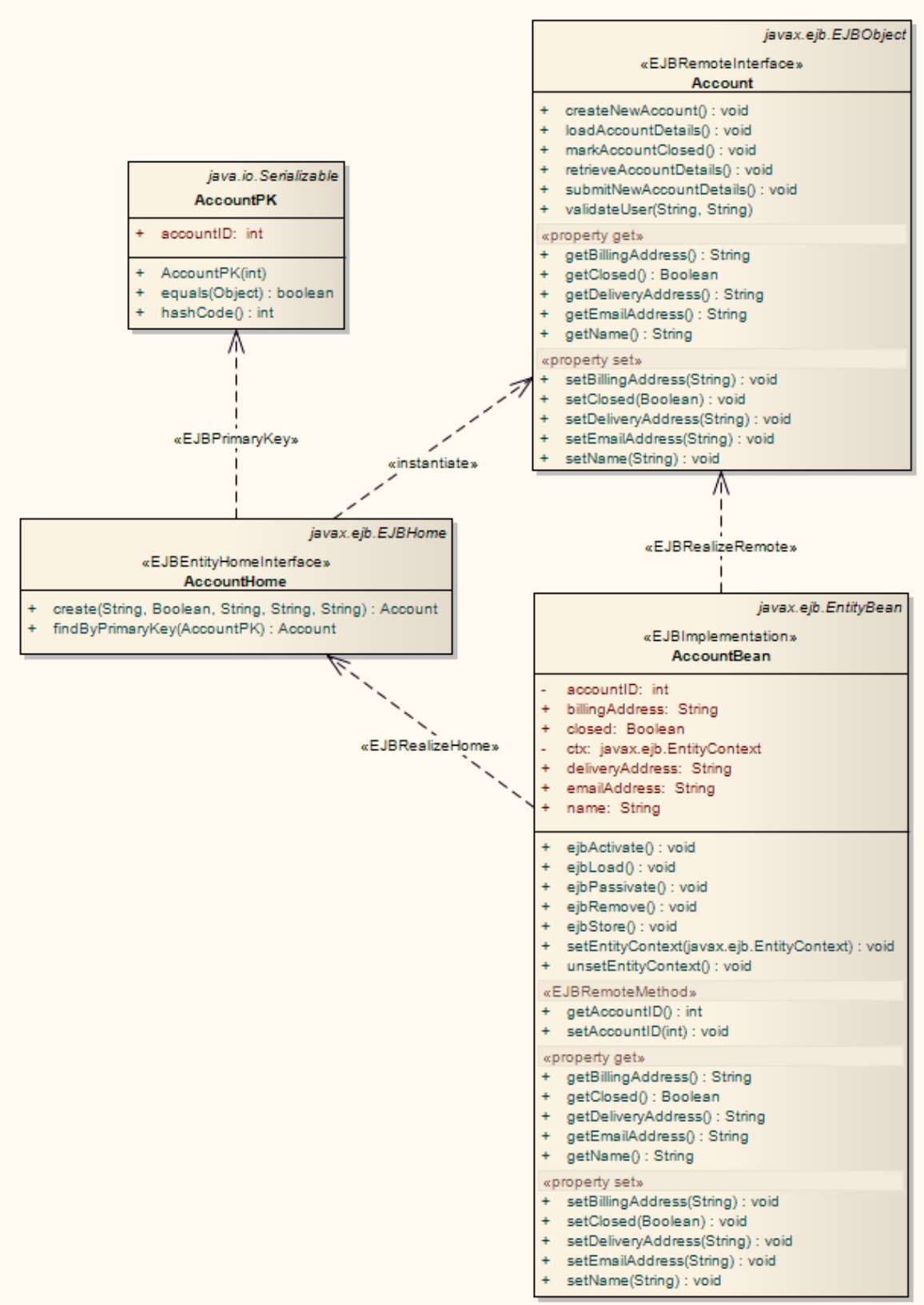
变换	细节
EJB 会话 Bean	此转换将单个类元素（包含 javax.ejb.*包生成代码所需的属性、操作和引用）转换为 • 一个实现类元素 • A家接口元素 • A远程接口元素
EJB实体Bean	此转换将单个类元素（包含 javax.ejb.*包生成代码所需的属性、操作和引用）转换为： • 一个实现类元素 • A家接口元素 • A远程接口元素 • A主键元素

示例

PIM 元素



转换后生成一组实体Bean，其中每个实体都采用以下形式（对于账户类）：


javax.ejb.EJBObject
«EJBRemoteInterface»
Account
+ createNewAccount() : void
+ loadAccountDetails() : void
+ markAccountClosed() : void
+ retrieveAccountDetails() : void
+ submitNewAccountDetails() : void
+ validateUser(String, String)
«property get»
+ getBillingAddress() : String
+ getClosed() : Boolean
+ getDeliveryAddress() : String
+ getEmailAddress() : String
+ getName() : String
«property set»
+ setBillingAddress(String) : void
+ setClosed(Boolean) : void
+ setDeliveryAddress(String) : void
+ setEmailAddress(String) : void
+ setName(String) : void
java.io.Serializable
AccountPK
+ accountID: int
+ AccountPK(int)
+ equals(Object) : boolean
+ hashCode() : int
«EJBPrimaryKey»
«instantiate»
javax.ejb.EJBHome
«EJBEntityHomeInterface»
AccountHome
+ create(String, Boolean, String, String, String) : Account
+ findByPrimaryKey(AccountPK) : Account
«EJBRealizeRemote»
«EJBRealizeHome»
javax.ejb.EntityBean
«EJBImplementation»
AccountBean
- accountID: int
+ billingAddress: String
+ closed: Boolean
- ctx: javax.ejb.EntityContext
+ deliveryAddress: String
+ emailAddress: String
+ name: String
+ ejbActivate() : void
+ ejbLoad() : void
+ ejbPassivate() : void
+ ejbRemove() : void
+ ejbStore() : void
+ setEntityContext(javax.ejb.EntityContext) : void
+ unsetEntityContext() : void
«EJBRemoteMethod»
+ getAccountID() : int
+ setAccountID(int) : void
«property get»
+ getBillingAddress() : String
+ getClosed() : Boolean
+ getDeliveryAddress() : String
+ getEmailAddress() : String
+ getName() : String
«property set»
+ setBillingAddress(String) : void
+ setClosed(Boolean) : void
+ setDeliveryAddress(String) : void
+ setEmailAddress(String) : void
+ setName(String) : void

ERD 到数据模型变换

实体关系图（图表）到数据模型的转换将 ERD 逻辑模型转换为针对默认数据库类型的数据模型，为生成运行语句以在系统支持的数据库产品之一中运行做好准备。在进行转换之前，为每个属性定义通用数据类型并选择一个数据库类型作为默认数据库。然后，您可以自动生成数据建模图。

该转换使用并演示了对许多特定于数据库的概念的中间语言的支持。

概念

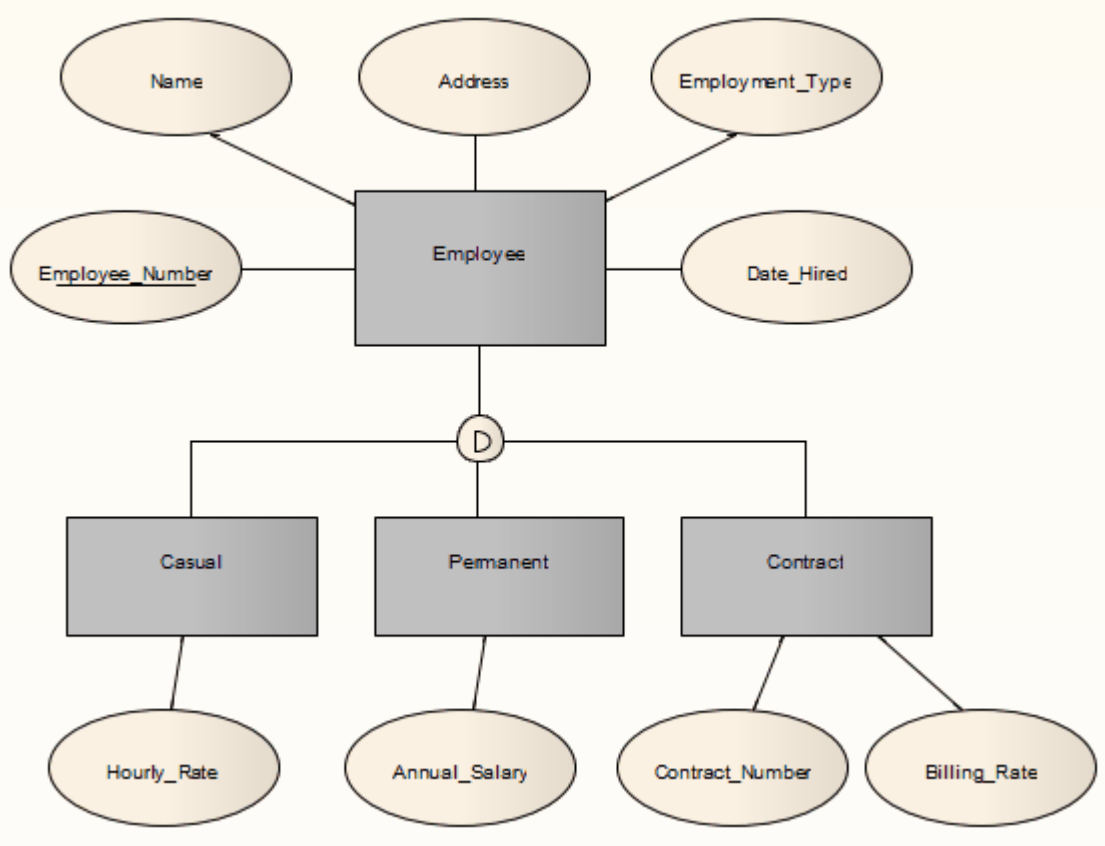
概念	定义
库表	将一对一映射到类元素上。
柱子	一对一映射到属性。
首要的关键	列出所有涉及的列，以便它们存在于类中，并为它们创建主键方法。
外键	A特殊的连接器，其中源和目标部分列出了所有涉及的列，以便： ● 列存在 ● 目标类中A匹配的主键，并且 ● 转换创建适当的外键

概括

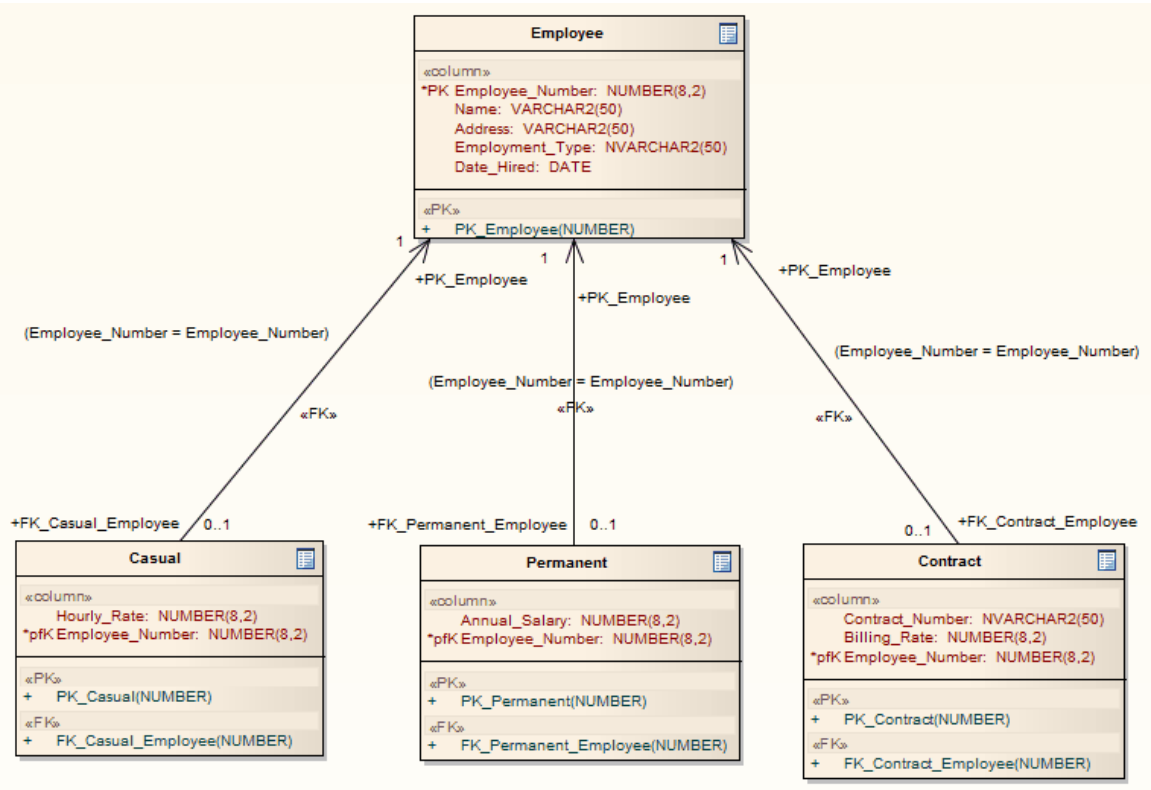
概括技术可以处理，如图所示。注记目前仅支持两个级别的向下复制继承。

示例

ERD 元素


Name
Address
Employment_Type
Employee_Number
Employee
Date_Hired
D
Casual
Permanent
Contract
Hourly_Rate
Annual_Salary
Contract_Number
Billing_Rate

改造后成为模型


Employee
«column»
*PK Employee_Number: NUMBER(8,2)
Name: VARCHAR2(50)
Address: VARCHAR2(50)
Employment_Type: NVARCHAR2(50)
Date_Hired: DATE
«PK»
+ PK_Employee(NUMBER)
1
1
1
+PK_Employee
+PK_Employee
+PK_Employee
(Employee_Number = Employee_Number)
(Employee_Number = Employee_Number)
(Employee_Number = Employee_Number)
«FK»
«FK»
«FK»
+FK_Casual_Employee
0..1
+FK_Permanent_Employee
0..1
0..1
+FK_Contract_Employee
Casual
«column»
Hourly_Rate: NUMBER(8,2)
*pfK Employee_Number: NUMBER(8,2)
«PK»
+ PK_Casual(NUMBER)
«FK»
+ FK_Casual_Employee(NUMBER)
Permanent
«column»
Annual_Salary: NUMBER(8,2)
*pfK Employee_Number: NUMBER(8,2)
«PK»
+ PK_Permanent(NUMBER)
«FK»
+ FK_Permanent_Employee(NUMBER)
Contract
«column»
Contract_Number: NVARCHAR2(50)
Billing_Rate: NUMBER(8,2)
*pfK Employee_Number: NUMBER(8,2)
«PK»
+ PK_Contract(NUMBER)
«FK»
+ FK_Contract_Employee(NUMBER)

注记

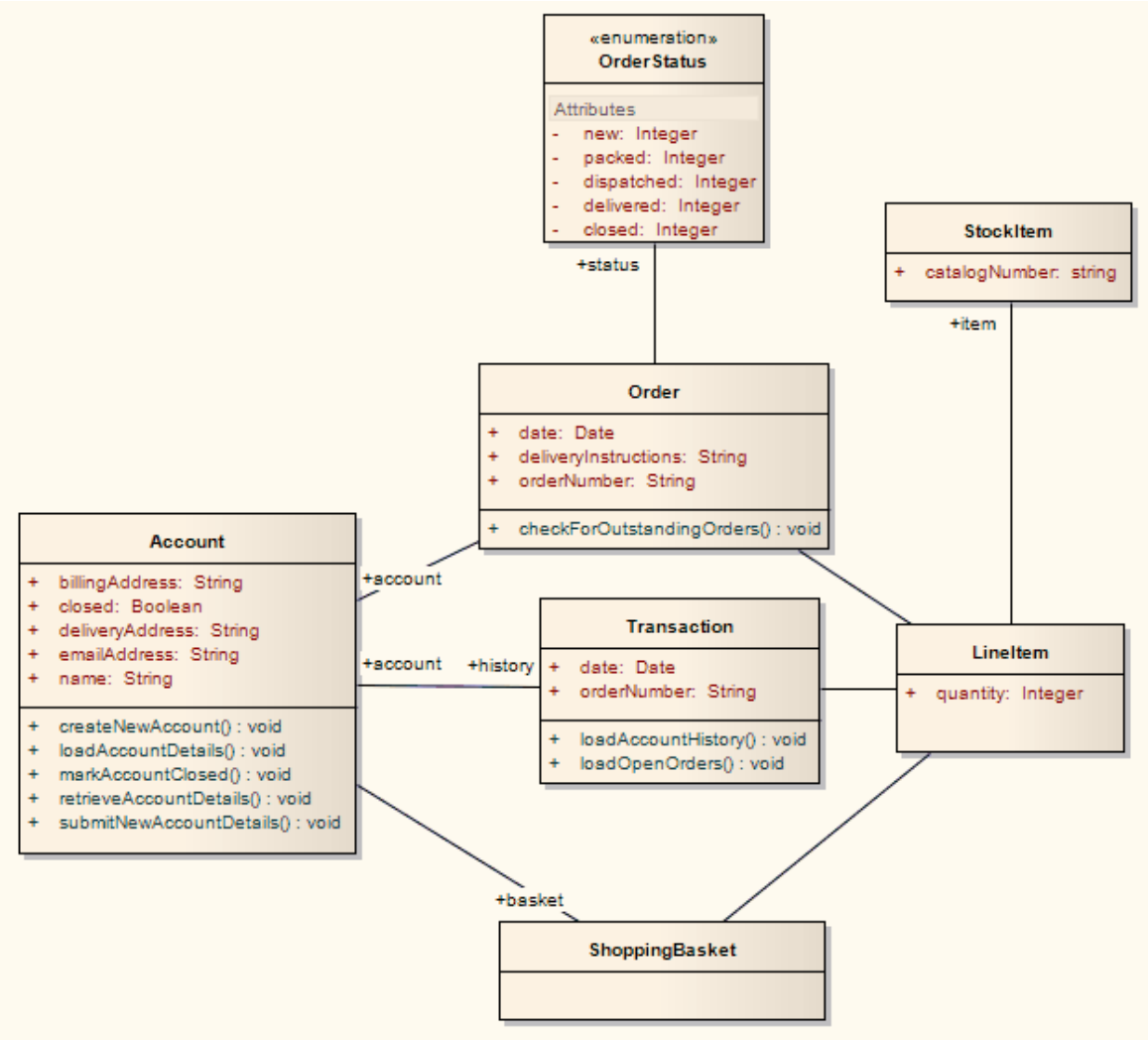
- 有时您可能会回到 ERD，进行一些更改，然后需要进行另一次转换；在这种情况下，为了达到更好的效果，在进行下一个变换之前，总是删除上一个变换包

Java变换

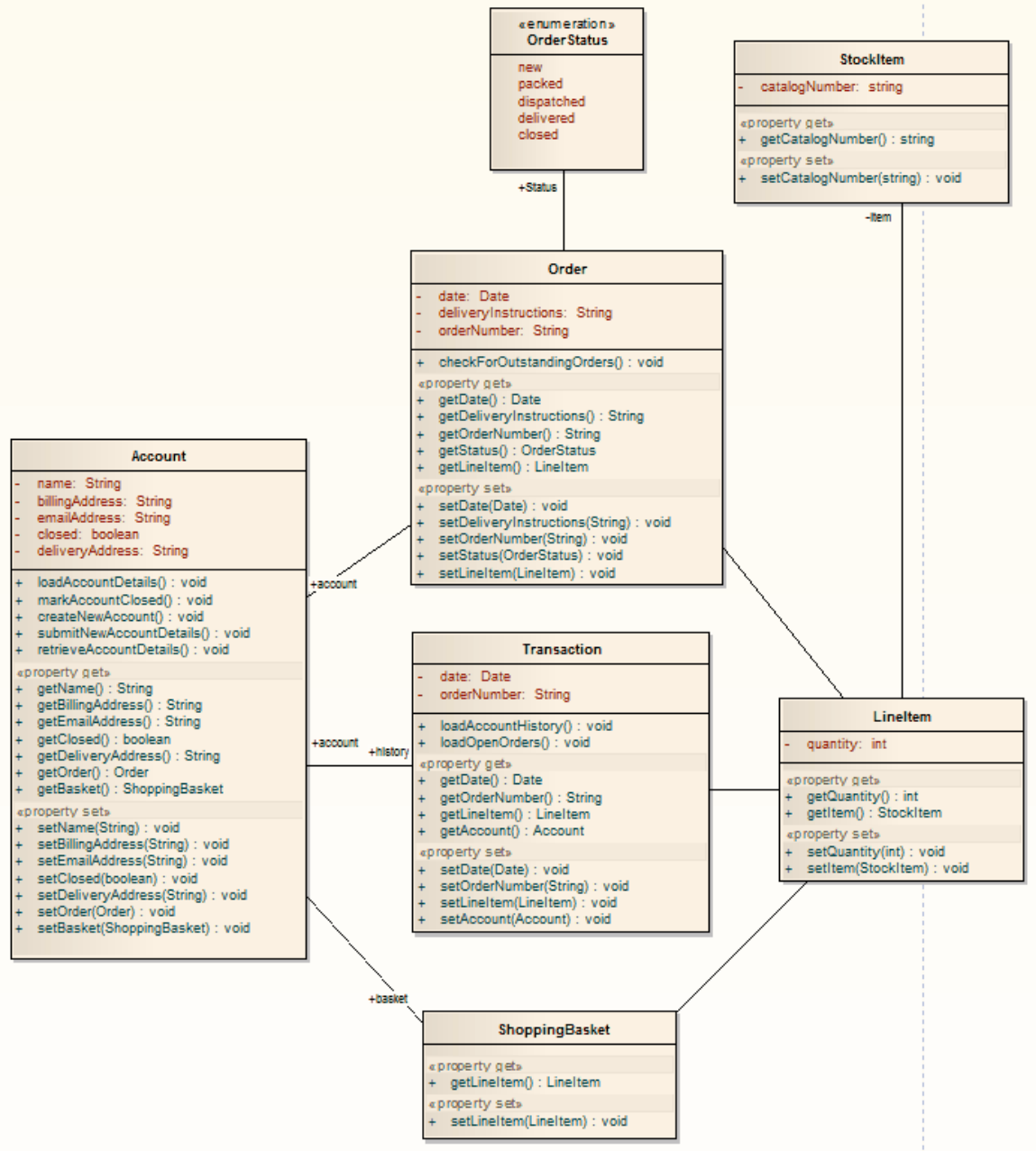
Java转换将平台模型(PIM)元素类型转换为特定于 Java 的类元素类型，并根据您设置的用于从Java属性创建属性的选项创建封装（生成 getter 和 setter）（在 ' Java Specifications ' 首选项对话框的页面）。请注记，PIM 中的公共属性将转换为 PSM 中的私有属性。接口中的所有操作都转化为纯虚方法。

示例

PIM 元素



改造后成为PSM元素

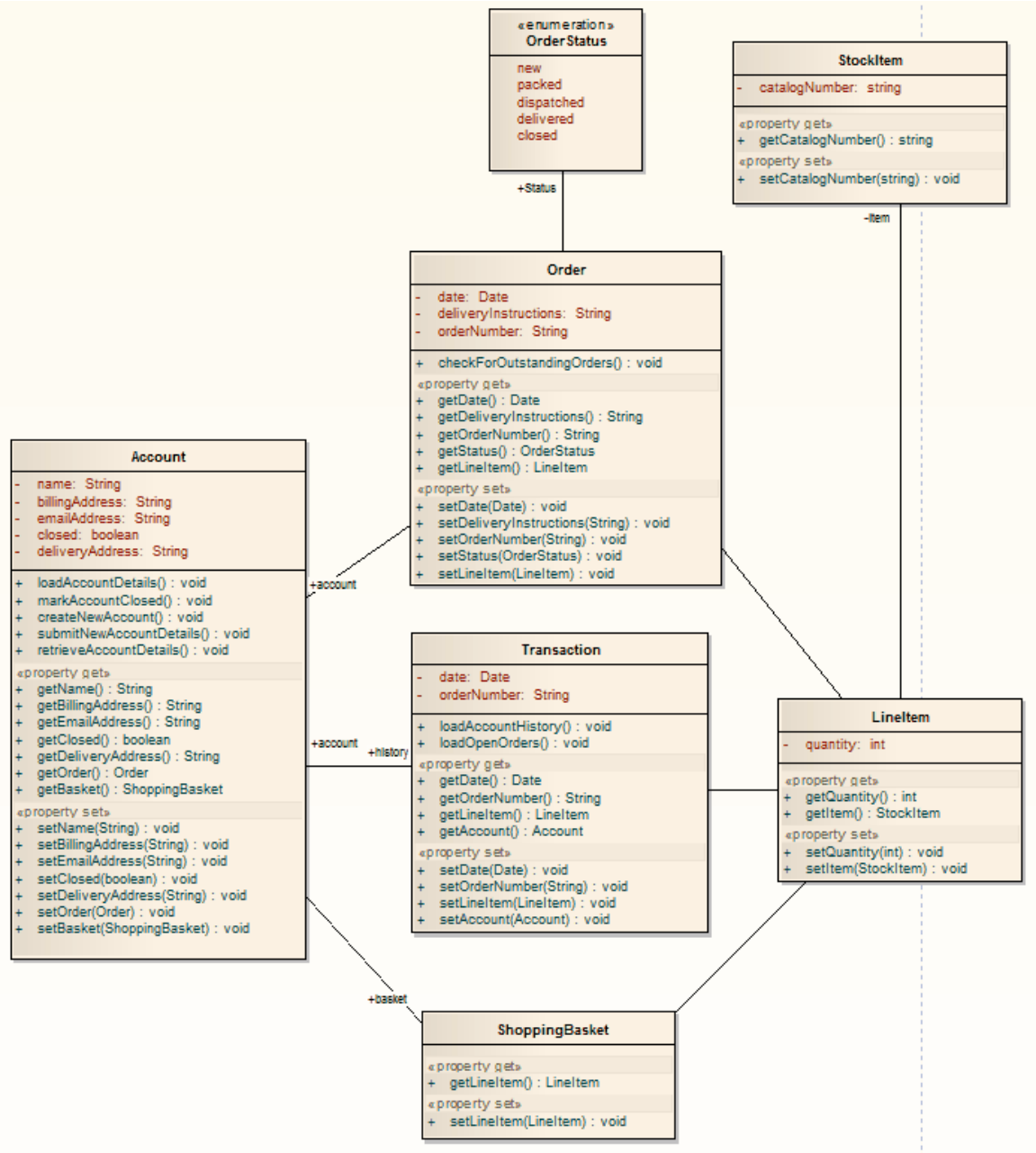

«enumeration»
OrderStatus
new
packed
dispatched
delivered
closed
+Status
StockItem
- catalogNumber: string
«property get»
+ getCatalogNumber() : string
«property set»
+ setCatalogNumber(string) : void
-Item
Order
- date: Date
- deliveryInstructions: String
- orderNumber: String
+ checkForOutstandingOrders() : void
«property get»
+ getDate() : Date
+ getDeliveryInstructions() : String
+ getOrderNumber() : String
+ getStatus() : OrderStatus
+ getLineItem() : LineItem
«property set»
+ setDate(Date) : void
+ setDeliveryInstructions(String) : void
+ setOrderNumber(String) : void
+ setStatus(OrderStatus) : void
+ setLineItem(LineItem) : void
Account
- name: String
- billingAddress: String
- emailAddress: String
- closed: boolean
- deliveryAddress: String
+ loadAccountDetails() : void
+ markAccountClosed() : void
+ createNewAccount() : void
+ submitNewAccountDetails() : void
+ retrieveAccountDetails() : void
«property get»
+ getName() : String
+ getBillingAddress() : String
+ getEmailAddress() : String
+ getClosed() : boolean
+ getDeliveryAddress() : String
+ getOrder() : Order
+ getBasket() : ShoppingBasket
«property set»
+ setName(String) : void
+ setBillingAddress(String) : void
+ setEmailAddress(String) : void
+ setClosed(boolean) : void
+ setDeliveryAddress(String) : void
+ setOrder(Order) : void
+ setBasket(ShoppingBasket) : void
+account
Transaction
- date: Date
- orderNumber: String
+ loadAccountHistory() : void
+ loadOpenOrders() : void
«property get»
+ getDate() : Date
+ getOrderNumber() : String
+ getLineItem() : LineItem
+ getAccount() : Account
«property set»
+ setDate(Date) : void
+ setOrderNumber(String) : void
+ setLineItem(LineItem) : void
+ setAccount(Account) : void
+account
+history
LineItem
- quantity: int
«property get»
+ getQuantity() : int
+ getItem() : StockItem
«property set»
+ setQuantity(int) : void
+ setItem(StockItem) : void
+basket
ShoppingBasket
«property get»
+ getLineItem() : LineItem
«property set»
+ setLineItem(LineItem) : void

JUnit变换

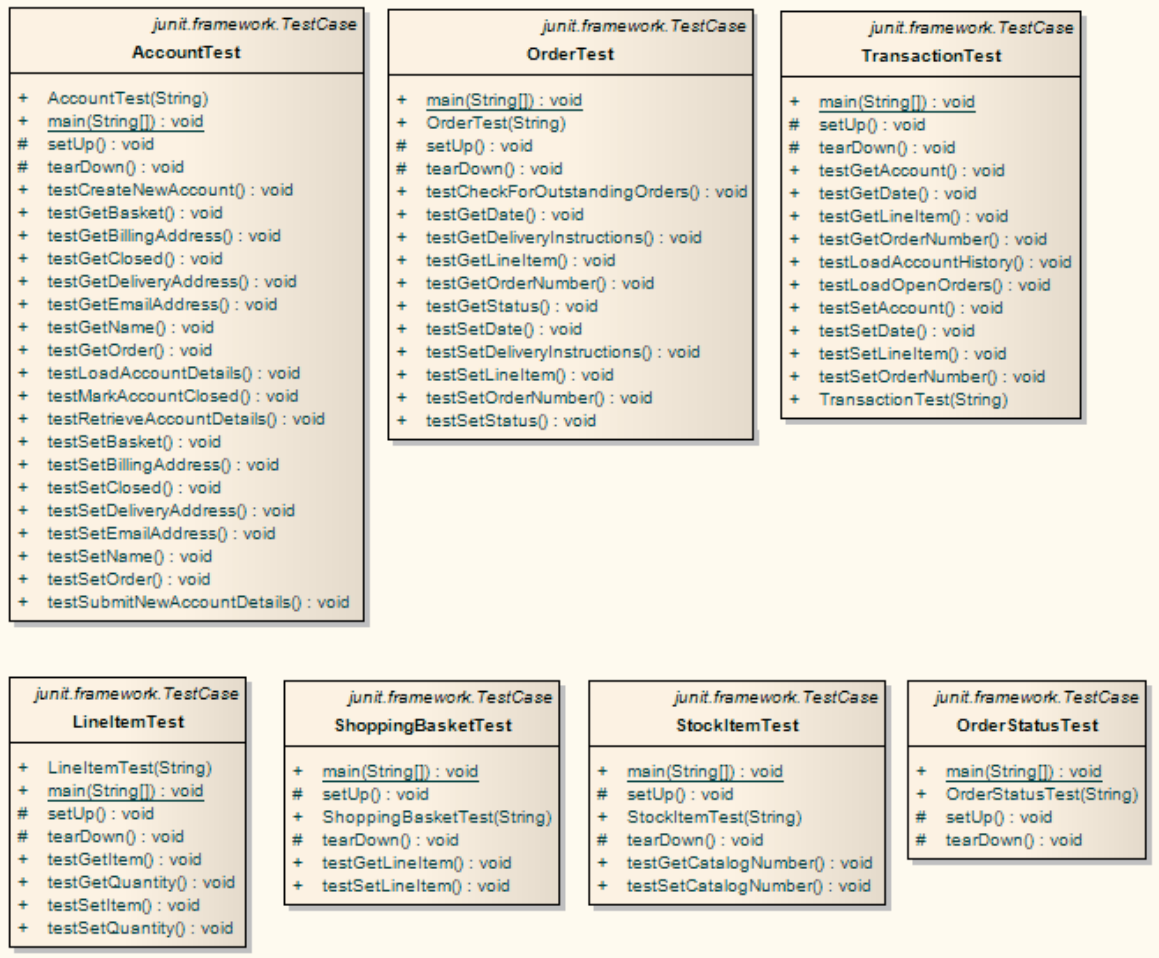
JUnit 转换将现有的具有公共方法的Java类转换为具有每个公共方法的测试方法的类。然后可以生成生成的类，并通过运行填写和运行测试。

示例

Java模型元素（最初由PIM转换而来）



改造后成为PSM元素



注记

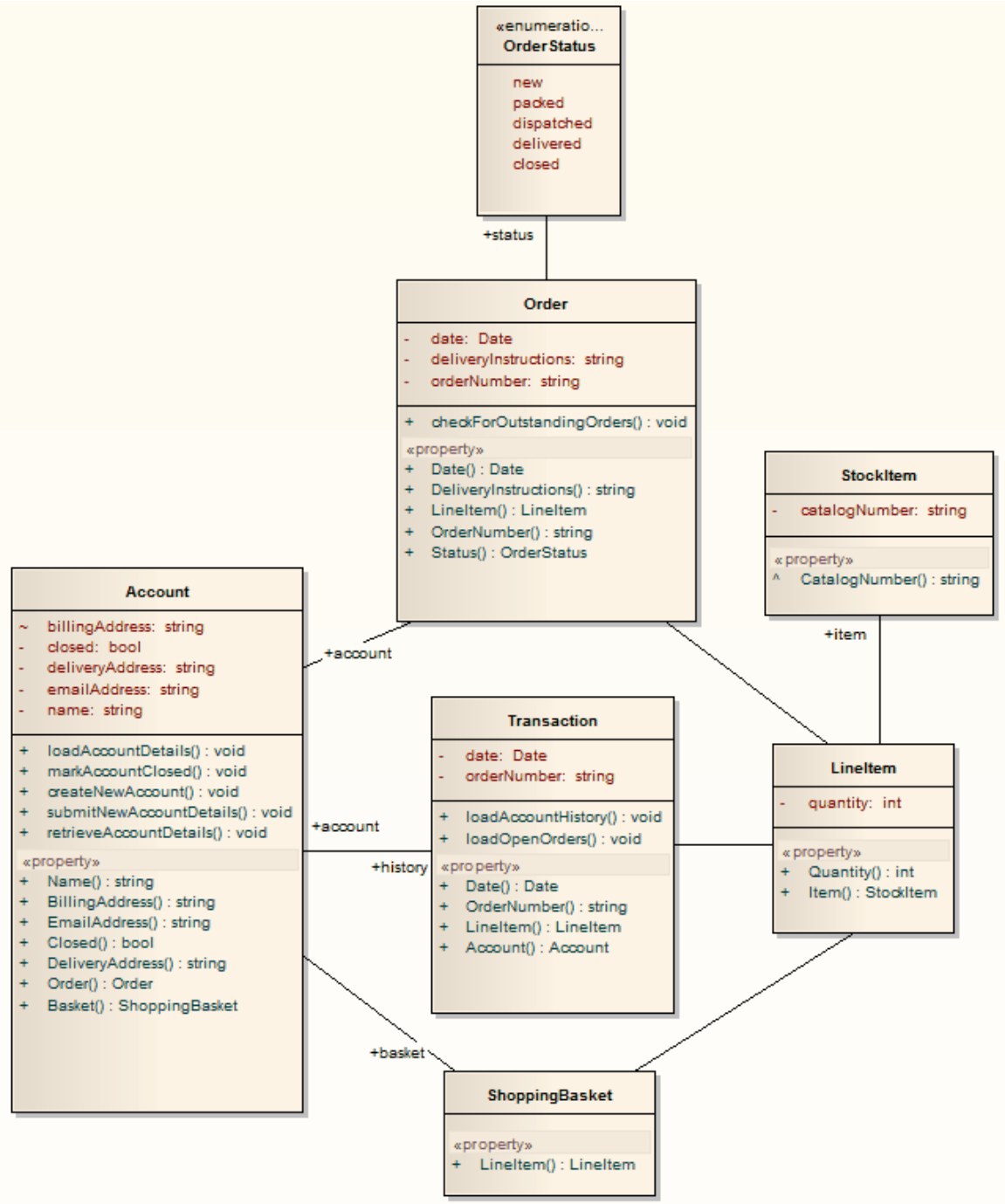
- 对于Java模型中的每个类，都创建了一个相应的测试类，其中包含源类中每个公共方法的测试方法，以及适当设置测试所需的方法；你填写每个测试的详细信息

NUnit变换

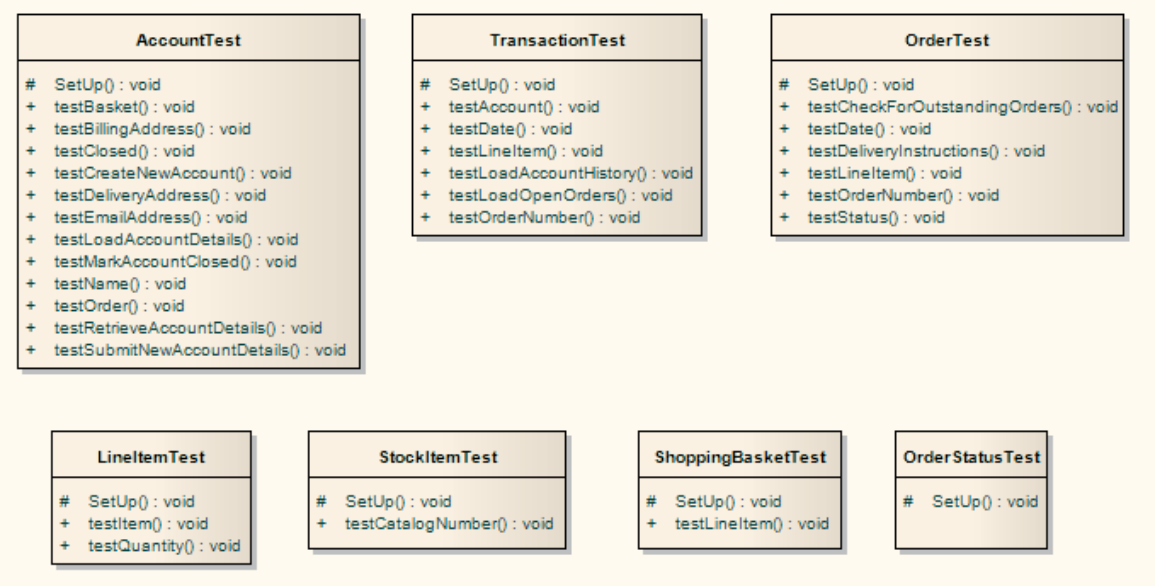
NUnit 转换将现有的具有公共方法的.NET兼容类转换为具有每个公共方法的测试方法的类。然后可以生成生成的类，并通过运行填写和运行测试。

示例

C# 元素（最初从 PIM 转换而来）



改造后成为PSM元素



注记

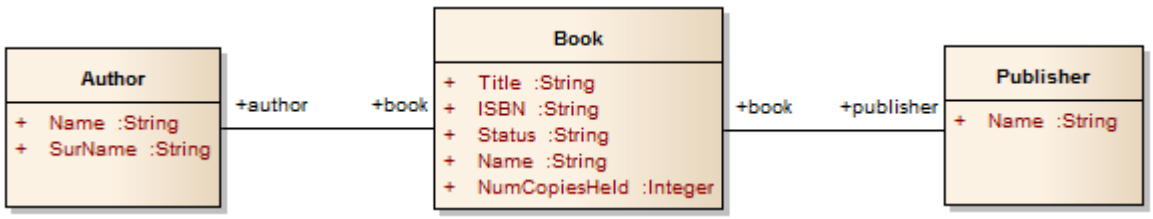
- 对于 C#模型中的每个类，都创建了一个相应的测试类，其中包含源类中每个公共方法的测试方法，以及适当设置测试所需的方法；你填写每个测试的详细信息

PHP变换

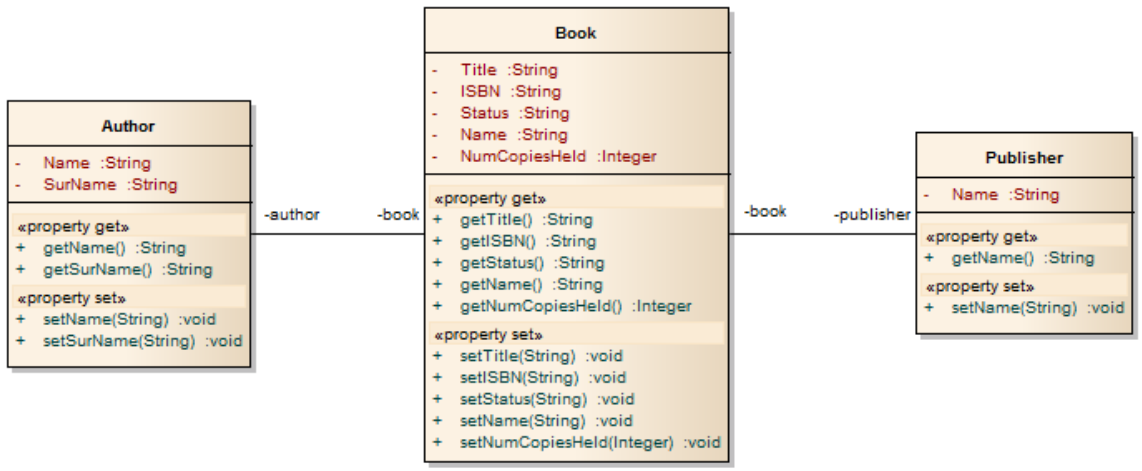
PHP 转换将平台模型(PIM)元素类型转换为特定语言的 PHP类元素类型，并根据您设置的用于从 PHP 属性创建属性的选项创建封装（生成 getter 和 setter）（在 'PHP Specifications ' 首选项对话框的页面）。请注记，PIM 中的公共属性将转换为 PSM 中的私有属性。

示例

PIM 元素



改造后成为PSM元素



序列/通讯图表变换

可以将序列图转化为通讯图，也可以将通讯图转化为序列图。在每种情况下，源图类型中的每个元素或消息都以1：1的方式映射到目标图中的匹配元素或消息。

访问

功能区	设计>包>变换>变换选区
键盘快捷键	Ctrl+Shift+H（变换当前包） Ctrl+H（变换选定元素）

执行变换

字段/按钮	行动
元素	列出并突出显示图表中将包含在转换中的所有元素。
转型	选择： ● 通讯”复选框，如果将序列图转换为通讯图，或 ● “ 列”复选框，如果将通讯图转换为序列图 无论哪种情况，都会显示 浏览项目”对话框。浏览并选择将在其中创建目标图的目标包，然后单击确定按钮。
做变换	单击此按钮以执行转换。 目标图已创建并在目标包下的浏览器窗口中列出，并带有名称（取决于您执行的转换）： <源图名>通讯或 <源图名>序列

注记

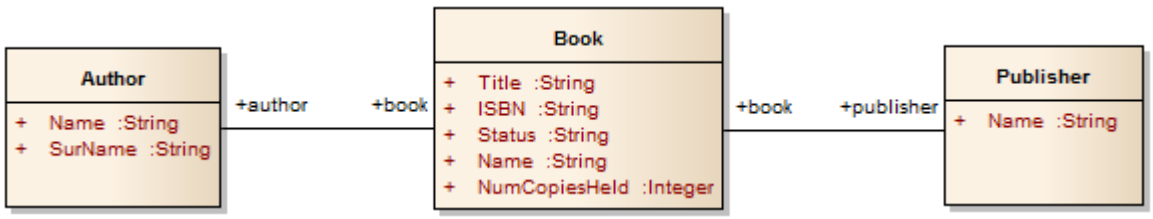
- 被转换的图表必须在主图表视图中打开，以便 通讯”或 序列”选项出现在 模型变换对话框中。
- 一旦您选择了 通讯”或 序列”复选框，这些转换将忽略对话框中除 目标包”之外的任何其他字段设置，并将执行源图中每个元素的直接转换

VB.Net变换

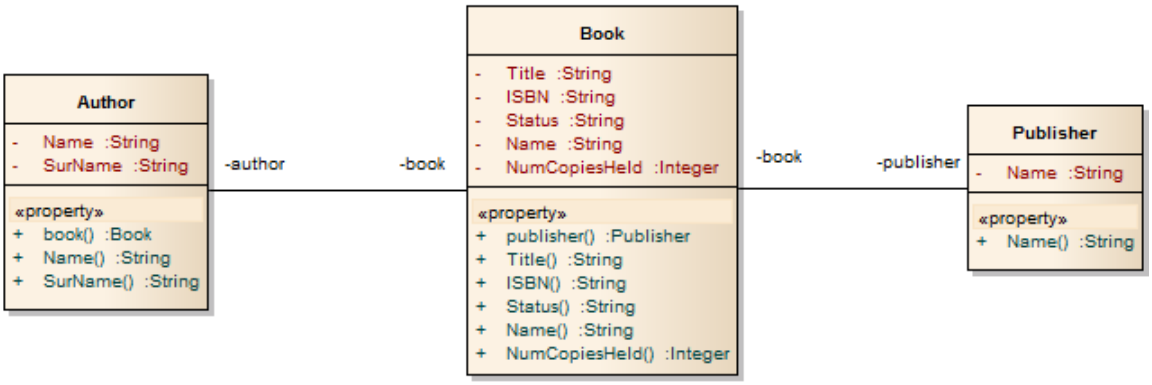
VB.Net 转换将平台模型（PIM）元素类型转换为特定语言的 VB.Net类元素类型，并根据您设置的用于从 VB.Net 属性创建属性的选项创建封装（在 "VB.Net"属性中）。 Net Specifications 页面）。注记PIM 中的公共属性被转换为 PSM 中的私有属性。

示例

PIM 元素

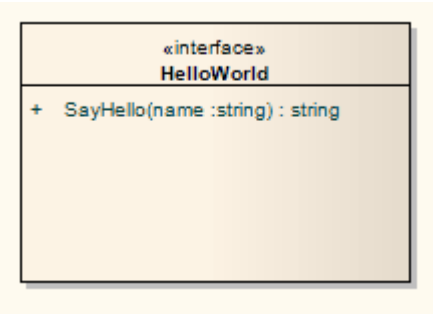


改造后成为PSM元素



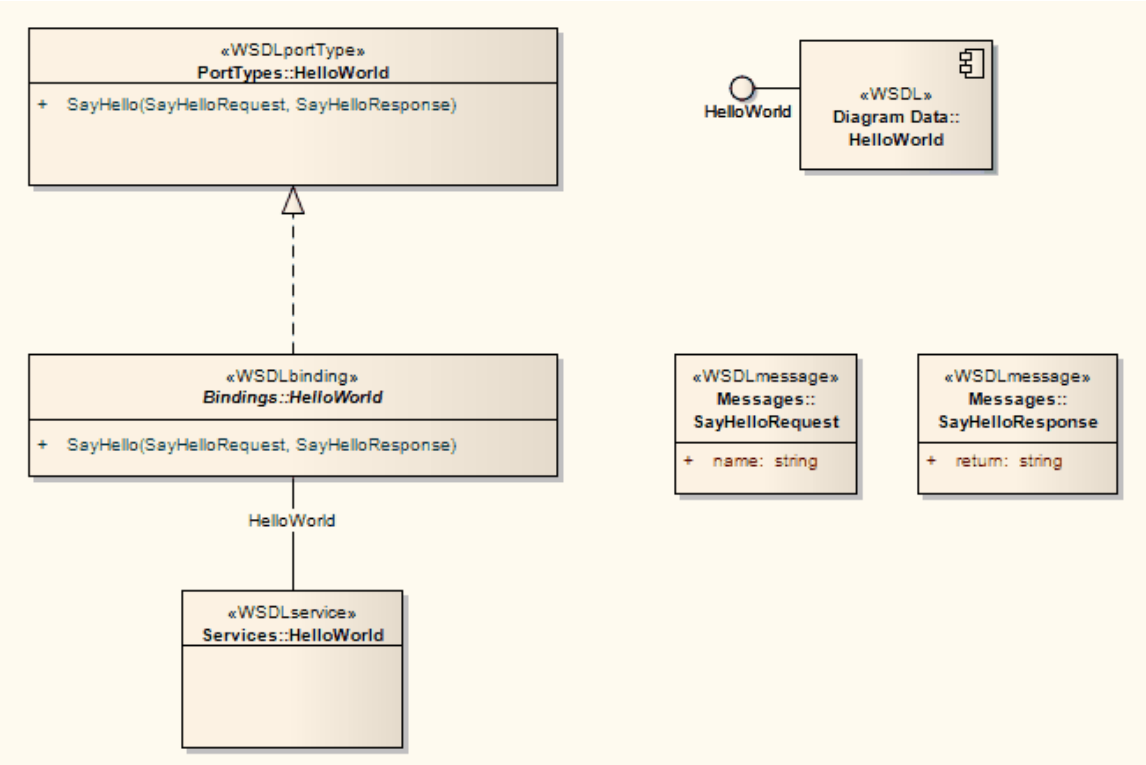
WSDL变换

WSDL 转换将简单模型转换为适合生成的 WSDL 接口的扩展模型。例如：



变换的变换生成相应的 WSDL部件、Service、端口类型、捆绑和 Messages：

- 类的处理方式与 XSD 中的变换
- 所有 'in' 参数都转换为请求消息中的 WSDL信息部分
- 返回值和所有 'out' 和 'return' 参数都转换为响应消息中的 WSDL信息部分
- 所有有返回值的方法都转化为 Request-Response 操作，所有不返回值的方法都转化为 OneWay 操作
- 转换不处理 Solicit-Response 和 Notification 方法或故障的生成



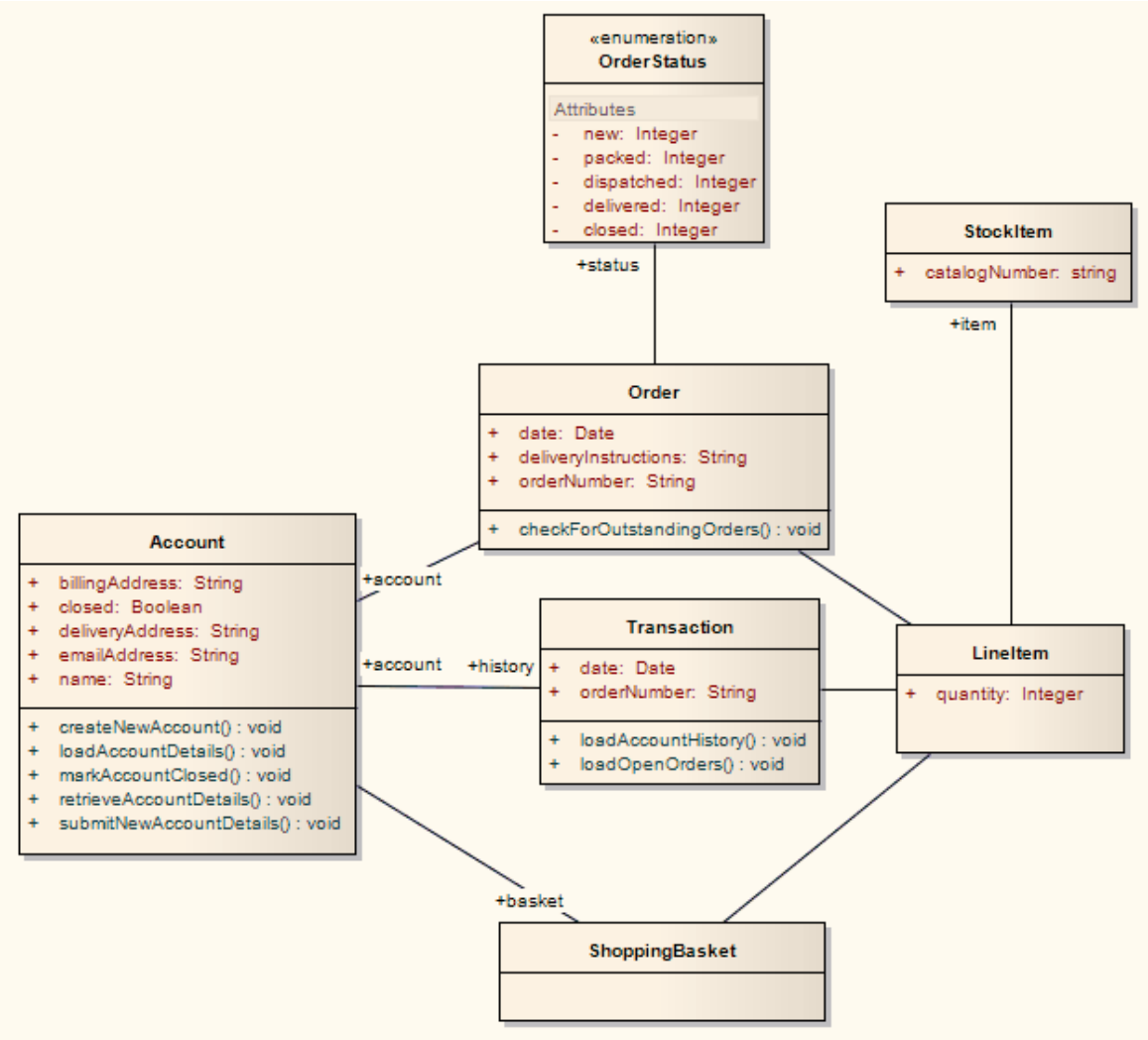
在生成的包中，您可以使用Enterprise Architect的 WSDL 编辑功能填写详细信息，最后使用 WSDL 生成工具生成包。

XSD变换

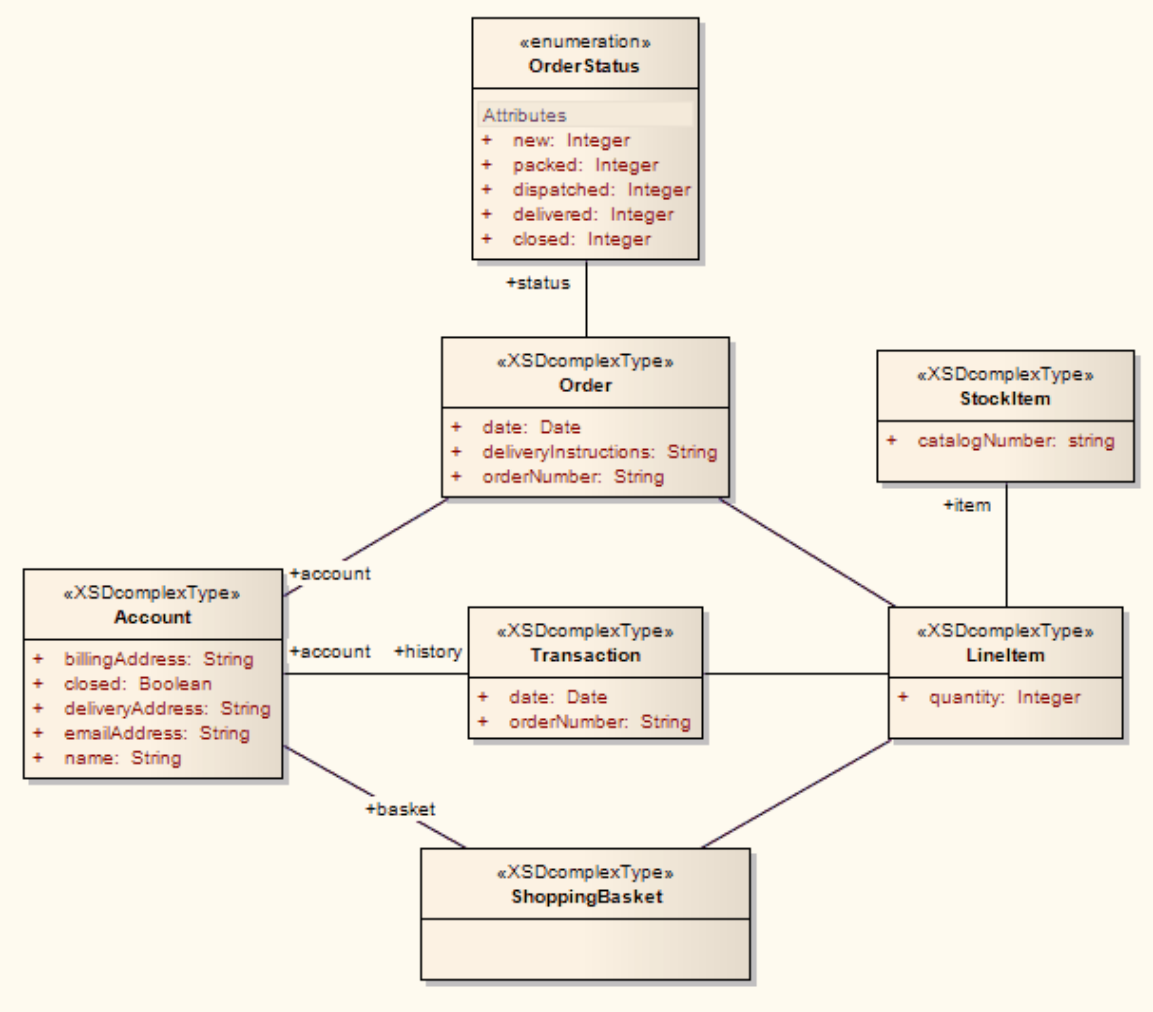
XSD 转换将平台模型(PIM) 元素转换为 XML 元素的UML配置文件，作为创建XML Schema的中间步骤。每个选定的 PIM类元素都被转换为一个 «XSDcomplexType»元素。

示例

PIM 元素



改造后成为PSM元素



这些反过来生成这个 XSD

<?xml 版本=" 1 .0" 编码="ISO-8859-1"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:元素="账户" type="账户"/>

<xs:complexType name="账户">

<xs:序列>

<xs:元素名="name" type="xs: string "/>

<xs:元素="billingAddress" type="xs: string "/>

<xs:元素名="emailAddress" type="xs: string "/>

<xs:元素名="关闭" type="xs:boolean"/>

<xs:元素名="deliveryAddress" type="xs: string "/>

<xs:元素="Order"/>

<xs:元素ref="ShoppingBasket"/>

</xs:sequence>

</xs:complexType>

<xs:元素名称="LineItem" type="LineItem"/>

<xs:complexType name="LineItem">

<xs:序列>

<xs:元素名="数量" type="xs:integer"/>

<xs:元素="StockItem"/>
</xs:sequence>
</xs:complexType>
<xs:元素名称="Order" type="Order"/>
<xs:complexType name="订单">
<xs:序列>
<xs:元素名称="日期" type="xs:日期"/>
<xs:元素="deliveryInstructions" type="xs: string "/>
<xs:元素名="orderNumber" type="xs: string "/>
<xs:元素="LineItem"/>
<xs:元素名称="状态" type="OrderStatus"/>
</xs:sequence>
</xs:complexType>
<xs:simpleType name="OrderStatus">
<xs:restriction base="xs: string ">
<xs:enumeration value="new"/>
<xs:enumeration value="packed"/>
<xs:enumeration value="dispatched"/>
<xs:enumeration value="delivered"/>
<xs:枚举值="关闭"/>
</xs:限制>
</xs:simpleType>
<xs:元素名称="ShoppingBasket" type="ShoppingBasket"/>
<xs:complexType name="ShoppingBasket">
<xs:序列>
<xs:元素="LineItem"/>
</xs:sequence>
</xs:complexType>
<xs:元素名称="StockItem" type="StockItem"/>
<xs:complexType name="StockItem">
<xs:序列>
<xs:元素="catalogNumber" type="xs: string "/>
</xs:sequence>
</xs:complexType>
<xs:元素="Transaction" type="Transaction"/>
<xs:complexType name="事务">
<xs:序列>
<xs:元素名称="日期" type="xs:日期"/>
<xs:元素名="orderNumber" type="xs: string "/>
<xs:元素="账户"/>
<xs:元素="LineItem"/>
</xs:sequence>

```
</xs:complexType>
</xs:schema>
```

编辑变换模板

单个转换应用多个转换模板，每个模板都定义了A转换中作用的object类型，以及对该类型对象执行的操作。系统提供了一系列内置的默认模板，每种类型的转换都使用这些模板的特定子集。通常，转换类型和模板子集是针对目标语言定制的。一组中的一些默认模板没有内容；这些是 潜在的”，表示对通常不包含在转换中但如果您想包含它完全有效的object进行操作的潜力。潜在模板的一个示例是C#转换中的类模板。

您可以通过多种方式定制转换模板，包括：

- 调整默认集中一个或多个模板中的代码
- 将代码添加到 潜在”默认模板
- 添加一个新的自定义模板，基于默认值之一，但服务于您定义的不同目的
- 添加一个新的转换类型，最初包含一组基本的默认模板
- 添加（或删除）模板的原型覆盖

仅A元素和/或特征属于指定的原型类型时，原型覆盖才指示转换使用修改后的模板。如果object或特征不属于该类型，则转换将应用基本模板。

访问

功能区	设计>包>变换>变换模板
键盘快捷键	Ctrl+Alt+H

编辑变换模板

选项	行动
语	单击下拉箭头并选择转换的名称。
新的变换类型	如果要创建新的转换，请单击此按钮。 显示转换名称A提示；输入名称并点击确定按钮。 模板”列表显示了一组默认的内置模板，您可以从中开发您的转换。除非您在转换中添加和/或编辑一个或多个模板，否则您的自定义转换不会保存或可供使用。
模板	列出当前转换的转换模板。 单击模板名称以突出显示它并在模板面板中显示其内容。 已修改”列指示您是否已为此转换编辑模板。
模板	显示当前选定模板的内容，并提供编辑器功能来修改模板（右键单击代码文本）。
构造型覆盖	列出活动基本模板。 'Modified' 列指示您是否修改了原型模板。
加新自定义模板	单击此按钮可创建自定义模板以添加到当前转换。

	将显示A对话框，提示您指定： • 此新模板将响应的object类型（基本模板类型）- 单击下拉箭头并选择名称（自定义模板类型不包含在此列表中） • 新模板的名称——在适当的文本中键入 点击确定按钮。新的模板名称被添加到模板列表中，并在模板编辑器中打开，供您添加其代码。
加新Stereotyped Override	单击此按钮可为当前选定的模板添加新的构造型覆盖。将显示A对话框，提示您指定： • 基类（类- 单击下拉箭头并单击列表中的类型）和/或 • 特征（单击下拉箭头并单击列表中的定型特征） 点击确定按钮。覆盖被添加到 构造型覆盖”列表中。
获取默认模板	单击此按钮以使用当前内置模板的默认版本更新编辑器显示或清除当前自定义模板的内容。 如果您保存了更改后的模板，重新设置默认版本是一种更改，因此 已修改”字段仍显示 是”字样。
节省	单击此按钮以保存新的或编辑的当前模板。您不能在不保存当前模板的情况下切换到另一个模板，因此这也有效地保存了过渡。
删除	单击此按钮可删除当前自定义模板或构造型覆盖，或对内置模板的最新更改（有效地将其返回到默认的基本内容）。您不能删除内置模板。 系统会提示您确认删除。
帮助	单击此按钮可显示此帮助主题。

注记

- 变换编辑非常强烈模板模板；有关编辑转换模板的更多信息，请参阅代码模板编辑器部分和编辑源代码主题

编写转换

Enterprise Architect功能了创建您自己的转换的功能；这对于从更一般的模型生成更具体的模型的过程自动化、重用转换并防止引入错误（如果模型是手动创建的）可能很有用。现有模板将提供有用的指南和参考，以帮助您创建新模板。

变换模板基于代码生成模板框架，了解这些模板的工作方式对于能够调整现有转换模板或创建新模板至关重要。因此，建议您在使用变换模板语言之前阅读并理解讨论代码生成模板的主题。

访问

功能区	设计>包>变换>变换模板
键盘快捷键	Ctrl+Alt+H

有关变换的模板

因素	细节
默认变换模板	Enterprise Architect提供了一组默认转换模板，您可以 按原样”使用或根据您的要求进行自定义。
中间语言的通用语法	Enterprise Architect中的转换生成转换中创建的模型的中间代码形式。您可以审阅并编辑此代码。
中介语言调试	您还可以通过检查从转换脚本生成的中间代码来调试转换脚本。
编辑转换模板和代码	编写转换时，您使用功能代码编辑器的功能。
代码模板框架	您使用代码模板框架来执行UML模型的正向工程。变换模板框架就是由此衍生而来的。
创建对象的语法	要在转换中生成对象或元素，请在模板脚本中应用特定语法。
创建连接器的语法	要在转换中生成连接器（关系），您还可以在模板脚本中应用特定语法。
转换重复信息	在许多转换中，需要复制大量信息。您可以使用宏从其源中读取它，而不是将这些信息放在模板中。
转换模板参数替换	在转换模板中，如果要转换模板捆绑连接器绑定参数替换，则可以使用模板参数替换宏。
转换类型	您可以应用各种方法将数据类型转换为不同的目标平台类型。
转换名称	您可以应用各种方法将元素名称转换为不同的目标平台命名约定。
交叉引用	在转换期间，您可以对转换后的元素执行交叉验证。

注记

- 通过对Enterprise Architect提供的变换模板的深入研究，可以收集到更多提示和技巧
- 变换模板编辑器应用功能代码编辑器的功能

默认变换模板

变换模板提供了以修改的方式在模型中表示现有信息的能力。创建新转换时， Enterprise Architect提供了一组默认转换模板，用于将源模型直接复制到目标模型。这使您可以考虑源模型和目标模型的不同之处。对于每个模板，您都可以防止属性被复制并添加其他信息，直到创建相应的目标模型。

您可以在变换编辑器中列出并检查默认模板。默认模板的组合因您要转换的语言而异。

访问

功能区	设计>包>变换>变换模板
键盘快捷键	Ctrl+Alt+H

注记

- 创建新转换时，您必须在新转换可用之前修改至少一个模板

中介语言

Enterprise Architect中的所有转换都创建了要生成的模型的中间语言形式。您可以使用外部编辑器访问和编辑包含此中间语言代码的文件。每个object在此语言中由object类型（例如类、行动、概括或属性）表示，然后是object及其特征； object描述的语法类似于：

元素：

elementName { (elementProperty |元素)* }

元素属性：

包裹名字

刻板印象

propertyName = "propertyValueSymbol*"

包裹名字：

名称=“propertyValueSymbol*”（ 。 p̋ropertyValueSymbol*”）*

刻板印象：

刻板印象=“propertyValueSymbol*”（ ，“propertyValueSymbol*”）*

属性值符号：

\\

\"

除 " (U+0022)、\ (U+005C) 以外的任何字符

- elementName 是一组元素类型中的任何一个
- propertyName 是一组属性中的任何一个

文字字符串可以通过 转义″引号字符包含在属性值中：

default = "\"一些string值。\""

中介语言调试

来自 MDA模板的脚本生成中间语言文本。但是，在生成模型时，此脚本可能会返回错误。发生错误时，您可以在外部查看和调试生成的文本，最好是在提示更新文件更改的编辑器中。

访问

功能区	设计>包>变换>变换选区
键盘快捷键	Ctrl+H（变换选定元素） Ctrl+Shift+H（变换当前包）

调试生成更改代码时返回错误时

节	描述
1	选择要转换的包，并选择 转换包”选项。 将显示 模型转换”对话框。
2	在 名称”列中，选中要更改的转换类型对应的复选框。
3	在 中间文件”字段中，单击 ... 按钮并设置生成代码的文件位置。
4	选中 始终写入”复选框，然后单击 立即写入”按钮以生成脚本。 这只会生成脚本，不会生成模型。
5	如果返回指定问题行号的错误，请在外部代码编辑器（使用行号）中打开文件并找到问题的行号。
6	更改模板代码以更正错误。
7	单击 执行转换”按钮以检查更改是否已纠正问题。

示例

对于MySQL数据库，模板代码可能类似于：

$enumFieldName = 测试”

柱子

{

name= %qt%% CONVERT_NAME ($enumFieldName, "Pascal Case", "Camel Case")%%qt%

type= %qt%% CONVERT_TYPE (genOptDefaultDatabase, "Enum")%%qt%

}

这会将生成的文本文件中的输出返回为：

柱子

{

名称= 测试”

类型= 枚举”

}

如果原始转换中存在错误，例如拼写错误 - 列” - 单击 执行转换”按钮将返回一条错误消息，该消息引用包含错误 列”的中间代码的第一行。

对象

对象在转换中生成为以下形式的文本：

对象类型
{
对象属性*
外部参照{外部参照}*
标签{标签}*
属性{属性}*
操作{操作}*
分类器{分类器}*
参数{参数}*
}

例如：

类
{
名称= 东例″
语言 = "C++"
标签
{
name = "defaultCollectionClass"
值 = "列表"
}
属性
{
名称= 计数″
类型=“ int ”
}
}

在转换中创建的每个object都应包含外部参照语法元素（请参阅本主题的末尾），因为它有助于系统与object同步，并可以在转换中创建到类的连接器。

代码中的语法元素

元素	细节
对象类型	objectType 是其中之一： • 行动 • 行动销 • 活动

	• 活动参数 • 活动分区 • 活动区域 • 参与者 • 关联 • 更改 • 类 • 协作 • 协作使用 • 部件 • 部署规范 • 图框 • 决策 • 入口点 • 事件 • 异常处理程序 • 执行环境 • 出口点 • 扩展节点 • 扩展区域 • 暴露接口 • 图形界面元素 • 交互片段 • 交互发生 • 交互状态 • 接口 • 可中断活动区域 • 问题 • 迭代 • 物件 • 物件节点 • 合并节点 • 消息端点 • 节点 • 包 • 参数 • 部件 • 端口 • 提供的接口 • 必需接口 • 需求 • 序列 • 状态 • 状态机

	• 状态节点 • 同步 • 库表 • 时间线 • 触发器 • UMLD图 • 用例
对象属性	objectProperties 为零，或以下一项或多项的一个实例： • 抽象的 • 别名 • 论据 • 作者 • 基数 • 分类器 • 复杂 • 并发 • 文件名 • 标题 • 导入 • 活跃 • 伊斯叶 • 是根 • 是规范 • 关键词 • 语 • 多样性 • 名称 • 注记 • 类型 • 持久性 • 相 • 范围 • 状态 • 构造型 • 版本 • 能见度
属性	Attribute 与 objectType 具有相同的结构，包括以下属性： • 别名 • 分类器 • 收藏 • 容器 • 遏制 • 持续的

	• 默认 • 衍生的 • 下界 • 名称 • 注记 • 已订购 • 范围 • 静止的 • 构造型 • 类型 • 上界 • 易挥发的 属性还包括以下元素： • 分类器 • 标签 • 外部参照
手术	Operation 与 objectType 具有相同的结构，并包括以下属性： • 抽象的 • 别名 • 行为 • 分类器 • 代码 • 持续的 • 查询 • 名称 • 注记 • 纯的 • 返回数组 • 范围 • 静止的 • 构造型 • 类型 操作还包括以下要素： • 分类器 • 参数 • 标签 • 外部参照
参数	参数与objectType具有相同的结构，并且包括Tag元素和这些属性： • 分类器 • 默认 • 固定的 • 名称 • 注记

	• 种类 • 构造型
标签	标记具有以下属性： • 名称 • 价值

特别案例

某些类型的object具有object定义语法的变体。

物件	细节
包	包在这些方面与其他对象不同： • 它们有一组简化的属性：别名、作者、名称、命名空间根、注记、范围、构造型和版本 • 属性namespaceRoot 只给包 • 必须为每个包指定A名称 • name属性可以是限定名；当指定了限定名称时，给定的属性仅应用于最终包 • 只有包可以包含其他包 • 包不能包含属性和操作
外部参照	交叉引用是使用转换语句定义的。属性包括： • 命名空间 • 名称 • 源 • 注记
表	表是一种特殊类型的object ，与其他object类型有以下区别： • 它们可以包括列和主键 • 它们不能包含属性
列	列类似于属性，但有一个包含 Startnum 及其增量的自动编号元素，以及这些添加的属性： • 长度 • 非空 • 精确 • 首要的关键 • 规模 • 独特的 在列定义中，您不能为 NotNull、PrimaryKey 或 Unique属性赋值。

连接器

在转换中创建连接器的过程与创建元素（对象）的形式相同。它有点复杂，因为您还定义了连接器的每一端 -源和目标。

连接器在中间语言中表示为：

连接器类型

{

连接器属性*

关联类 {associationClassProperties*}

源{sourceProperties*}

目标{targetProperties*}

}

例如：

关联

{

名称="一个协会"

刻板印象=""

方向= 未指定"

源

{

访问="私人"

navigability="未指定"

}

目标

{

访问="私人"

多重性=" 1 ..*"

}

}

代码中的语法元素

元素	细节
连接器类型	ConnectorType 是其中之一： • 抽象 • 聚合 • 组装 • 关联 • 协作 • 控制流

	• 连接器 • 代表 • 依赖 • 部署 • 外键 • 概括 • 信息流 • 实例化 • 接口 • 中断流 • 显现 • 嵌套 • 注意链接 • 对象流 • 包 • 实现 • 序列 • 替代 • 模板绑定 • 转移 • 用途 • 用例 • 用途
连接器属性	connectorProperties 为零，或者是以下一项或多项的一个实例： • 别名 • 方向 • 注记 • 姓名 • 刻板印象 • 标签 • 外部参照
关联类属性	AssociationClassProperties 是其中的一个实例： • 分类器 • 外部参照
源属性 目标属性	sourceProperties 和 targetProperties 都是对元素和零的引用，或者是其中一个或多个的一个实例： • 聚合 • 别名 • 允许重复 • 多变 • 约束 • 遏制 • 通航性

	• 会员类型 • 多样性 • 注记 • 订购 • 限定词 • 角色 • 范围 • 刻板印象 • 标签
元素参考	元素引用要么是引用转换之前已存在的元素的 guid，要么是引用由转换创建的元素的外部参照。 • 向导 • 外部参照

注记

- 每个连接器在两个末端对象上都进行了变换，因此连接器可能会在变换中出现两次；这不是问题，尽管您应该仔细检查连接器的生成方式完全相同，无论当前类在哪一端

转换连接器

转换连接器时，您可以使用两种不同类型的类作为连接器的结尾：由转换创建的类或您已经知道GUID的现有类。

连接到由 a变换创建的类

最常见的连接是通过转换创建的类；要创建此连接，您需要使用三项信息：

- 原类GUID
- 转换名称
- 转换类的名称

这种类型的连接器是使用 TRANSFORM_REFERENCE函数宏创建的；当元素在当前转换中时，它可以安全地从转换中省略。最简单的例子是，当您从一个转换中的单个类创建多个类时，您希望它们之间有一个连接器；考虑 EJB实体转换中的这个脚本：

```
依赖
{
%TRANSFORM_REFERENCE("EJBRealizeHome",classGUID)%stereotype="EJBRealizeHome"
源
{
%TRANSFORM_REFERENCE("EJBEntityBean",classGUID)%
}
目标
{
%TRANSFORM_REFERENCE("EJBHomeInterface",classGUID)%
}
}
```

在此脚本中，TRANSFORM_REFERENCE 宏有三种用途：一种用于识别连接器以用于同步目的，另两种用于识别末端；这三个都使用相同的源GUID ，因为它们都来自一个原始类。这三个都不必指定转换，因为这两个引用指向当前转换中的某些内容 - 然后它们中的每一个都只需要标识转换名称。

也可以从另一个连接器创建一个连接器。您可以创建一个连接器模板并从类级别模板中列出连接到一个类的所有连接器；您不必担心只生成一次连接器，因为如果您为连接器创建了 TRANSFORM_REFERENCE，那么系统会自动同步它们。

此脚本复制源连接器：

```
%连接器类型%
{
%TRANSFORM_CURRENT()%
%TRANSFORM_REFERENCE("连接器",connectorGUID)%
源
{
%TRANSFORM_REFERENCE("类",connectorSourceGUID)%
%TRANSFORM_CURRENT("源")%
}
目标
{
```

%TRANSFORM_REFERENCE("类",connectorDestGUID)%

%TRANSFORM_CURRENT("目标")%

}

}

连接到您知道GUID的类

可以用作连接器端的第二类是您知道当前GUID的现有元素。要创建此连接，请在源端或目标端指定目标类的GUID ；此脚本从转换中创建的类创建依赖关系，类其转换为：

依赖

{

%TRANSFORM_REFERENCE("SourceDependency",classGUID)%

刻板印象="transformedFrom"

源

{

%TRANSFORM_REFERENCE("类",classGUID)%

}

目标

{

GUID=%qt%%classGUID%%qt%

}

}

注记

- 每个连接器在两个末端对象上都进行了变换，因此连接器可能会在变换中出现两次；这不是问题，尽管您应该仔细检查连接器的生成方式完全相同，无论当前类在哪一端

转换外键

Enterprise Architect支持将逻辑模型中实体之间定义的许多不同类型的关系转换为外键。

物理模型中的每个外键都由原型连接器和每个相关表中的操作的组合表示。外键转换是通过 DDL 语言中的 连接器"模板实现的。此模板生成一个中间数据集，然后由Enterprise Architect的转换引擎解释以创建所有必需的物理实体和连接器。

默认情况下， Enterprise Architect支持以下连接器类型的转换：

- 概括是-这种连接器将创建一个Foreign Key，其源1的重数为0..1，目标中的重数为1
- 关联类——这种连接器将创建一个连接源和目标表的 连接"表
- 关联/聚合——这些类型的连接器使用逻辑模型关系中定义的多重性来连接源和目的地表

所有外键定义都将导致在源表和目标表中添加一个新的整数（或等效）列，它将充当源库表中的主键和目标表库表中的外键列。新列的默认名称将是库表名称，加上后缀 ID"，而外键的名称将使用 FK DDL模板自动生成。

复制信息

在许多转换中，需要复制大量信息。

将所有公共信息输入到模板中以便将其复制到转换后的类中会很乏味；另一种方法是使用 TRANSFORM_CURRENT 和 TRANSFORM_TAGS函数宏。

使用宏

客观的	细节
复制物件	TRANSFORM_CURRENT (<listOfExcludedItems>) 该函数生成当前项的所有属性的精确副本，除了在 <listOfExcludedItems> 中命名的项。
复制连接器	当转换连接器以复制连接器的任一端时，函数的另一种形式是可用的： TRANSFORM_CURRENT (<connectorEnd>, <listOfExcludedItems>) 除了在 <listOfExcludedItems> 中命名的项目之外，这将生成由 <connectorEnd>（源或目标）指定的连接器端的精确副本。
复制标签	TRANSFORM_TAGS (<listOfExcludedItems>) 该函数生成当前项目所有标记值的精确副本，<listOfExcludedItems> 中命名的项目除外。

转换类型

不同的目标平台几乎肯定需要不同的数据类型，因此您通常需要一种在类型之间进行转换的方法。这是由宏提供的：

CONVERT_TYPE (<destinationLanguage>, <originalType>)

该函数使用模型中定义的数据类型和通用类型将<originalType> 转换为<destinationLanguage> 中的相应类型，其中<originalType> 假定为独立于平台的通用类型。

A常见数据类型转换为指定数据库的数据类型时，可以使用类似的宏：

CONVERT_DB_TYPE (<destinationDatabase>, <originalType>)

该函数将<originalType> 转换为模型中定义的<destinationDatabase> 中的相应数据类型； <originalType> 指的是独立于平台的通用数据类型。

转换名称

不同的目标平台使用不同的命名约定，因此您可能不想将元素的名称直接复制到转换后的模型中。为了满足这一要求，转换模板提供了一个 CONVERT_NAME函数宏。

转换名称的另一种方法是使用 REMOVE_PREFIX 宏从原始名称中删除前缀。

CONVERT_NAME (<originalName>, <originalFormat>, <targetFormat>)

此宏将假定在 <originalFormat> 中的 <originalName> 转换为 <targetFormat>。

支持的格式有：

- Camel Case：第一个单词以小写字母开头，后续单词以大写字母开头；例如，我的变量表
- Pascal Case：每个单词的首字母大写；例如，MyVariableTable
- Spaced：单词之间用空格隔开；字母的大小写被忽略
- 下划线：单词之间用下划线分隔；字母的大小写被忽略

原始格式还可能指定要使用的分隔符列表。例如，只要找到空格或下划线，'_'的值就会中断单词。目标格式也可以使用一个格式string来指定每个单词的大小写和它们之间的分隔符。它采用这种形式：

<firstWord> (<delimiter>) <otherWords>

- <firstWord> 控制第一个单词的大小写
- <delimiter> 是单词之间生成的string
- <otherWords> 适用于第一个单词之后的所有单词

<firstWord> 和 <otherWords> 都是两个字符的序列。第一个字符表示该单词第一个字母的大小写，第二个字符表示所有后续字母的大小写。大写字母强制输出为大写，小写字母强制输出为小写，任何其他字符都保留原始大小写。

示例1 ：将每个单词的首字母大写并用空格分隔多个单词：

"Ht()Ht" 输出"我的变量库表"

示例2：生成 Camel Case 的等价物，但颠倒了大写和小写的角色；也就是说，除了第一个单词之后的每个单词的第一个字符之外，所有字符都是大写的：

“HT()hT″输出 我的变量表″

REMOVE_PREFIX(<原始名称>,<prefixes>)

此宏从 <originalName> 中删除在 <prefixes> 中找到的任何前缀。前缀在分号分隔的列表中指定。

该宏通常与 CONVERT_NAME 宏结合使用。例如，此代码根据Java的选项创建一个获取属性名称：

$propertyName=%REMOVE_PREFIX(attName,genOptPropertyPrefix)%

%if genOptGenCapitalisedProperties==" T "%

$propertyName=%CONVERT_NAME($propertyName, "camel case", "pascal case")%

%万一%

注记

- 从 Camel Case 或 Pascal Case 转换时不支持首字母缩略词

交叉引用

交叉引用是转换的重要组成部分。您可以使用它们：

- 找到要与之同步的转换类
- 在转换的类之间创建连接器
- 指定一个类型的分类器
- 确定未来转型的转型目标

每个交叉参考都包含三个不同的部分：

- A命名空间，对应于生成元素的变换
- 名称，它是对可以在转换中生成的事物A唯一引用，以及
- A源，它是创建此元素的元素的GUID

在为转换编写模板时，最容易使用为此目的定义的宏生成交叉引用：

TRANSFORM_REFERENCE (<name>, <sourceGuid>, <namespace>)

这三个参数是可选的。该宏会生成一个类似于以下内容的引用：

外部参照{namespace="<namespace>" name="<name>" source="<sourceGuid>"

- 如果未指定 <name>，则宏获取当前模板的名称
- 如果未指定 <sourceGUID>，则宏获取当前类的GUID
- 如果未指定 <namespace>，则宏获取当前转换的名称

唯一应该指定交叉参考的时间是在创建连接到在不同转换中创建的类时。

使用交叉引用的A很好的例子是Enterprise Architect提供的 DDL 转换。在类模板中创建了一个名为 库表"的交叉参考。然后最多可以创建两个不同的连接器，每个连接器都必须标识它使用交叉引用连接的两个类，同时具有自己独特的交叉参考。

指定分类器

对象、属性、操作和参数都可以引用模型中的另一个元素作为它们的类型。从转换创建此类型时，您必须使用交叉引用来指定它，使用宏：

TRANSFORM_CLASSIFIER (<name>, <sourceGuid>, <namespace>)

此宏在分类器元素中生成交叉引用，其中参数与 TRANSFORM_REFERENCE 宏相同，但生成的是名称分类器而不是外部参照。

如果目标分类器在转换之前已经存在于模型中，则TRANSFORM_CLASSIFIER是不合适的，因此可以直接将GUID赋予一个分类器属性。

如果为任何类型指定了分类器，它将覆盖该类型。

变换模板参数替换

如果您想在转换模板中提供对与模型中模板捆绑连接器的绑定参数替换转换有关的数据的访问，您可以使用模板参数替换宏。

变换中的变换

因素	细节

中介语言	模板参数替换在中间语言中表示为：
	模板参数替换
	{
	正式的 { FormalProperties }
	实际 { ActualProperties }
	}
	
	例如：
	模板参数替换
	{
	正式的
	{
	name=%qt%%parameterSubstitutionFormal%%qt%
	}
	实际的
	{
	name=%qt%%parameterSubstitutionActual%%qt%
	%TRANSFORM_CLASSIFIER("类", parameterSubstitutionActualClassifier)%
	}
	}

形式属性或实际属性	FormalProperties 和 ActualProperties 为零，或以下属性之一的一个实例： • 姓名 • 分类器

参数的变换Substitution 实际参数	如果 Actual 参数被分配了一个字符串表达式，它将转换为 Actual 名称。如果您知道GUID ，则可以分配实际分类器：
	模板参数替换
	(
	正式的
	{
	name=%qt%%parameterSubstitutionFormal%%qt%
	}
	实际的
	{

	name=%qt%%parameterSubstitutionActual%%qt%
	分类器=%qt%%parameterSubstitutionActualClassifier%%qt%
	}
	}
	
	如果您希望对 Actual 参数进行转换，以便为其分类器分配一个已转换的元素，则使用 TRANSFORM_CLASSIFIER 或 TRANSFORM_REFERENCE，如下所示：
	
	模板参数替换
	{
	正式的
	{
	name=%qt%%parameterSubstitutionFormal%%qt%
	}
	实际的
	{
	name=%qt%%parameterSubstitutionActual%%qt%
	%TRANSFORM_CLASSIFIER("类", parameterSubstitutionActualClassifier)%
	}
	}
	
	或者
	
	模板参数替换
	{
	正式的
	{
	name=%qt%%parameterSubstitutionFormal%%qt%
	}
	实际的
	{
	name=%qt%%parameterSubstitutionActual%%qt%
	%TRANSFORM_REFERENCE("类", parameterSubstitutionActualClassifier)%
	}
	}