

模型驱动的软件开发技术

Model-driven Software Development Technology

课程介绍

提纲

- 课程主要目标
- 引例
- 课程特色
- 在模型驱动领域的研究背景与成果
- 授课方式与课程要求
- 课程内容与计划
- 课程的准备知识、学习方法
- 本课程所用到的工具集
- 课程分数评定
- 参考资料
- 研究项目与原型展示

课程主要目标

- 了解软件开发方法学在面向对象软件开发方法之后的最新发展趋势与关键技术，如基于构件，面向特征，面向特征等技术。
- 掌握模型驱动软件开发方法的基本概念与主要思想。
- 掌握模型驱动软件开发方法中的关键技术，如元建模技术、模型转换和代码生成的规范与方法。
- 初步掌握将模型驱动软件开发方法运用到特定领域的方法，技术与工具。
- 改变对软件开发的传统思路，更深理解软件工程。
- 掌握前沿软件开发方法，增强职业竞争力。
- 提高研究能力，开阔学术视野，培养把握前沿软件开发方法的研究与应用能力。

引例一软件企业面临的问题

- 攸关系统
 - 汽车领域 航天领域
 - 恒润科技 <http://www.hirain.com/>
- SaaS企业
 - 某软件开发公司的主营业务是为中小型企业开发Web信息管理系统，而每个企业都有自身的需求，公司人员已达8000人，但仍忙不过来。
 - 中企动力 <http://www.300.cn/>
- 创新公司
 - 某公司计划在短期内开发一种非常有创意的软件产品，以便快速占领市场，但不精通开发技术，人手不够。
- 大型IT企业
 - 异构性的问题
 - 代码重构
 - 如何充分利用多年来积累的开发经验、制品与代码。
 - 某大型软件公司的软件开发人员每天苦于建立千篇一律的设计模型、编写千篇一律的程序，耗时而乏味。
 - 某软件设计人员建立了一个设计模型，但无法验证。
 - IBM 华为 上海证券
- 涉及软件外包的企业
 - 软件外包过程中，甲乙双方沟通问题。
 - 软件外包过程中，甲乙双方知识产权问题。
 - 金融公司 银行

课程特色

- 深入、系统、完整地讲解模型驱动软件开发方法的产生背景及其中的关键技术与规范。
- 深入讲解UML/MOF/QVT语言的高级技术细节，讲解完整的模型驱动软件开发方法。
- 展示模型驱动方法中的工具集。
- 综合最新研究文献与多年研究成果，集理论性、实用性、研究性、前沿性一体。
- 提供丰富的案例,便于快速掌握关键技术
- 通过完整案例，展示模型驱动软件开发的过程。

在模型驱动领域的研究背景

- 博士论文《**UML扩展机制及其支持技术研究**》**2003**
 - 针对**UML**扩展机制及其在使用中出现的问题，对**UML**的衍型扩展机制进行了精确的定义与分析，研究了元模型扩展机制的分级定义以及元模型语法扩展的支持技术。
 - 特别是对**UML**扩展机制和元模型方面的理论研究方面，已有两篇论文发表在**UML**领域最重要的国际会议**UML2004**
 - 论文中关于元建模工具原理与原型的论述与近期**Eclipse GMF**中的工作原理非常相似，却比后者早了**6**年
- 博士后科学基金项目《**扩展UML的过程策略研究及其在web领域建模与模拟中的应用**》**2004~2005**
- 博士后出站报告《**MDA在特定领域的应用与基础理论研究**》**2006**
 - 模型驱动的体系结构研究综述
 - 基于**MDA**的**Web**领域用况模型研究
 - 基于**MDA**与**SOA**的系统集成技术研究
 - 扩展**MDA**的建模语言的过程策略研究
- 指导**MDA**相关的综合实践四项、硕士论文三篇
- 主持国家自然科学基金项目《**软件外包领域模型驱动开发方法中模型伪装与转换理论研究**》**2012**
- 主持华为合作研究项目《**下一代网络架构和协议的广义模型和转换**》**2013**

在模型驱动领域的研究背景

- 1. On the Formalized Semantics of Static Modeling Elements in UML, *Jiang Yan-bing Shao Wei-zhong Ma Zhi-yi and Feng Yao-dong*, Formal Methods And Software Engineering 2002, Proceedings Lecture Notes In Computer Science, Springer-Verlag Berlin. (SCI收录) 获得北京大学2004年优秀论文奖
- 2. On the Classification of UML's Meta Model Extension Mechanism, *Yanbing Jiang, Weizhong Shao, Lu Zhang, Zhiyi Ma, Xiangwen Meng and Haohai Ma*, UML2004, Proceedings Lecture Notes In Computer Science, Springer-Verlag Berlin. (SCI收录 引用1次) 这是在有关UML最权威的国际会议上发表的论文, 并对UML的发展提出建设性建议。
- 3. Applying OO Metrics to Assess UML Meta-Models, *Haohai Ma, Weizhong Shao Lu Zhang, Zhiyi Ma and Yanbing Jiang*, UML2004, Proceedings Lecture Notes In Computer Science, Springer-Verlag Berlin. (SCI收录)
- 4. On Procedure Strategy of Constructing SOA's Modeling Language, *Yanbing Jiang, Chunxiao Xing, Wei he and Jijiang Yang*, Proceedings of IEEE International Workshop on service-oriented system Engineering 2005, IEEE Computer Society Press
- 5. 模型驱动的体系结构研究综述, 蒋严冰 邢春晓 南京大学学报(自然科学版), 计算机科学专辑 2005.10
- 6. UML中衍型的精确定义与分析, 蒋严冰 邵维忠 张路 麻志毅 电子学报 2003.12a (EI收录)
- 7. UML现存问题与发展道路, 邵维忠 蒋严冰 麻志毅, 计算机研究与发展, 2003.4 (EI收录)
- 8. 面向对象的建模工具——JBOO3.0的研究与开发, 麻志毅 蒋严冰 戴耀飞 李劲宇 电子学报 2002.12a (EI索引源)
- 9. 基于规则的UML元模型语法扩展的支持技术研究, 蒋严冰 麻志毅 朱志高 张能斌, 中国计算机大会论文集, 清华大学出版社, 2003年11月

课程内容与计划

课次	内容	学时	时间
1	课程介绍 绪论:模型驱动软件开发技术综述与后面向对象 方法与模型驱动	3学时	2013.2.26 星期二 晚 18: 00~21: 00
2	元建模技术——概述 元建模技术——UML的元模型体系结构与扩展 机制	3学时	2013.3.5 星期二 晚 18: 00~21: 00
3	元建模技术——元建模过程与支撑工具 元建模技术—— 实战案例	3学时	2013.3.12 星期二 晚 18: 00~21: 00
4	元建模技术—— 实战案例	3学时	2013.3.19 星期二 晚 18: 00~21: 00
5	元建模技术—— 展示与辅导	3学时	2013.3.26 星期二 晚 18: 00~21: 00

课程内容与计划

课次	内容	学时	时间
6	模型转换技术——概述 模型转换技术——基于QVT relation的转换语言与工具	3学时	2013.4.2 星期二 晚 18: 00~21: 00
7	模型转换技术——基于QVT relation的案例	3学时	2013.4.9星期二 晚 18: 00~21: 00
8	模型转换技术——基于QVT operational的转换语言与工具	3学时	2013.4.16星期二 晚 18: 00~21: 00
9	模型转换技术——基于QVT operational的案例	3学时	2013.4.23 星期二 晚 18: 00~21: 00
10	模型转换技术——基于QVT operational的案例	3学时	2013.5.7星期二 晚 18: 00~21: 00

课程内容与计划

课次	内容	学时	时间
11	模型转换技术——展示与辅导	3学时	2013.5.14 星期二 晚 18: 00~21: 00
12	模型驱动的代码生成技术——基于Jet的代码生成语言与工具	3学时	2013.5.21 星期二 晚 18: 00~21: 00
13	模型驱动的代码生成技术——基于QVT M2T的代码生成语言与工具Acceleo	3学时	2013.5.28 星期二 晚 18: 00~21: 00
14	模型驱动的代码生成技术——展示与辅导	3学时	2013.6.4 星期二 晚 18: 00~21: 00
15	MDA综合案例	3学时	择机
16	分组项目答辩	3学时	择机

授课方式与课程要求

- 课堂讲授与讨论相结合
- 鼓励相关论文规范阅读
- 分组项目与展示
- 理论学习与项目实践相结合
- 注重相关工具集的使用

课程的准备知识、学习方法

➤ 准备知识

- 一种面向对象的编程语言
- 面向对象技术
- **UML**

➤ 学习方法

- 理论与实践相结合
- 学习与研究相结合

本课程所用到的工具集

➤ Eclipse modeling

➤ <http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/indigosr1>

➤ Medini QVT

➤ <http://projects.ikv.de/qvt/>

本课程所用到的工具集

工具	作用	基于工具	配置与修改方案
元建模工具	可视化地建立各种模型的ecore元模型	Eclipse modeling Ecore tools	直接可视化的使用 Eclipse modeling Ecore tools 中 Ecore diagram 可视化的建立各种模型的ecore元模型,并存储为.ecore文件。
可扩展的建模工具	以ecore元模型为输入,将其可视化的绑定建模元素的表示法与行为,形成支持各种ecore元模型的建模工具。用以生成各种模型,并以.xmi的方式存储。	Eclipse modeling GMF	利用 GMF 的提供的配置向导可产生简单的建模工具的插件,如对表示法和行为有特殊的要求,可通过修改所生成的建模工具的插件的源代码。
基于QVT的模型转换	根据既定的QVT元模型转化规则,以一种web领域模型为输入,通过转换输出另外一种模型	Eclipse modeling operational QVT ;mediniQVT	大部分转换可通过在这些模型转换工具中输入以QVT规范编写的转换语言后,即可自动实现。其中 Eclipse modeling operational QVT 支持过程化的转换描述, mediniQVT 支持关系声明型的转换描述。
代码生成	可通过QVT及EMF工具分别以模型转换的方法或模板的方法实现。	Eclipse modeling JET Acceleo	代码生成可通过EMF中的 JET 工具,以模板的方式实现。

课程分数评定

- 考核方式
 - 无笔试 无编码项目
 - 平时成绩 **30%**
 - 项目成绩 **70%**
- 项目要求
 - 基于**Eclipse modeling**等模型驱动支撑工具，建立特定领域的模型驱动工具，包括元建模、模型转换及代码生成等部分。
 - 要求：
 - ①明确描述需求
 - ②重点描述元模型与基于**QVT**的转换方案及实验案例
 - ③代码生成方案

课程项目选取领域

- **Web**领域代码自动生成工具
- 手机领域代码自动生成工具
- 基于设计模式的模型自动转换
- 其他

课程参考资料

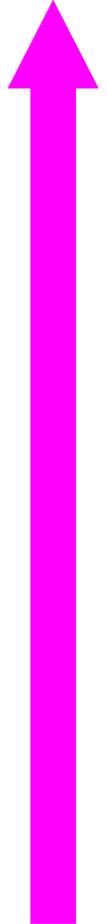
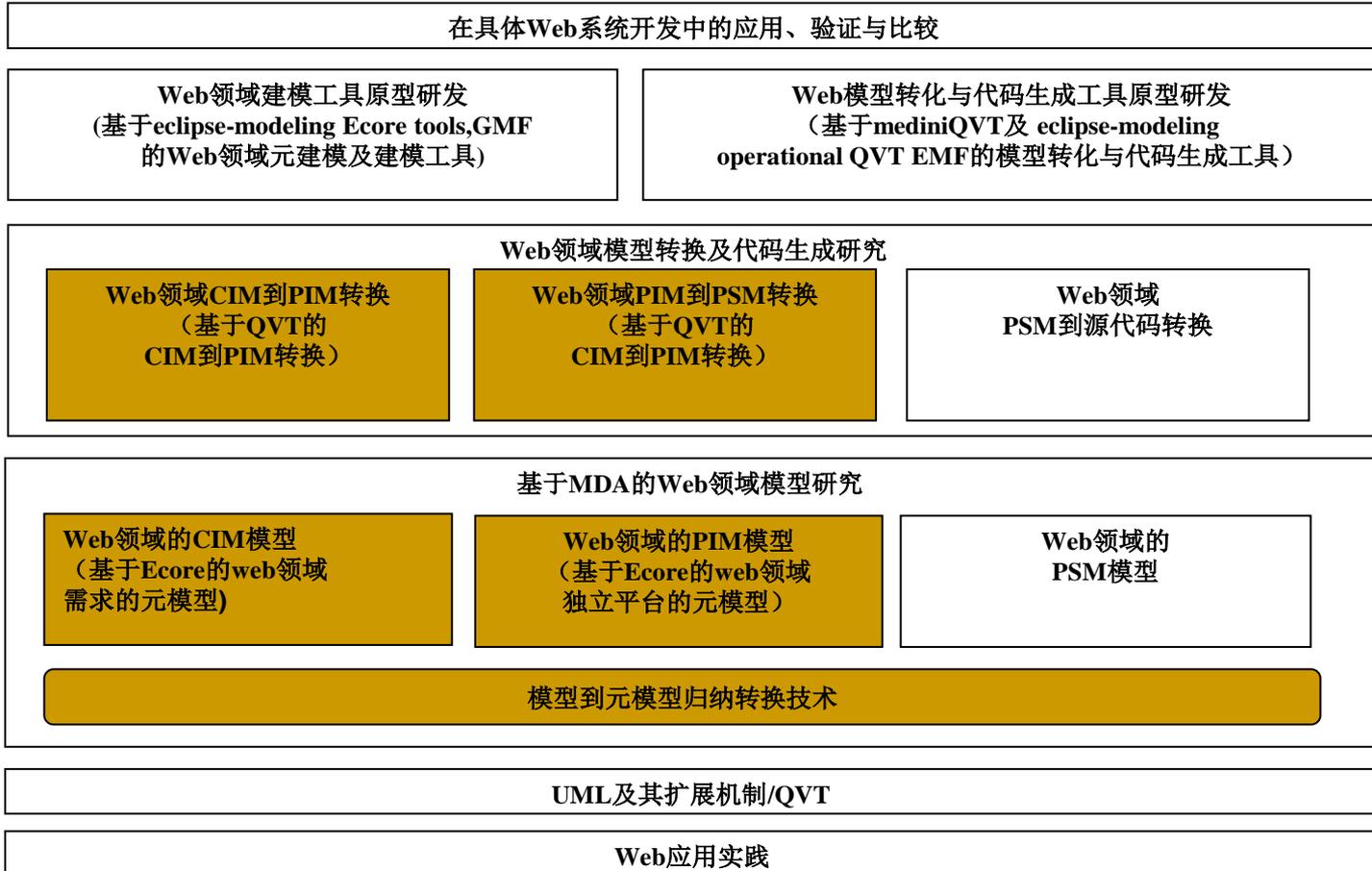
- **Stahl, T., Volter, M.等著, 杨华, 高猛 译,模型驱动软件开发: 技术、工程与管理,清华大学出版社,2009年1月第一版**
- **S.J.Mellor,M.J.Balcer,Executable UML, A Foundation For MDA,科学出版社 影印, 2003年5月第一版**
- **QVT-Partners,Revised submission for MOF 2.0 Query/Views/Transformations RFP,Version 1.1 (2003/08/18),<http://qvtp.org/>**
- **UML 2.0 Superstructure Specification <http://www.omg.org/>**
- **UML 2.0 Infrastructure Specification <http://www.omg.org/MOF™ Query / Views / Transformations>
<http://www.omg.org/technology/documents/formal/uml.htm>**
- **MDA Guide V1.0.1 <http://www.omg.org/cgi-bin/doc?omg/03-06-01>**

模型驱动的研究项目与原型

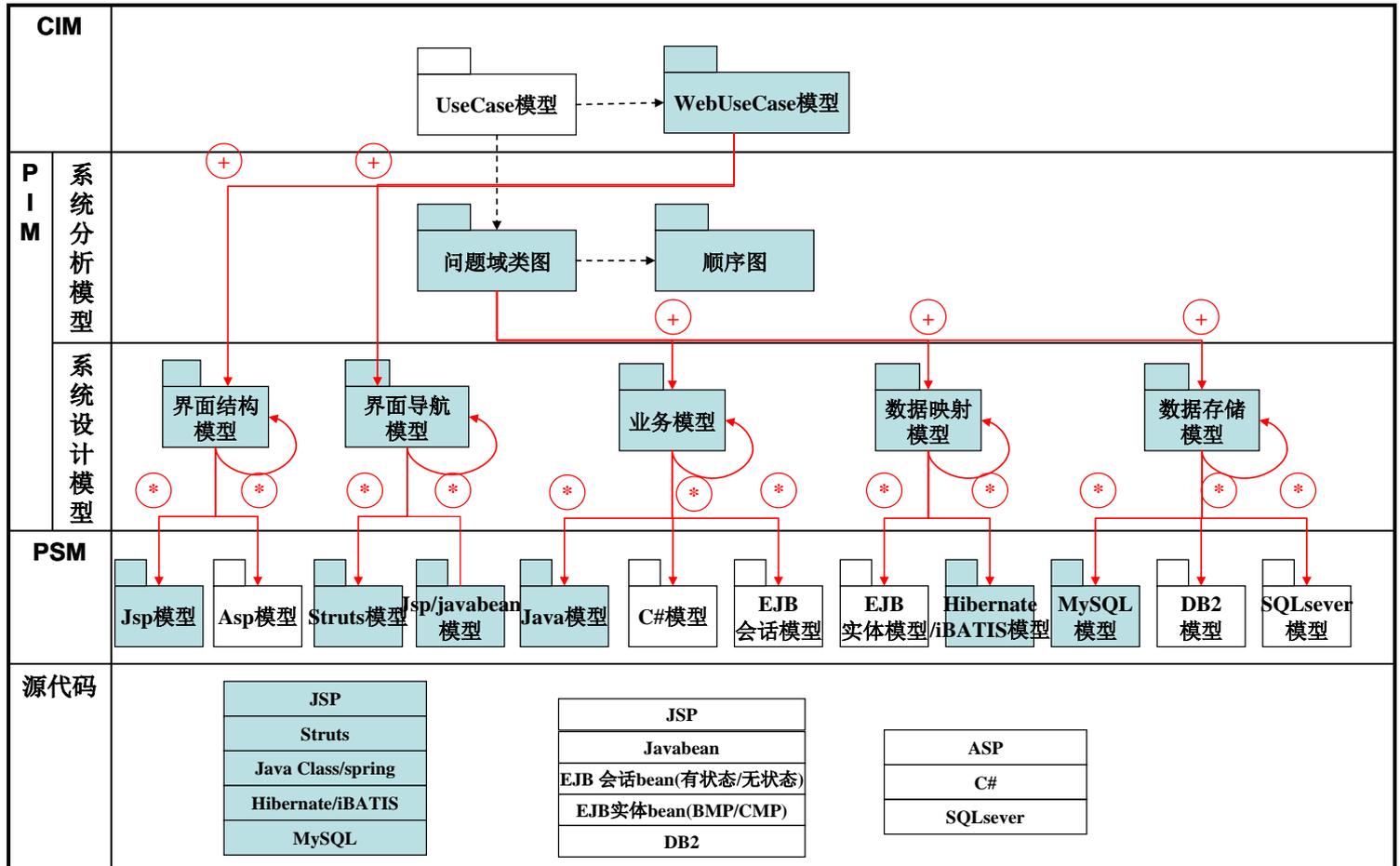
- 模型驱动的**Web**领域软件模型体系、模型转换及代码生成技术与原型
- 模型驱动的**J2EE**部署模型工具
- **WBM**到**SOMA**的模型转化系统
- 软件外包领域模型驱动开发方法中模型伪装与转换
- 下一代网络架构和协议的广义模型和转换

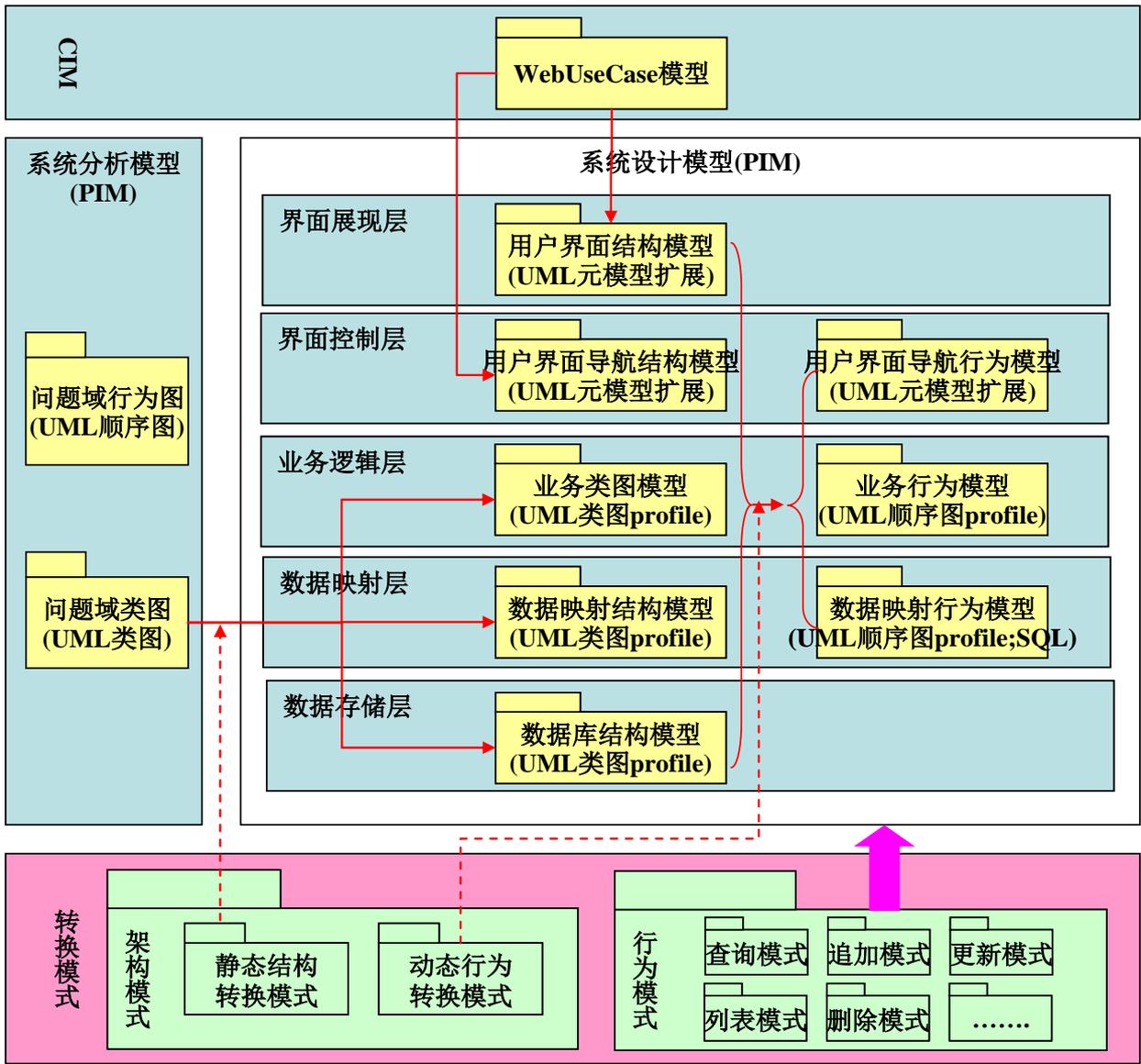
模型驱动的Web领域软件 模型体系、模型转换及代 码生成技术与原型

Web领域代码自动生成工具



模型体系与模型转换概要





实例：博客领域模型驱动的开发

元建模

模型转换

代码生成

基于web三层模型进行分层分工，
每层分别进行元建模、模型转换和代码生成

表现层

陈定胜
耿耘

90%

数据持久化层

刘洋
骆云

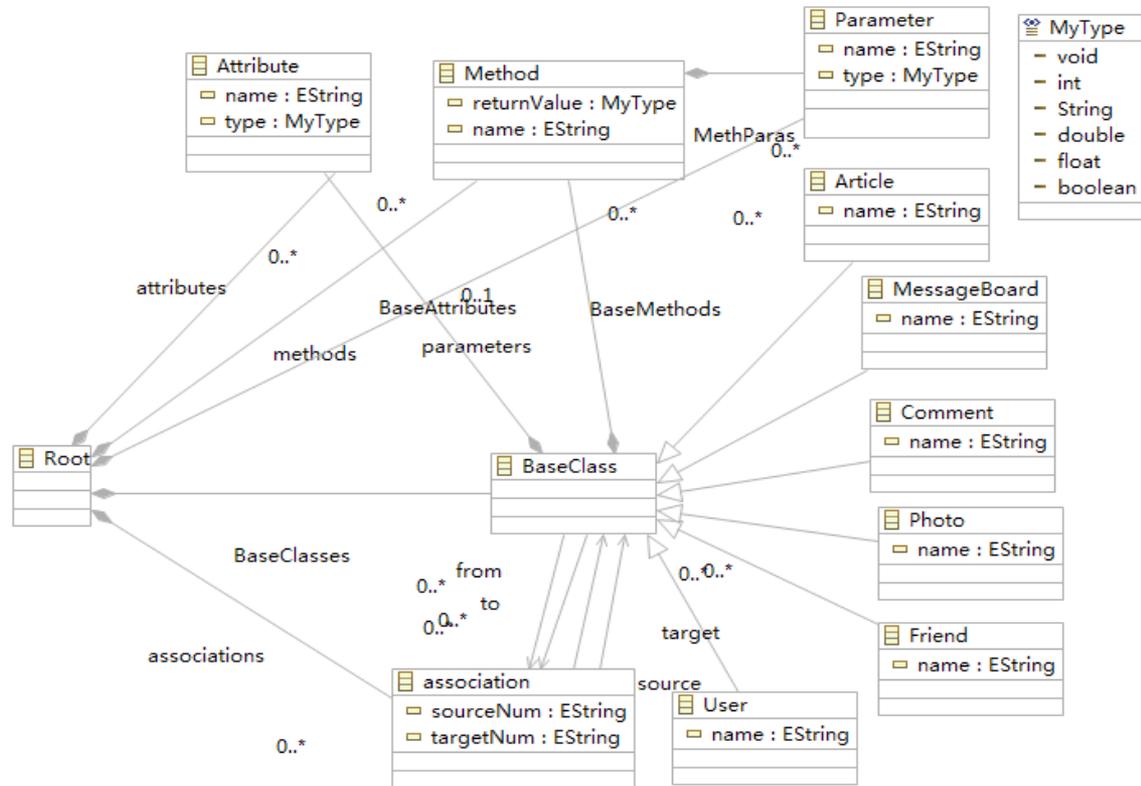
100%

业务逻辑层

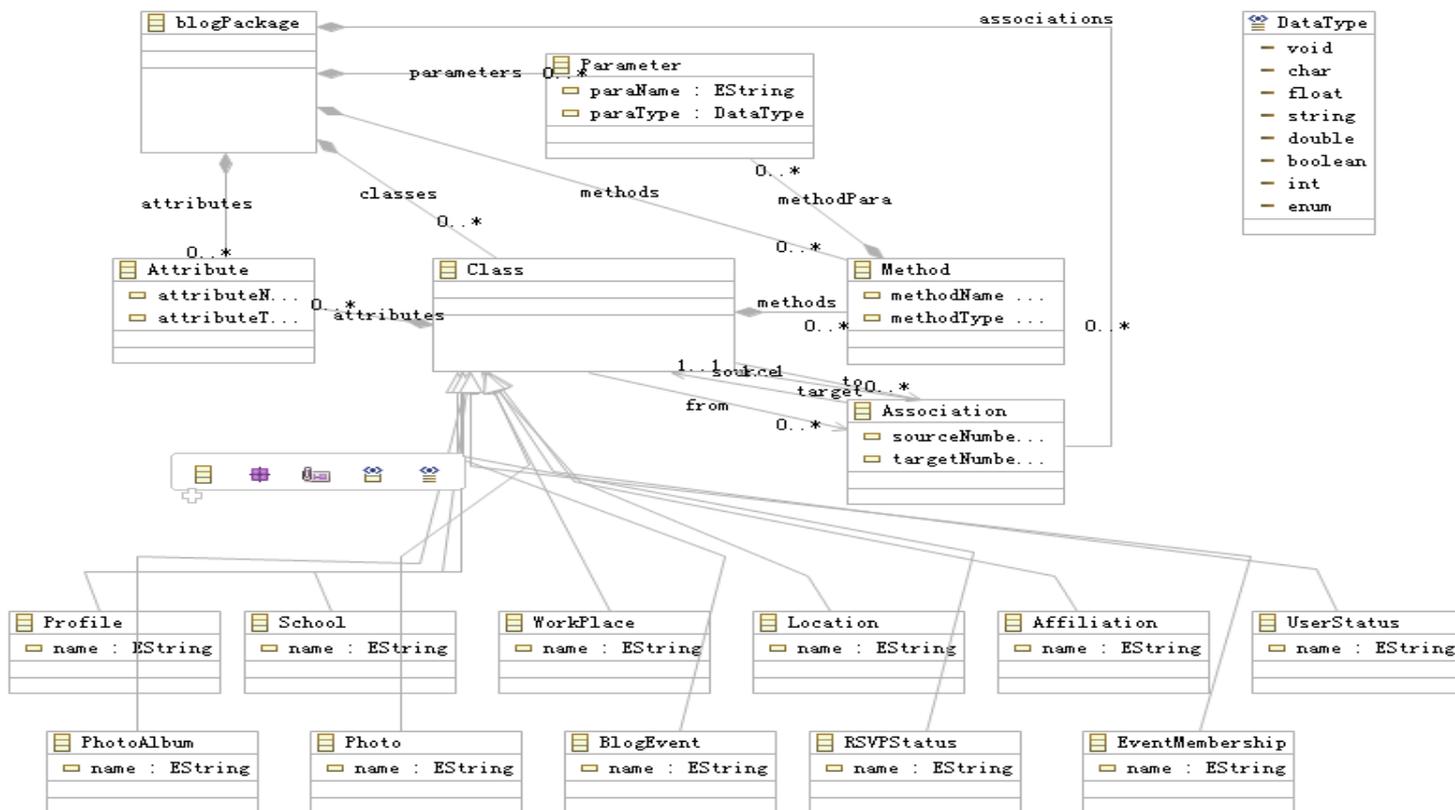
朱晓文
赵越月

100%

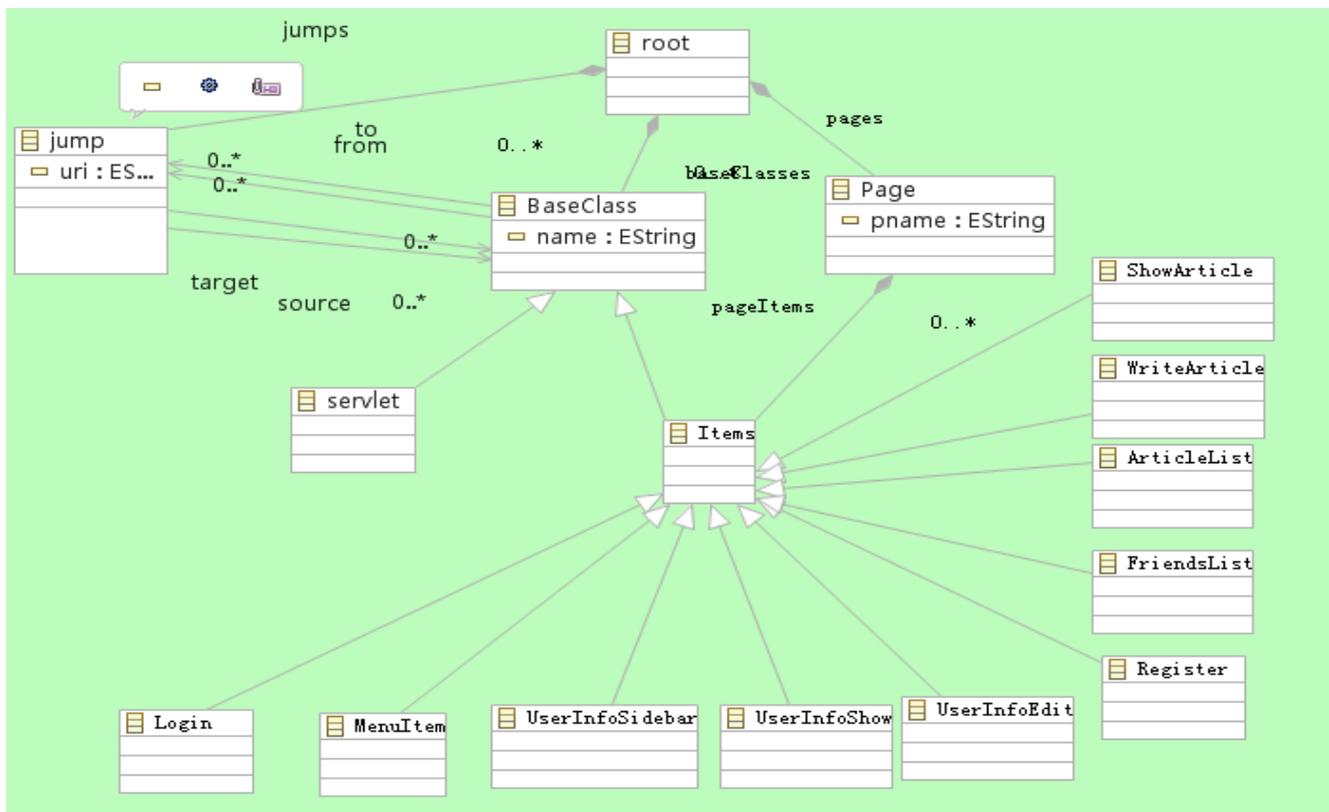
分析模型元模型



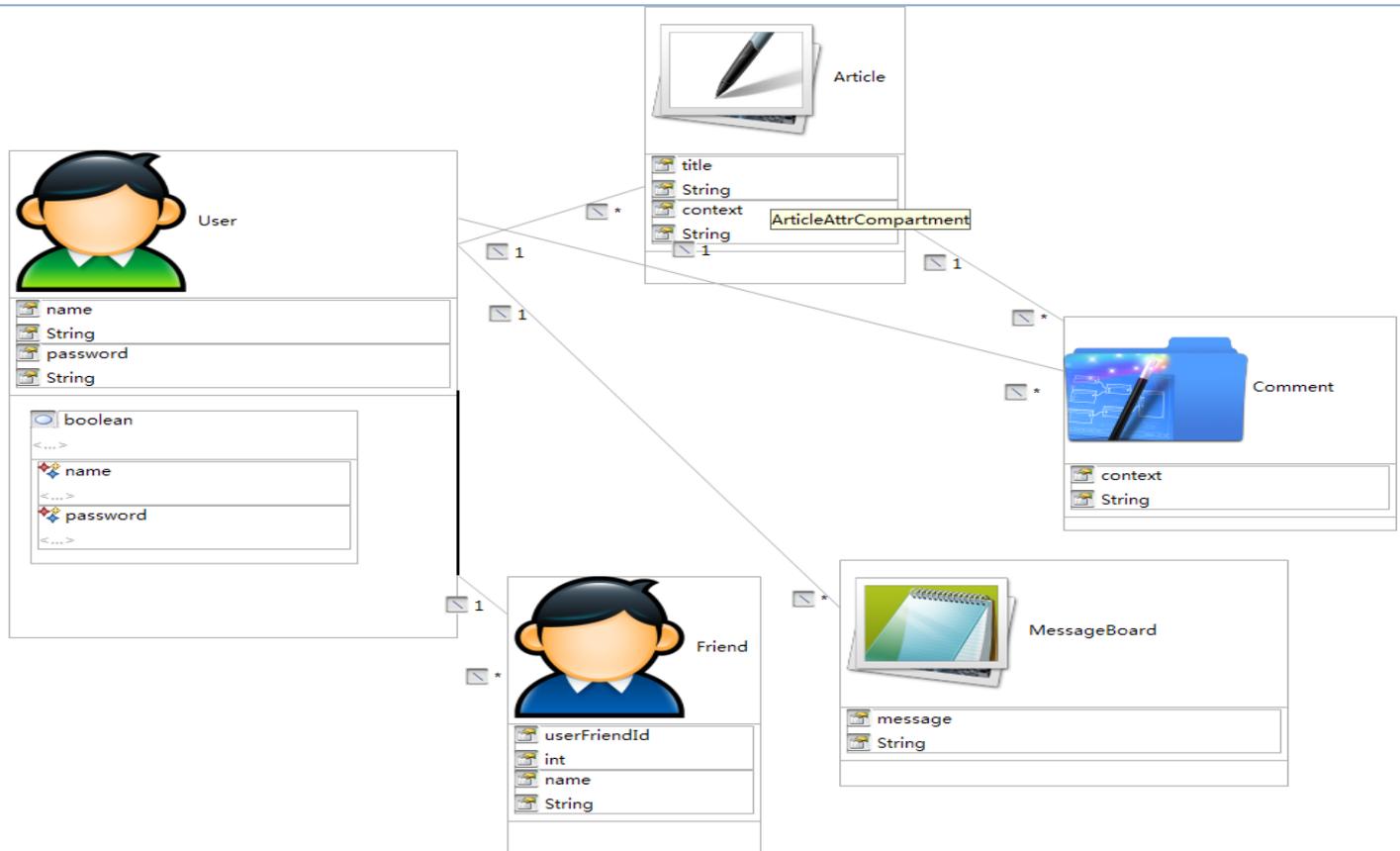
业务逻辑层元模型



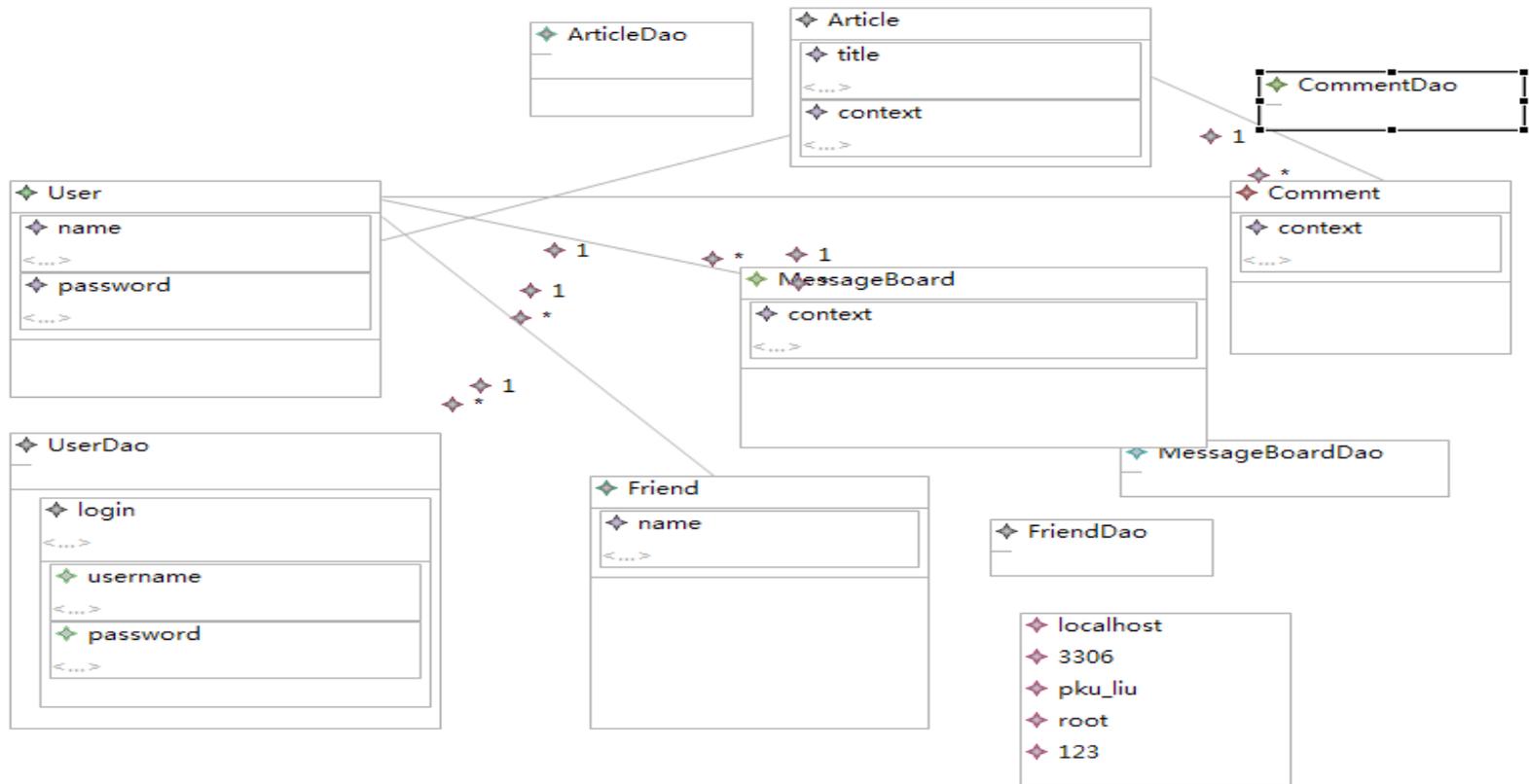
展现层元模型



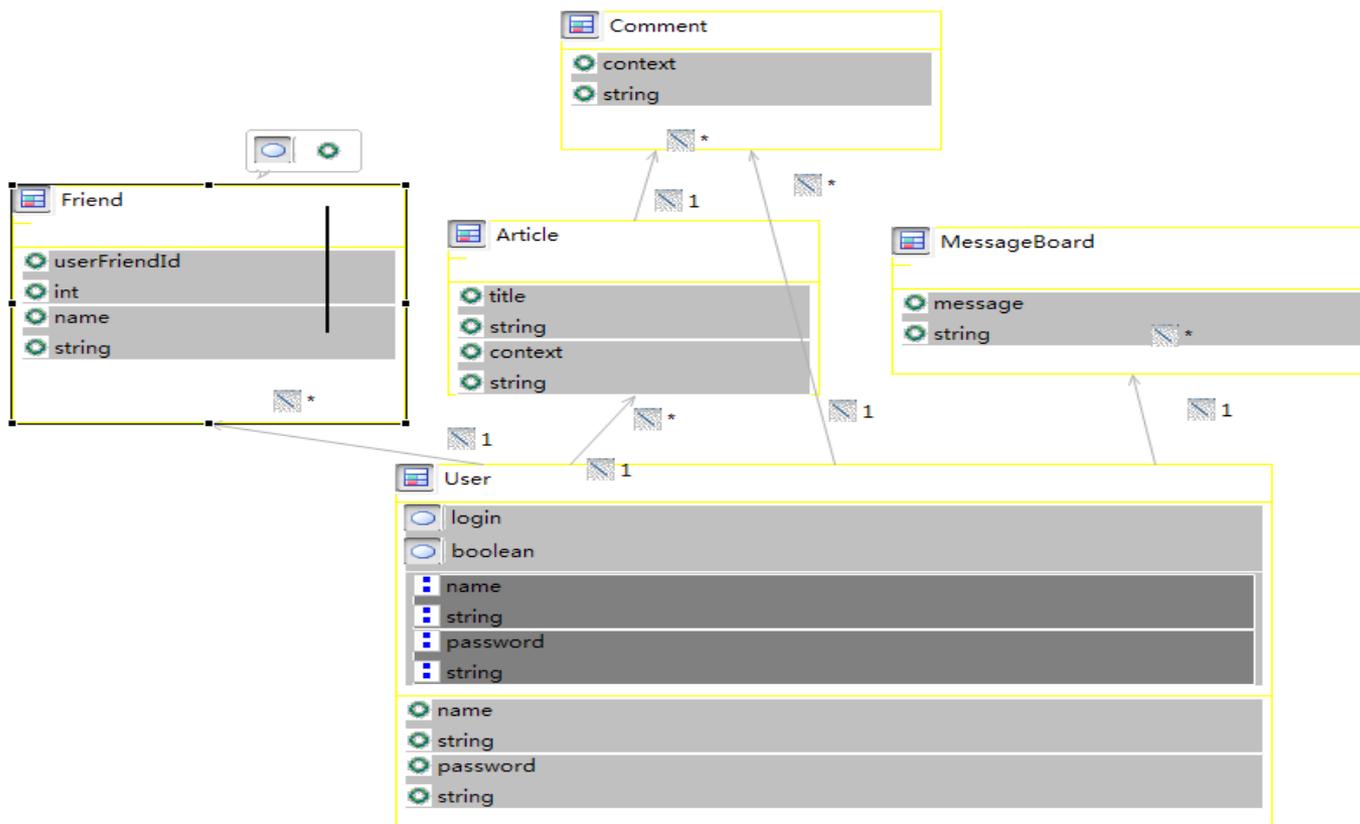
分析模型



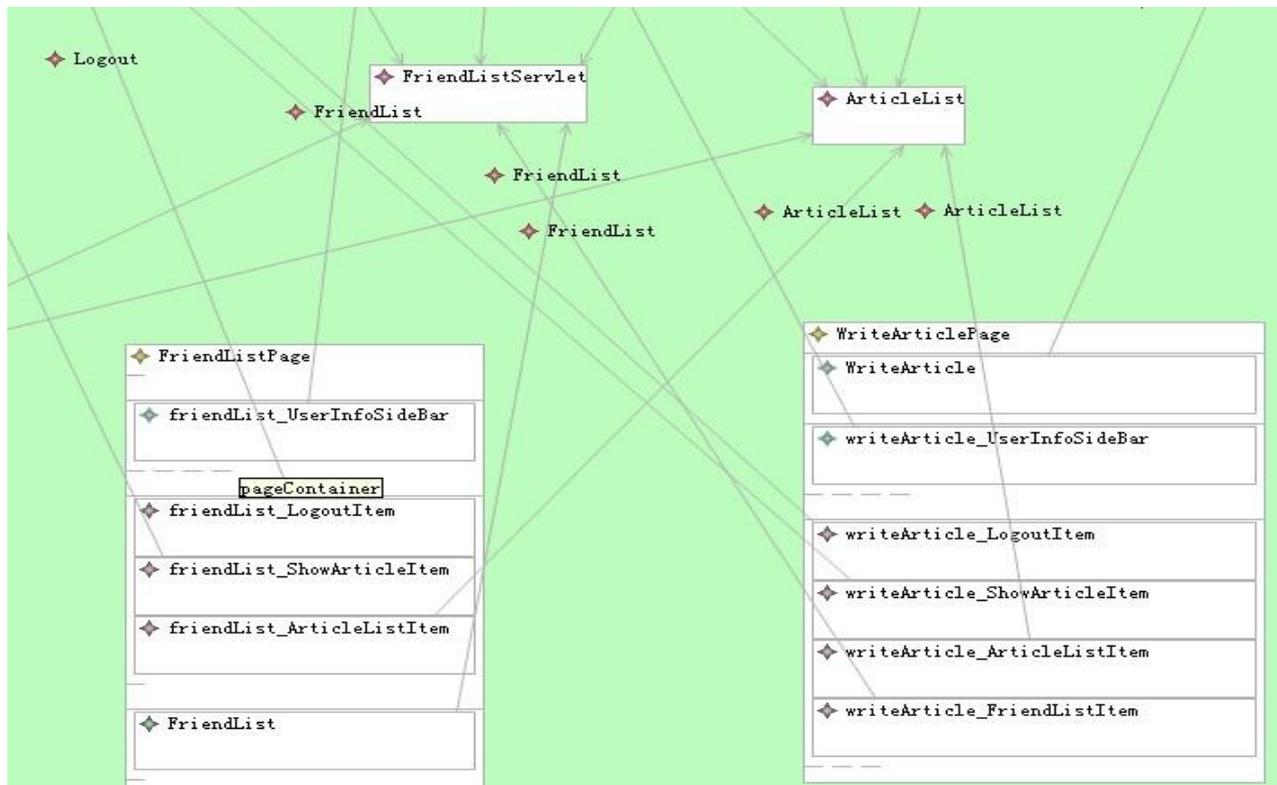
数据访问层设计模型



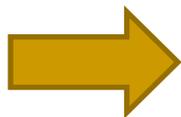
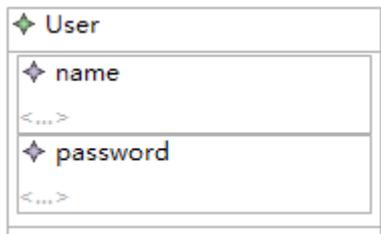
业务逻辑层设计模型



展现层设计模型



数据访问层的代码生成



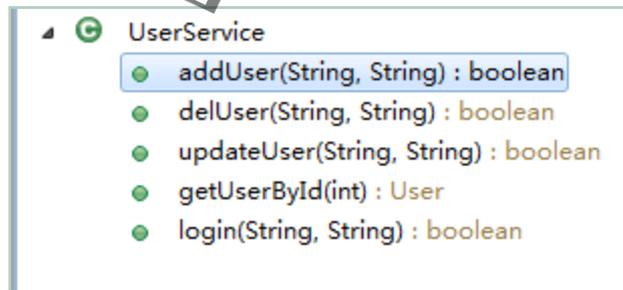
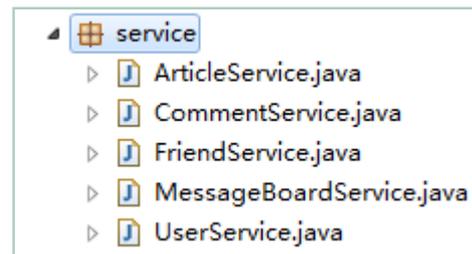
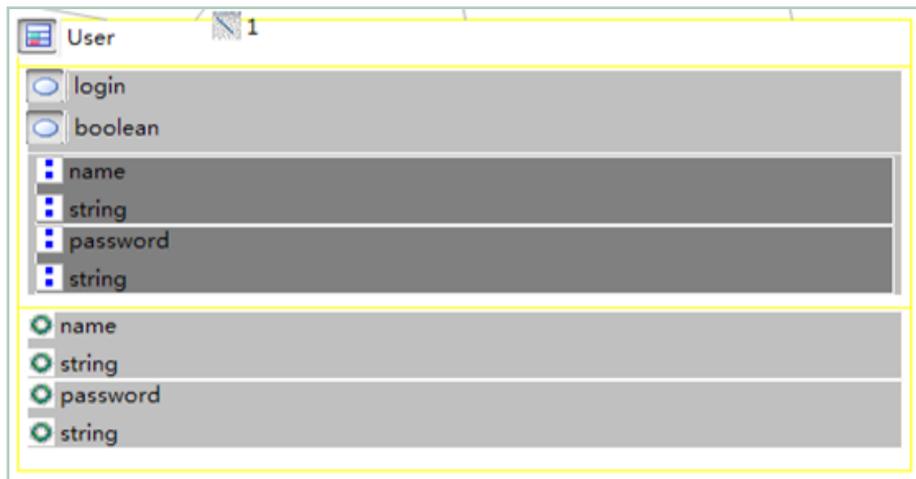
```
User
  name : String
  password : String
  UserId : int
  setId(int) : void
  getUserId() : int
  getName() : String
  setName(String) : void
  getPassword() : String
  setPassword(String) : void
```

```
dao
  Article.java
  ArticleDao.java
  Comment.java
  CommentDao.java
  DBUtil.java
  Friend.java
  FriendDao.java
  MessageBoard.java
  MessageBoardDao.java
  User.java
  UserDao.java
  CreateDB.sql
  createTable.sql
```

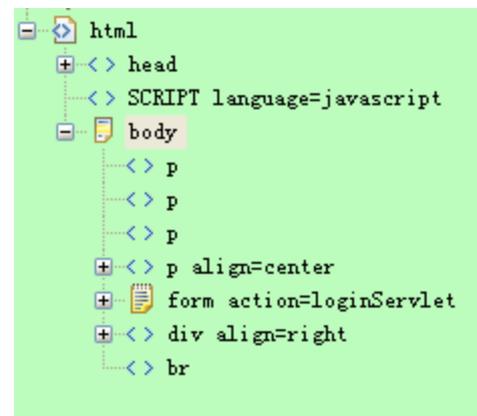
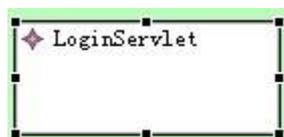
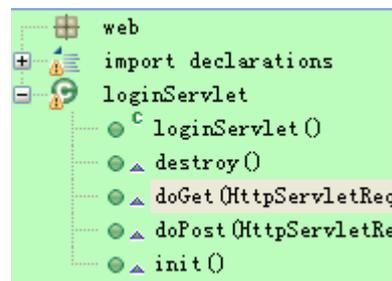
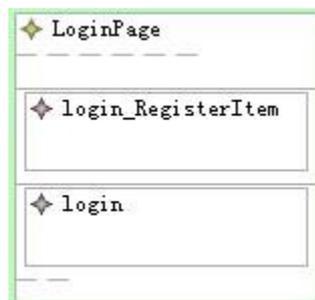


```
UserDao
  login(String, String) : boolean
  getArticleList(int) : List
  getCommentList(int) : List
  getMessageBoardList(int) : List
  getFriendList(int) : List
  addUser(User) : boolean
  delUser(User) : boolean
  updateUser(User) : boolean
  getUserById(int) : User
```

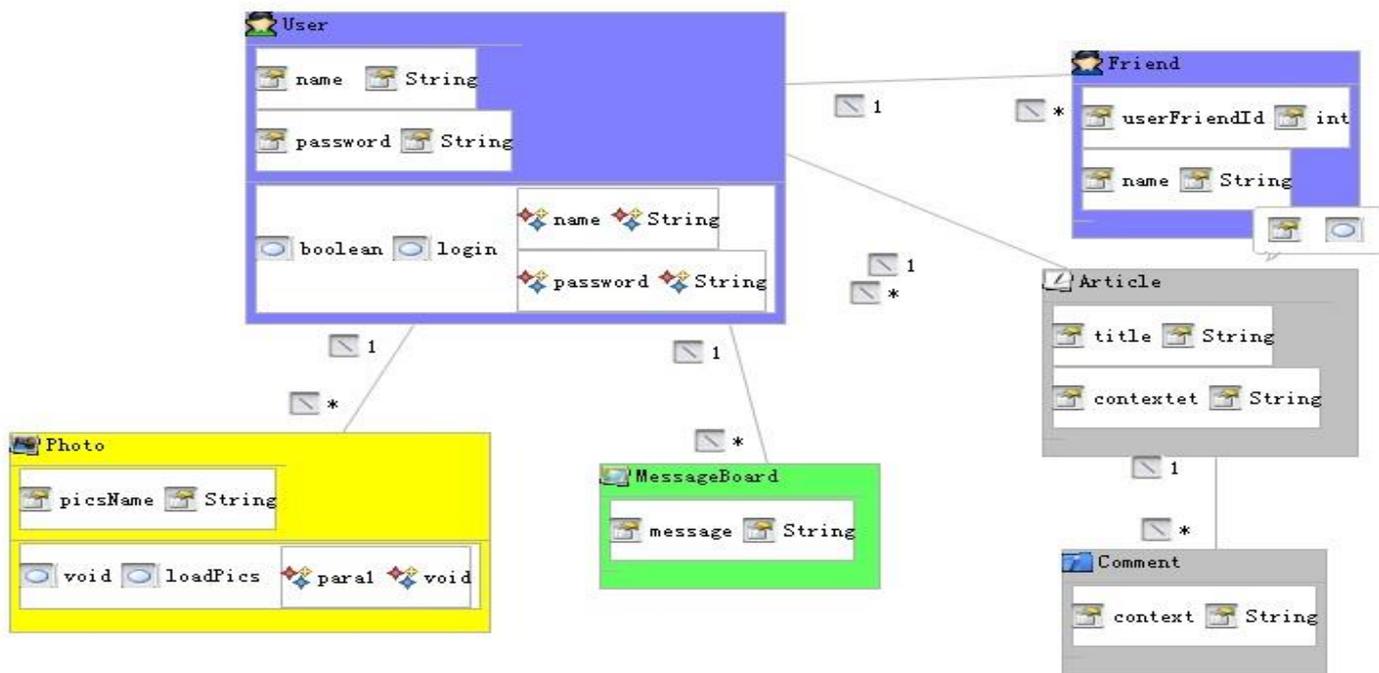
业务逻辑层代码生成



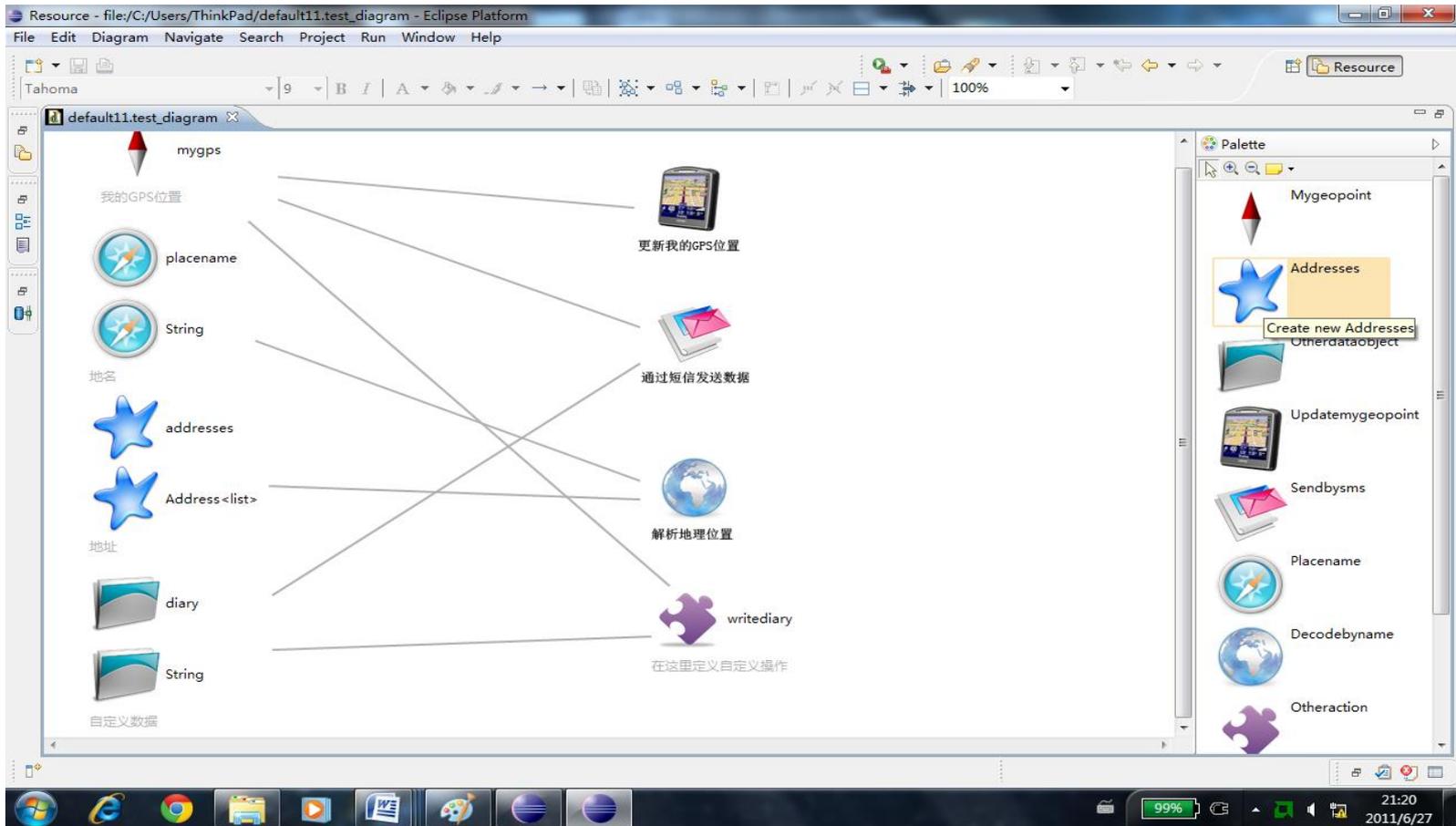
展现层代码生成



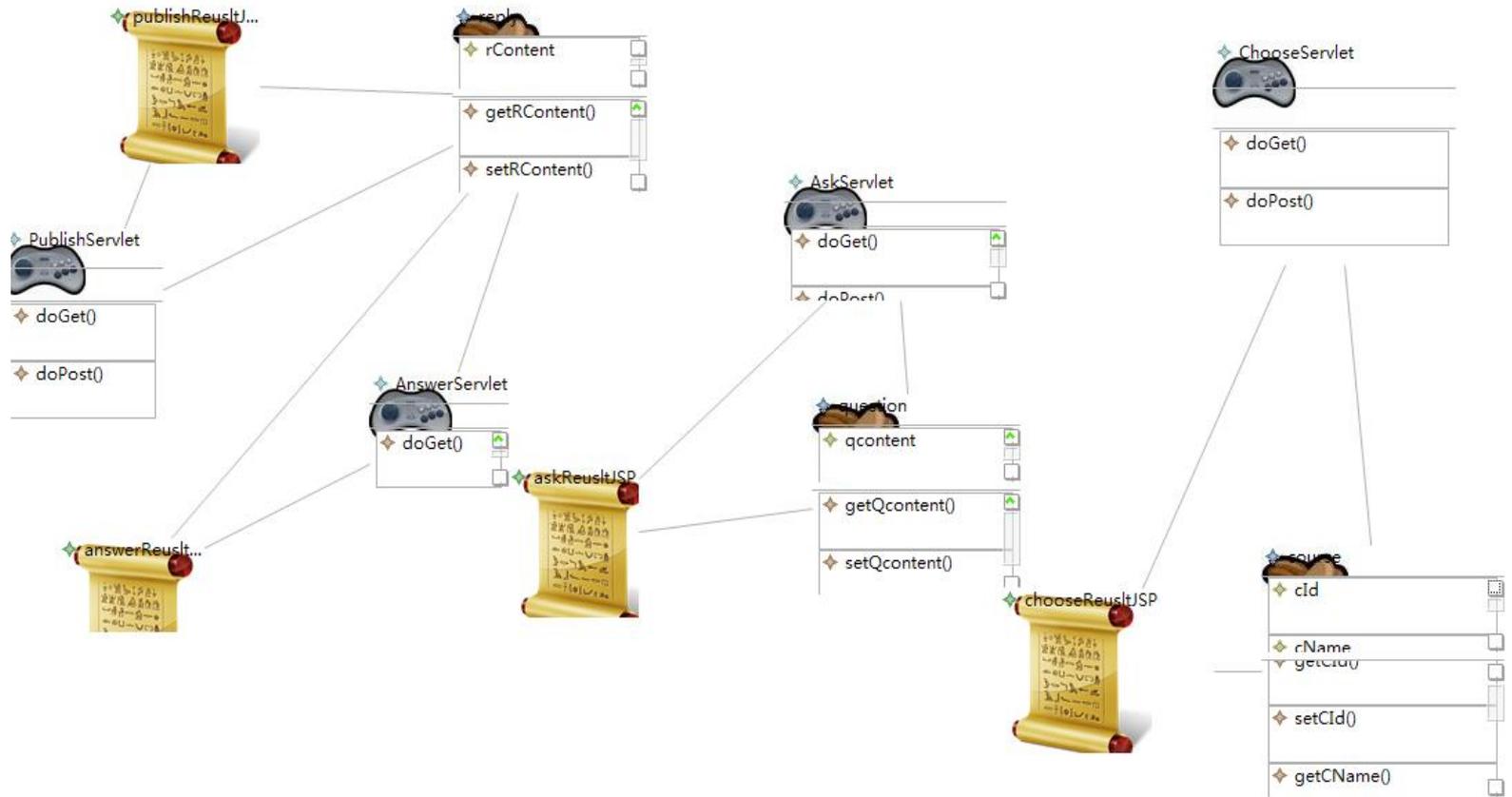
展现层



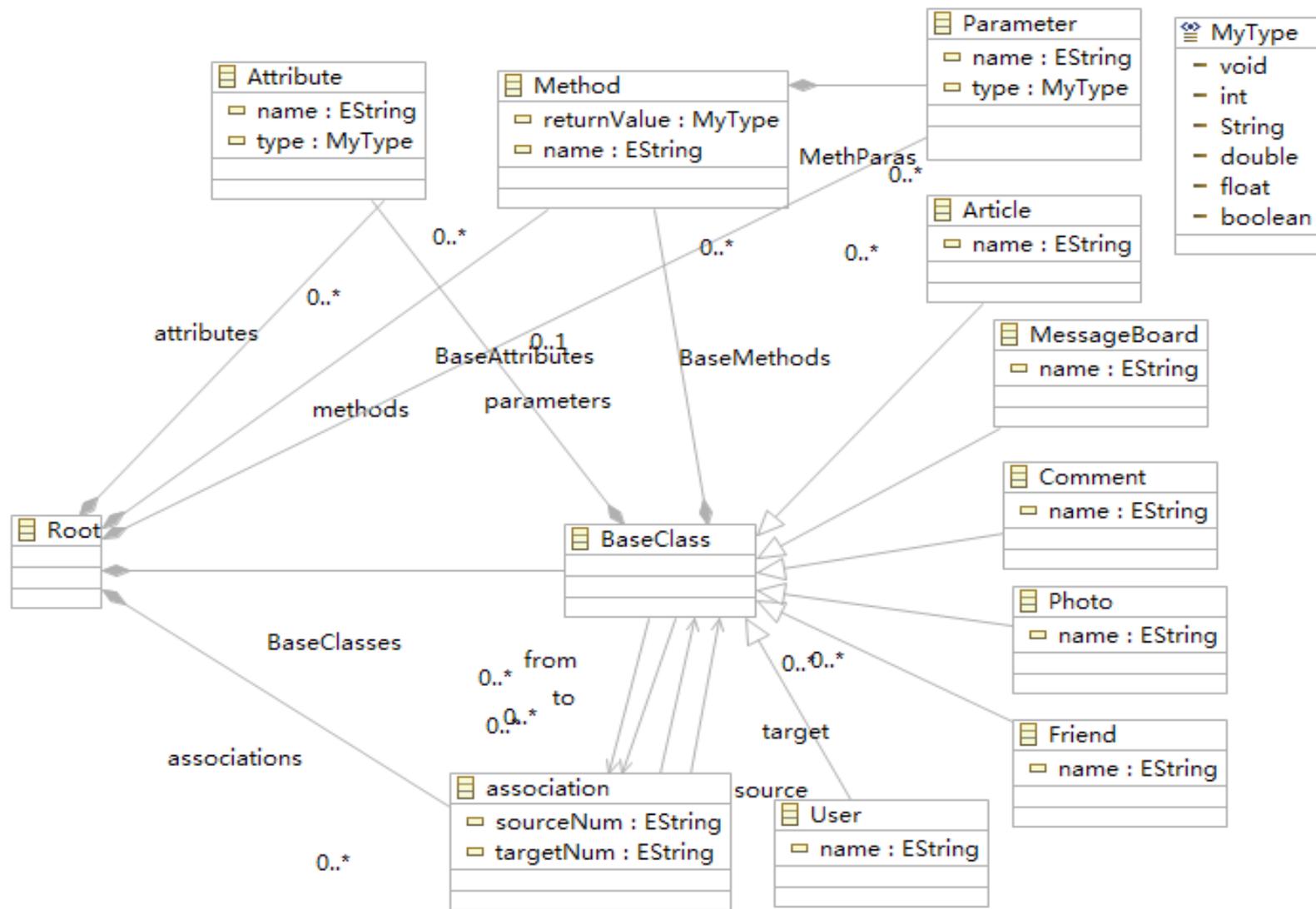
基于android的手机代码框架生成系统



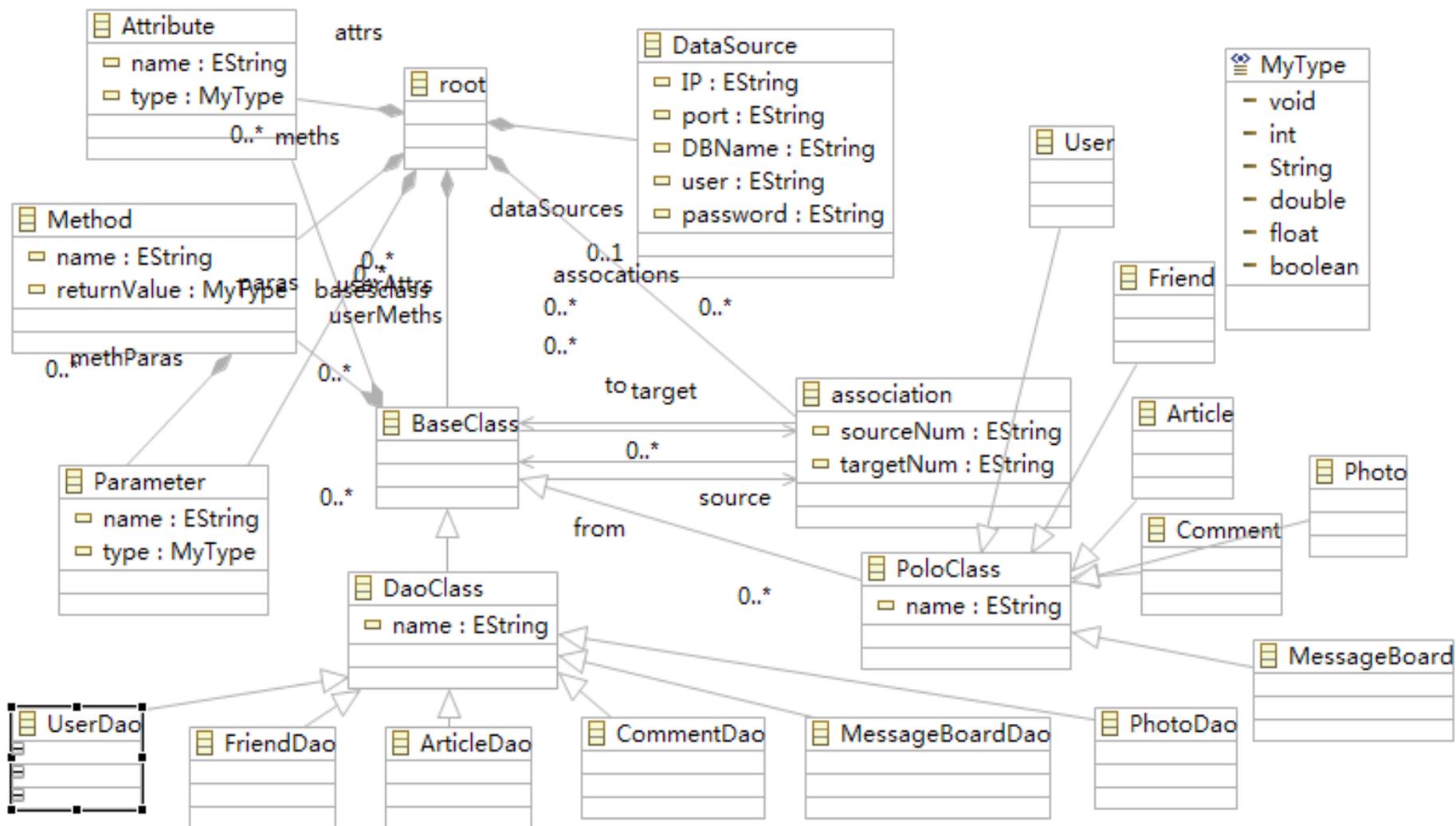
JSP Servlet代码框架生成系统



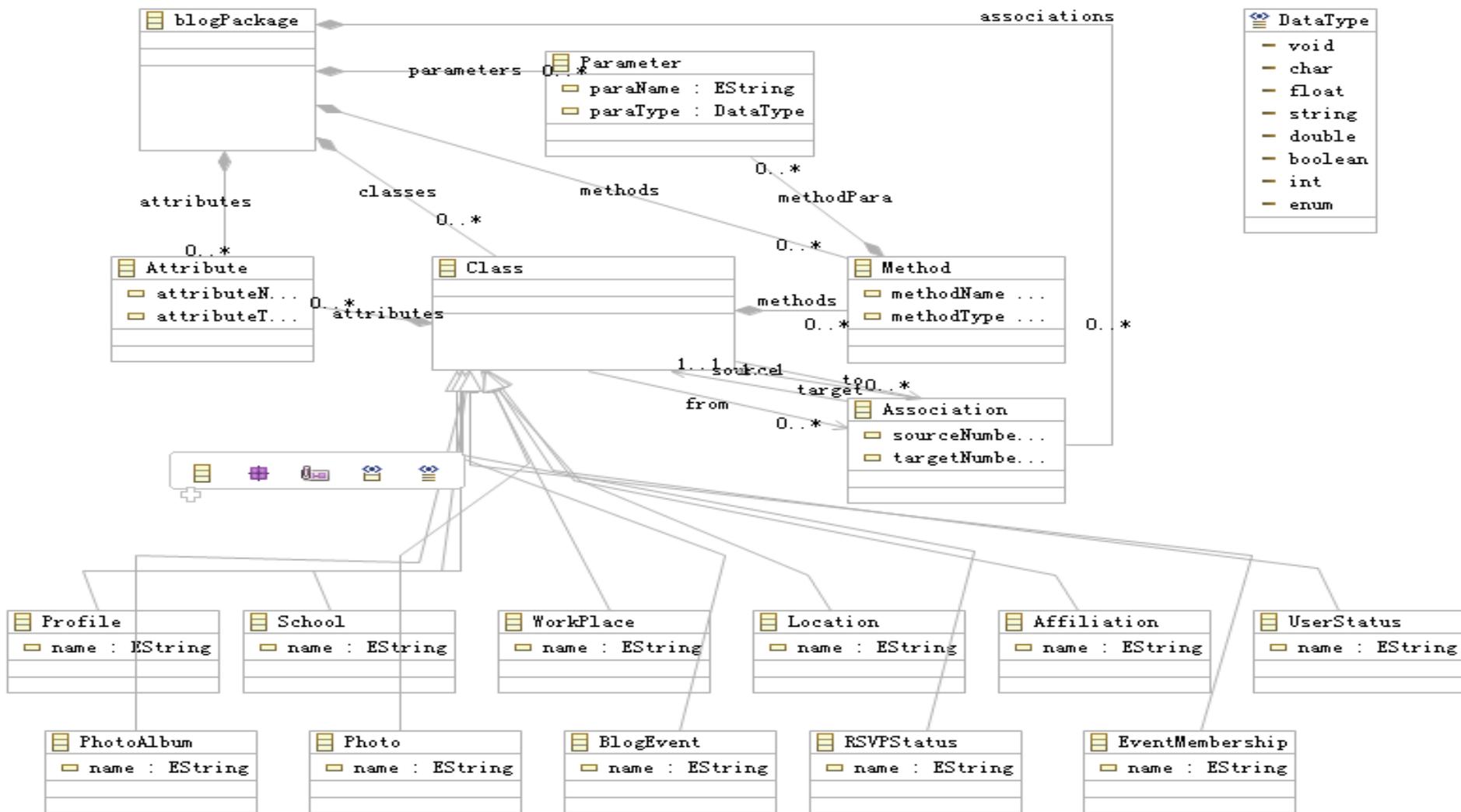
分析模型元模型



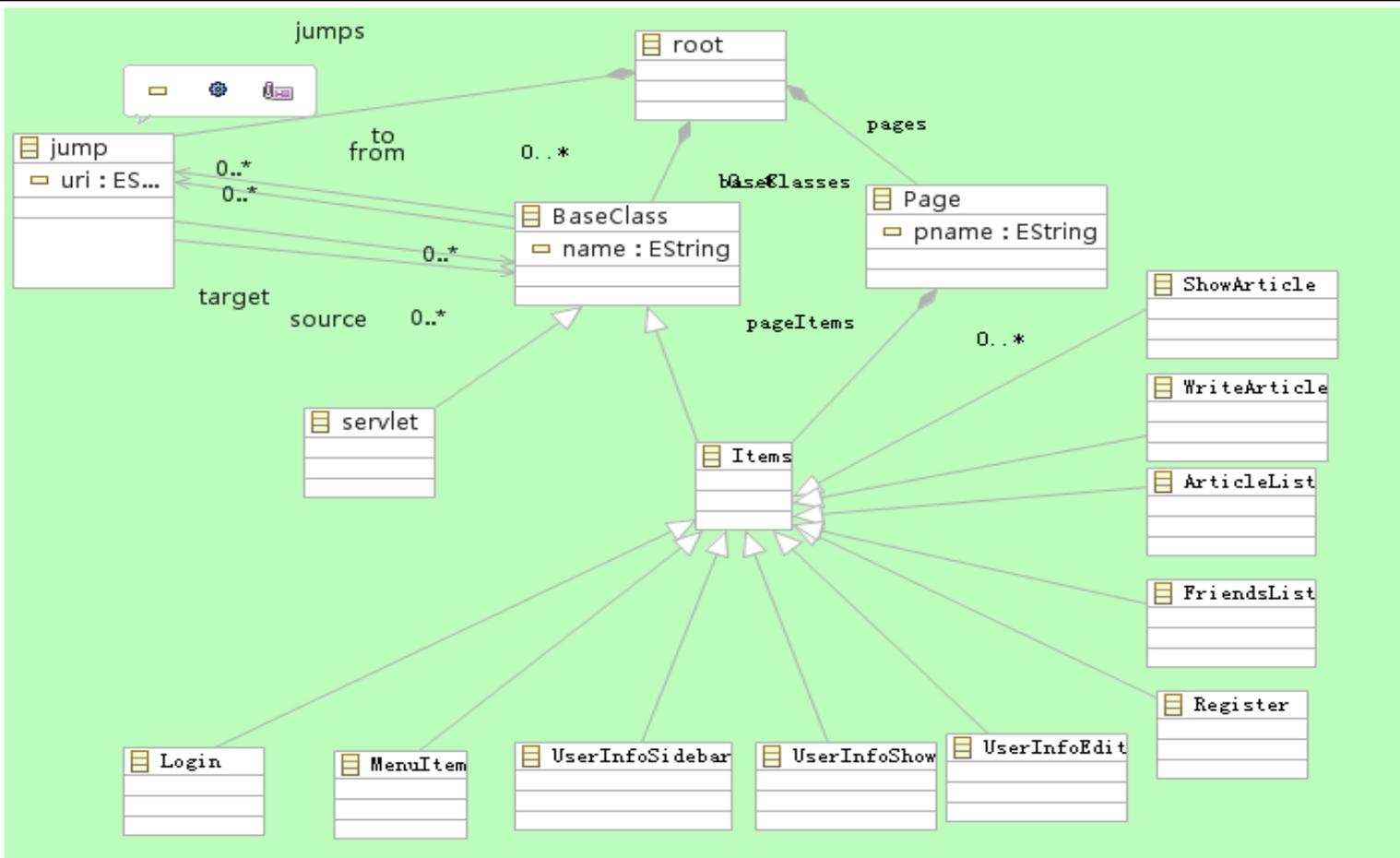
数据访问层元模型



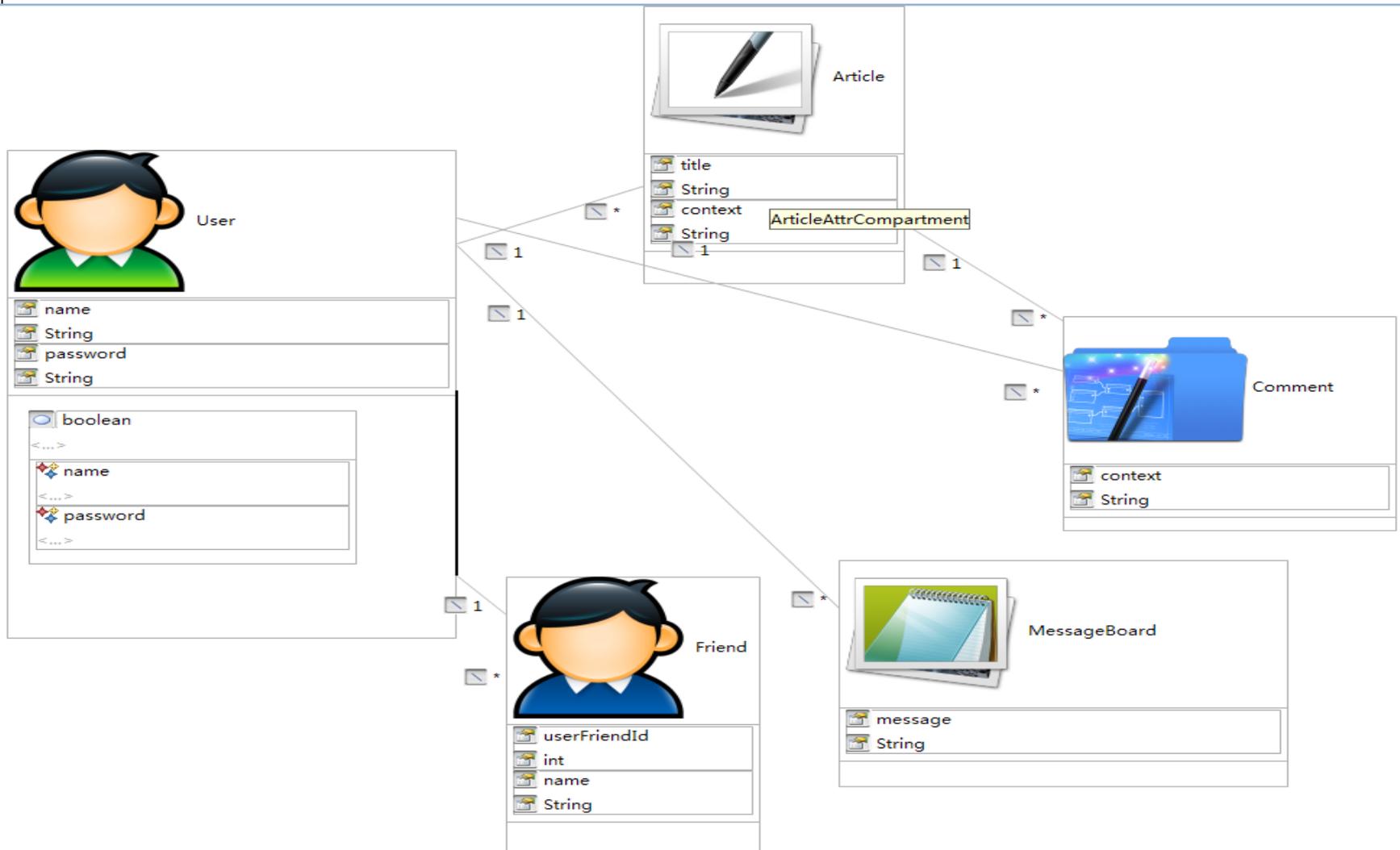
业务逻辑层元模型



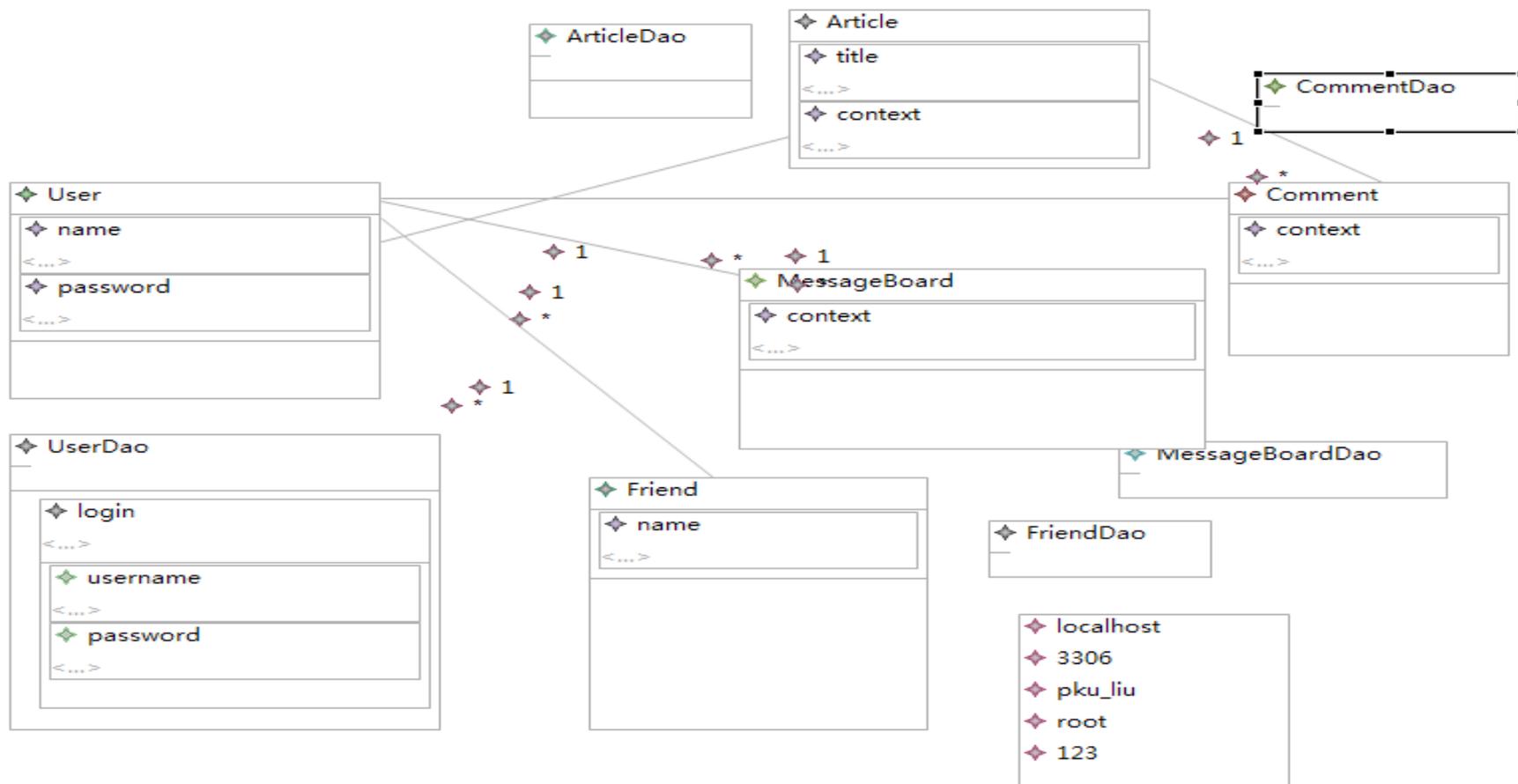
展现层元模型



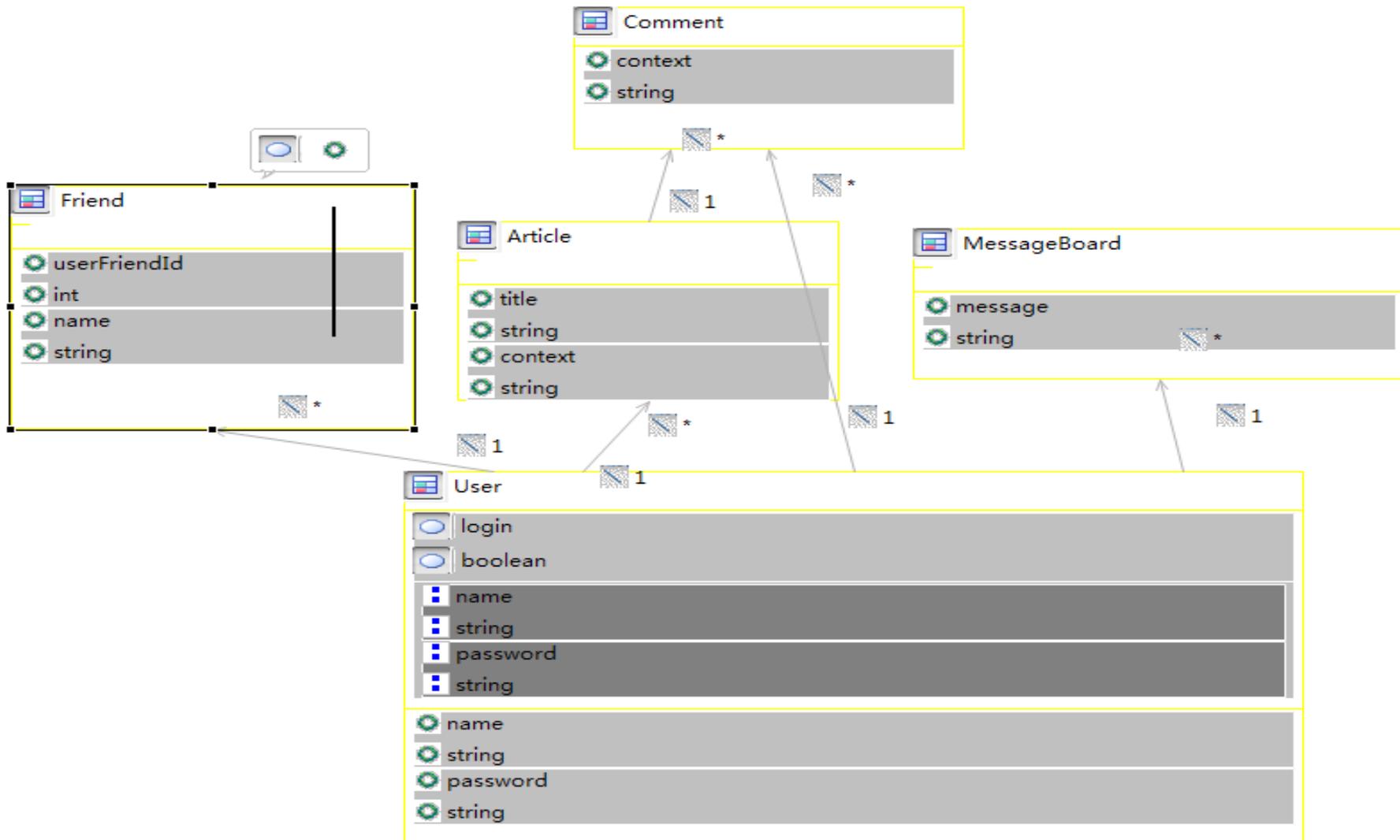
分析模型



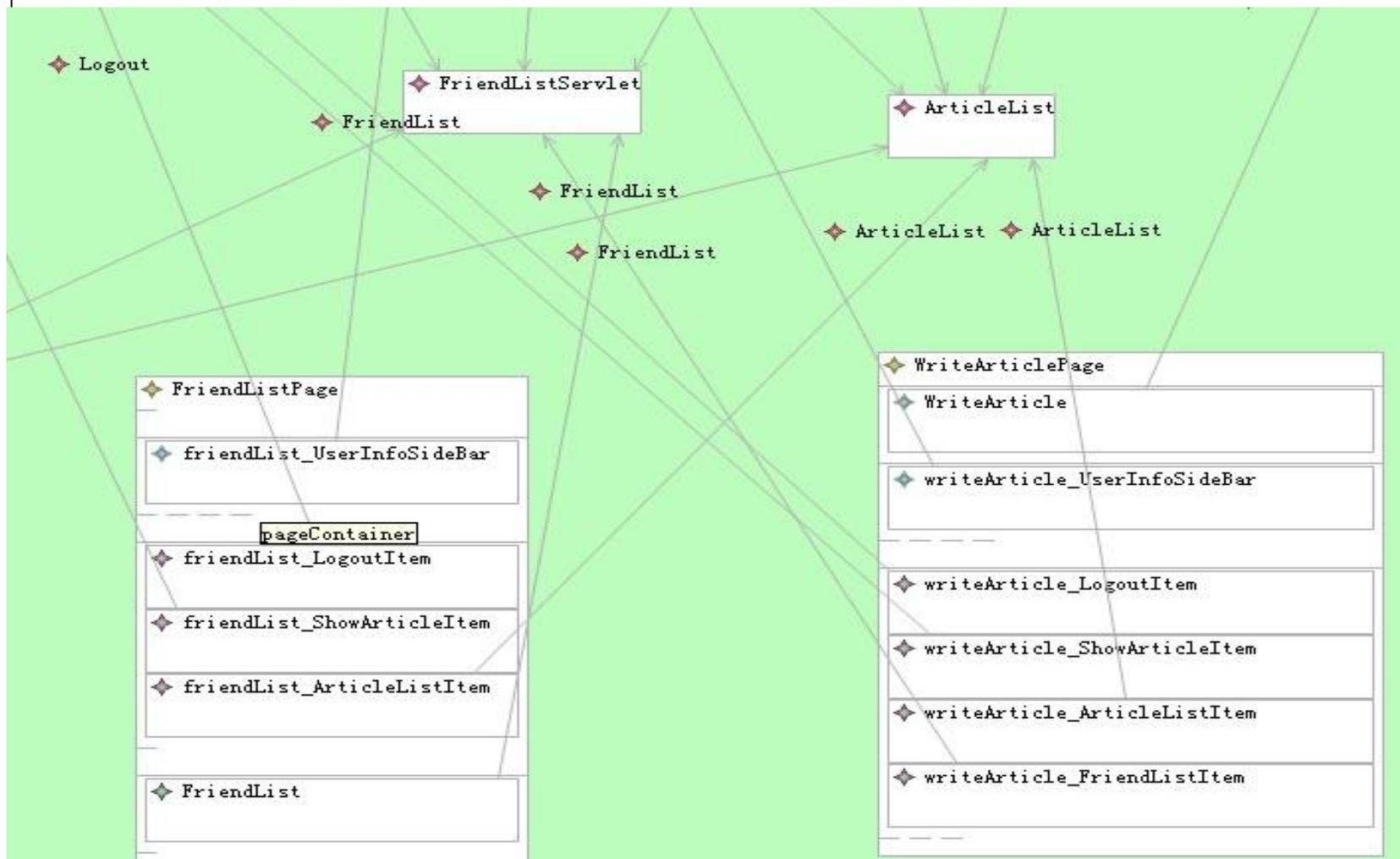
数据访问层设计模型



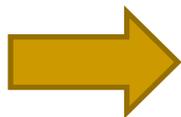
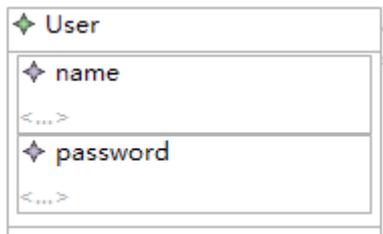
业务逻辑层设计模型



展现层设计模型



数据访问层的代码生成



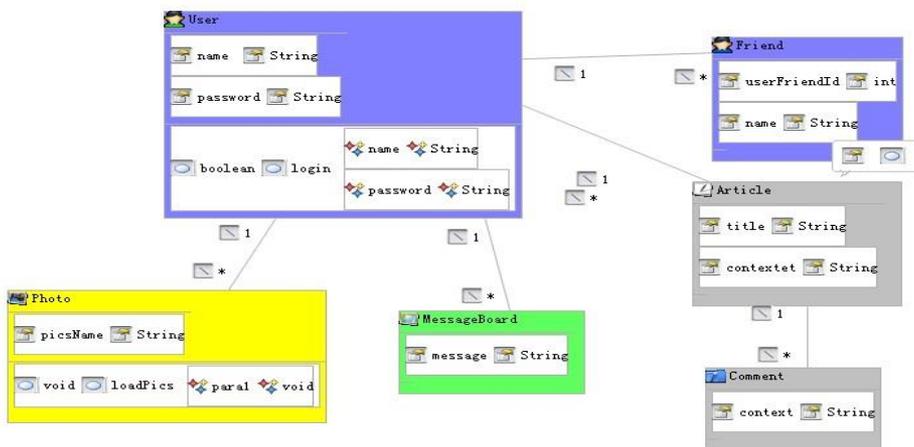
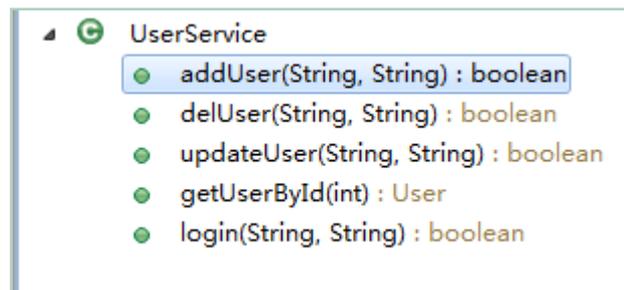
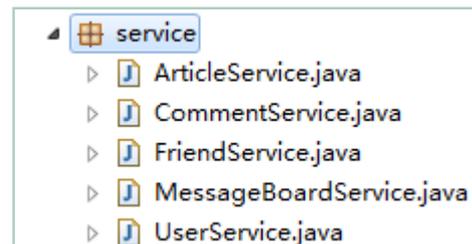
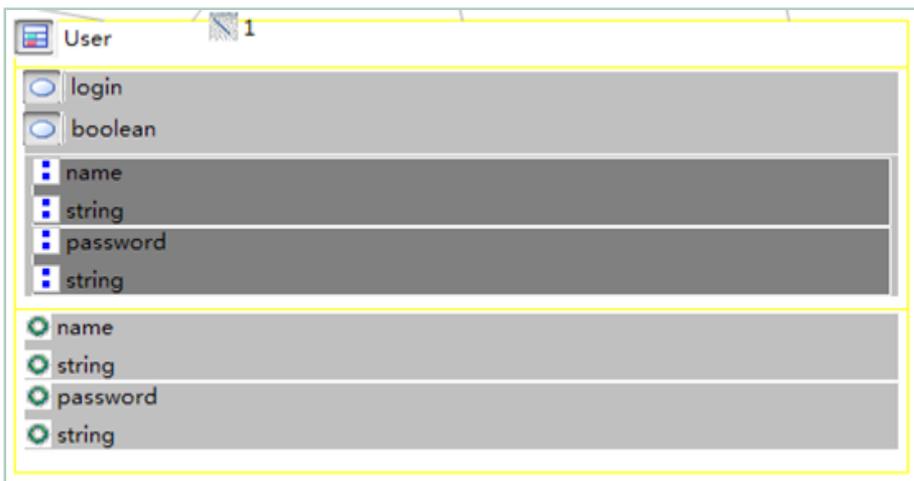
```
User
  name : String
  password : String
  UserId : int
  setId(int) : void
  getUserId() : int
  getName() : String
  setName(String) : void
  getPassword() : String
  setPassword(String) : void
```

```
dao
  Article.java
  ArticleDao.java
  Comment.java
  CommentDao.java
  DBUtil.java
  Friend.java
  FriendDao.java
  MessageBoard.java
  MessageBoardDao.java
  User.java
  UserDao.java
  CreateDB.sql
  createTable.sql
```

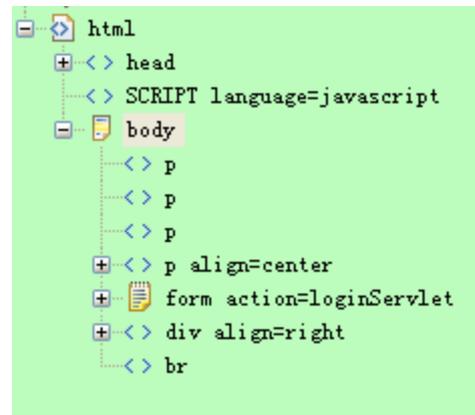
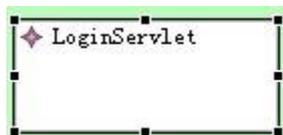
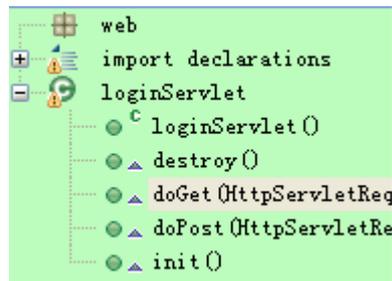


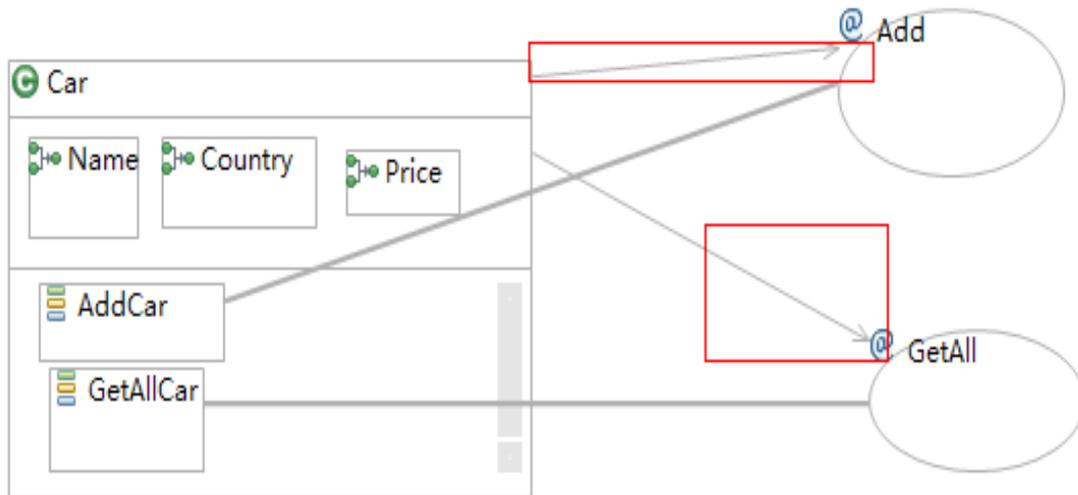
```
UserDao
  login(String, String) : boolean
  getArticleList(int) : List
  getCommentList(int) : List
  getMessageBoardList(int) : List
  getFriendList(int) : List
  addUser(User) : boolean
  delUser(User) : boolean
  updateUser(User) : boolean
  getUserById(int) : User
```

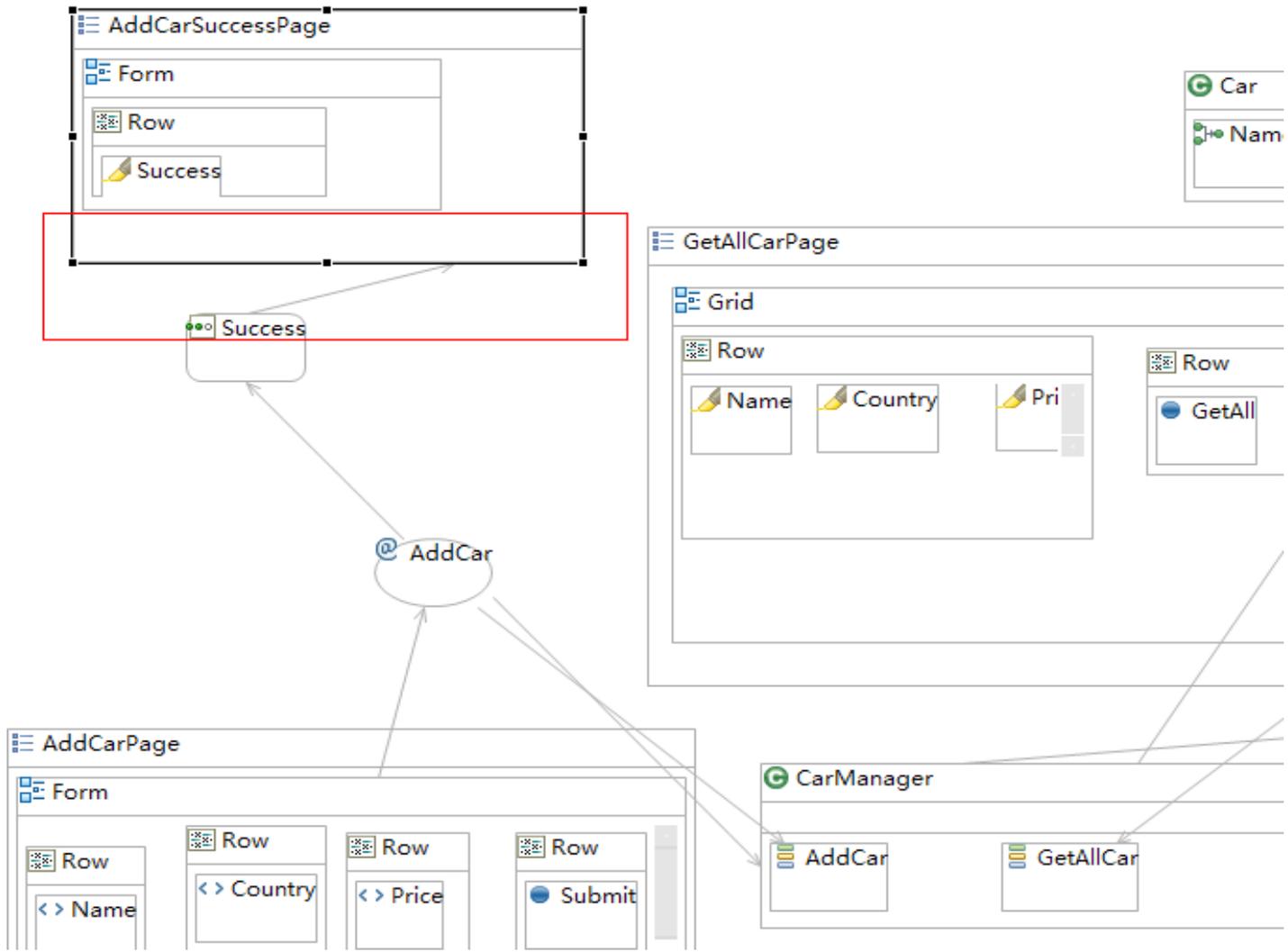
业务逻辑层代码生成



展现层代码生成







← → 🚫 💰 http://localhost:8080/mda.web/AddCarPage.jsp

Name:

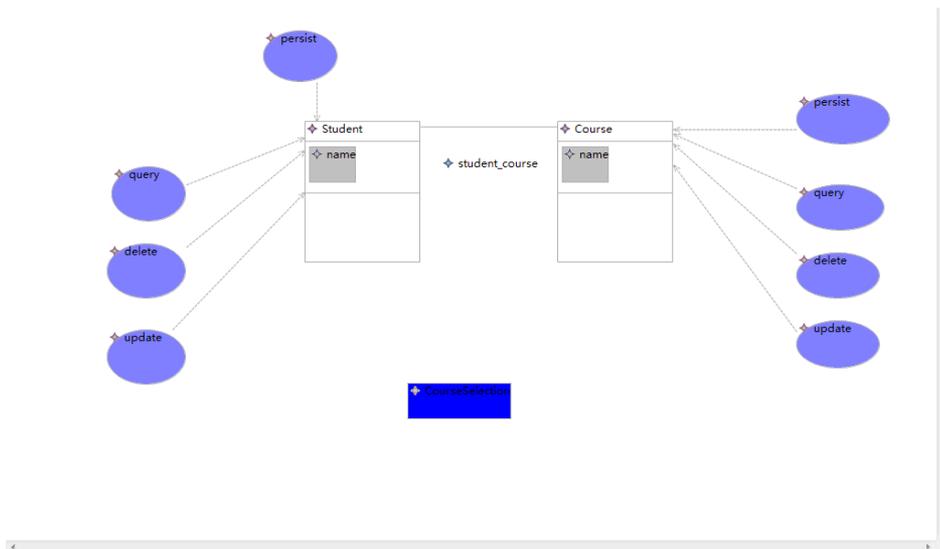
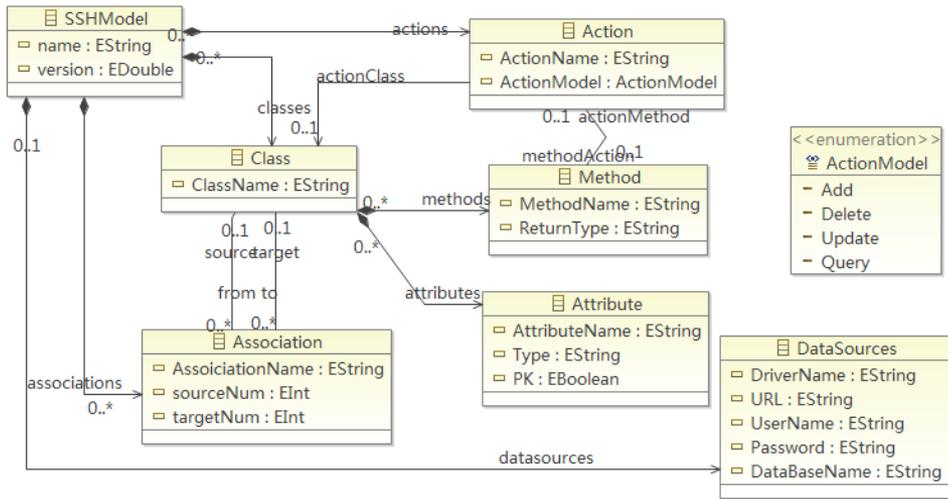
Country:

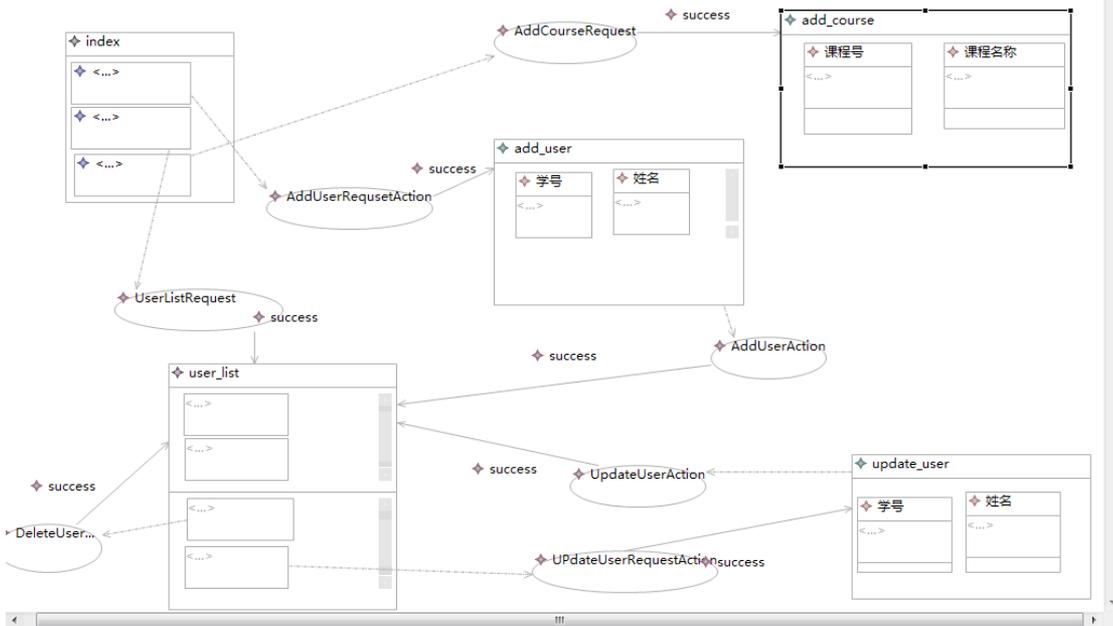
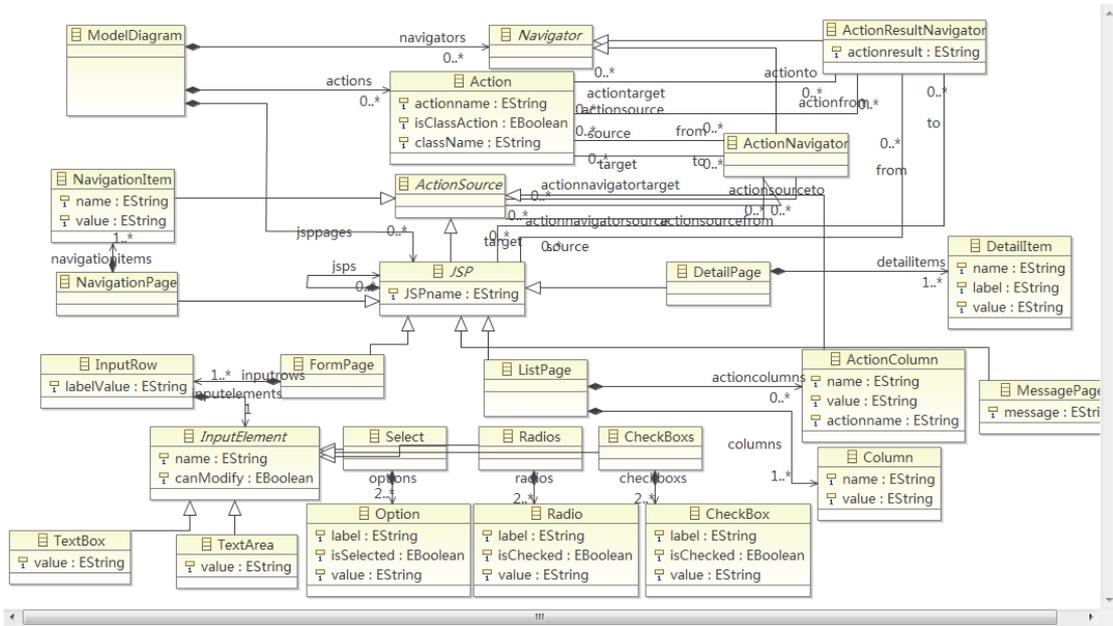
Price:

Submit

← → 🚫 💰 http://localhost:8080/mda.web/GetAllCar.action ▼

Name	Country	Price
qq	china	40000





1101210749

李鹏

已选课程

课程号	课程名称	
0005	信息检索	删除
0002	软件实现技术	删除

未选课程

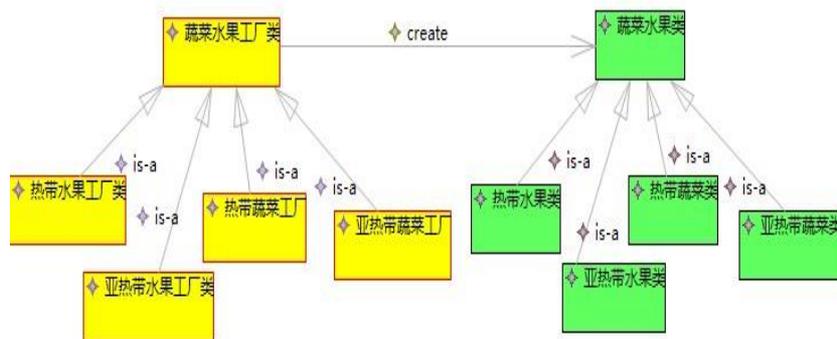
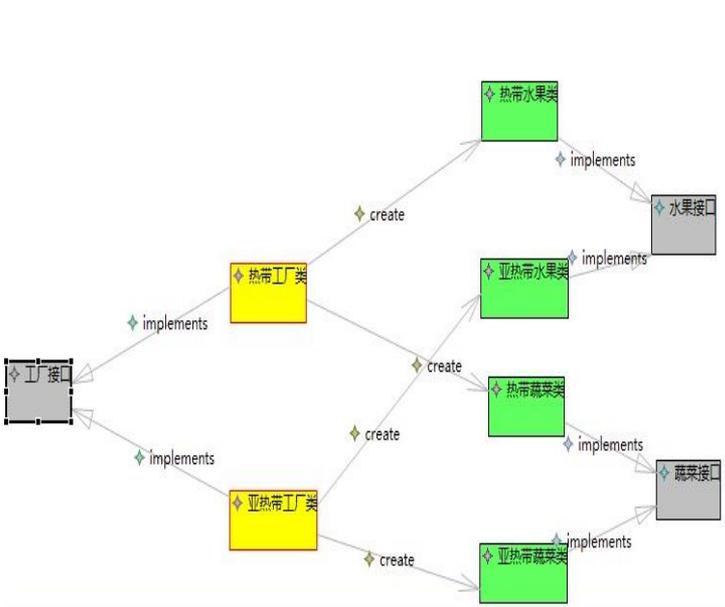
课程号	课程名称	
0001	模型驱动开发	选课
0003	机器学习	选课
0004	操作系统高级课程	选课
0006	体系结构	选课
zzz	dddd	选课

[离开](#)

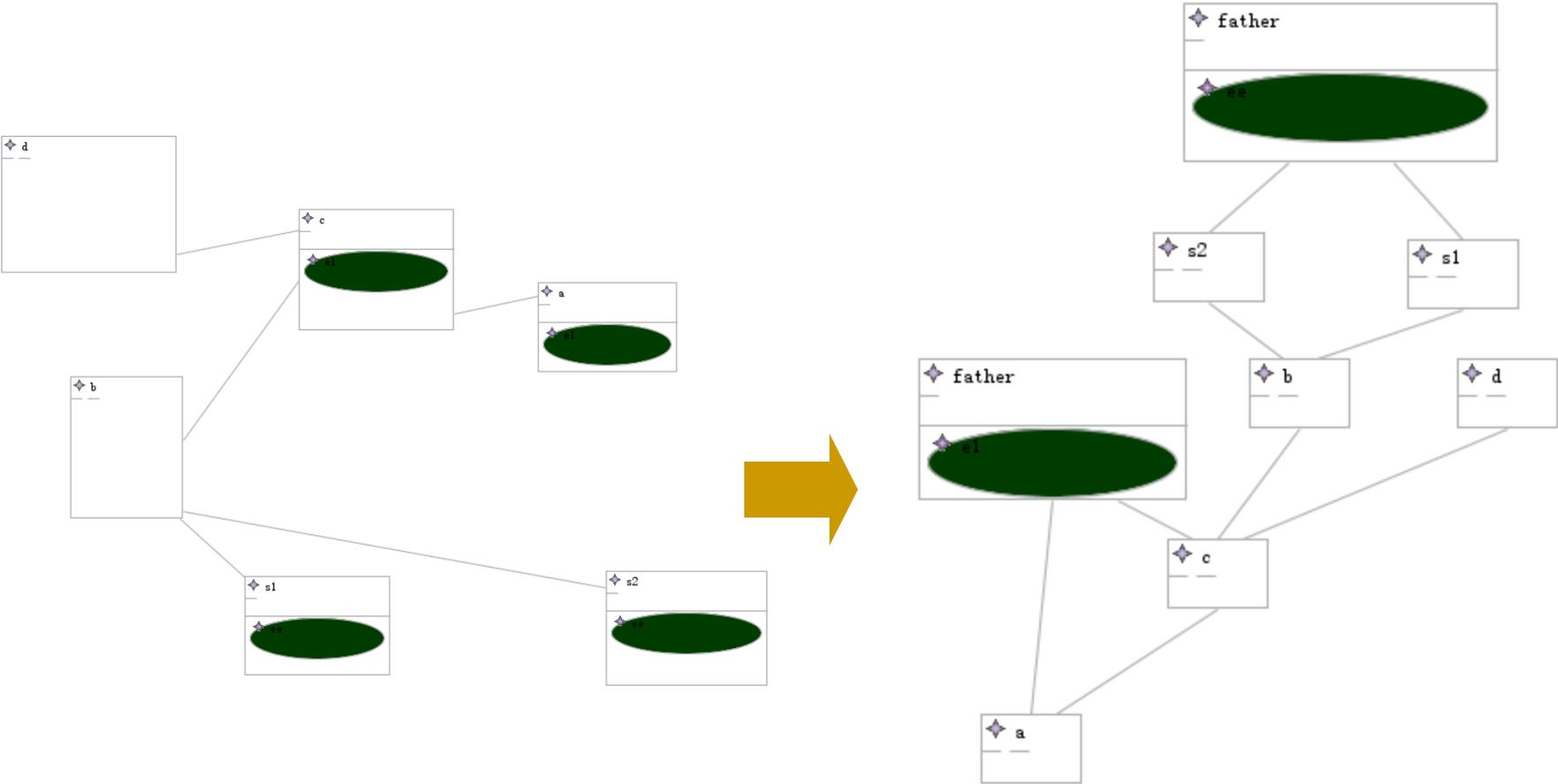
例6.1:取消目标模型元素的自关联

- mapping Line::lineToline():Line
- when {
- not (not self.source->resolveoneIn(ellipse::eTc)->isEmpty() and
- not self.tareget->resolveoneIn(ellipse::eTc)->isEmpty())
- and self.source->resolveoneIn(ellipse::eTc)=self.tareget->resolveoneIn(ellipse::eTc))
- }
- {
-
- }
- 在lineToline中加入when 子句

抽象工厂模式转化为一般工厂模式



将相同的子结构提取出来



将相同的子结构提取出来

```
> modeltype simplegraph uses 'http://simplegraph/';
> transformation NewTransformation(in source : simplegraph, out target:simplegraph);
> main() {
>   source.objectsOfType(simplediagram)->map SDToSD();
> }
> mapping simplediagram::SDToSD():simplediagram {
>   self.nodes->forEach(r)
>   {
>     // result.nodes+=self.findNodeWithSameEllipse(r);
>     if self.findNodeWithSameEllipse(r)->isEmpty() or self.findNodeWithSameEllipse(r)=null
then result.nodes+=r.map NodeTNode()
>     else
>     { var f:=resolveIn(abstractsubnode, node)->select(e|e.nodename='father' and
e.components[ellipse]->first().nodename=r.components[ellipse]->first().nodename);
>     if f->notEmpty() then {continue} endif;
>     self.findNodeWithSameEllipse(r)->forEach(k){
>       result.nodes+= map abstractsubnode(r,k).father;
>       result.lines+= map abstractsubnode(r,k).l1;
>       result.nodes+= map abstractsubnode(r,k).son;
>     }
>   }
>   endif
> };
> result.lines+=self.lines->map lineToline();
> }
```

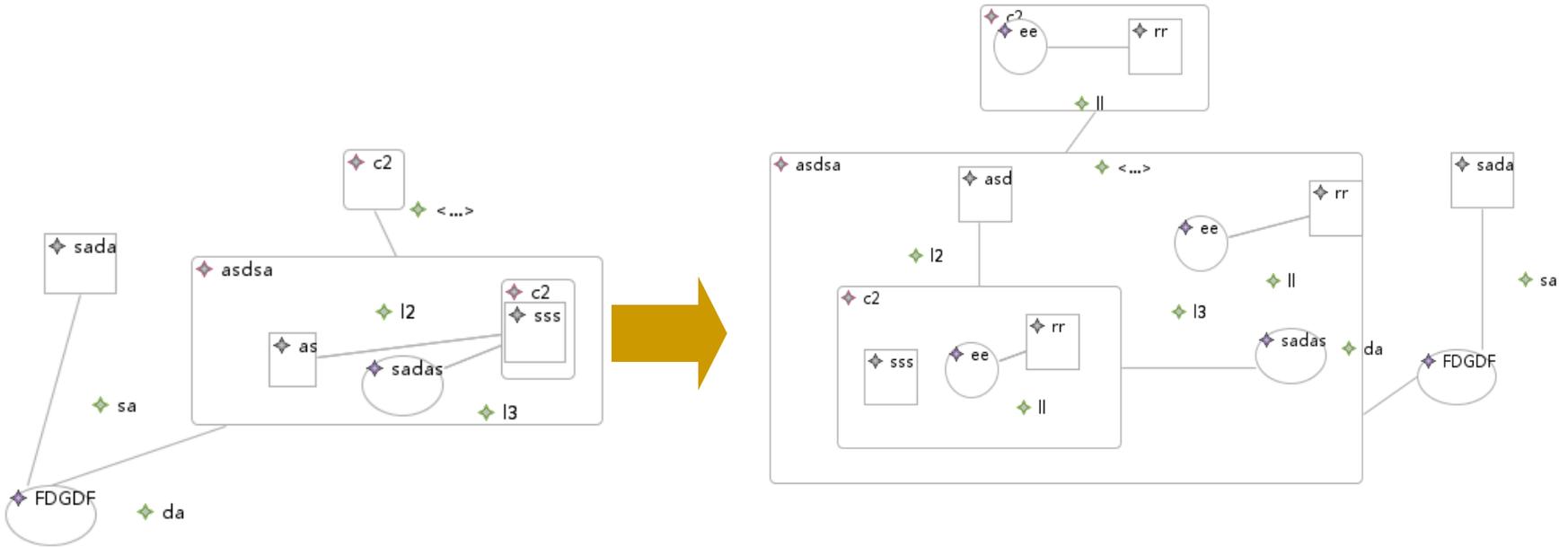
将相同的子结构提取出来

```
> mapping node::NodeTNode():node{
>   result.nodename:=self.nodename;
>   result.to:=self.to->map lineToline();
>   result._from:=self._from->map lineToline();
>   result.components:=self.components[ellipse]->map ellipseToellipse();
>   result.components+=(self.components[rectangle]->map rectangleTorectangle());
> }
> helper simplediagram::findNodeWithSameEllipse(n:node) : Set(node) {
>   var renodes: Set(node) ;
>   //if n.components[ellipse]=null then return null endif;
>   var e :ellipse=n.components[ellipse]->first();
>   self.nodes->forEach(m){
>     m.components[ellipse]->forEach(f){
>       if f.nodename=e.nodename then {
>         renodes+=m;
>       } endif;
>     }
>   };
>   return renodes;
> }
> mapping line::lineToline():line{
> }
> mapping ellipse::ellipseToellipse():ellipse{
>   result.nodename:=self.nodename;
> }
> mapping rectangle::rectangleTorectangle():rectangle{
>   result.nodename:=self.nodename;
> }
```

将相同的子结构提取出来

```
> mapping abstractsubnode(in n1:node,in n2:node): father:node, son:node,l1:line
> {
>   init{
>     var x:String =n1.components[ellipse]->first().nodename;
>     var f:=resolveIn(abstractsubnode, node)->select(e|e.nodename='father' and e.components[ellipse]-
>first().nodename=x);
>     if f->notEmpty() then father:=f->first() else{
>       object father:node{
>         nodename:='father';
>         components+=object ellipse{nodename:=x};
>       };
>     } endif;
>
>     object son:node {
>       son.nodename:=n2.nodename;
>       son.to:=n2.to->map lineToline();
>       son._from:=n2._from->map lineToline();
>       //son.components:=n2.components[ellipse]->map ellipseToellipse();
>       son.components+=(n2.components[rectangle]->map rectangleTorectangle());
>     };
>     object l1:line{
>       l1.source:=son;
>       l1.target:=father
>     };
>   }
> }
```

产生内部结构:内部结构的连线



产生内部结构:内部结构的连线

```
modeltype myfirstGMF uses 'http://myfirstGMF/';
transformation NewTransformation(in source : myfirstGMF, out
target : myfirstGMF);
main() {
source.objectsOfType(myfirstDiagram)->map SDToSD();
}
mapping myfirstDiagram::SDToSD():myfirstDiagram {
// result.component:= self.component[certainernode]->map cTc();
self.component[certainernode]->forEach(c){
    result.component+=c.map cTc(result);
};
result.component+= self.component[rectangle]->map cTc(result);
result.component+= self.component[ellipse]->map cTc(result);
result.lines+=self.lines->map lineToline();
}
```

产生内部结构:内部结构的连线

- mapping Node::nTn(inout m:myfirstDiagram):Node{
- result.name:=self.name;
- result.to:=self.to->map lineToline();
- result._from:=self._from->map lineToline();
- }
- mapping certainernode::cTc(inout m:myfirstDiagram):certainernode inherits Node::nTn {
- sname:=self.sname;
- result.sons+= self.sons[certainernode]->map cTc(m);
- result.sons+= self.sons[rectangle]->map cTc(m);
- result.sons+= self.sons[ellipse]->map cTc(m);
- **m.lines+=map addinterStruct(result).sl;**
- }
- 增加参数**inout m:myfirstDiagram**的目的是为了在产生**line**时,将其加入到**myfirstDiagram**容器, **inout** 目的是需要修改传入的参数.

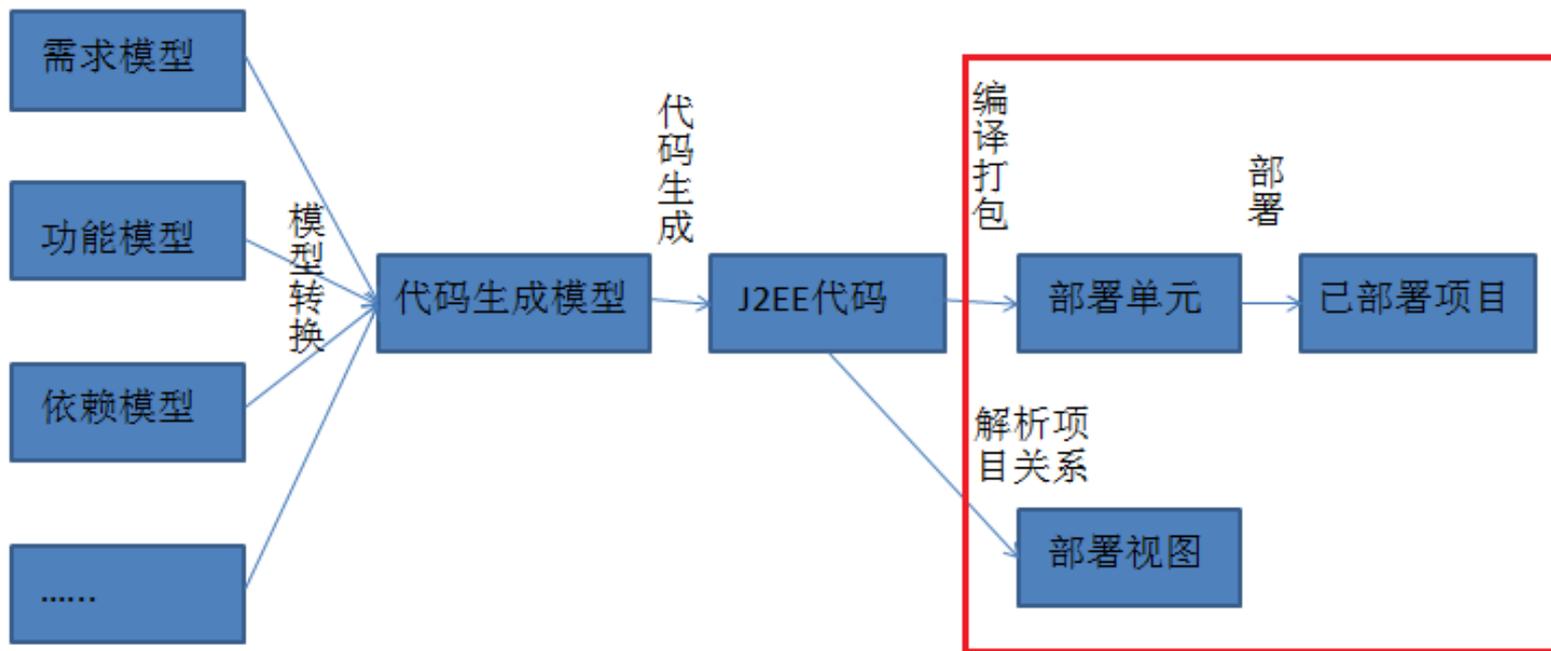
产生内部结构:内部结构的连线

- mapping addinterStruct(inout c: certainernode):e:ellipse,r:rectangle,sl:Line
- {
- init{
- e:= new ellipse('ee');
- r:= object rectangle { name:='rr' };
- sl:=object Line{ Lname:='ll' };
- }
- c.sons+=e;
- c.sons+=r;
- sl.source:=e;
- sl.tareget:=r;
- }
- **addinterStruct**用于将**certainernode** 中增加新的子结构,注意**init{}**此处不能省略

特点

- 1.自动化程度高.
- 通过实验原型的研发,我们已经完全掌握并熟练运用了完整的从元建模到模型转换再到代码生成的全部关键技术,实现特定领域的从需求模型到设计模型的转化,从设计模型到代码的转换,实现整个软件开发流程的模型自动转换（基于**QVT**）和代码的自动生成（基于**QVT M2T**）.而**IBM**相应的研发团队只是在模型转换代码生成方面，并且用到的代码生成技术（基于**JET**）不如我们的先进，模型转换完全采用编程方式实现。
- 2. 软件开发效率高。我们的研发团队可以在很短的时间内针对特定领域搭建特定领域的图形化建模工具、模型转换工具和代码生成工具。从而在很大程度上提高软件的开发效率。
- 3. **100%**的代码生成。代码生成技术不仅能够提高软件的开发效率。而且也能够保证代码的质量。并且模型驱动的方法不同于以往的代码生成技术，可以通过不同阶段的模型的方式，对代码进行维护。并不存在传统观念中认为的生成的代码可维护性差的问题。

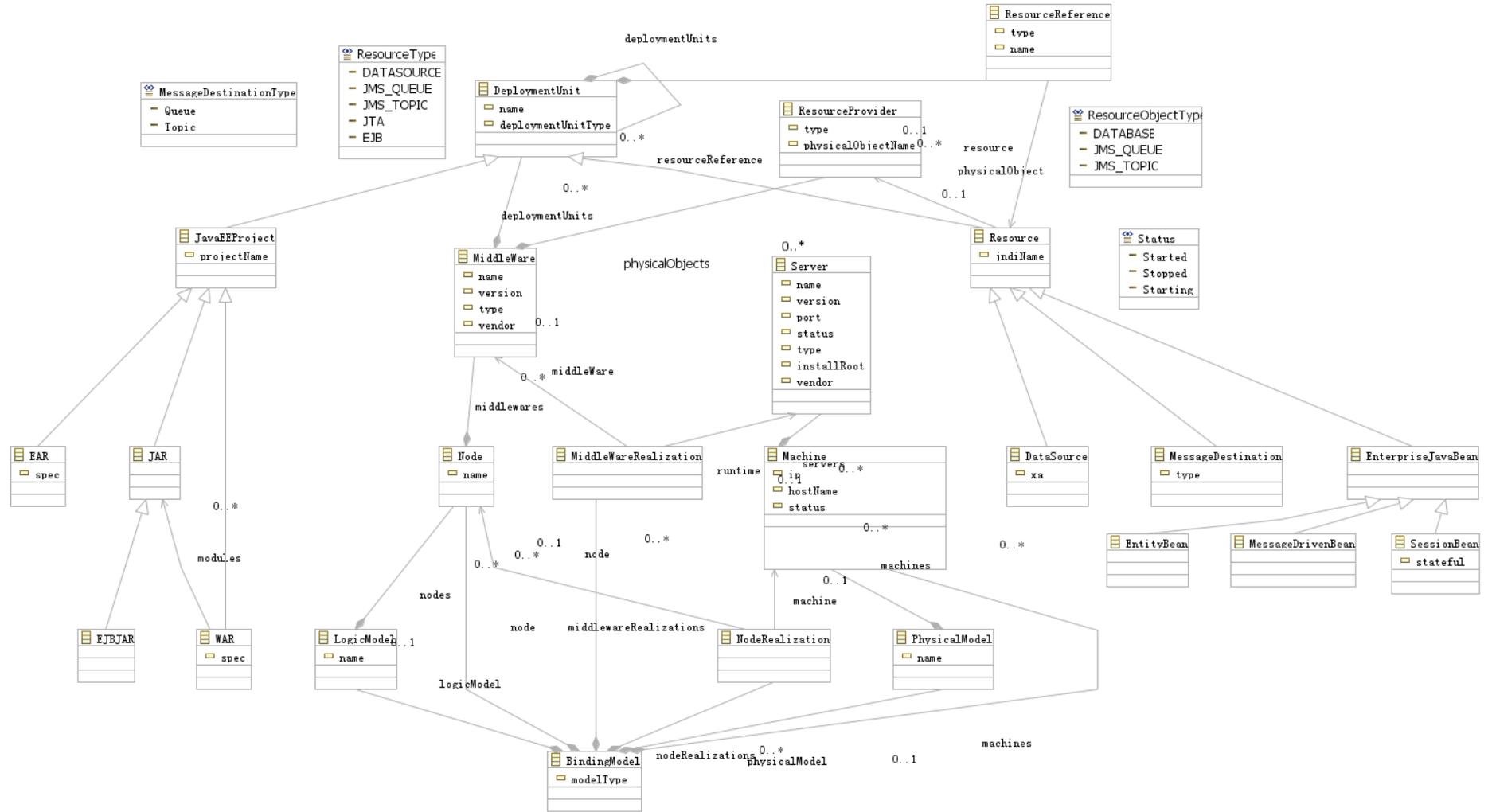
基于模型驱动的J2EE部署模型工具



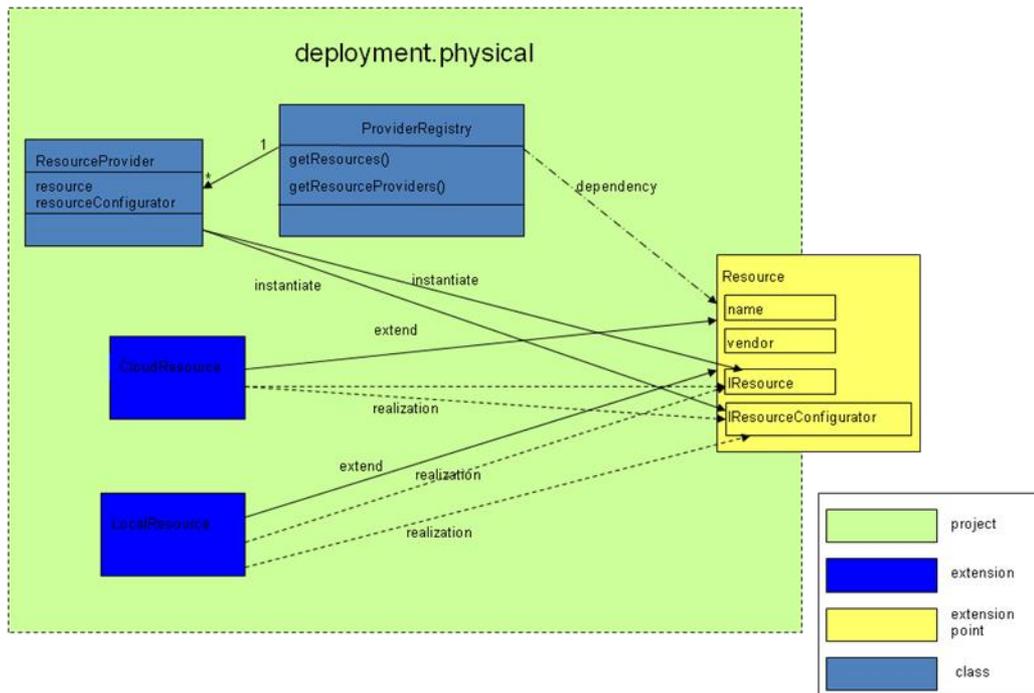
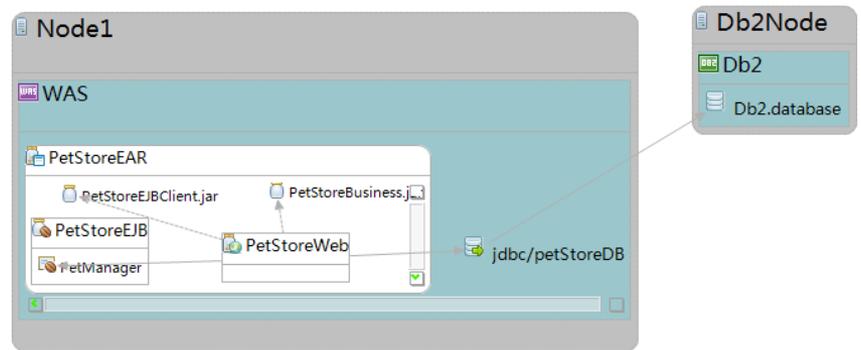
大型项目的部署的问题

- 为大型项目的部署提供一个宏观的部署视图：
 - 很多部署协助工具能够完成编译、打包、拷贝的功能，但是这些一般都是针对结构相对简单的项目，而且没有一个图形化的界面提供给用户，不能让用户对整个部署的拓扑结构一目了然。
 - 如何自动生成正确的编译、部署脚本，并与项目保持一致：
 - 大多数人可能不会部署脚本的编写；保持部署脚本和项目的一致性问题，一旦项目的结构发生了变化，很可能需要重新对脚本进行修改。
- 远程服务器的项目的部署
 - **Eclipse**或者**RSA**都提供了相应的插件能够很好的满足与本地服务器的集成，比如返回本地服务器的状态，将项目编译、打包并拷贝到本地服务器的相应目录完成部署，但是针对与远程的服务器，一般还是得部署人员或者编写部署脚本，或者直接登录到相应的机器，进行远程的操作。

部署元模型



部署模型



部署结构建模工具

The screenshot displays a software development environment for deployment structure modeling. The interface is divided into several panes:

- Outline / Project Explorer:** Located at the top left, showing a hierarchical tree of the deployment model. A red box highlights this area with the label "缩略视图" (Thumbnail View).
- Task List / Buddy:** Located below the Outline, containing a search field and a list of tasks.
- Deployment Model 1.0 Table:** Located at the bottom, showing properties for the deployment model. A red box highlights this area with the label "属性视图" (Property View).
- Graphical View:** The central area, labeled "图形视图" (Graphical View), shows a detailed diagram of the deployment structure. It features two nodes: "Node1" and "Db2Node". "Node1" contains a "WAS" (WebSphere Application Server) component, which includes "jdbc/pets" and "PetStoreEAR". "PetStoreEAR" is further decomposed into "PetStoreEJB", "PetStoreBu", and "PetStoreEJB". "PetStoreWeb" is also shown, connected to "PetStoreEJB". "Db2Node" contains a "Db2" component, which includes "Db2.database".
- Tool View:** Located on the right side, labeled "工具视图" (Tool View), it shows a palette of components and resources available for the deployment model, such as "Nodes", "Middlewares", "Deployment Units", "Java EE Resources", and "Database".

Core	Property	Value
	Name	
Rulers & Grid	ResourceResolvers	LocalResource1
Appearance	Version	1.0

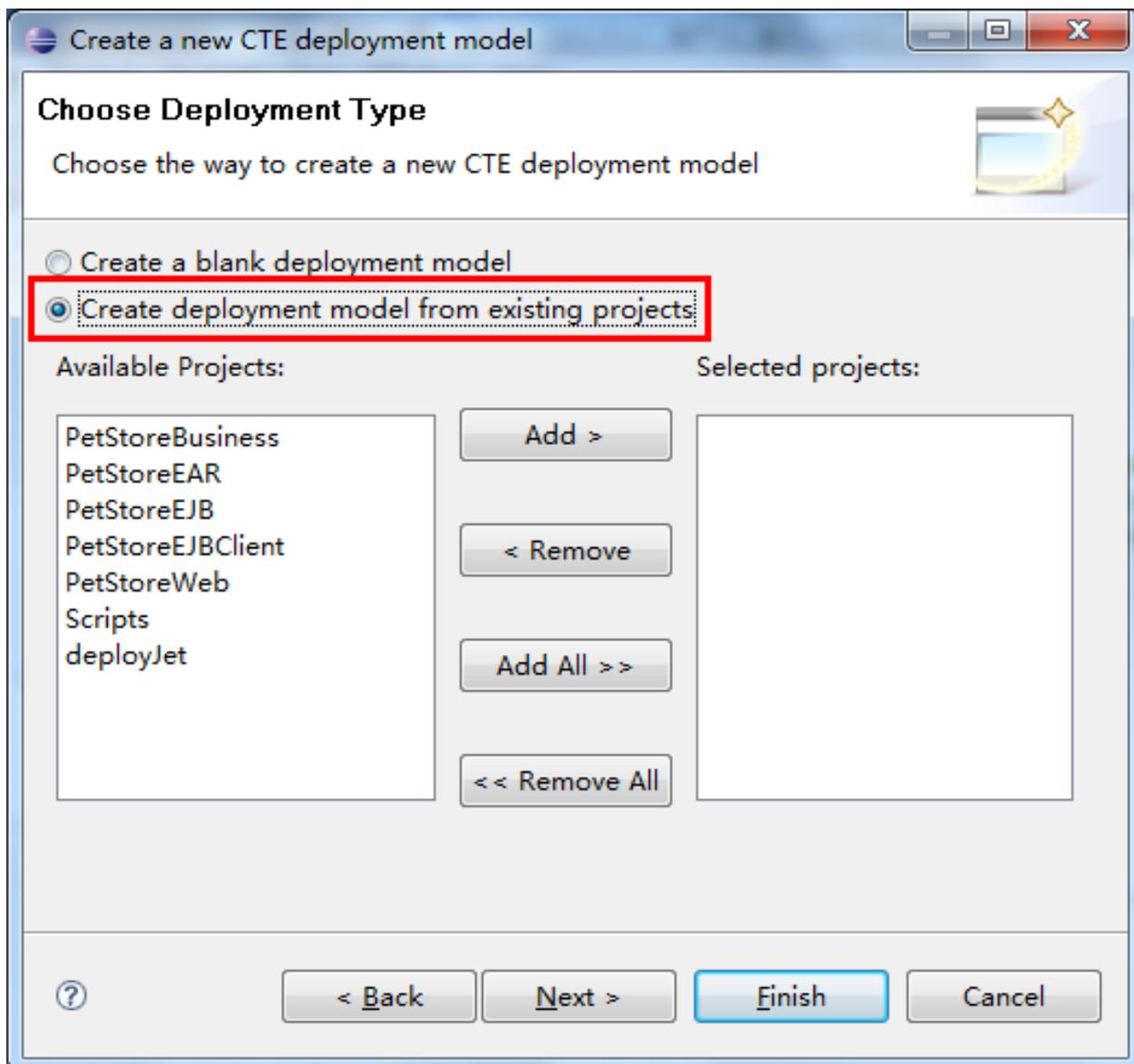
一个典型的J2EE项目作为实例演示

序号	项目名称	项目作用
1	PetStoreBusiness	提供一些必要常用的API供其它项目引用
2	PetStoreEAR	用于部署打包的EAR项目
3	PetStoreEJB	包含多个Session Bean和Entity Bean，完成整体项目的业务逻辑
4	PetStoreEJBClient	客户端利引用此接口对服务端的服务进行调用，从而完成业务调用
5	PetStoreWeb	负责显示和交互的逻辑

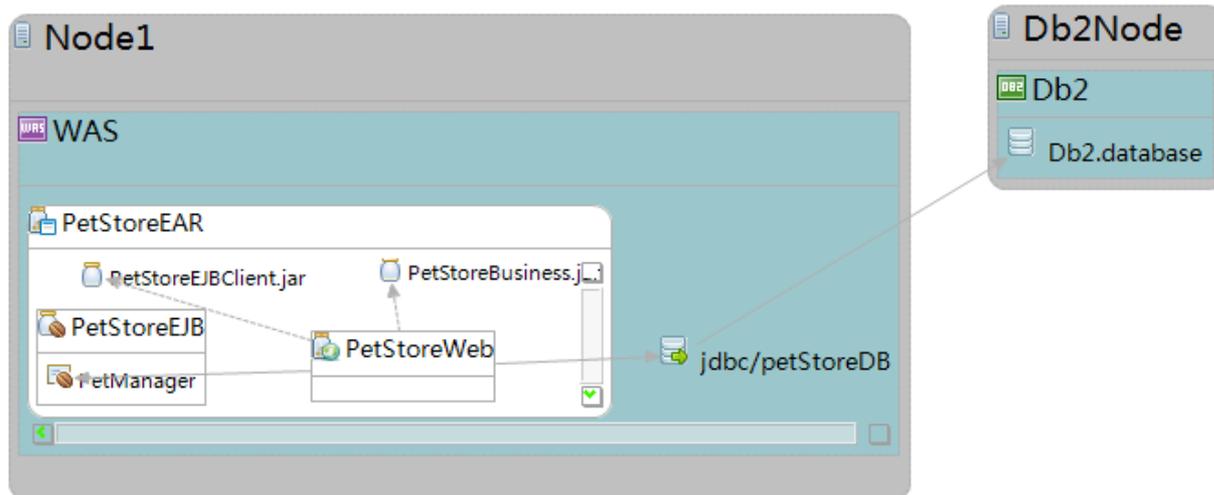
➤ 当将**Workbench**集成到**Eclipse**中去后，运行**Eclipse**插件后，用户可以选择根据项目产生部署模型，下图展示了启动之前的效果：



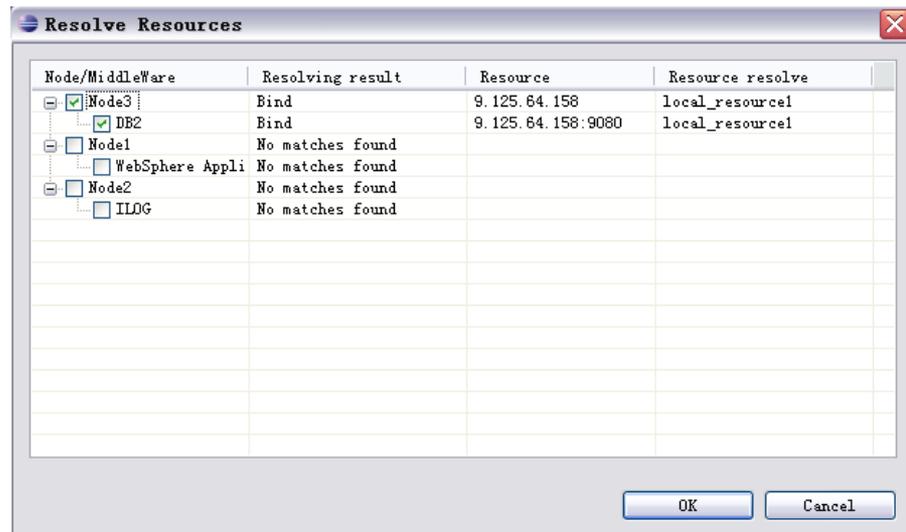
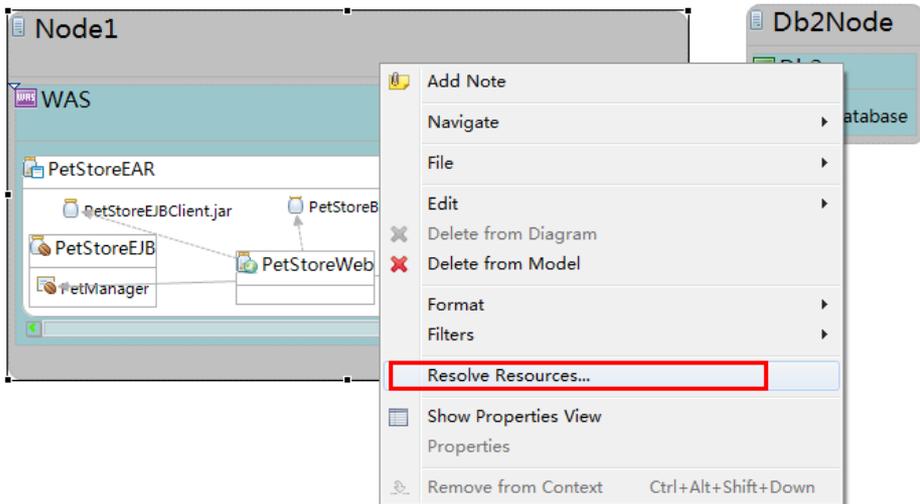
产生部署模型



产生逻辑部署模型



逻辑模型绑定到物理模型



生成部署模型

```
> <!-- PetStoreEAR properties -->
>     <property name="PetStoreEAR.name" value="PetStoreEAR" />
> <property name="PetStoreEAR.build.dir" value="${build.dir}/${PetStoreEAR.name}" />
> <property name="PetStoreEAR.dest.dir" value="${PetStoreEAR.build.dir}/classes" />
> <path id="PetStoreEAR.classpath">
>     <fileset dir="${PetStoreEAR.jar.dir}" includes="*.jar" />
> </path>
> <!-- PetStoreEAR Init -->
> <target name="PetStoreEAR_init" depends="global_init">
> <tstamp />
>     <echo message="---- Deployment Unit[${PetStoreEAR.name}] ----" />
>     <echo message="" />
> <echo message="Deployment unit type = ${PetStoreEAR.type}" />
> <mkdir dir="${PetStoreEAR.build.dir}" />
> </target>
```

➤ 成功运行项目后的效果图

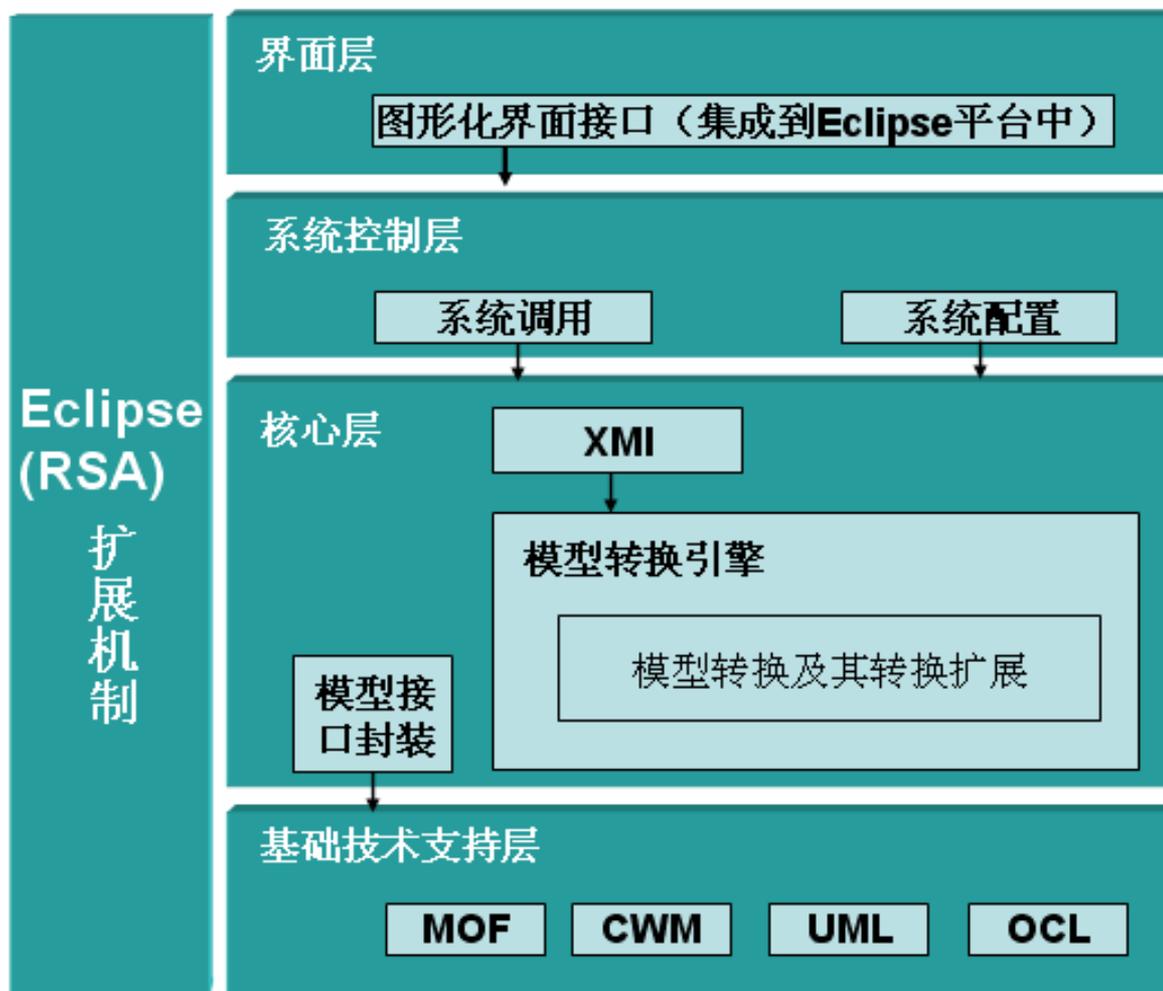
http://localhost:8080/PetStoreWeb/login.html

Login Page

UserName:

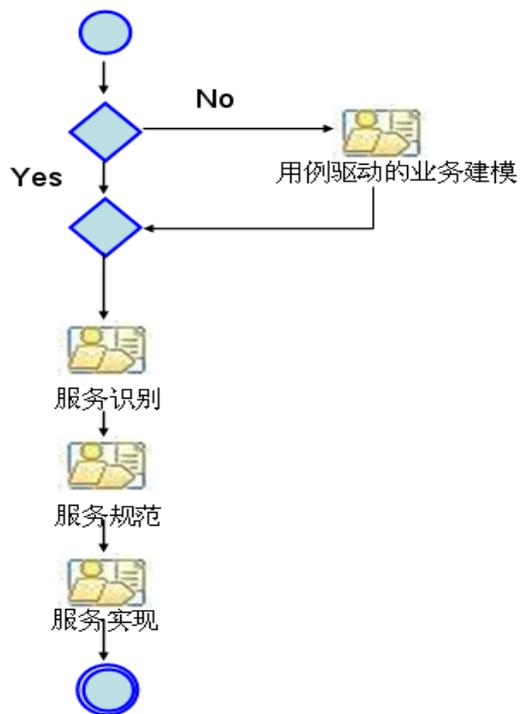
PassWord:

WBM到SOMA的模型转化系统

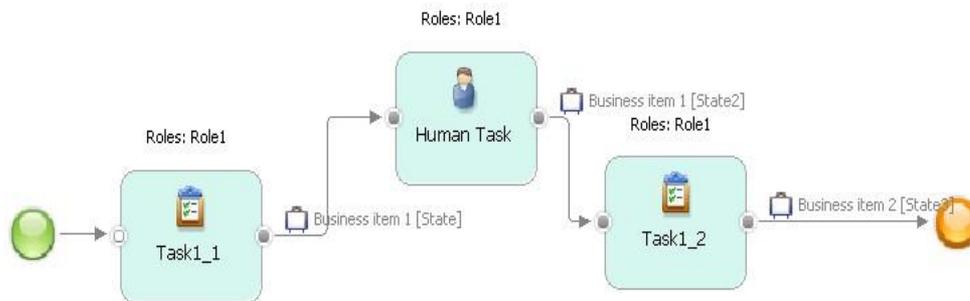


- 首先对要进行转换的三个模型进行分析并设计出它们之间模型元素的映射规则，利用**RSA**框架里的模型转换引擎根据映射规则将业务流程模型转换为服务模型，并最后转换为代码生成模型。
- 实现该设计方案的具体做法是先建立业务流程的领域模型，然后建立模型转换的**UML**类图，并通过**SWT/JFace**为该模型转换工具提供了图形化用户界面的实现，最后利用**Eclipse**的扩展机制以及**GMF**插件对模型进行转换。

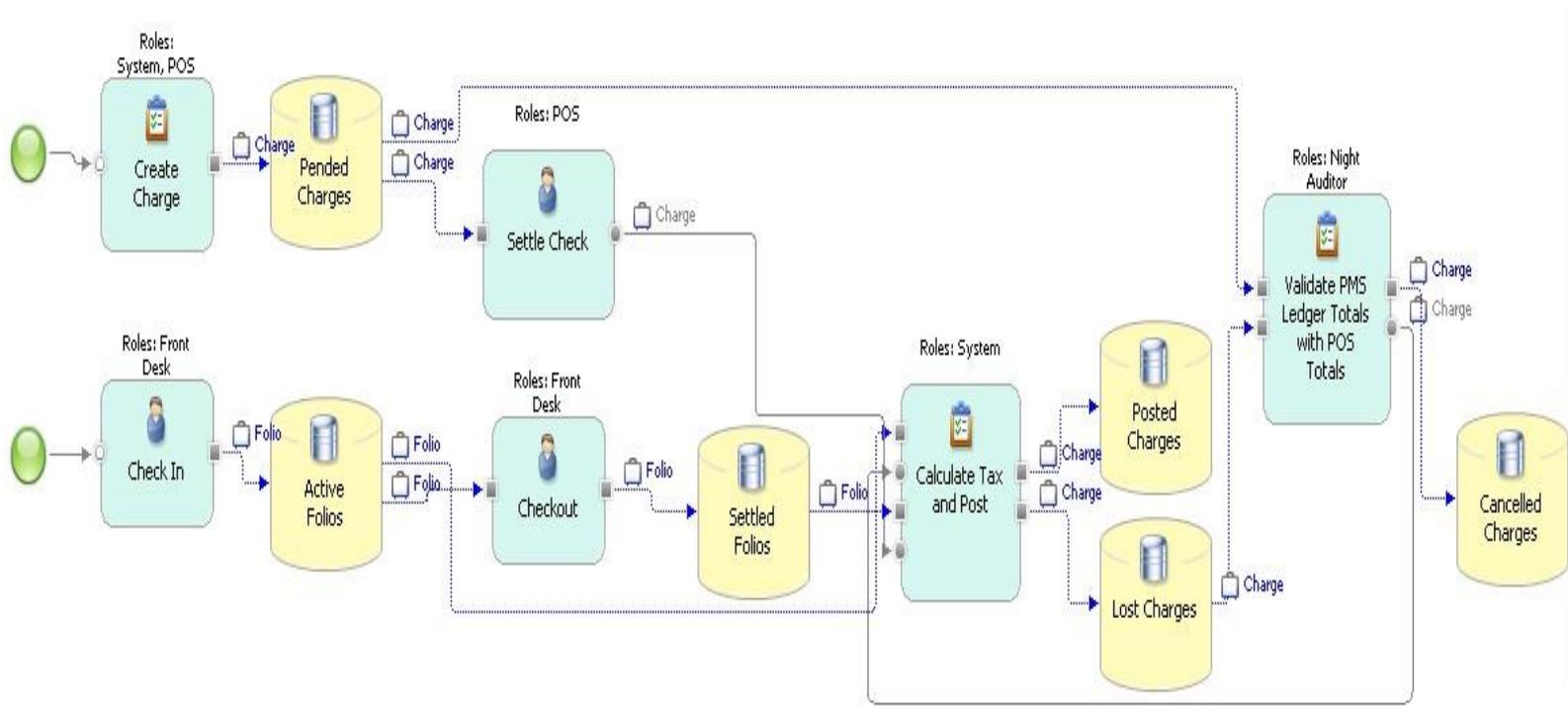
◆ SOMA解决方案 workflow



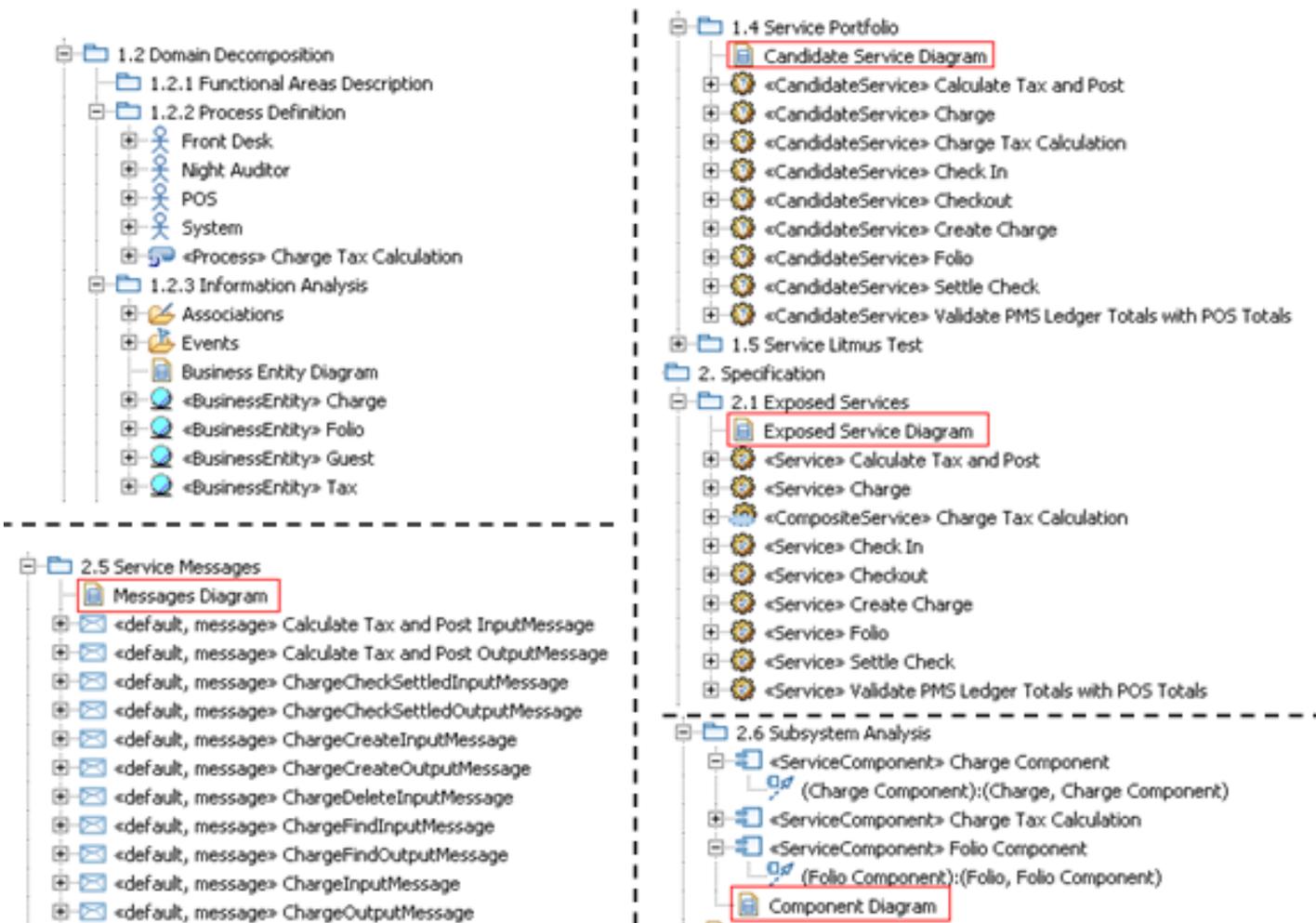
◆ WBM模型



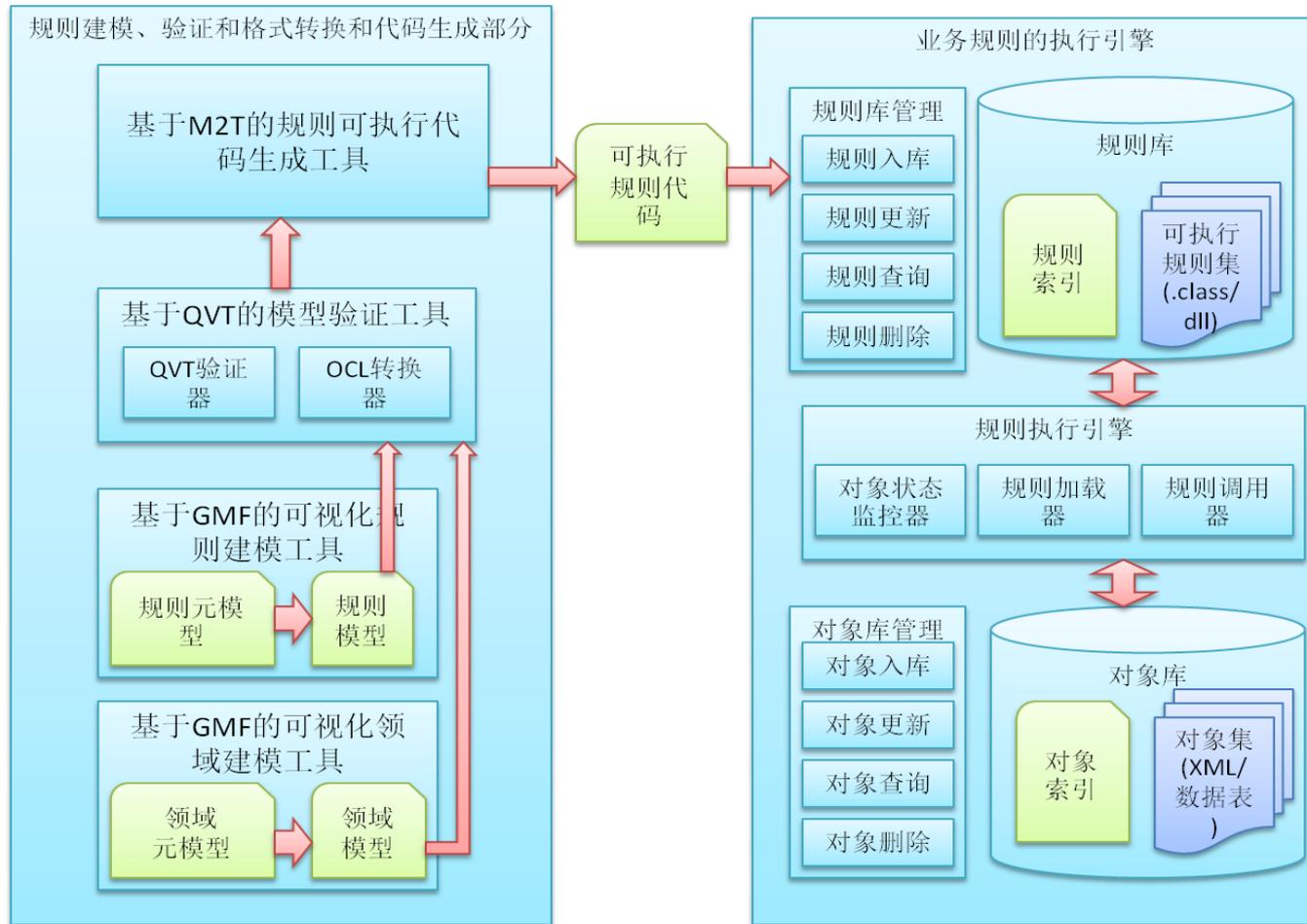
Hotel WBM业务流程模型



生成的SOMA服务模型元素构成



业务规则管理框架解决方案

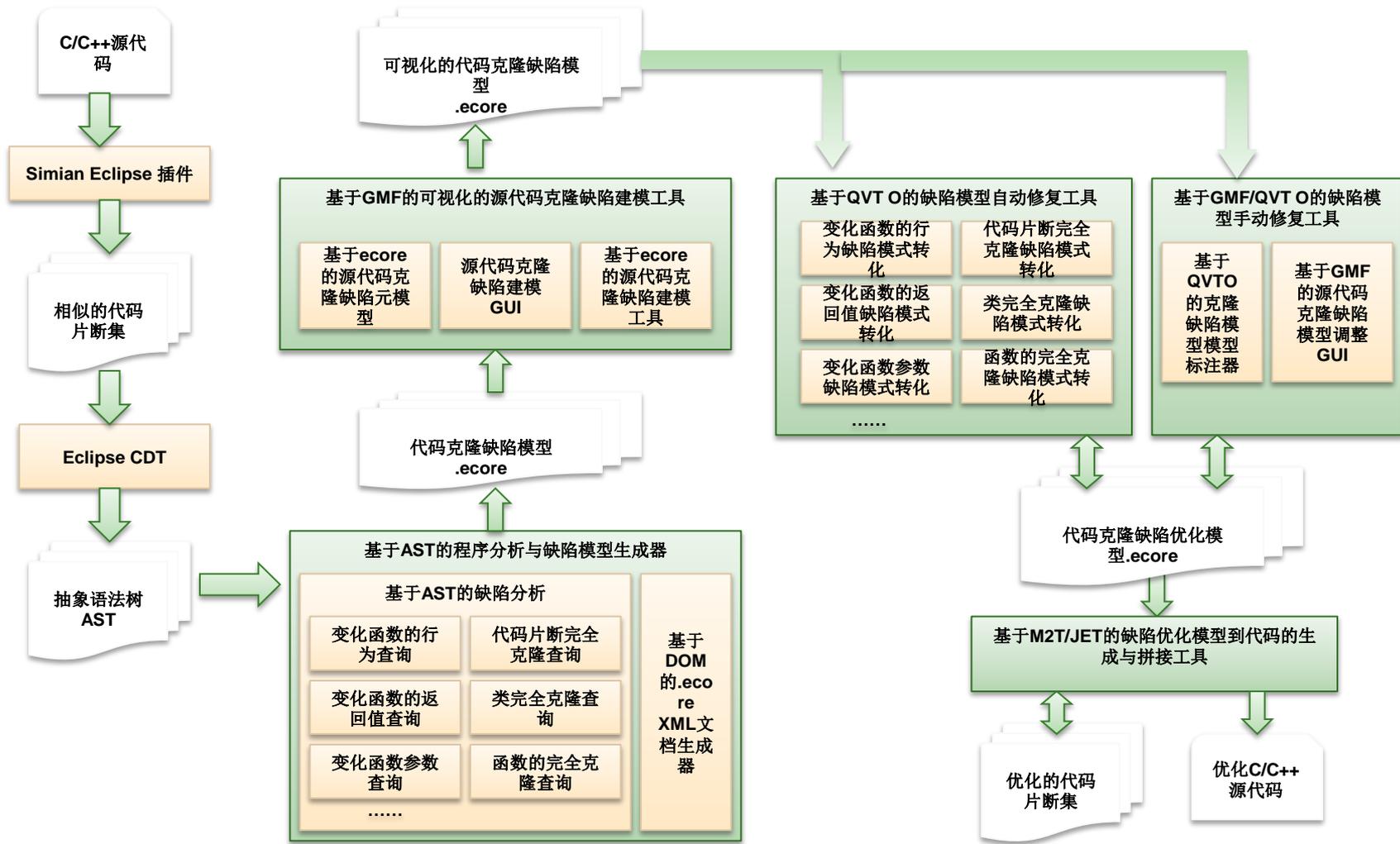


规则自动验证与到代码的自动转化工具

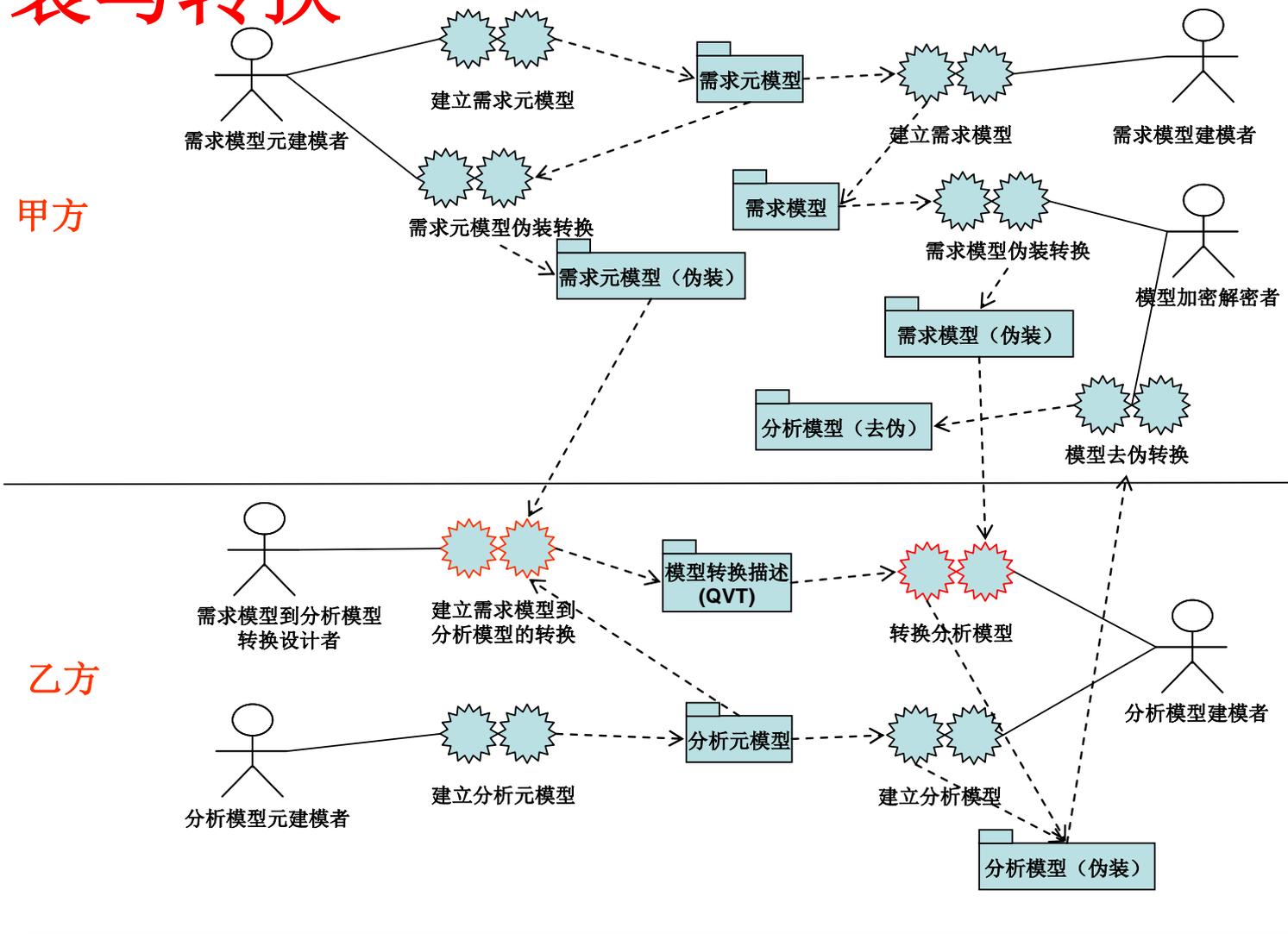
- 使用**GMF**技术建立规则适用的领域元模型与**RCP**建模工具,用于建立适用规则的可视化模型.
- 通过在元模型上编写**OCL**语言描述的约束,可以在模型层面验证**OCL**规则语义上的正确性。
- 针对复杂的规则，单纯**OCL**规则并不能完全满足要求。针对这种情况，使用**GMF**技术建立可视化的规则编辑器用于编辑规则，并通过与领域模型编辑器编辑的模型一起作为输入，进行验证和代码生成。

业务规则执行引擎

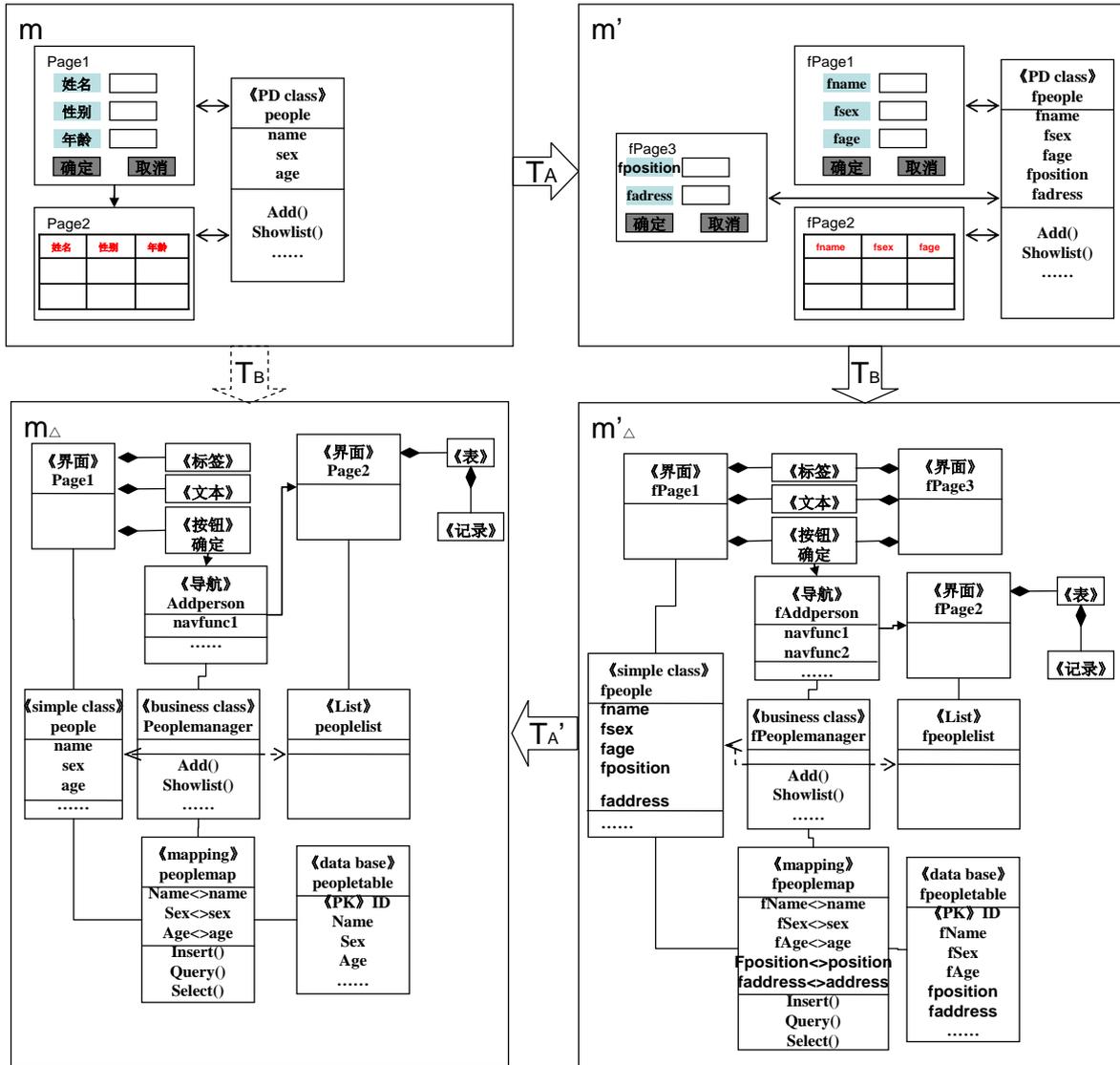
- 综合运用程序框架技术、程序动态加载与执行技术（**Java/dll**）、数据库管理技术，对第一部分中生成的可执行的规则代码进行全生命周期管理
- 同时对规则中引用的对象也进行生命周期管理，并提供一种程序自动加载与调用框架，根据其监控的各个对象的状态自动加载相应的规则代码，根据规则调用相应的执行动作。



软件外包领域模型驱动开发方法中模型伪装与转换

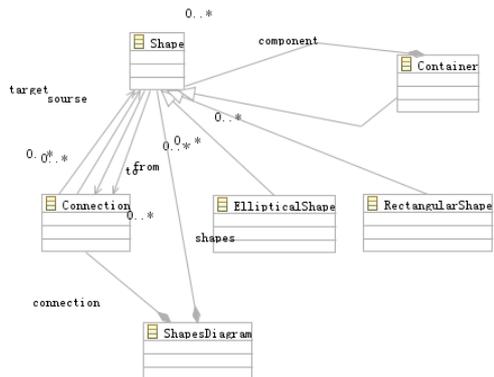


模型伪装与转换示意图



基于QVT的转换机制

源元模型

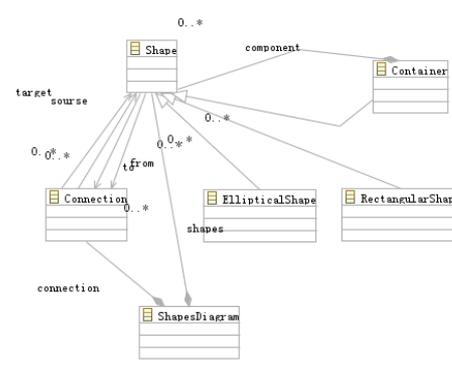


基于QVT operational的转换描述

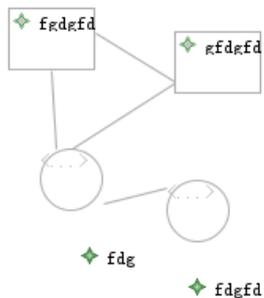
```

modeltype test2 uses 'http://jybtst2/';
transformation NewTransformation(in source : test2, out target:
test2);
main() {
source.objectsOfType(ShapesDiagram)->map SDToSD();
}
mapping ShapesDiagram::SDToSD():ShapesDiagram {
connection:=self.connection->CToC1SC2().c1;
self.connection->CToC1SC2().s->forEach(i){
shapes:=shapes->append(i)
};
self.connection->CToC1SC2().c2->forEach(i){
connection:= connection->append(i)
};
self.shapes->map ShapesToShapes()->forEach(i){
shapes:=shapes->append(i)
};
};
mapping Shape::ShapesToShapes():RectangularShape{
init{
object result:RectangularShape{
};
};
name:=self.name;
source:=self.source->map CToC1SC2().c1;
result.target:=self.target->map CToC1SC2().c2;
}
mapping
Connection::CToC1SC2():c1:Connection,s:RectangularSh
ape,c2:Connection{
object c1:Connection{
};
};
object s:RectangularShape{
};
object c2:Connection{
};
c1.to:=s;
s.name:="ss";
c2._from:=s;
}
    
```

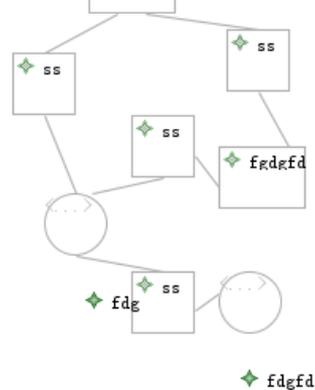
目标元模型



源模型



目标模型



下一代网络架构和协议的广义模型和转换

基于Acceleo的下一代交换机软件代码生成工具开发

控制器端代码生成工具

人机界面

控制

数据管理

通讯

交换机端代码生成工具

通讯

控制

交换

基于QVT的开放协议模型转换工具开发

交换机PIM结构模型到PSM结构模型的转换工具

交换机行为模型到PSM行为模型的转换工具

基于GMF的开放协议建模工具开发

PIM建模工具

结构Ecore元模型

行为Ecore元模型

结构OCL规则

行为OCL规则

PSM建模工具

PSM结构Ecore元模型

PSM行为Ecore元模型

PSM结构OCL规则

PSM行为OCL规则

谢谢！

欢迎选修模型驱动的软件开发技术！

