UML Profile/MDA

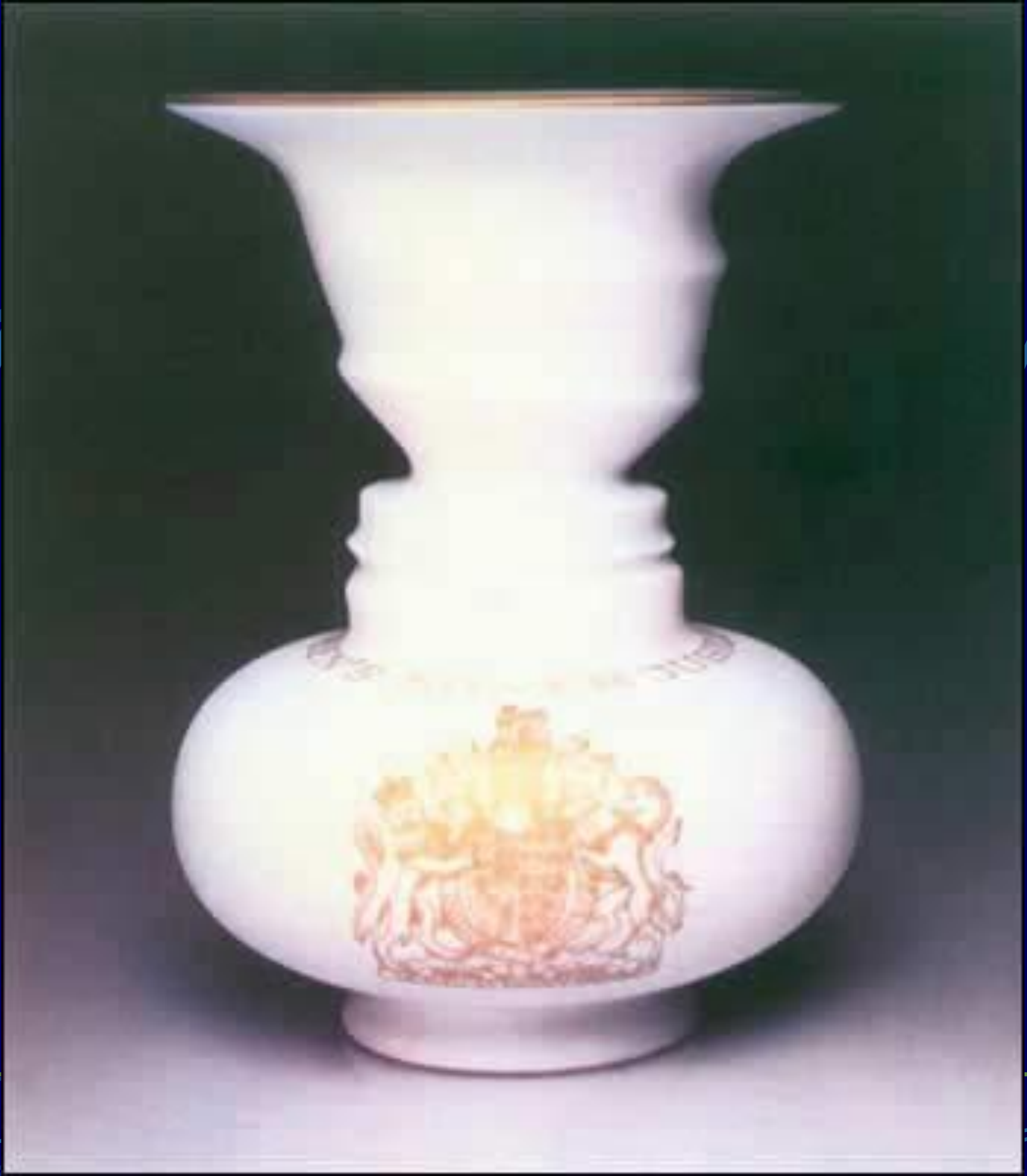
Xu Yingxiao

xuyingxiao@126.com

Fudan University

Review

- Why Modeling?



Problem

- no UML model will represent more than one type of application
  - gaming systems
  - .Net systems
  - e-commerce system
  - billing system

Solution

- Profile
  - dialect of UML
  - standard mechanism to extend UML.

  - a stereotype gives specific roles to elements

  - records additional context-specific information in tagged values .

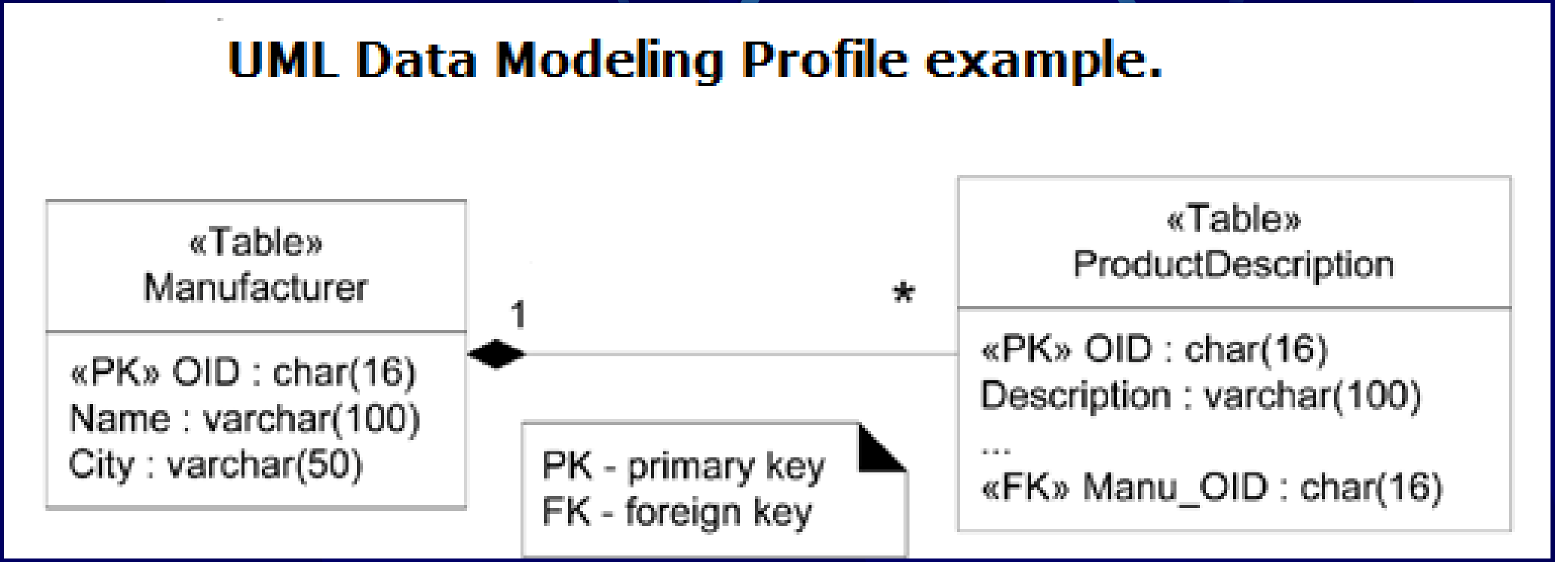
  - Profiles also include constraints that ensure integrity of use.

UML Profiles and Related Specifications

- Platform Independent Model (PIM) & Platform Specific Model (PSM) for Software Radio Components (also referred to as UML Profile for Software Radio)
- UML Profile for CORBA®
- UML Profile for CORBA® Component Model (CCM)
- UML Profile for CORBA® and CORBA® Component Model (CCM) [This specification, nearly complete, will supersede the separate profiles listed just above.]
- UML Profile for Enterprise Application Integration (EAI)
- UML Profile for Enterprise Distributed Object Computing (EDOC)
- UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms
- UML Profile for Schedulability, Performance and Time
- UML Profile for System on a Chip (SoC)
- UML Profile for Systems Engineering (SysML)
- UML Testing Profile

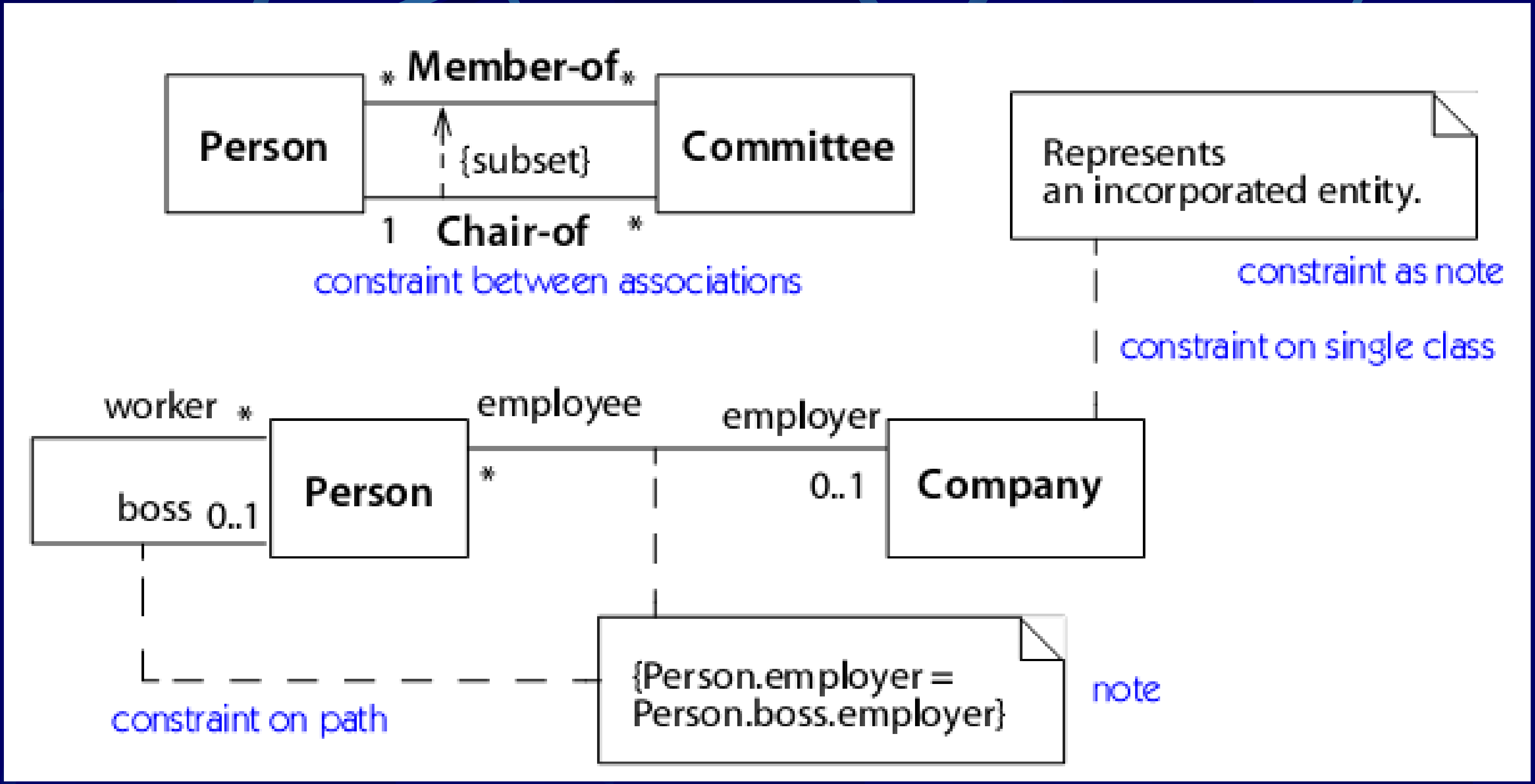
构造型(Stereotype)

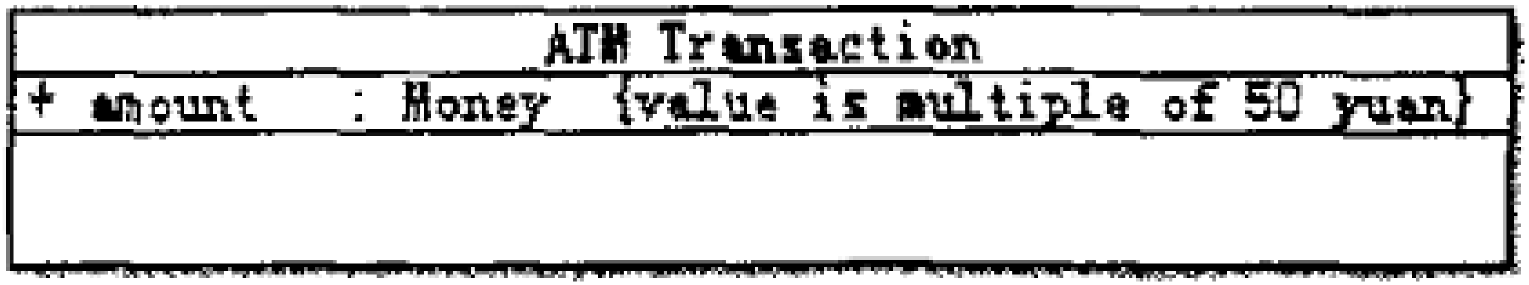
- A stereotype denotes a variation on an existing modeling element with the same form but with a modified intent.



约束 (constraint)

- A constraint refines a model element by expressing a condition or a restriction in a textual statement to which the model element must conform

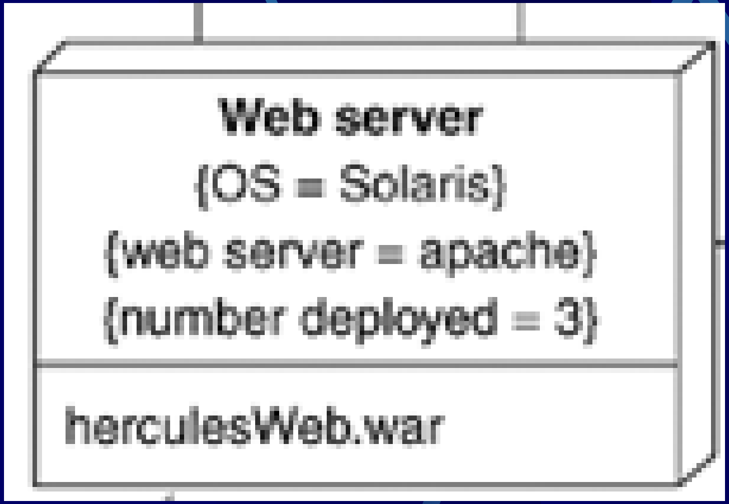

Person
Member-of
*
*
{subset}
Committee
1
Chair-of
*
constraint between associations
Represents an incorporated entity.
constraint as note
constraint on single class
worker *
boss 0..1
Person
employee
*
employer
0..1
Company
{Person.employer = Person.boss.employer}
note
constraint on path


ATM Transaction

+ amount : Money {value is multiple of 50 yuan}

标记值 **(Tagged Value)**

- A *tagged value* can be used to add properties to any model

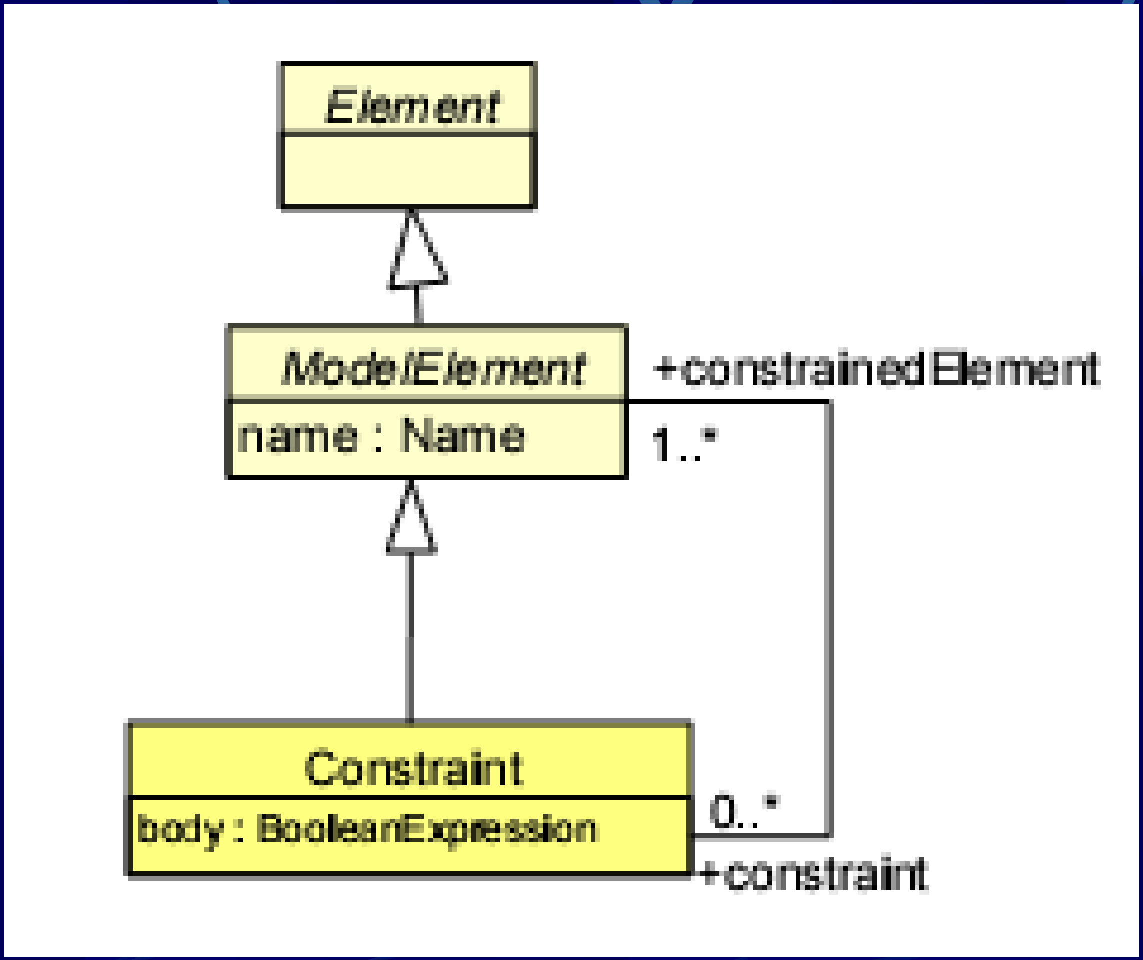

Web server
{OS = Solaris}
{web server = apache}
{number deployed = 3}
herculesWeb.war

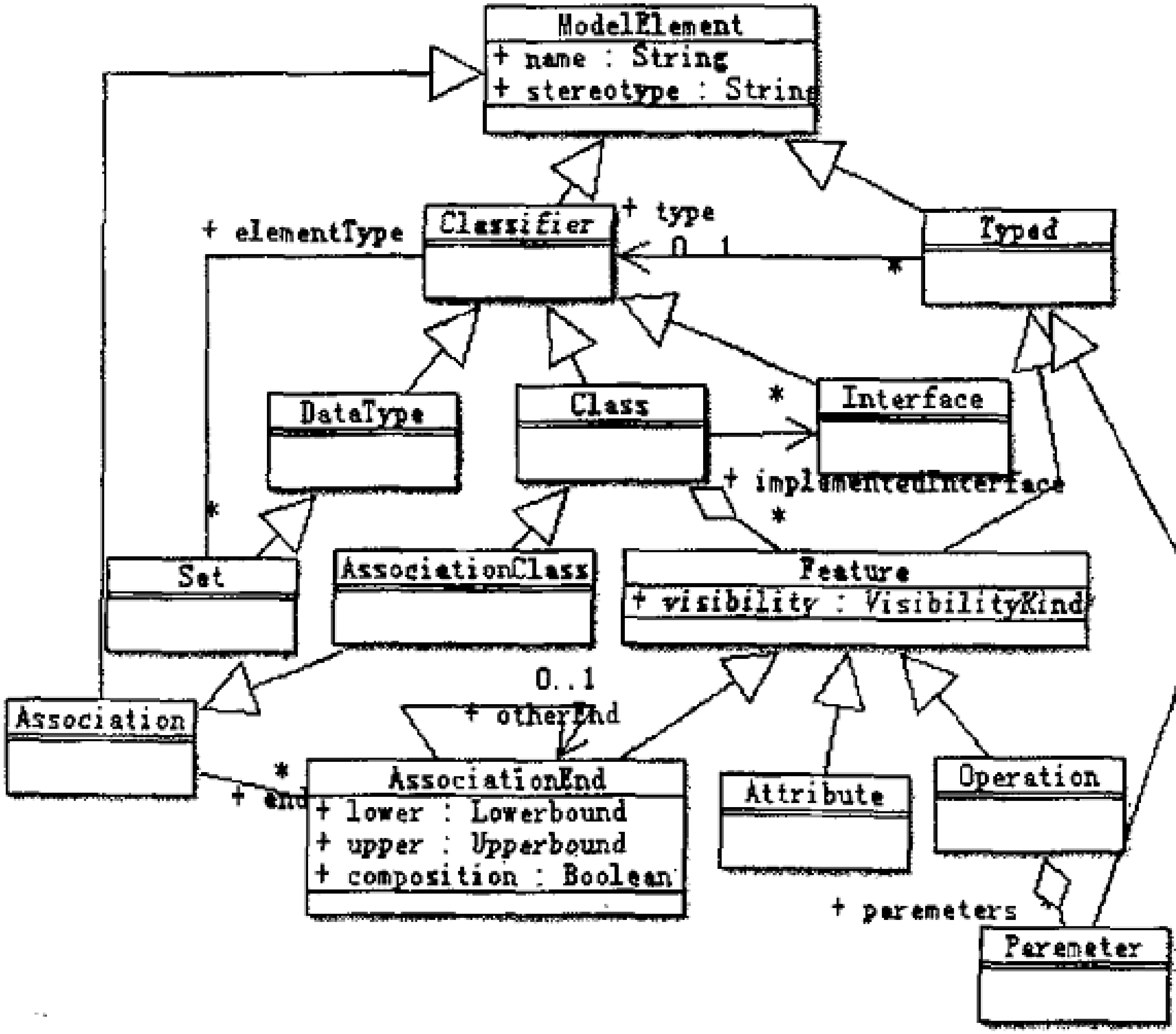
UML Metamodel

- The *UML metamodel* defines model elements and their relationships for UML diagrams

- Every model element can have zero, one or more constraint.
- Constraint is a model element

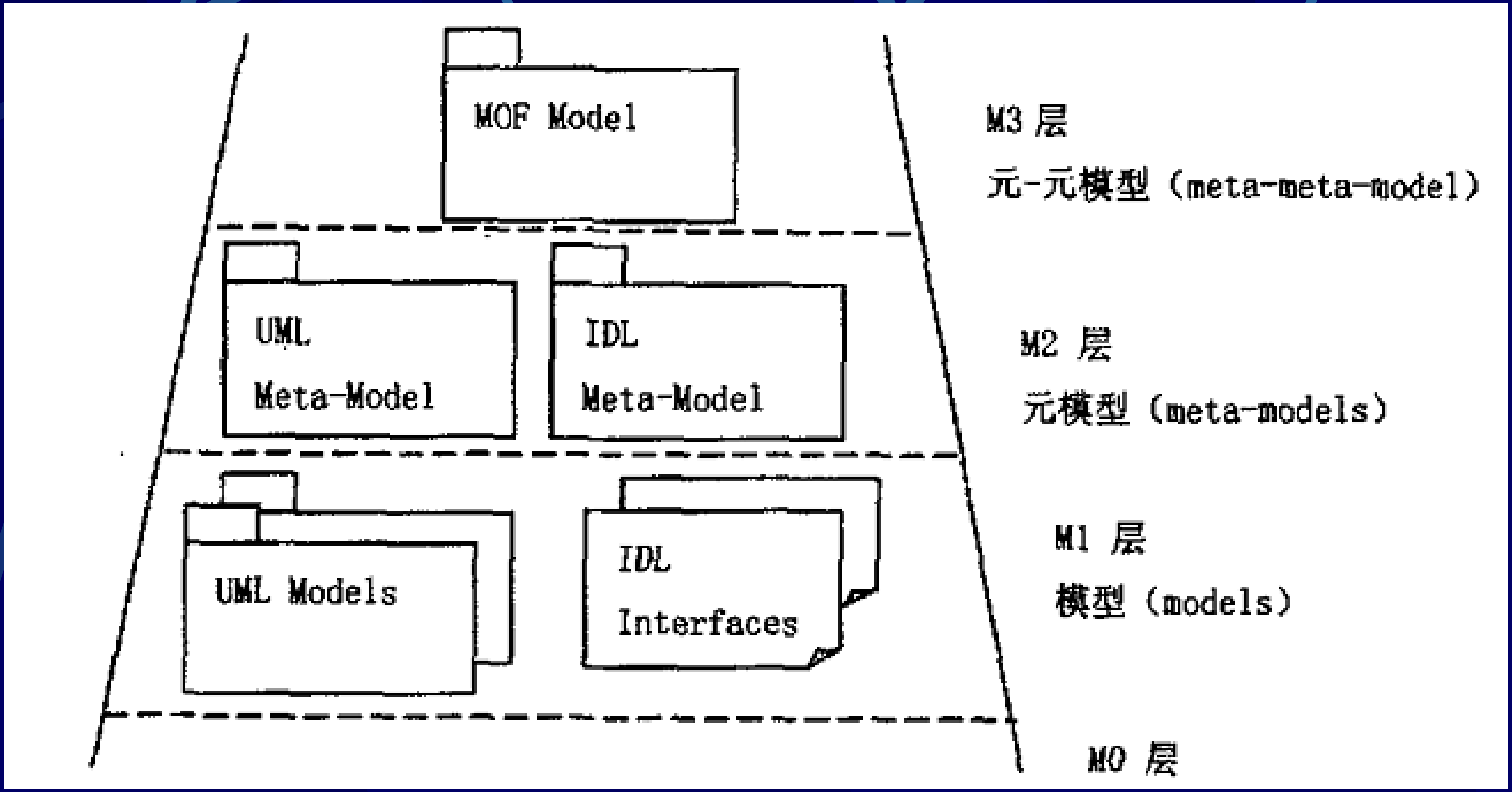
- Draw metaModel

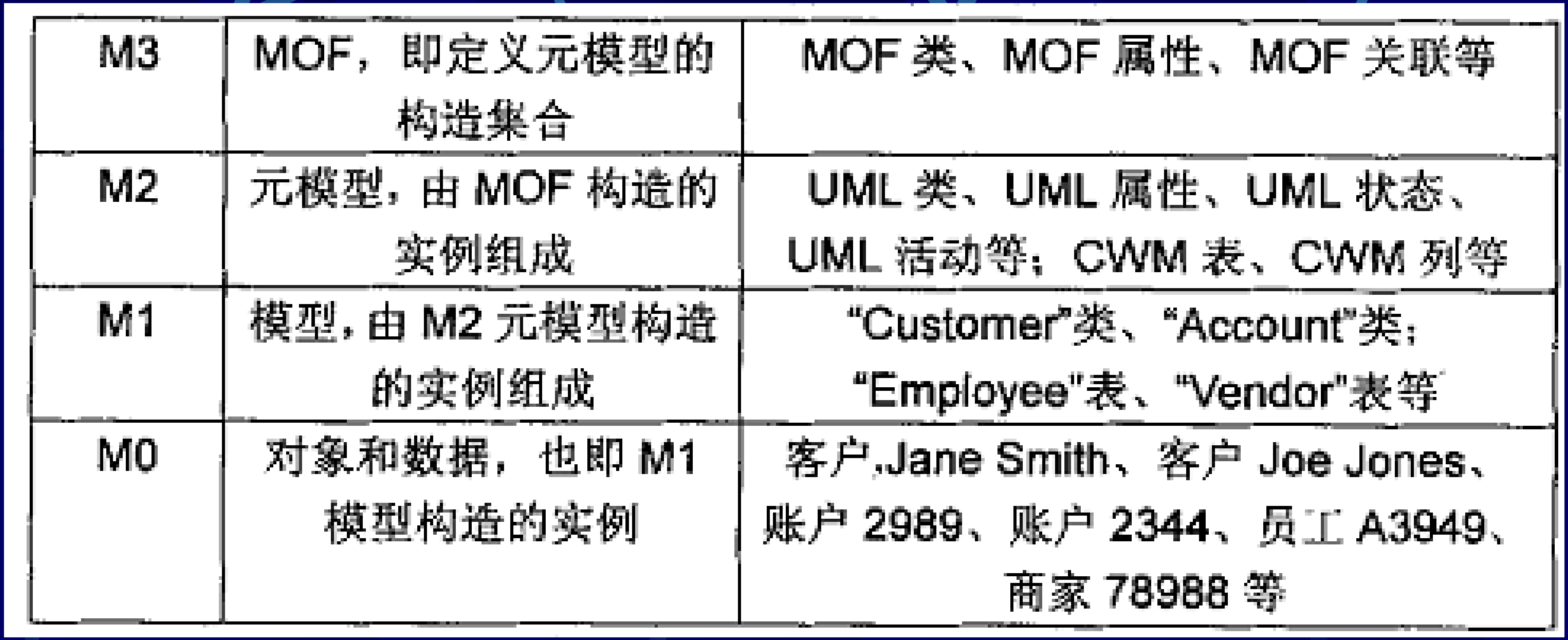

Element
ModelElement
name : Name
+constrainedElement
1..*
Constraint
body : BooleanExpression
0..*
+constraint



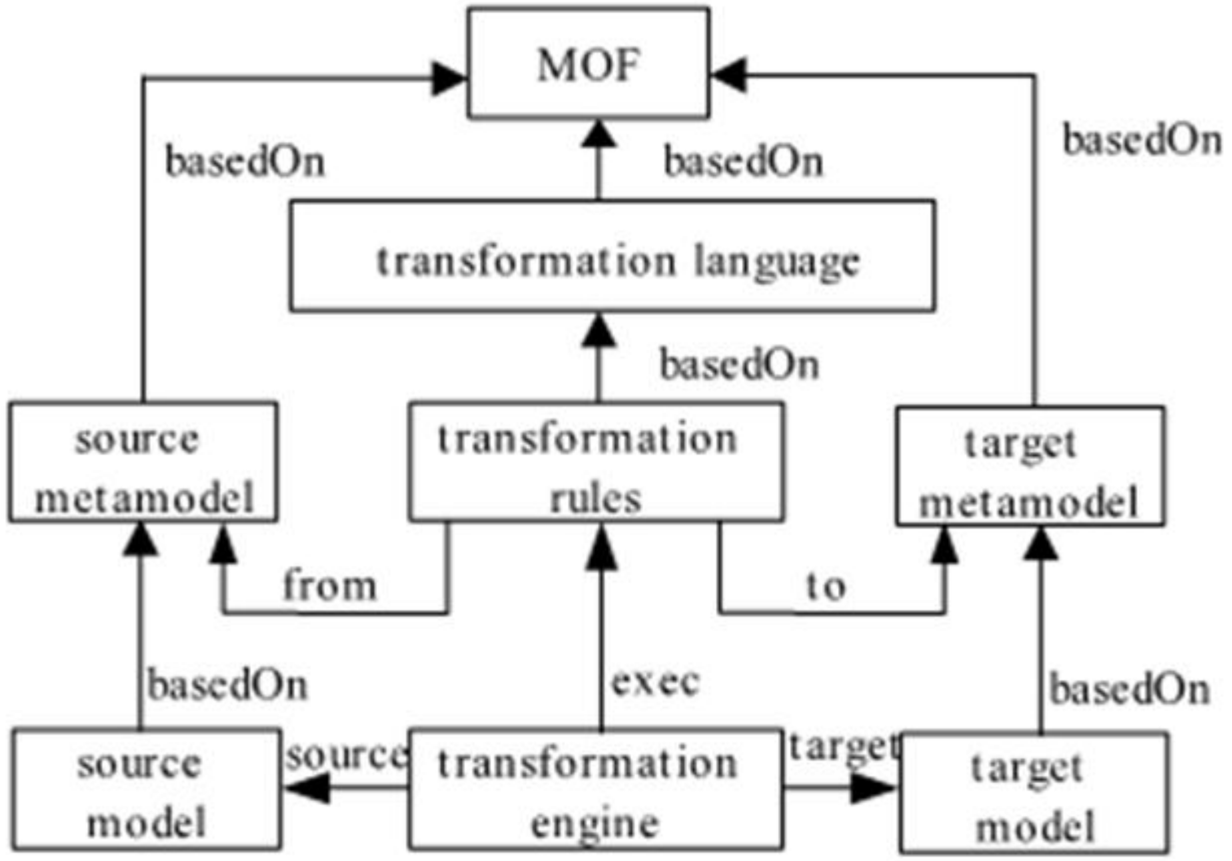
ModelElement
+ name : String
+ stereotype : String
Classifier
+ elementType
+ type
0..1
*
Typed
DataType
Class
Interface
*
+ implementedInterface
*
*
Set
AssociationClass
Feature
+ visibility : VisibilityKind
0..1
+ otherEnd
Association
*
+ end
AssociationEnd
+ lower : Lowerbound
+ upper : Upperbound
+ composition : Boolean
Attribute
Operation
+ paremeters
Paremeter

元对象设施 (Meta-Object Facility/MOF)





M3	MOF，即定义元模型的构造集合	MOF 类、MOF 属性、MOF 关联等
M2	元模型，由 MOF 构造的实例组成	UML 类、UML 属性、UML 状态、UML 活动等；CWM 表、CWM 列等
M1	模型，由 M2 元模型构造的实例组成	"Customer"类、"Account"类；"Employee"表、"Vendor"表等
M0	对象和数据，也即 M1 模型构造的实例	客户,Jane Smith、客户 Joe Jones、账户 2989、账户 2344、员工 A3949、商家 78988 等



Transformation in MDA

Sample

- UML→Java

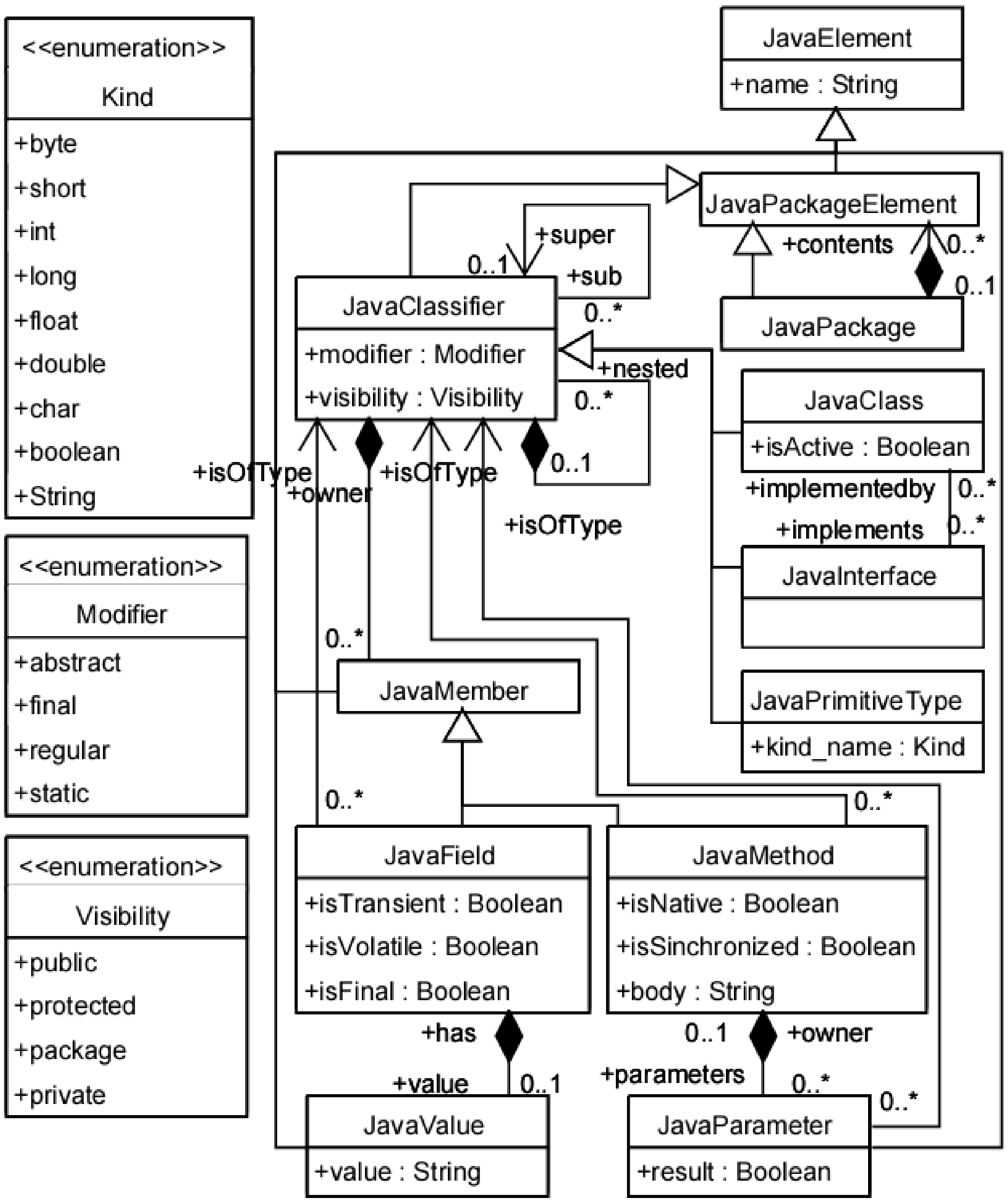


Figure 4 . Java Meta model

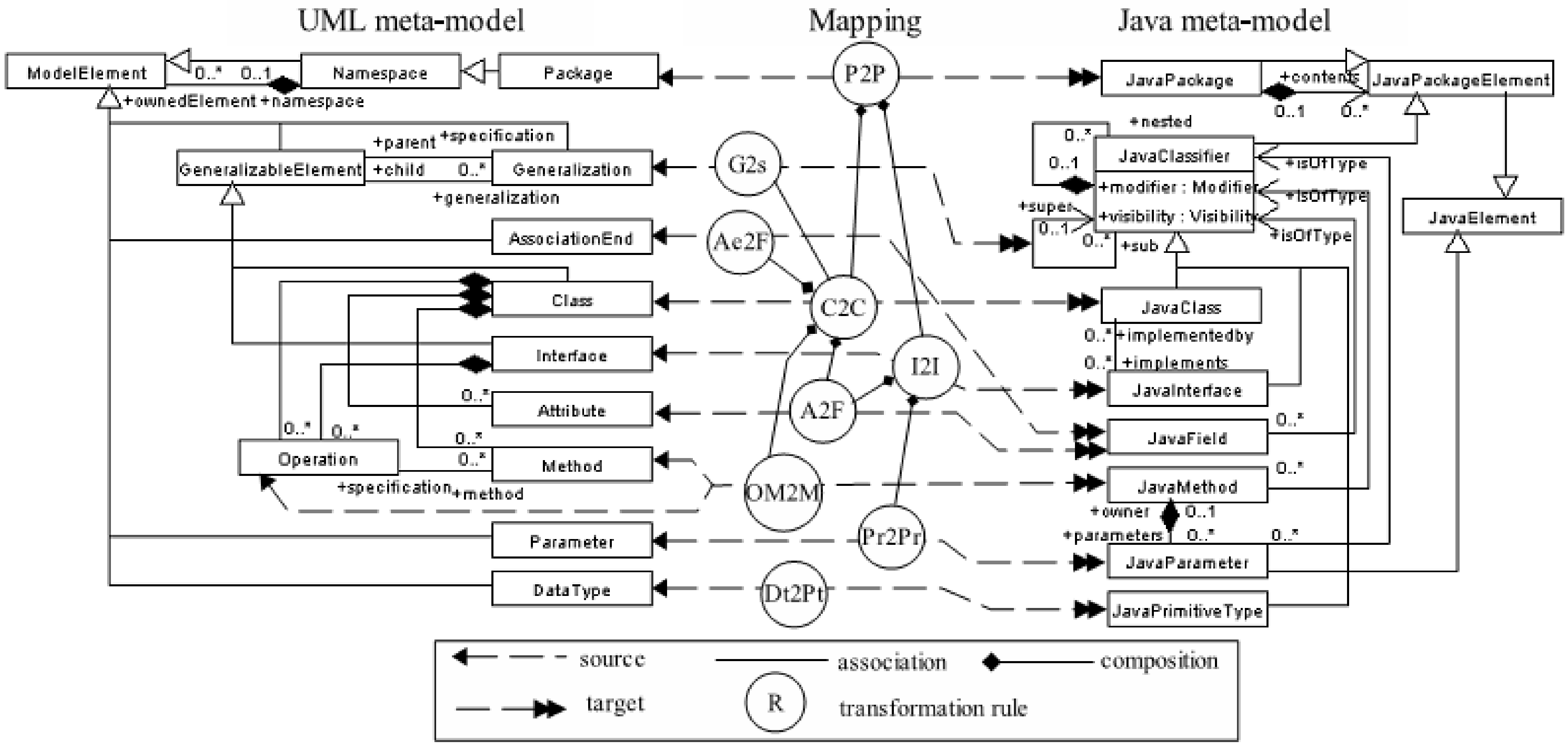
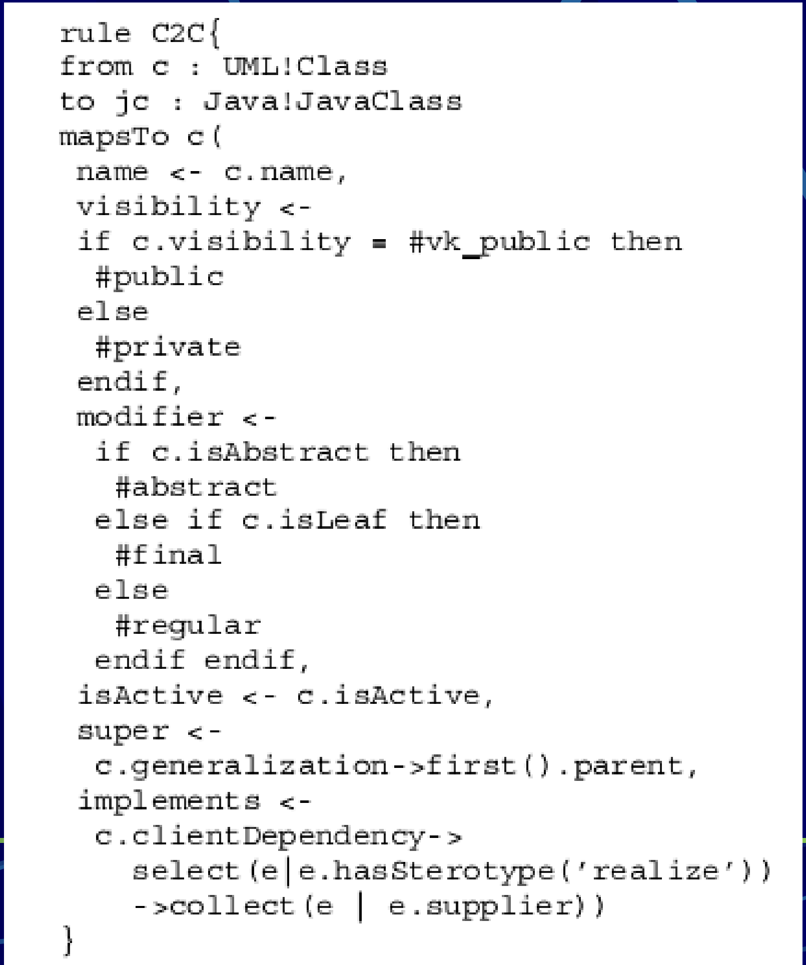


Figure 5. Mapping from UML to Java Meta-model (fragment)

- UML class is mapped to Java Class through the rule C2C



```
rule C2C{
from c : UML!Class
to jc : Java!JavaClass
mapsTo c(
 name <- c.name,
 visibility <-
 if c.visibility = #vk_public then
  #public
 else
  #private
 endif,
 modifier <-
  if c.isAbstract then
   #abstract
  else if c.isLeaf then
   #final
  else
   #regular
  endif endif,
 isActive <- c.isActive,
 super <-
  c.generalization->first().parent,
 implements <-
  c.clientDependency->
    select(e|e.hasSterotype('realize'))
    ->collect(e | e.supplier))
}
```

The Productivity Problem

- the fact that the code is the driving force of software development. The only phases in the development process that are really productive are coding and testing.

The Portability Problem

- Existing software is either ported to the new technology, or to a newer version of an existing technology
- Many companies need to follow these new technologies for good reasons:
  - The technology is demanded by the customers (e.g., Web interfaces).
  - It solves some real problems (e.g., XML for interchange or Java for portability).
  - Tool vendors stop supporting old technologies and focus on the new one (e.g., UML support replaces OMT).

The Interoperability Problem

- Web-based systems need to get information from existing back-end systems.
- A system uses Enterprise Java Beans (EJB), it also needs to use relational databases as a storage mechanism.、
- Components need to interact with each other

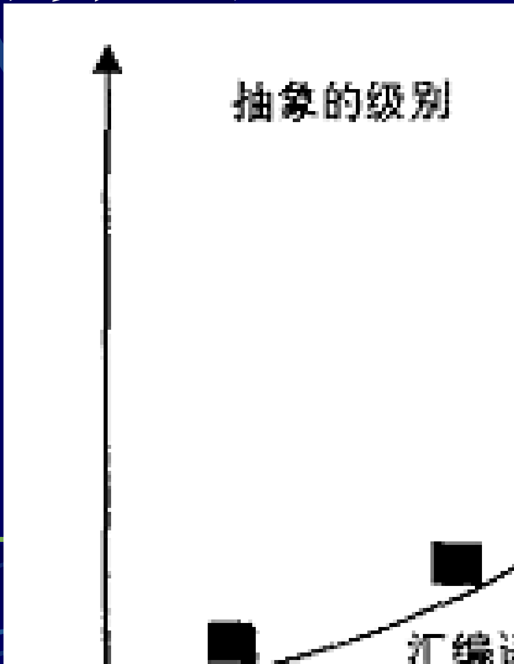
The Maintenance and Documentation Problem

- Most developers feel their main task is to produce code
- With every change in the code the documentation needs to be changed as well——by hand!

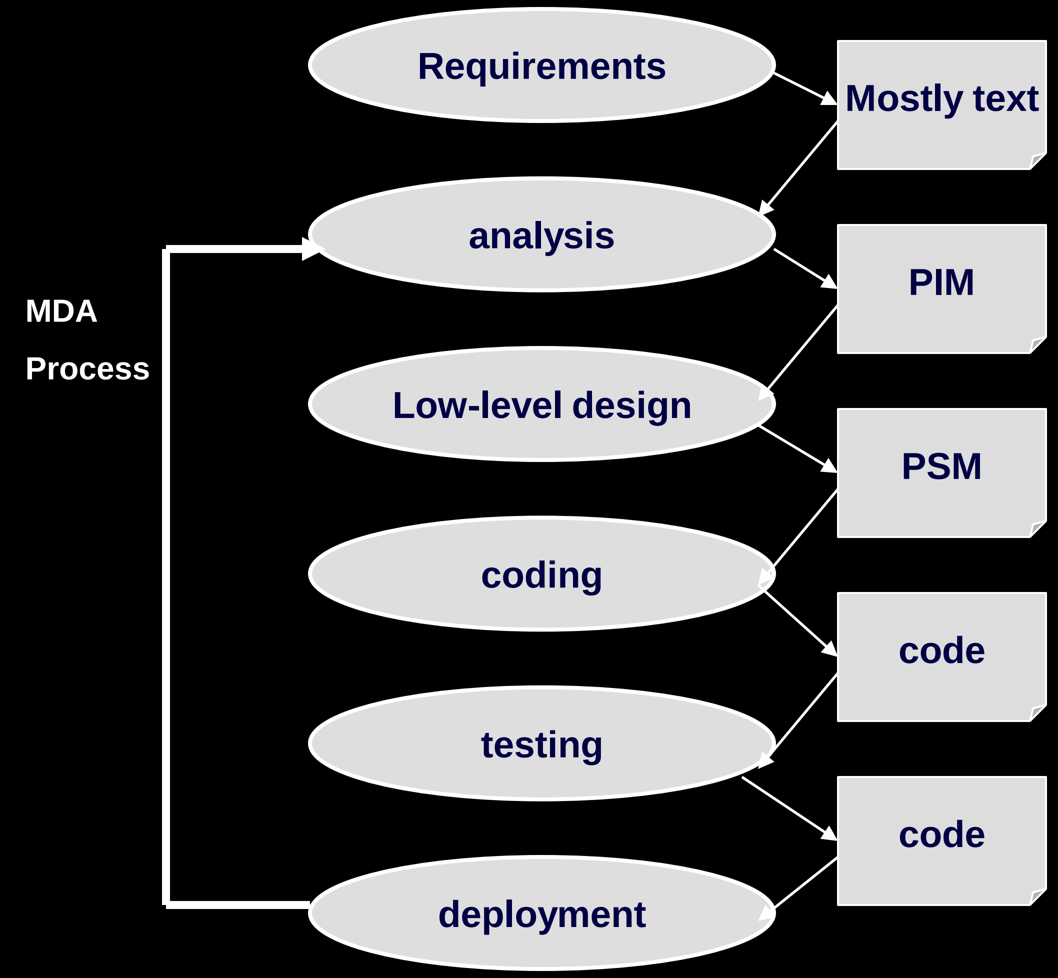
MDA

- 用模型来引导对系统的理解、设计、构造、配置、维护、改进的整个过程。模型不再是一种辅助工具，而是一个产品，一种编程语言
- 模型必须是以精确定义的语言表述的。精确定义的语言是具有精确语法和语义的语言，可以由机器自动解释

- 提升了软件开发的抽象层次并分离出了两个抽象级别的模型，PIM是一个纯粹的不考虑实现技术的分析模型，而PSM可以视为是一个基于特定的实现技术的模型，通过MDA工具把针对特定计算平台的编码工作交由机器自动完成



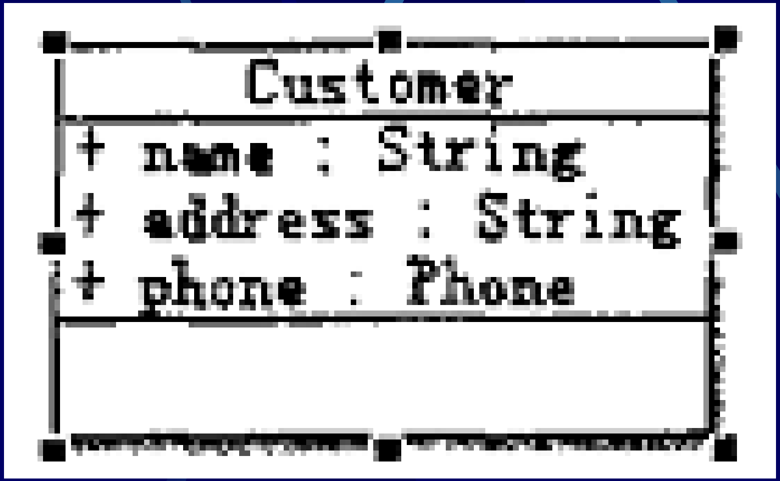
MDA software development life cycle

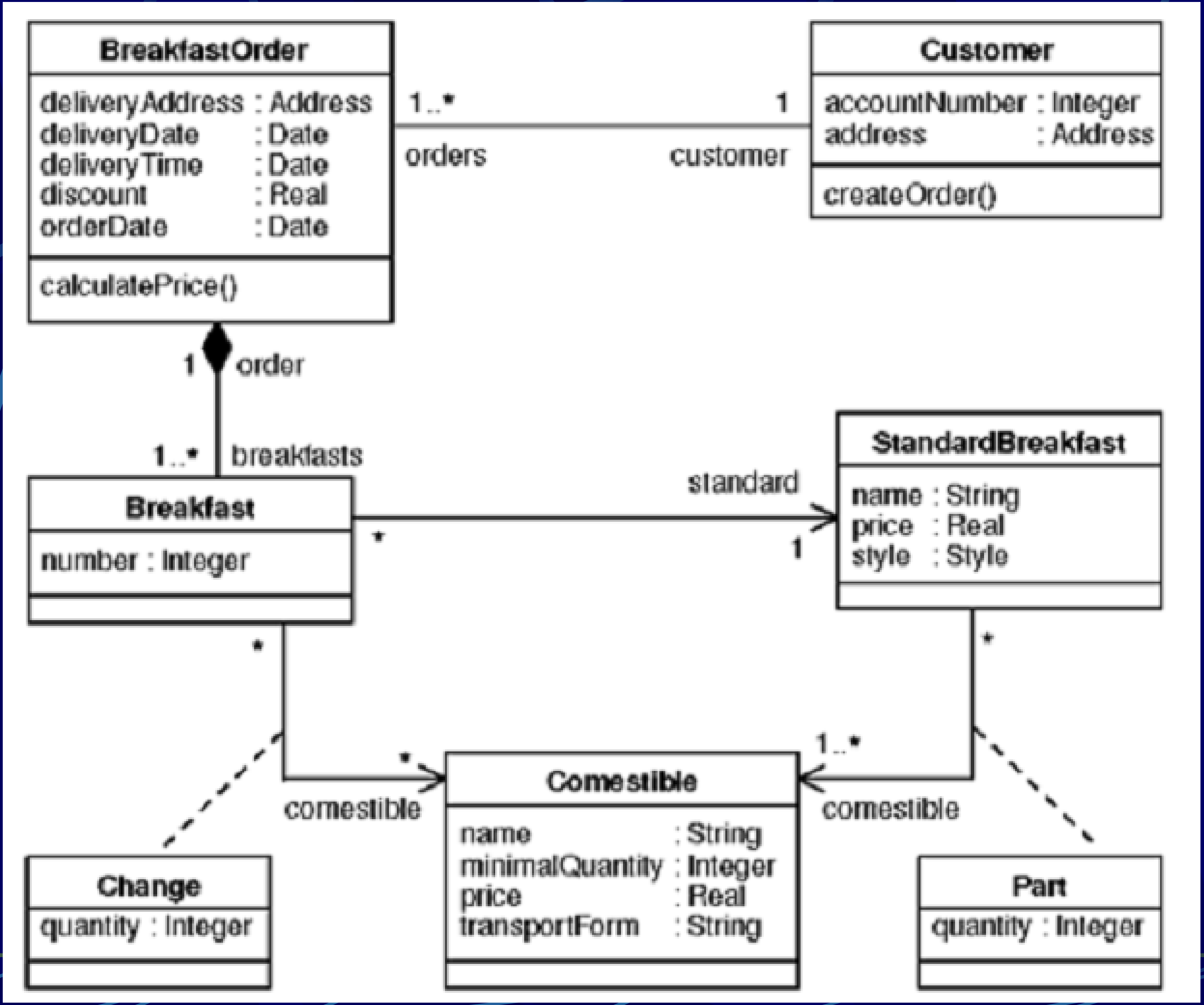


Source: Kleppe et al. MDA Explained: The model Driven Architecture: practice and Promise

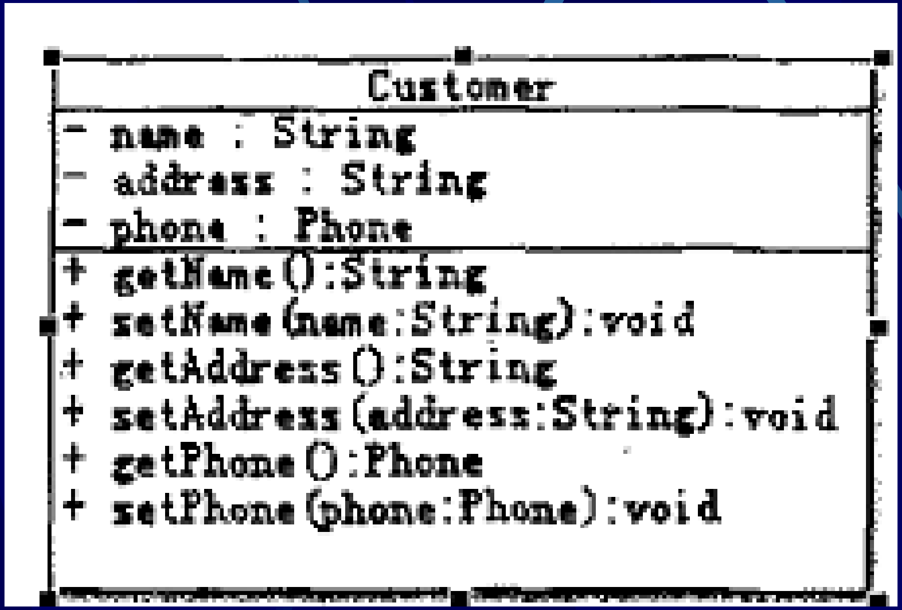
MDA中的模型

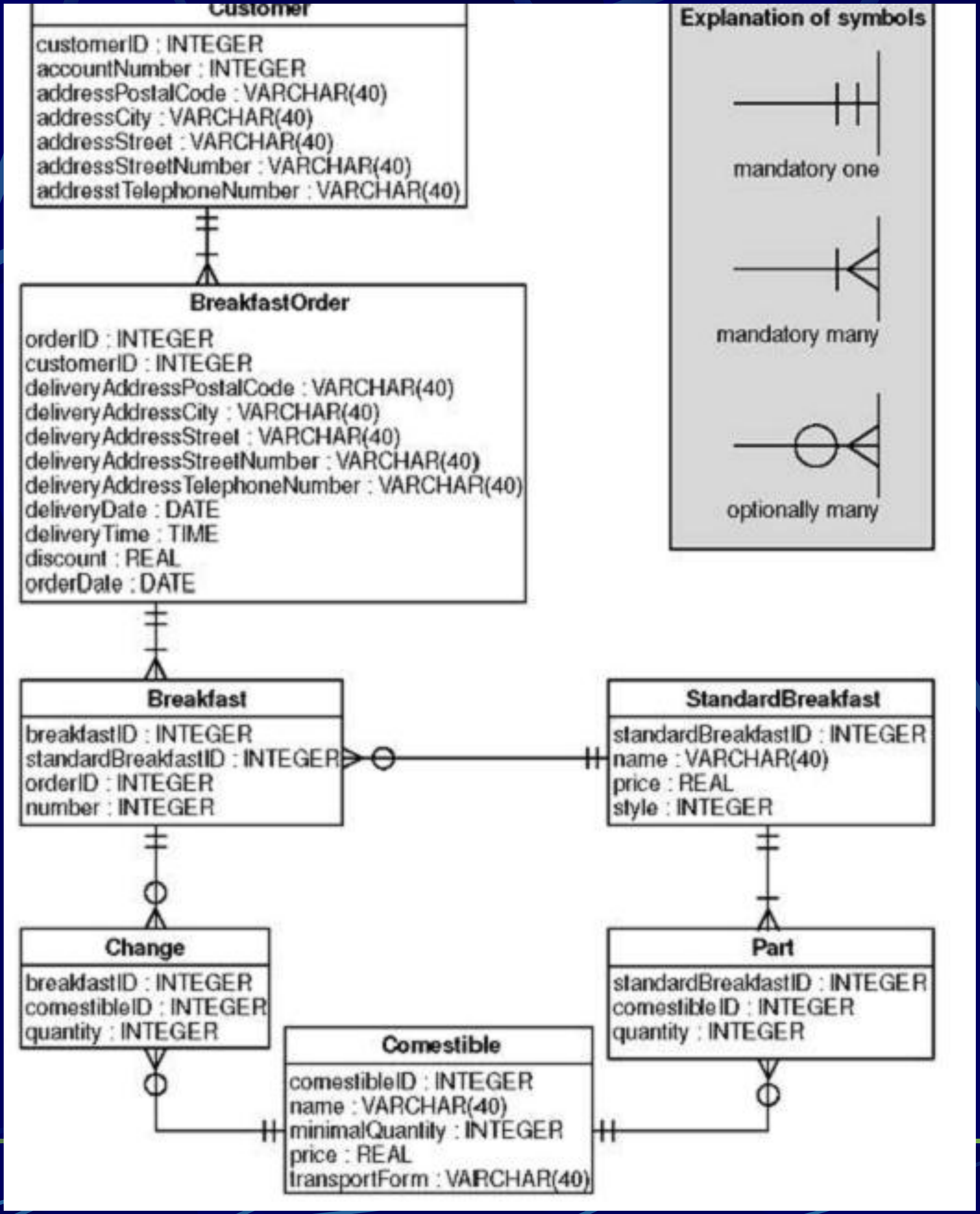
- 平台无关模型 platform-independent **model** (PIM)

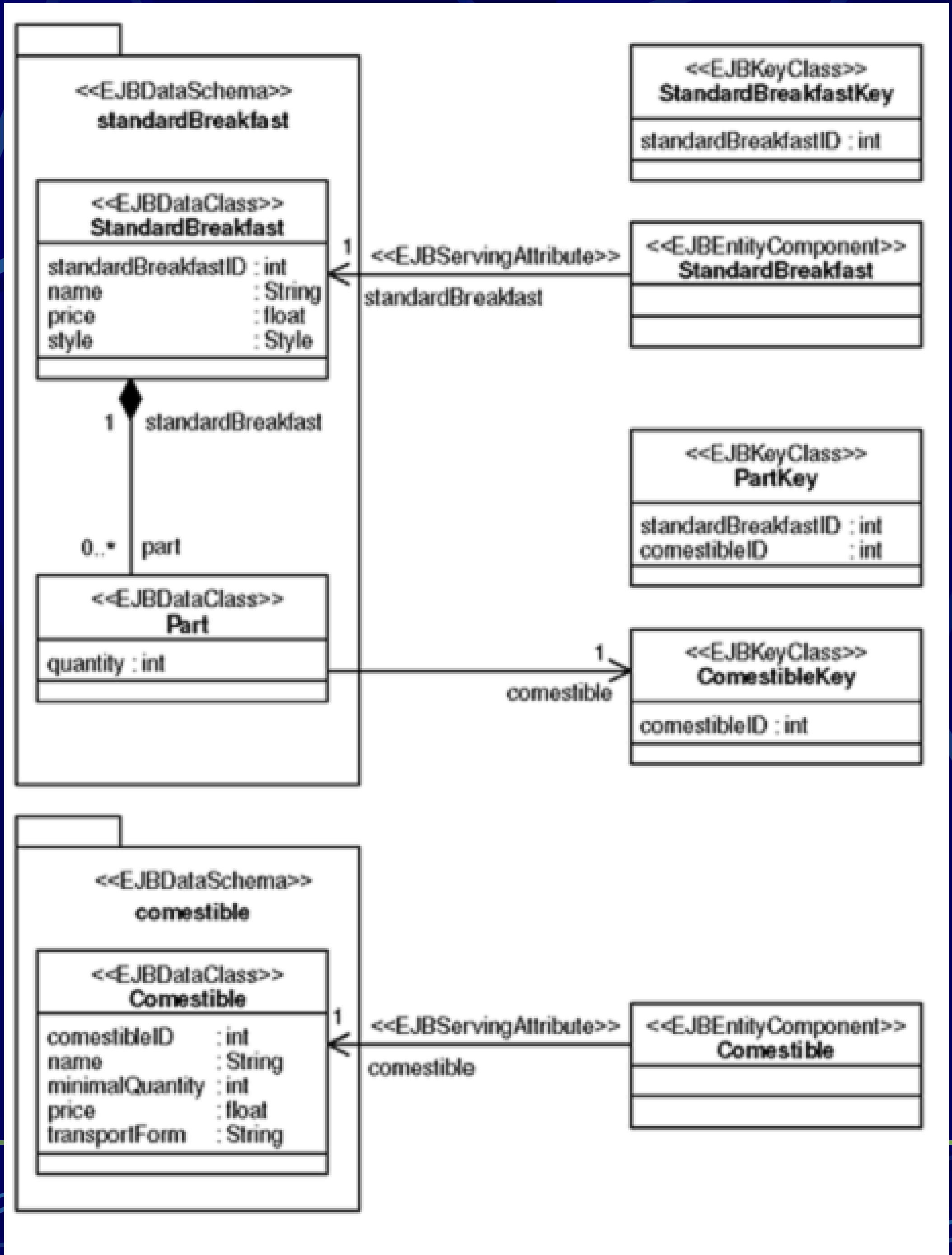



BreakfastOrder
deliveryAddress : Address
deliveryDate : Date
deliveryTime : Date
discount : Real
orderDate : Date
calculatePrice()
1..*
orders
1
customer
Customer
accountNumber : Integer
address : Address
createOrder()
1
order
1..*
breakfasts
Breakfast
number : Integer
*
standard
1
StandardBreakfast
name : String
price : Real
style : Style
*
*
comestible
*
1..*
comestible
Comestible
name : String
minimalQuantity : Integer
price : Real
transportForm : String
Change
quantity : Integer
Part
quantity : Integer

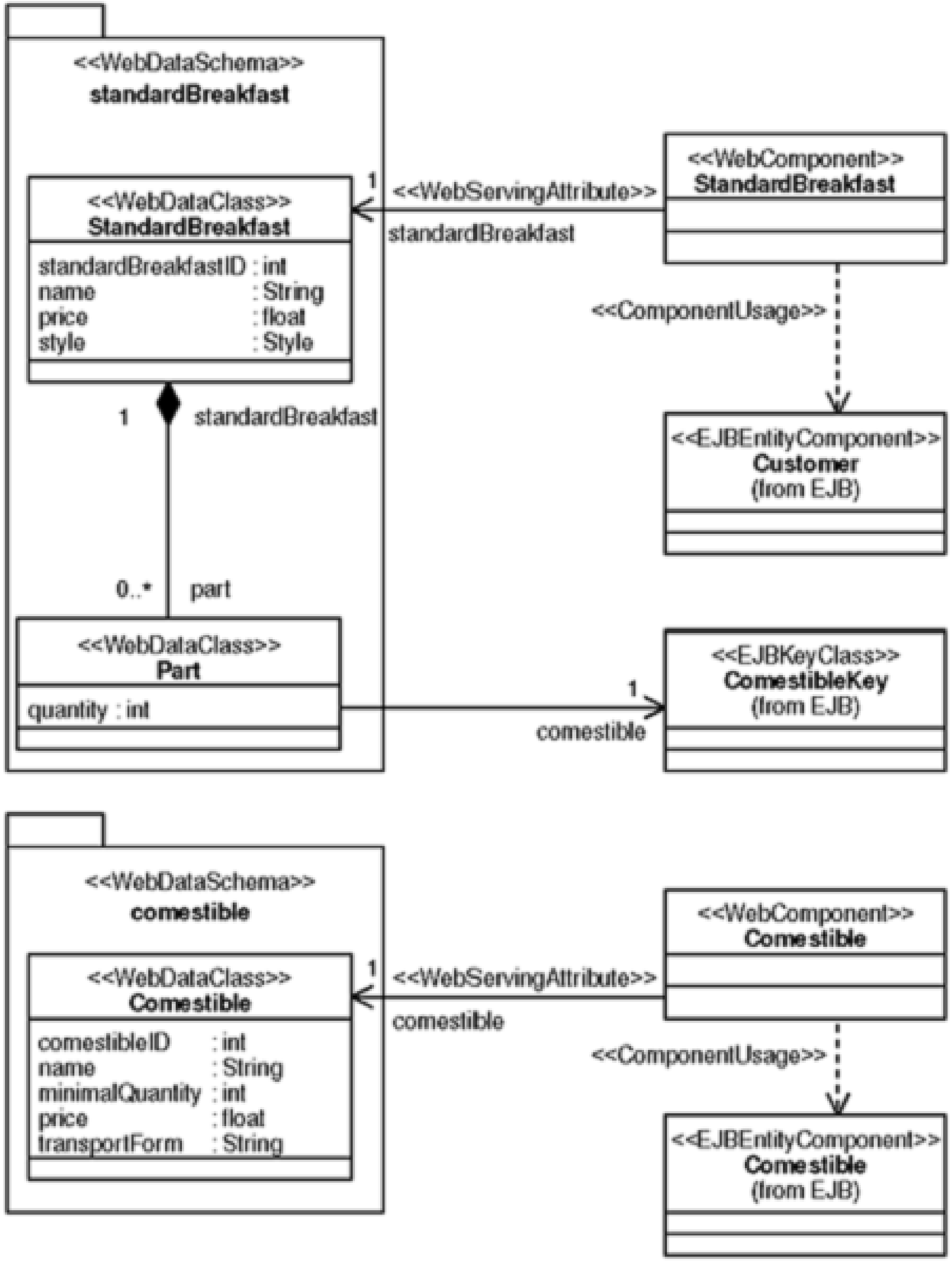
- 平台相关模型platform-specific-model (PSM)




Customer
customerID : INTEGER
accountNumber : INTEGER
addressPostalCode : VARCHAR(40)
addressCity : VARCHAR(40)
addressStreet : VARCHAR(40)
addressStreetNumber : VARCHAR(40)
addresstTelephoneNumber : VARCHAR(40)
Explanation of symbols
mandatory one
mandatory many
optionally many
BreakfastOrder
orderID : INTEGER
customerID : INTEGER
deliveryAddressPostalCode : VARCHAR(40)
deliveryAddressCity : VARCHAR(40)
deliveryAddressStreet : VARCHAR(40)
deliveryAddressStreetNumber : VARCHAR(40)
deliveryAddressTelephoneNumber : VARCHAR(40)
deliveryDate : DATE
deliveryTime : TIME
discount : REAL
orderDate : DATE
Breakfast
breakfastID : INTEGER
standardBreakfastID : INTEGER
orderID : INTEGER
number : INTEGER
StandardBreakfast
standardBreakfastID : INTEGER
name : VARCHAR(40)
price : REAL
style : INTEGER
Change
breakfastID : INTEGER
comestibleID : INTEGER
quantity : INTEGER
Part
standardBreakfastID : INTEGER
comestible ID : INTEGER
quantity : INTEGER
Comestible
comestibleID : INTEGER
name : VARCHAR(40)
minimalQuantity : INTEGER
price : REAL
transportForm : VARCHAR(40)



<<EJBDataSchema>>
standardBreakfast
<<EJBDataClass>>
StandardBreakfast
standardBreakfastID : int
name : String
price : float
style : Style
1
standardBreakfast
0..*
part
<<EJBDataClass>>
Part
quantity : int
1
<<EJBServingAttribute>>
standardBreakfast
<<EJBKeyClass>>
StandardBreakfastKey
standardBreakfastID : int
<<EJBEntityComponent>>
StandardBreakfast
<<EJBKeyClass>>
PartKey
standardBreakfastID : int
comestibleID : int
1
comestible
<<EJBKeyClass>>
ComestibleKey
comestibleID : int
<<EJBDataSchema>>
comestible
<<EJBDataClass>>
Comestible
comestibleID : int
name : String
minimalQuantity : int
price : float
transportForm : String
1
<<EJBServingAttribute>>
comestible
<<EJBEntityComponent>>
Comestible


<<WebDataSchema>>
standardBreakfast
<<WebDataClass>>
StandardBreakfast
standardBreakfastID : int
name : String
price : float
style : Style
1
<<WebServingAttribute>>
standardBreakfast
<<WebComponent>>
StandardBreakfast
<<ComponentUsage>>
<<EJBEntityComponent>>
Customer
(from EJB)
1
standardBreakfast
0..*
part
<<WebDataClass>>
Part
quantity : int
1
comestible
<<EJBKeyClass>>
ComestibleKey
(from EJB)
<<WebDataSchema>>
comestible
<<WebDataClass>>
Comestible
comestibleID : int
name : String
minimalQuantity : int
price : float
transportForm : String
1
<<WebServingAttribute>>
comestible
<<WebComponent>>
Comestible
<<ComponentUsage>>
<<EJBEntityComponent>>
Comestible
(from EJB)

模型映射 (mapping)

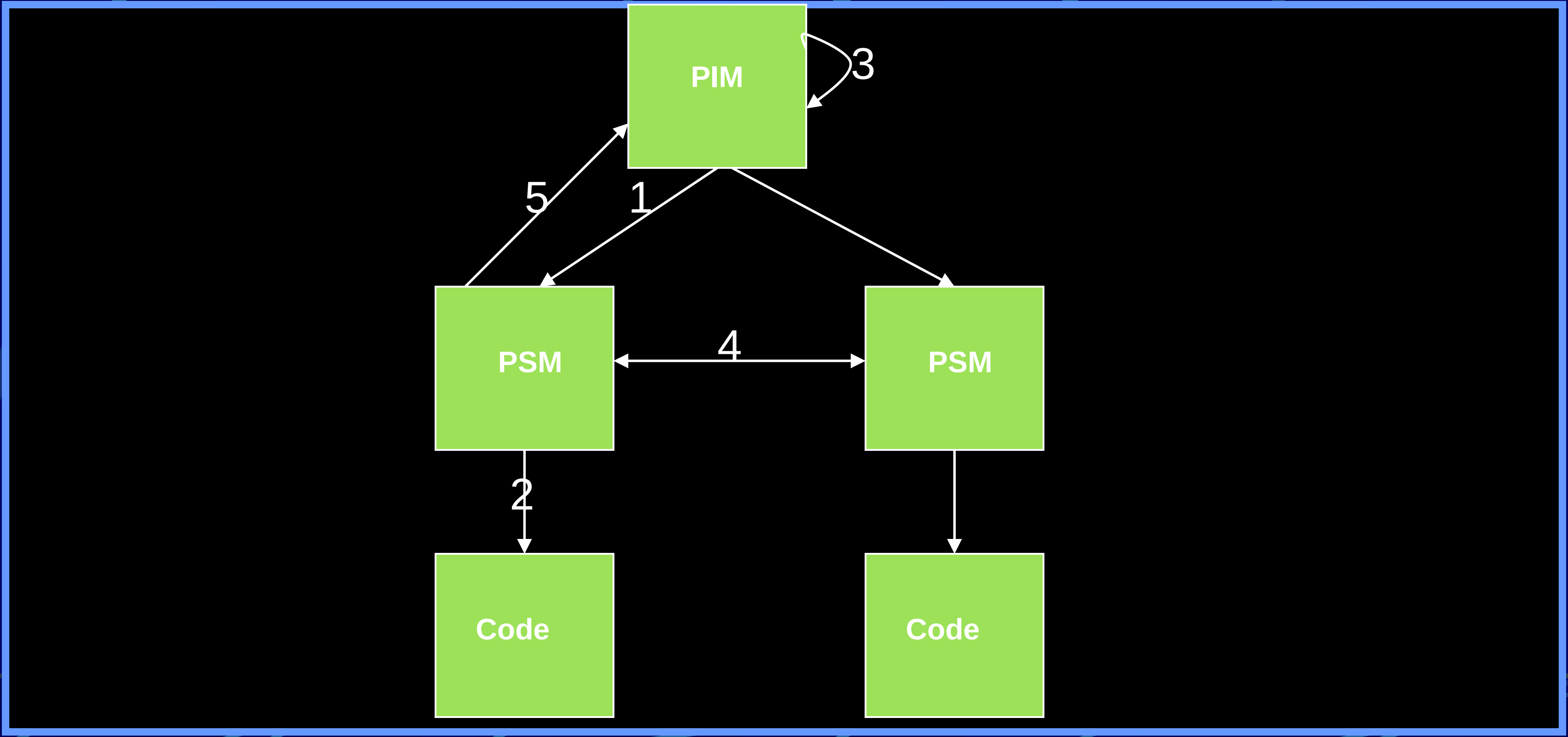
- MDA中的模型转换是基于元模型的。映射就是将一个模型变换为另一个模型的规则和技术集

Transformation types

1. PIM to PSM : models are transformed by adding platform *artifacts that are specific to a given platform.*
2. PSM to Code : Code is generated from a platform-specific model

Others mappings

3. PIM to PIM : models refinement without platform information
4. PSM to PSM : models refinement that needs platform information
5. PSM to PIM : Abstracting existing models to a platform independent model. (Reverse-engineering)



Source: Kleppe et al. MDA Explained: The model Driven Architecture: practice and Promise

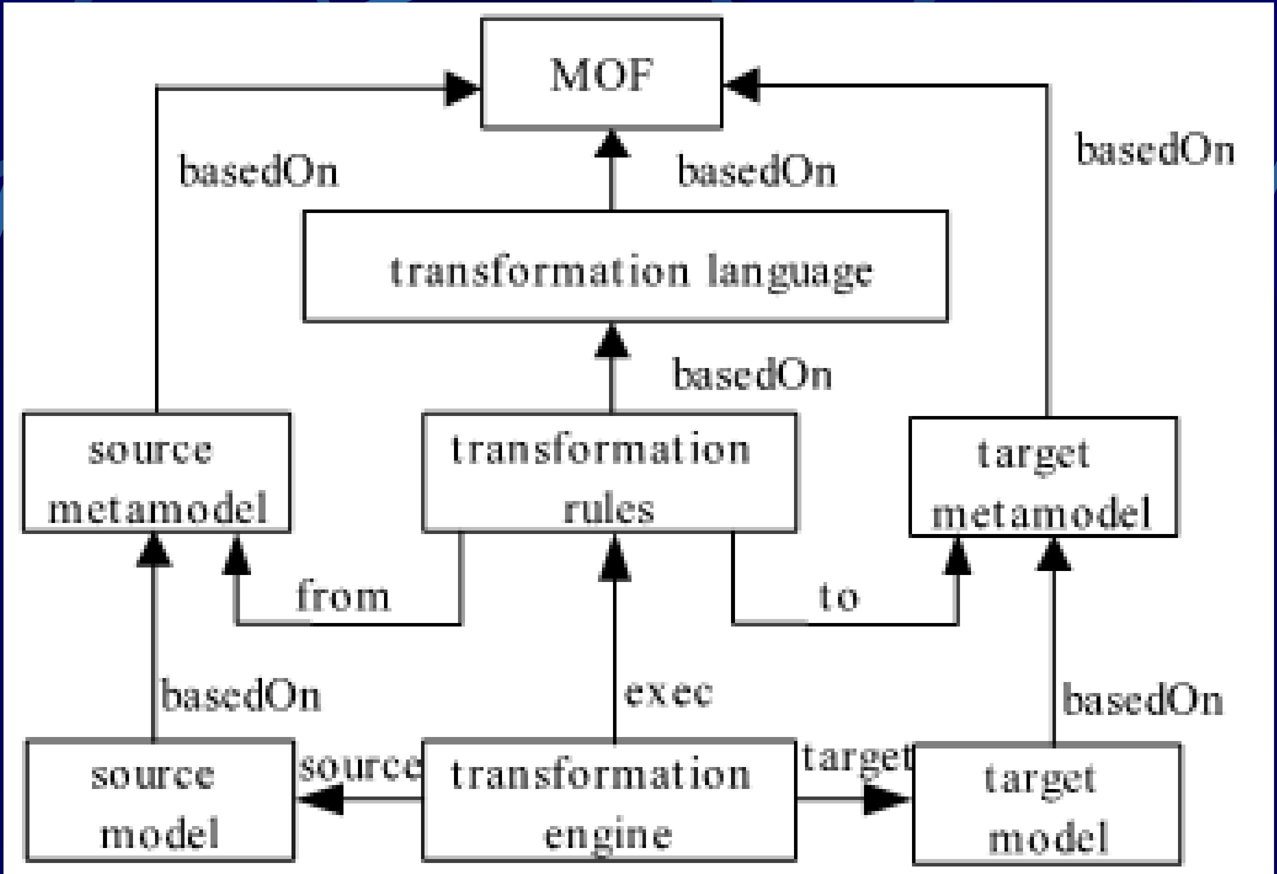


Figure 1. Transformation in MDA

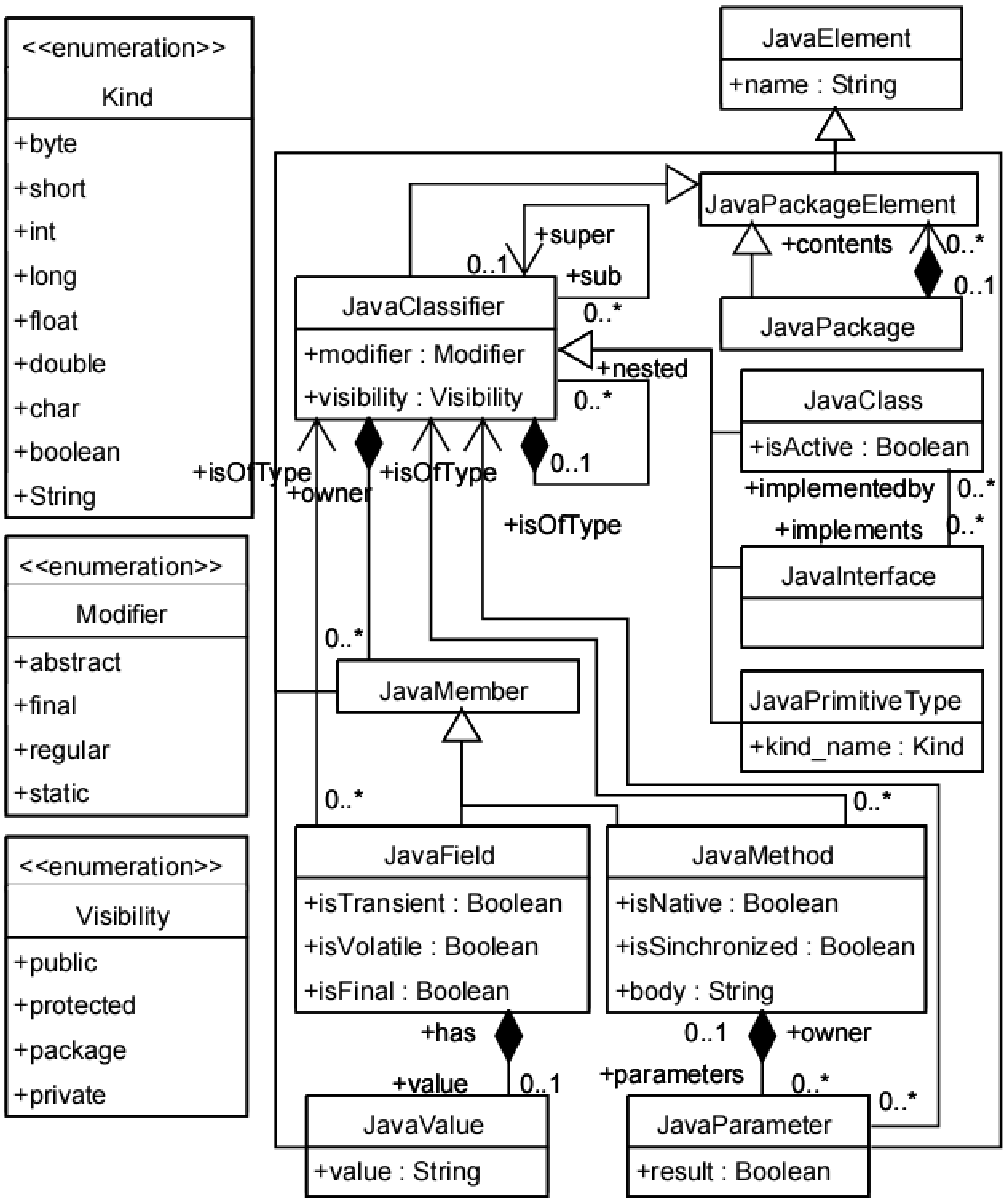
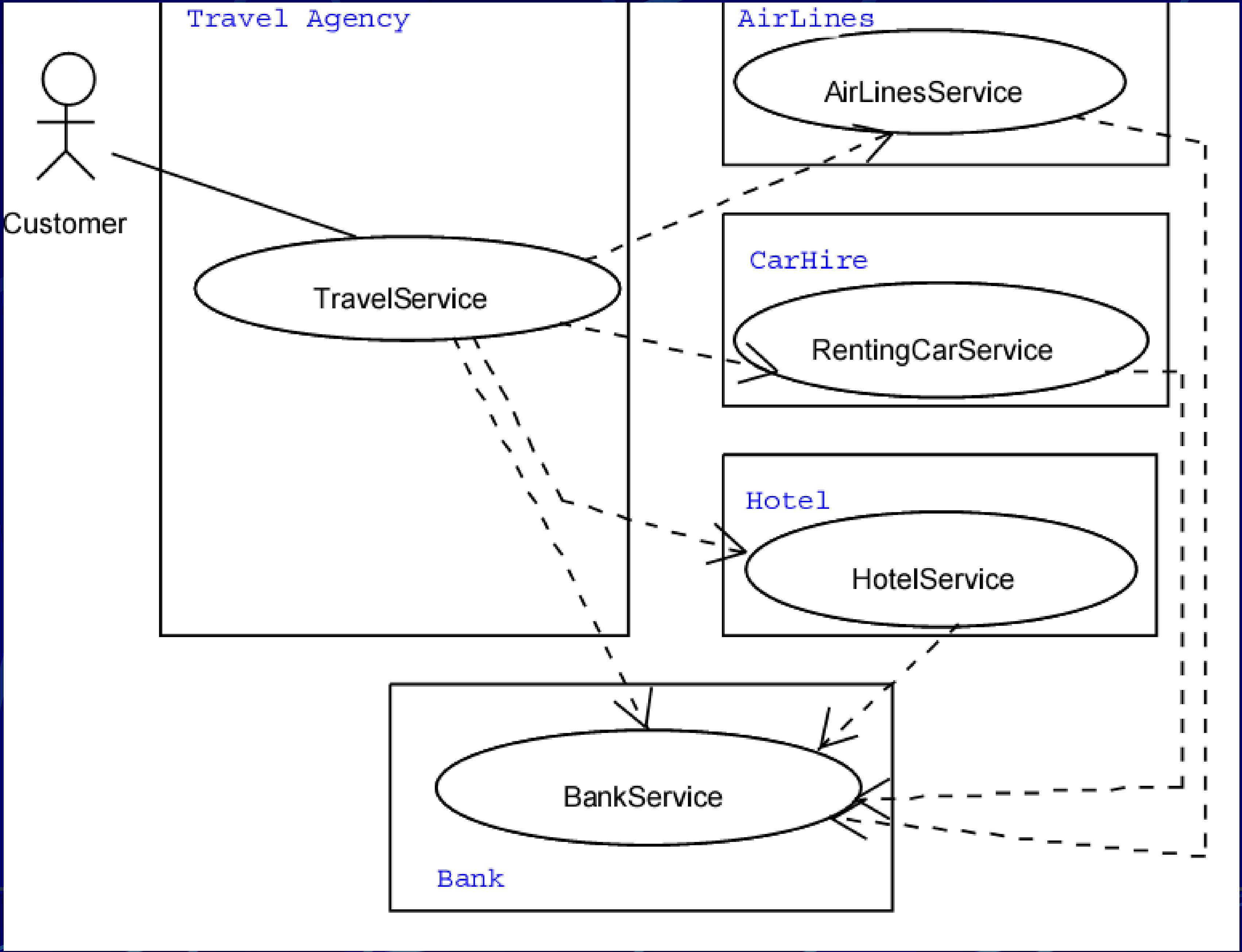
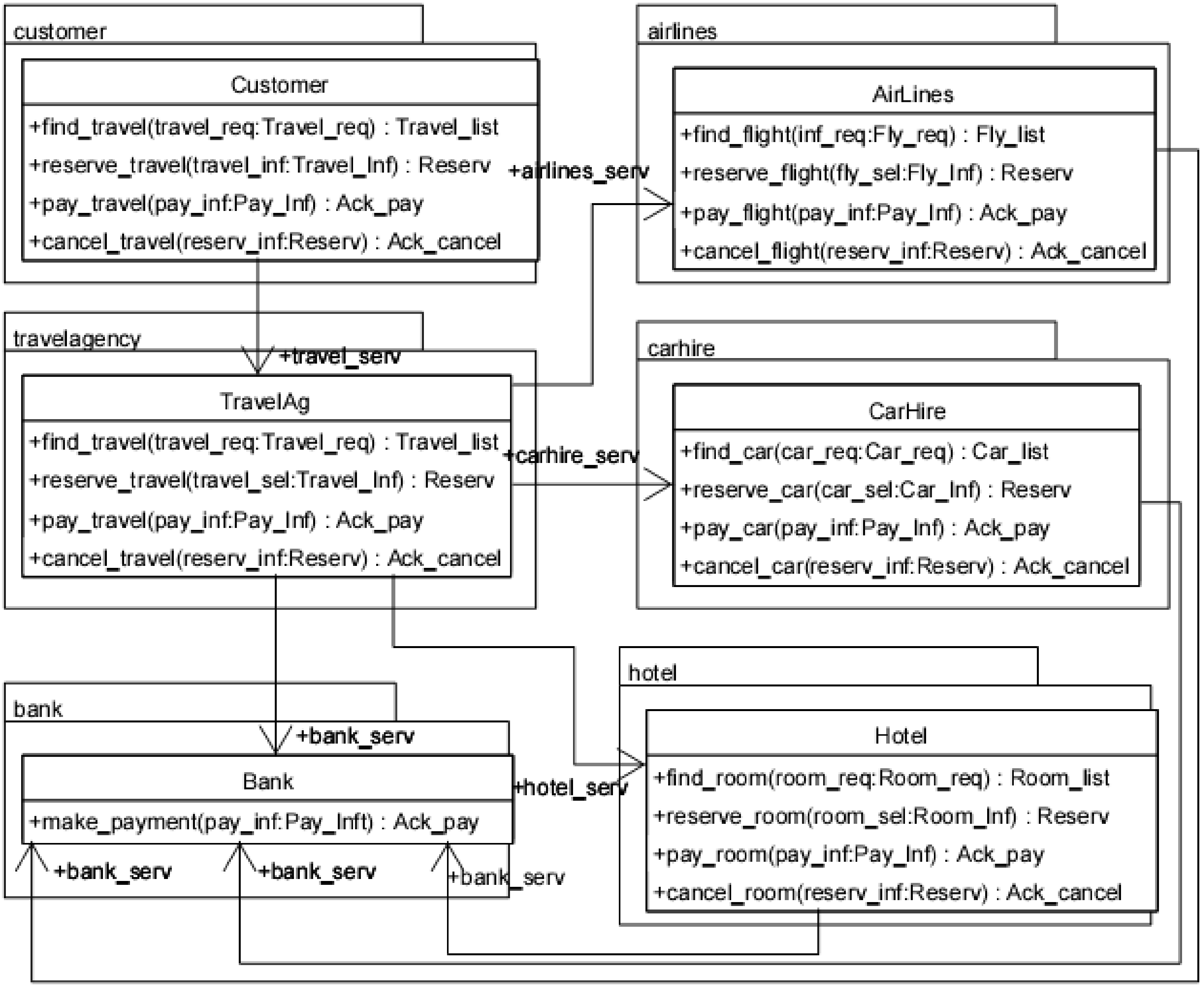


Figure 4 . Java Meta model


Customer
Travel Agency
TravelService
AirLines
AirLinesService
CarHire
RentingCarService
Hotel
HotelService
BankService
Bank


customer
Customer
+find_travel(travel_req:Travel_req) : Travel_list
+reserve_travel(travel_inf:Travel_Inf) : Reserv
+pay_travel(pay_inf:Pay_Inf) : Ack_pay
+cancel_travel(reserv_inf:Reserv) : Ack_cancel
airlines
AirLines
+find_flight(inf_req:Fly_req) : Fly_list
+reserve_flight(fly_sel:Fly_Inf) : Reserv
+pay_flight(pay_inf:Pay_Inf) : Ack_pay
+cancel_flight(reserv_inf:Reserv) : Ack_cancel
+airlines_serv
travelagency
+travel_serv
TravelAg
+find_travel(travel_req:Travel_req) : Travel_list
+reserve_travel(travel_sel:Travel_Inf) : Reserv
+pay_travel(pay_inf:Pay_Inf) : Ack_pay
+cancel_travel(reserv_inf:Reserv) : Ack_cancel
carhire
CarHire
+find_car(car_req:Car_req) : Car_list
+reserve_car(car_sel:Car_Inf) : Reserv
+pay_car(pay_inf:Pay_Inf) : Ack_pay
+cancel_car(reserv_inf:Reserv) : Ack_cancel
+carhire_serv
bank
+bank_serv
Bank
+make_payment(pay_inf:Pay_Inft) : Ack_pay
+hotel_serv
hotel
Hotel
+find_room(room_req:Room_req) : Room_list
+reserve_room(room_sel:Room_Inf) : Reserv
+pay_room(pay_inf:Pay_Inf) : Ack_pay
+cancel_room(reserv_inf:Reserv) : Ack_cancel
+bank_serv
+bank_serv
+bank_serv

From PIM(UML) to PSM(Java)

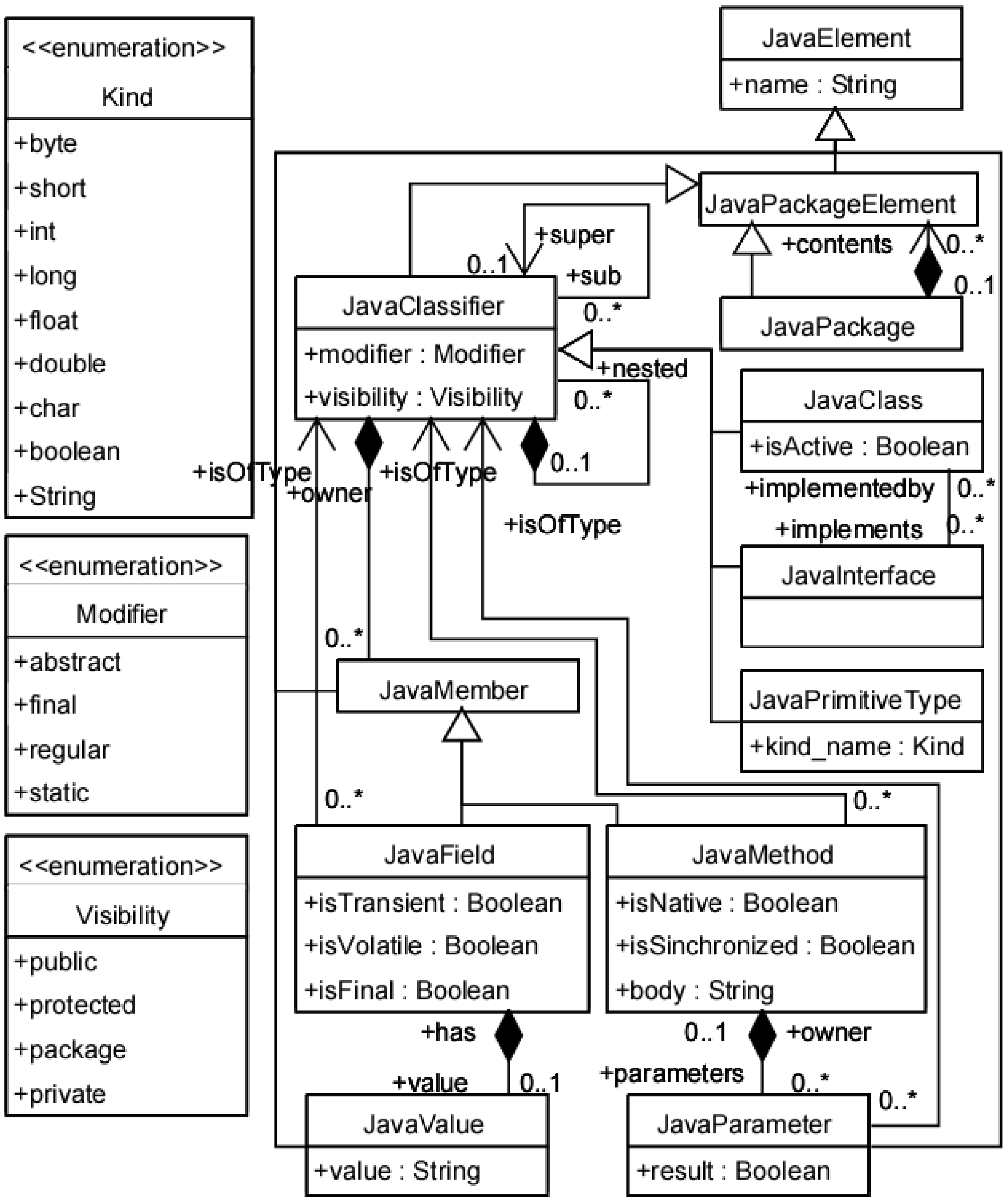


Figure 4 . Java Meta model

- The model transformation needs to determine what elements of the UML meta-model will be mapped to the elements of the Java meta-model.

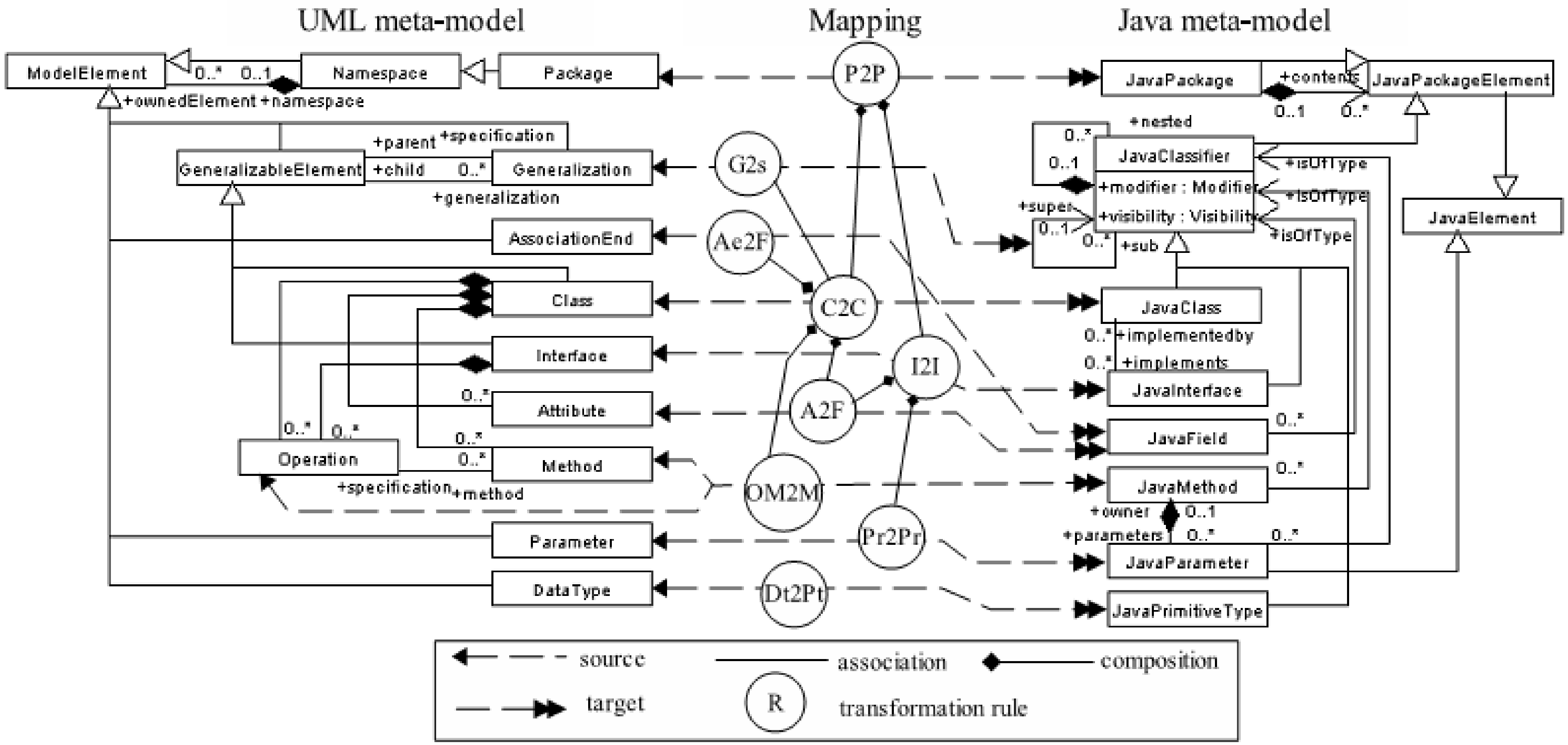
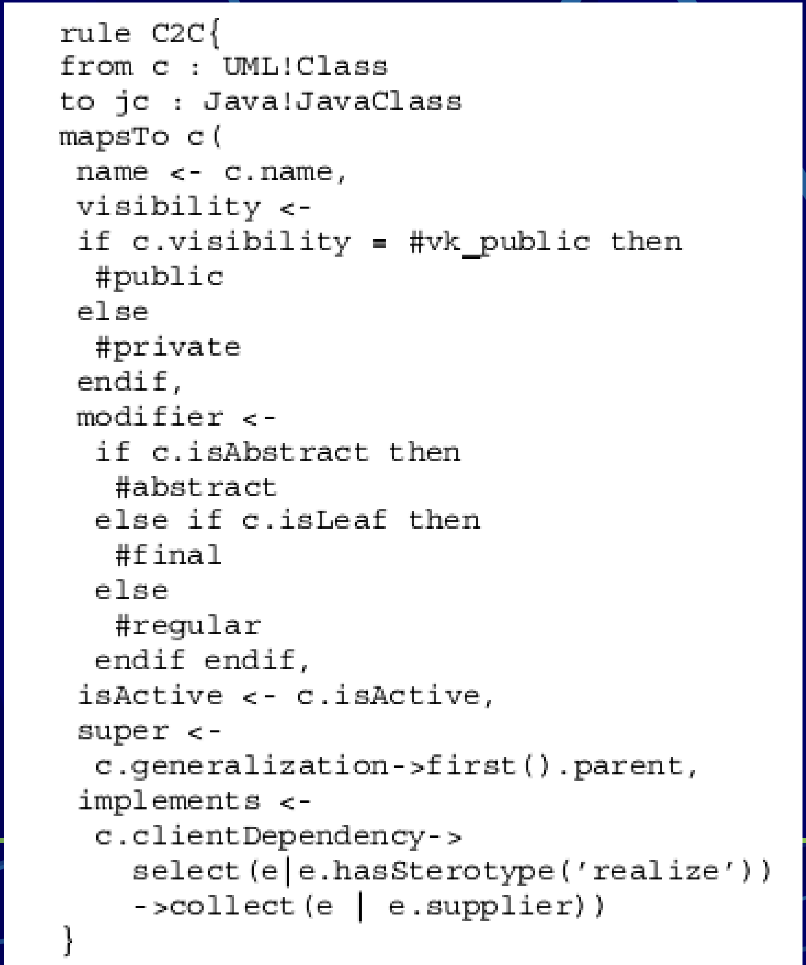


Figure 5. Mapping from UML to Java Meta-model (fragment)

create transformation rule

- eXtensible Stylesheet Language Transformation (XSLT)
- Logical languages
- ATL
- other transformation language based on Object Constraint Language(OCL) as ATL.

- UML class is mapped to Java Class through the rule C2C



```
rule C2C{
from c : UML!Class
to jc : Java!JavaClass
mapsTo c(
 name <- c.name,
 visibility <-
 if c.visibility = #vk_public then
  #public
 else
  #private
 endif,
 modifier <-
  if c.isAbstract then
   #abstract
  else if c.isLeaf then
   #final
  else
   #regular
  endif endif,
 isActive <- c.isActive,
 super <-
  c.generalization->first().parent,
 implements <-
  c.clientDependency->
    select(e|e.hasSterotype('realize'))
    ->collect(e | e.supplier))
}
```

From PIM(UML)to PSM(Web Service)

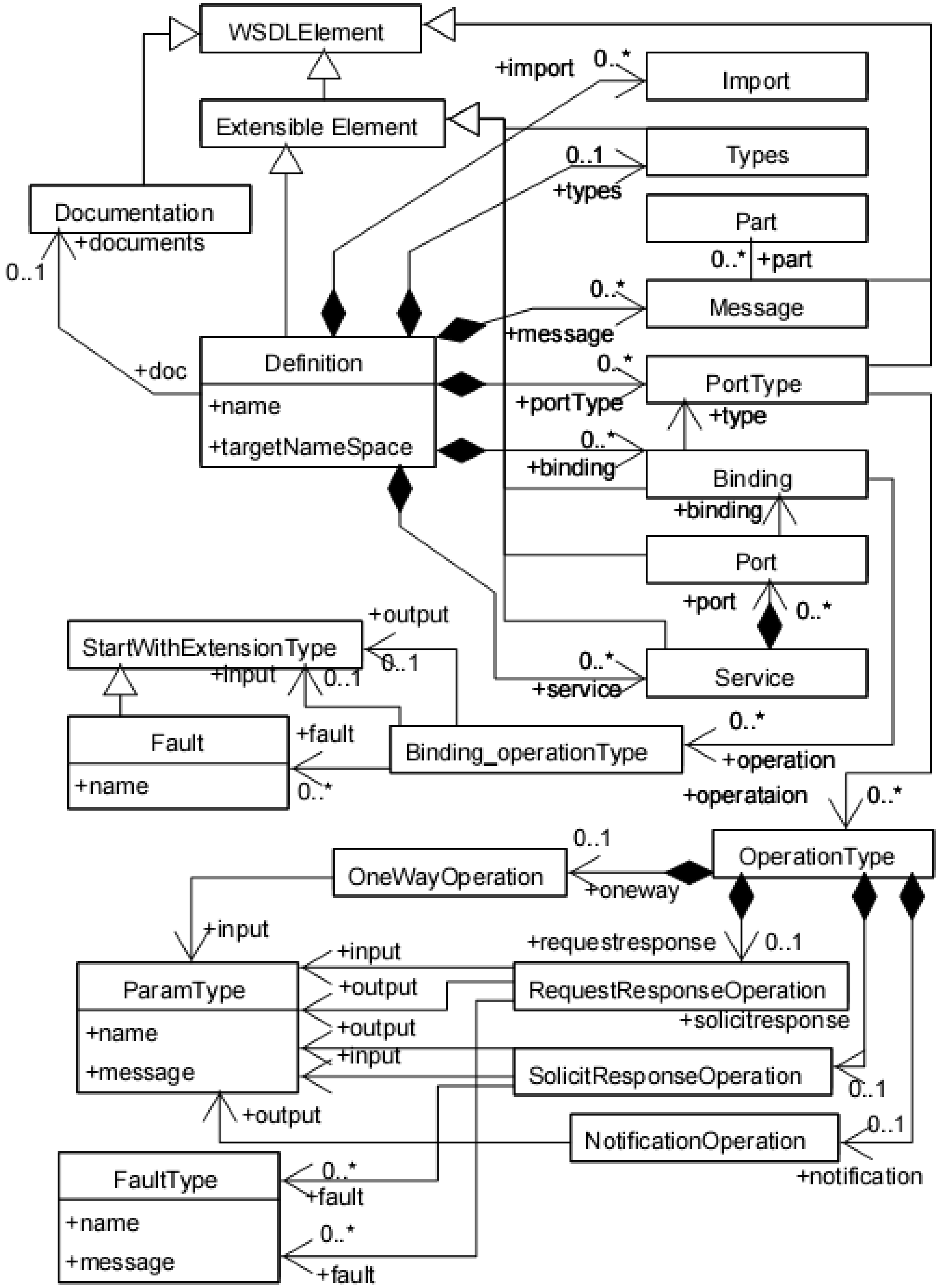


Figure 6. WSDL Meta-model

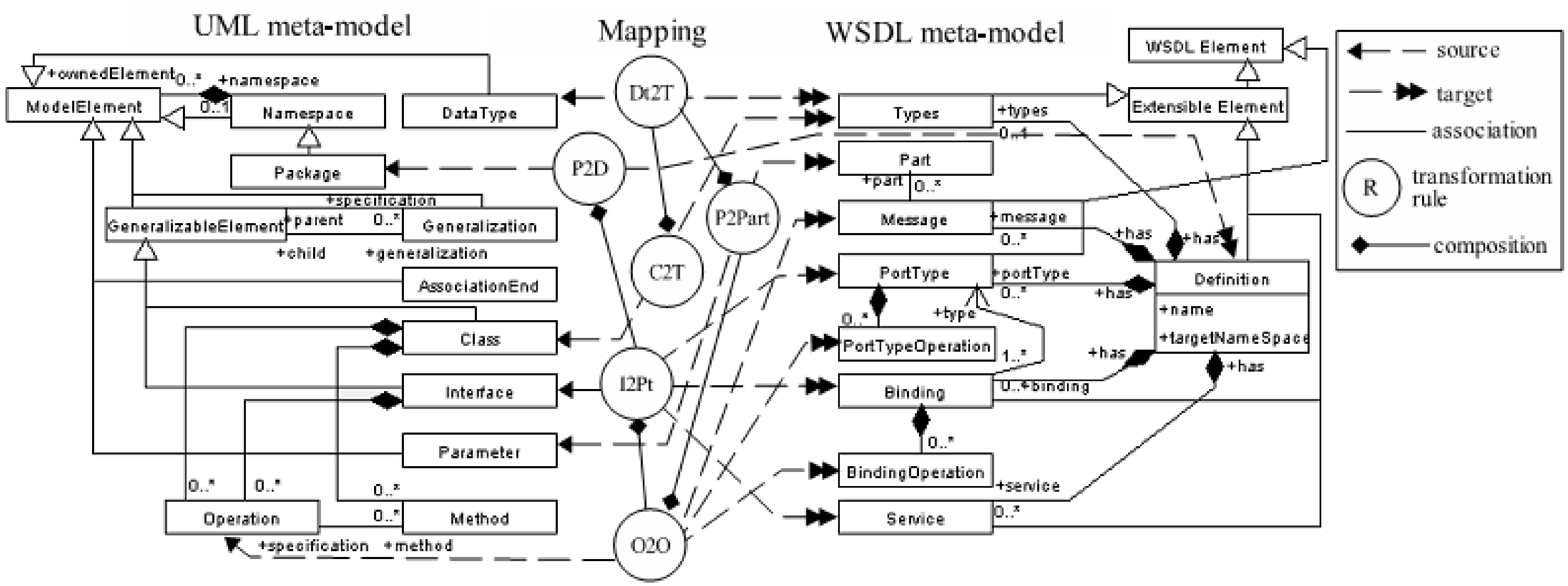


Figure 7. Mapping from UML to WSDL Meta-model (fragment)

MDA的优势-效率

- 具体应用程序的开发人员的注意力转移到开发PIM上
  - 问题：很多时候仍然需要做很多复杂的工作定义额外的信息来指导模型的转换

- MDA工具的开发人员定义变换规则
  - 一个PIM可以用于不同的技术平台，实现了系统设计复用;支持MDA的工具可以用于各个具体应用软件的开发。并且对于PSM和代码来说，开发人员只需要手工编写
  - 问题：定义变换规则的工作量是比较大的

MDA的优势-可移植性

- 对于新出现的技术，软件工业将会开发相应的变换规则。这使得我们迅速地将PIM变换到新的技术平台。

MDA的优势- 互操作性

- 如果我们能够把一个PIM变换成为两个不同平台上的PSM，则可以知道一个PSM中的每个模型元素是从PIM中的哪个元素变换而来的，还知道PIM中这个元素对应于另一个PSM中的哪个元素。因此，我们可以推断出两个PIM中元素的对应关系
- 因而可以在产生PSM的同时产生PSM桥接器。桥接器使我们可以在两个PSM之间进行变换;也可以定义一种新的模型变换，使生成的PSM跨越两个平台，为不同平台上的子系统提供互操作性

MDA的优势-维护和文档

- PIM完全可以当作设计文档来对待。与传统的文档相比，一个很大的不同就是当完成PIM的创建之后，PIM不会像传统的文档那样被“抛弃”。
- 对系统的改变首先是要更改PIM，再从PIM重新生成PSM和代码，代码和文档总是保持同步。

MDA的优势-软件质量

- 从模型中自动生成实现代码，将开发者从代码中解放出来，他们可以花更多的时间去解决商业逻辑的问题
- 代码生成器是由经验丰富的程序员开发的，确保产生高质量的代码，同时也避免了开发人员水平参差不齐的问题。