南京大學

本 科 毕 业 设 计

院　　系 计算机科学与技术系

专　　业 计算机科学与技术

题　　目 一种基于结构查询的 UML 设计模式识别方法

年　　级 2009　　学 号 091220132

学生姓名 许涵斌

指导老师 张天　　职 称 副教授

论文提交日期 2013 年 6 月

一种基于结构查询的UML设计模式识别方法

UML design pattern recognition method based on structured query

南京大学本科生毕业论文（设计）中文摘要

毕业论文题目： 一种基于结构查询的UML设计模式识别方法

计算机科学与技术 院系 计算机科学与技术

专业 2009 级本科生姓名： 许涵斌

指导教师（姓名、职称）： 张天 副教授

摘要:

随着模型驱动技术的逐渐成熟和广泛应用，大量反映程序结构、行为以及性质的模型产生于软件的开发过程中,并成为软件文档的重要部分保存下来。其中，尤以UML模型的应用最为广泛，也因此形成了通过理解UML模型来理解大规模、高复杂性软件系统的研究思路。

对UML模型理解的一个难点是如何有效地从大量复杂的模型中,快速查找并定位到具有一定结构特征的模型片段。幸运的是，设计模式的普遍应用为我们快速、高效地理解和定位模型提供了一条重要的线索。本文旨在通过分析和理解设计模式的结构特征，从而识别UML模型中的设计模式，以达到灵活和高效地理解软件系统的目的。

本文首先提出一种基于结构匹配的模型查询技术,然后利用UML模型自身语义特征对匹配算法做出一定的优化,最后使用这种基于UML的结构匹配的模型查询技术对当前主流的23中设计模式进行设计模式的识别。

本文的主要工作包括以下几个部分:

1、首先提出一种基于结构匹配的模型查询技术，包括这项技术中的UML结构信息抽取方法和模型匹配方法，并基于开源平台Eclipse开发相应模块;

2、通过逆向开源程序库的方式进行试验，从而分析UML模型自身的特点，然后根据分析结果对模型匹配算法做出改进,有效降低了UML模型查找过程的复杂度。同时，基于Eclipse平台进行了算法以及原型工具核心模块的实现;

3、使用基于结构匹配的模型查询技术对主流的23种设计模式进行设计模式的识别，针对其中具体有代表性和常用的几种模式进行实例研究和展示。

关键词：模型查询技术； 统一建模语言； 信息抽取； 设计模式;

南京大学本科生毕业论文（设计）英文摘要

THESIS: UML design pattern recognition method based on Structured Query

DEPARTMENT: Computer Science and Technology

SPECIALIZATION: Computer Science and Technology

UNDERGRADUATE: Hanbin XU

MENTOR: Associate Professor. Tian Zhang

ABSTRACT:

As model-driven techniques matured and widely used, more and more models, reflecting structures, behaviors and features of program, have been produced in the process of software development. And models also preserve as important parts of software documentations. Among them, UML models are the most widely used. Therefore, comprehension of UML models is thought as a good way to the comprehension of large-scale, highly complex software systems.

One of the difficulties to comprehension of UML models is how to find and locate effectively a certain structural feature of model fragments from a large number of complex models. Fortunately, the wide application of design patterns provides an important clue that how to understand and locate model quickly and efficiently. This paper aims to analyze and understand the structural features of design patterns in order to identify design patterns in UML models. In this way, we can achieve the purpose that we can understand software system flexibly and efficiently.

This paper introduces first a model query approach based on structure matching. Then, matching algorithm is improved more efficient by using the semantic features of UML model. In addition, the current 23 design patterns are analyzed and recognized based on the presented model query approach.

The main work includes the following sections:

1, Describe a model query approach based on structure matching, including the

UML structure information extraction techniques and models matching technology. And realize it based on the open source platform Eclipse;

2, Analyze the characteristics of UML models by reversing open source library. And then analyze the results to improve the model matching algorithm, in order to reduce complexity of finding the UML model. Realize algorithm and develop core modules on the Eclipse platform;

3, Use this model query approach based on structure matching to recognize the current mainstream 23 design patterns design. Select several design patterns which are representative and widely used to perform Case Study and Examples.

Keywords: model query technology; Unified Modeling Language; Information Extraction; design patterns;

目录

1. 引言

1.1 研究背景

人类社会在不断快速地发展着，总共经历了三次工业革命分别是以蒸汽机的广泛使用为代表的第一次工业革命，以电力的广泛使用为标志的第二次工业革命，还有以信息技术为标志的第三次工业革命，第三次工业革命后计算机软件在大量的领域得到了广泛地使用，它促使了生物学，化学，医学这些基础学科产生了新的分支学科如生物学中基因工程，医学中的药物图谱分析，化学中的模拟合成，计算化学等等，计算机软件技术还促进着现代通信技术的尤其是移动通信技术的爆发式发展，它还引导一些传统行业如印刷业向现代技术的过渡，如各种文字处理软件，排版印刷软件等。在计算机软件应用领域的不断扩大延伸的同时，现代软件的规模也不断地扩大，软件的复杂性也程指数级的提高。但是落后的软件的生产模式无法满足快速增长的计算机软件需求，从而导致软件开发与维护过程中出现一系列严重问题，这便是软件危机[1]。

为了应对软件危机，北大西洋公约组织在 1968 年联邦德国召开国际会议上提出了软件工程[2]的概念。软件工程是一门系统的研究软件在生产中的规律的学科，它的目标是降低软件的生产成本，提高软件的生产效率，改善软件的产品质量。人们通过软件工程学找到了许多解决软件危机的方法，基于设计模式识别的软件复用就是其中的一种。

1.2 研究意义

设计模式（Design pattern）是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结[3]。设计模式因为在实际生产中被反复地使用，验证了设计模式的可靠性，因而使用设计模式来控制软件产品的设计，可以保证软件产品质量的可靠性，同时由于各种主流的设计模式拥有大量的完备的软件产品，因而我们可以通过复用使用相同设计模式的软件产品，达到代码复用的目的，这样无疑可以降低软件生产的成本，提高软件的生产效率。同时设计模式在软件维护中的作用也不可小觑，我们可以通过设计模式了解到可以在软件系统的哪一

部分添加和扩展所需的功能;在对软件系统的功能进行添加和扩展之后我们还可以通过软件的设计模式对整个软件系统进行验证以检测对系统的修改是否引入了新的错误和缺陷。

要实现软件设计模式的复用和通过设计模式对软件的系统的可靠性进行验证首先必须对软件设计模式进行识别,但是设计模式保存到文本文件将生成大量的冗余的结构信息,而且一般对设计模式的识别是在模型成面上识的别,即对软件的 UML 模型直接识别以找出设计模式,由于现代的软件规模十分庞大,复杂性也很高,十分的难以识别。如果使用人工对设计模式进行识别,那么花费的时间成本和人工成本将十分的高昂,而且准确性和可靠性不能得到保证。本文介绍了一种基于结构匹配的模型查询技术,使用该技术对主流的 23 种设计模式[4]进行设计模式的识别可以节省大量的人工和时间,有利于降低软件的生产成本,提高软件的生产效率,同时对于增强软件的可维护性也将有不小的帮助。

1.3 本文的工作

在各种软件设计模式日益成熟的今天,通过复用已经成熟的设计模式可以大大提高软件开发的效率。本文将首先介绍一种基于结构匹配的模型查询技术,该技术包括 UML 模型文件的信息抽取算法和 UML 模型匹配算法两部分,本文还将对 UML 模型匹配算法进行改进,以提高匹配算法的匹配效率。本文还将对设计模式进行分析,并针对其中一些使用广泛,具有代表性的设计模型进行深入地分析和阐述,并为这些具有代表性的设计模式建立模型结构和实例模型,同时本文将使用一个开源项目对软件设计模式识别的功能进行验证。最后本文将使用基于结构匹配的模型查询技术对设计模式进行识别方法的阐述,并对其中的一些设计模式进行实例识别的展示。

1.4 论文的结构

本文一共分为七个部分。

第一部分引言介绍了本文的研究背景，研究意义，研究的主要内容以及论文的整体结构。

第二部分介绍了相关的技术背景。

第三部分介绍了如何对一个由 UML 模型保存得到的文本文件进行结构信息的抽取，以及如何对 UML 模型进行深度优先的模型匹配。

第四部分介绍了针对模型匹配算法的一些缺陷进行的改进。

第五部分着重对设计模型进行分析，并选取其中一些有代表性的，应用广泛的设计模型进行深入地阐述和分析。并介绍了如何使用这种基于结构匹配的模型查询技术对设计模式进行识别进行。

第六部分通过一个实例对本文使用的设计模式识别方法进行展示。

第七部分将本文与其他相关工作做了比较并对本文进行了总结展望。

2.技术背景

2.1 Unified Modeling Language——统一建模语言

统一建模语言（Unified Modeling Language，UML）[5][7]即我们通常所说的UML语言,统一建模语言统一了Jacobson、Rumbaugh以及Booch所提出的OOSE、OMT和Booch方法，并在这三种方法的基础上作出了进一步地发展，最终形成了一种为大众广泛使用标准建模语言。

UML语言是一种能够支持模型化和图形化的语言，它能够为软件生产的各个阶段提供模块化和可视化地建模支持。作为一种建模语言，统一建模语言的定义包括两部分： UML语义（UML Semantics）和UML表示法（UML Notation）

1、UML语义（UML Semantics）

UML语义（UML Semantics）[16]的模型比较复杂，它采用元-元模型（meta-metamodel),元模型(metamodel),模型(model),用户对象(user object)这样的四级元模型体系结构。其中元-元模型（meta-metamodel）是元模型的基础架构，定义了元模型的一种描述语言。元模型（metamodel）则是一个元-元模型的具体的例子，定义了模型的一种描述语言.模型（model）则是一个元模型的具体的例子，定义了信息领域的一种描述语言.用户对象（user object)：则是一个元模型的具体的例子，定义了实际使用的某一个信息领域。

2、UML表示法（UML Notation）

UML表示法（UML Notation）[5]定义了一系列的文本语法和图形，通过这些文本语法和图形，为使用UML进行建模提供了一个切实可行的标准。这些图形和文本都属于UML语义四级模型中的模型（model）层次，是元模型的一系列实例。

以下是UML构造块的结构图示：

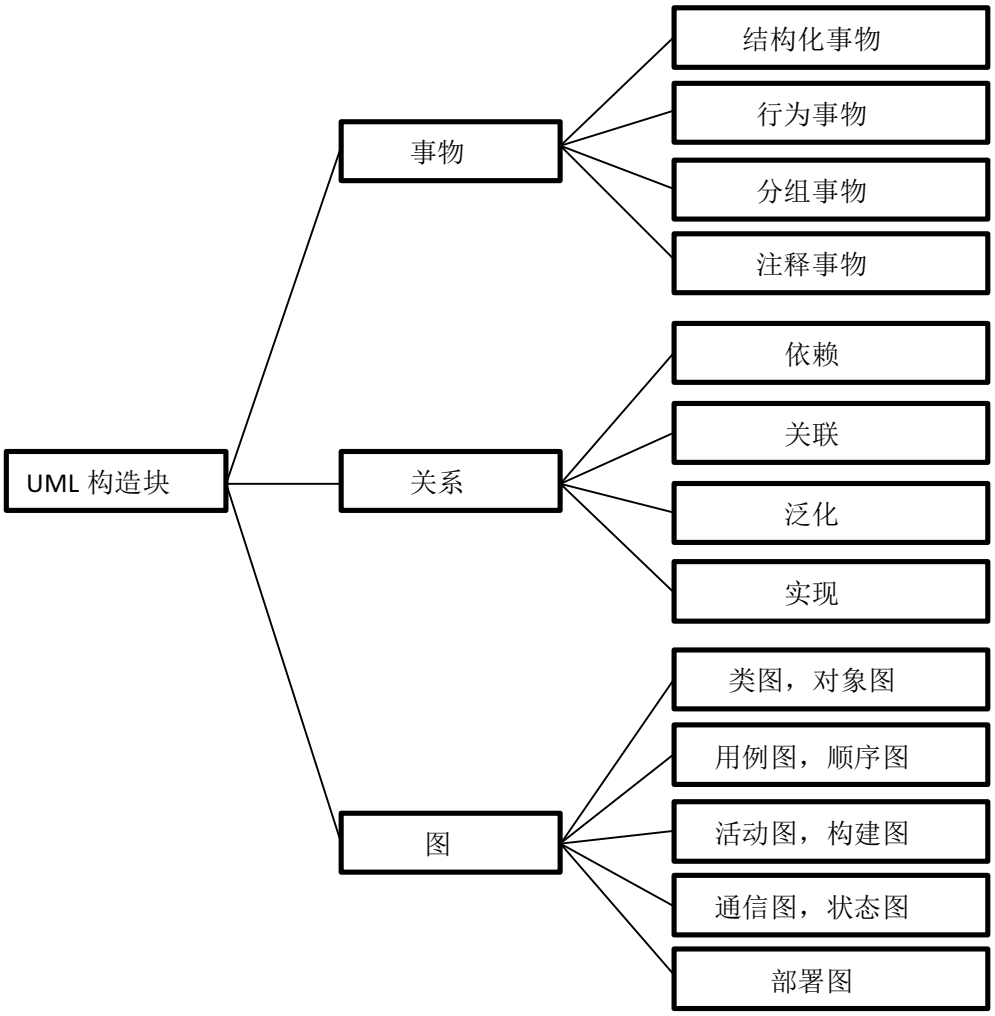


图 2-1 UML 构造块的结构图

统一建模语言中最重要的部分是以下几类图：

(1) 静态图(Static Diagram)，静态图中有类图，包图，对象图这三类。其中类图用来定义类的结构，包括类元的结构定义和类间的关系定义。

类元的结构定义主要有三部分：名称（Name），属性（Attribute），方法（Operation）。名称(Name)是一个字符串用于描述类元的名称,属性(Attribute)描述了实体的一些特征值包括特征值的类型，类型的名称和数值得范围；方法（Operation）描述对象的一些操作，主要包括返回值类型，方法名，参数名，参数的类型和参数的默认值。其图形结构如下

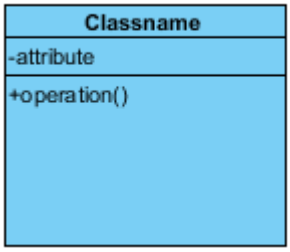


图 2-2 类元结构图

类间关系主要有四类：依赖关系（Dependency），关联关系（Association），实现关系（Realization）和泛化关系（Generalization）其描述作用如下：

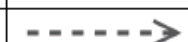
关系	描述	符号
依赖（Dependency）	两个模型元素之间的关系	
关联（Association）	描述两个类实例之间连接关系	
泛化（Generalization）	描述特殊/一般的关系， 用于模型元素之间的继承关系	
实现（Realization）	描述接口标准 与其具体实现之间的关系	

表2-1 类间关系表[5]

对象图是类图的一个具体实例，对象图可以有多个对象的实例。包图用来描述包与包之间的关系，系统的层次结构通常要由包图来描述。

(2) 用例图（Use Case Diagram），用例图（Use Case Diagram）是站在用户的角度来分析整个系统的各个功能，并指出不同的用户所能够执行的不同的系统操作，常用于需求分析阶段。

(3) 交互图（Interactive Diagram），交互图（Interactive Diagram）主要用于分析系统对象之间的交互关系，交互图主要有两种：通信图和顺序图。通信图强调不同系统对象之间的上下级的关系，说明的是系统对象之间的动态协作以及通信的关系。顺序图用于强调顺序和时间，说明的是消息在系统对象之间的发送顺序。

(4) 行为图（Behavior Diagram），行为图（Behavior Diagram）是用来描述组成对象和动态模型之间的交互关系。行为图包括状态图和活动图，状态图用于描述类的对象的所有可能出现状态，并在遇到不同的外界条件时状态是如何转

移的。活动图则用来描述满足要求的活动以及这些满足要求的活动间的约束条件，常在进行活动的并行分析时使用。

(5) 实现图(Implementation Diagram)，实现图(Implementation Diagram)主要包括构件图和部署图，其中构件图用来描述代码部件的物理结构和各个代码部件之间的依赖关系。部署图用来描述系统中软间和硬件之间的物理结构。

在这些 UML 表示法中由于类图可以比较直观的体现系统的逻辑结构因而被广泛运用于设计模式的建模表示。因而基于结构查询的 UML 设计模式识别可以运用设计模式的静态图作为识别的单元。

2.2 XML 文件解析工具

XML(eXtensible Markup Language)即可扩展标签语言[6][15]，是一种为了克服 HTML 语言的一些缺陷而提出的结构化描述语言，XML 作为 HTML 的一种补充在 internet 上得到了非常广泛地使用，Microsoft, Oracle，IBM 等国际著名的软件提供商都在各自的产品中对 XML 进行了支持，XML 语言在 1998 年被 World Wide Web Consortium 组织所认可，有了一个系统化的国际标准。

现在对于 XML 文件的解析工具大致有两种，一种是以 DOM [10]（Document Object Model）为基础的 XML 文件解析工具[9]，另一种是以 SAX（Simple API for XML）[11]为基础的 XML 文件解析工具[8]。

以 DOM（Document Object Model）为基础的 XML 文件解析工具是一种基于对象的 API，它将 XML 文件整个读入内存中，然后将文档表示成树状结构，将 XML 文档一次性的解析成一棵对象树，这种解析 XML 文件的方式对内存占用大，对 XML 文件较小的情况使用。

以 SAX（Simple API for XML）为基础的 XML 文件解析工具是一种基于事件的 API,它并不在内存中生成文档树，而是在 XML 文档遍历的过程中采用一边对文档进行深度优先地遍历，一边在遍历的过程中对遇到每一个文档元素进行解析的方式进行文档处理对系统内存的占用较小比较适合处理大型的 XML 文件。

2.3 Depth-First Search——深度优先搜索

图的深度优先遍历[13]递归定义描述如下：

首先假设图的所有节点均未被访问，那么可以从图中任取一点 v 开始如下操作:首先访问所取的原点 v，将该点 v 设为已经访问，然后寻找该点 v 的所有未访问节点取出其中的一点 w，重复点 v 的操作即以 w 为新的源点进行图的深度优先遍历，直到所取的点已经没有相邻的点未被访问为止，如果此时在图中还有没有被访问的点说明该图有多个连通分支重新选取一个未访问的点作为源点，进行深度优先的遍历。

其深度优先遍历算法[12]的伪代码描述如下：

```
 DFS(G)
1    for each vertex u∈V[G]
2        do color[u]←White
3         π[u]←NIL
4          time←0;
5    for each vertex u∈V[G]
6      do if color[u]==White
7           then DFS_Visit(u);
```

```
DFS_Visit(u);
1   color[u]←Gray;          Δwhite vertex u has just been discovered
2   time←time+1;
3   d[u]←time;
4   for each vertex v∈Adj[u]     Δexplore (u,v)
5     do if color[v]==White
6          then  π[v]←u;
7               DFS_Visit(v);
8   color[u]←Black;             Δblacken u;it is finished
9   f[u]←time←time+1;
```

3.模型查询技术

基于结构匹配的模型查询技术可分为两部分，第一部分为UML模型文件的结构信息抽取算法；第二部分为UML模型文件的匹配算法。本章将详细介绍这种基于结构匹配的模型查询技术的运行原理和运行流程。

3.1 UML模型文件的信息抽取

本节将介绍基于结构匹配的模型查询技术的第一部分：UML模型文件的结构信息抽取算法。由于UML模型保存得到的XMI格式文本文件满足OMG所制定的文档类型定义(Document Type Definition, DTD)[14]，XMI格式文本文件具有极强的结构特征，在遍历XMI格式文本文件过程中可以根据XMI的结构特征规律进行UML模型元素的信息抽取，为下面的模型元素匹配做准备。

3.1.1 模型元素结构抽取思想

由于模型文件的格式为XMI文件因而可以借助XML解析工具遍历整个XMI文本文件，本文主要使用以SAX（Simple API for XML）为基础的XML文件解析工具，这种XML文件解析工具是一种基于事件的解析工具。

首先定义的是对整个XML文件处理的方法和对XML文件中的节点的处理方法的接口,接口中需要定义以下方法：

startDocument()：在开始访问XMI文件时调用，用于进行遍历文档开始时的一系列操作。

endDocument()：在结束访问XMI文件时调用，用于进行遍历文档结束后的一系列操作。

characters ()：用于保存XML元素节点的信息的方法。

startElement()：访问每个XML元素节点前均调用此方法，根据所需抽取的信息进行不同地实现

endElement()：访问每个元素节点后均调用此方法，根据所需抽取的信息进行不同地实现

然后是使用上面定义的对XMI文件的文档处理的方法和元素节点的处理方

法对文档进行模型元素节点的信息记录和抽取。本文使用深度优先的算法进行遍历，在遍历的过程中对节点进行结构信息的抽取与储存。算法的伪代码描述如下：

```
VISITELEMENT (element)
1  for  every child of element
2    do startElement();
3       VISITELEMENT(child);              Δ访问子节点
4       endElement();
```

遍历算法的图示：

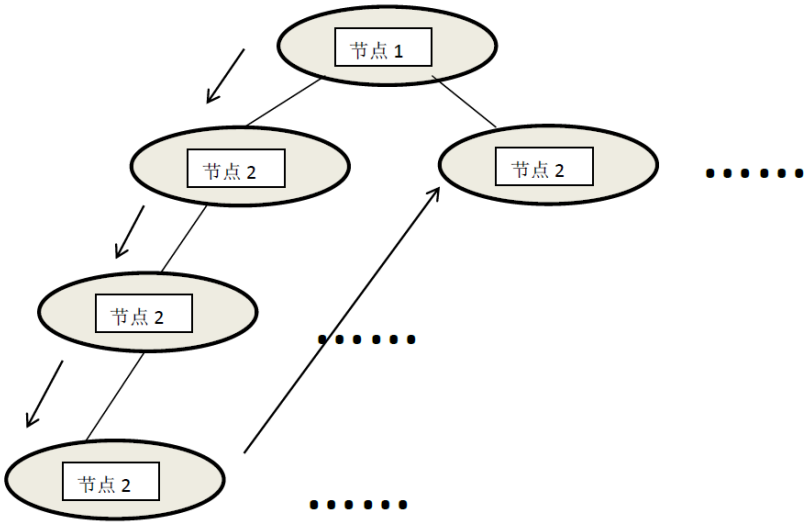


图 3-1 信息抽取算法遍历图示

最后将以上的算法综合可以得到一个文件信息抽取算法，其伪代码描述如下：

```
ExtractInformation (file)
1  root←startDocument(file);
2  VISITELEMENT(root);          Δ从根节点开始访问
3  endDocument(file);
```

3.1.2 UML 结构信息的抽取

为了实现不同的UML 建模工具生成的XMI 能够相互兼容使用，OMG 特别为UML 语言生成的模型文件制定了文档类型定义(Document Type Definition,DTD)[14]，使用 DTD 后各个 UML 建模工具生成的 XMI 文本文件具有一致的文本结构，为我们抽取 UML 模型的结构信息提供了条件。

本文以 SAX 为主要的 UML 解析工具，其解析的模型如下：

SAX 模型：

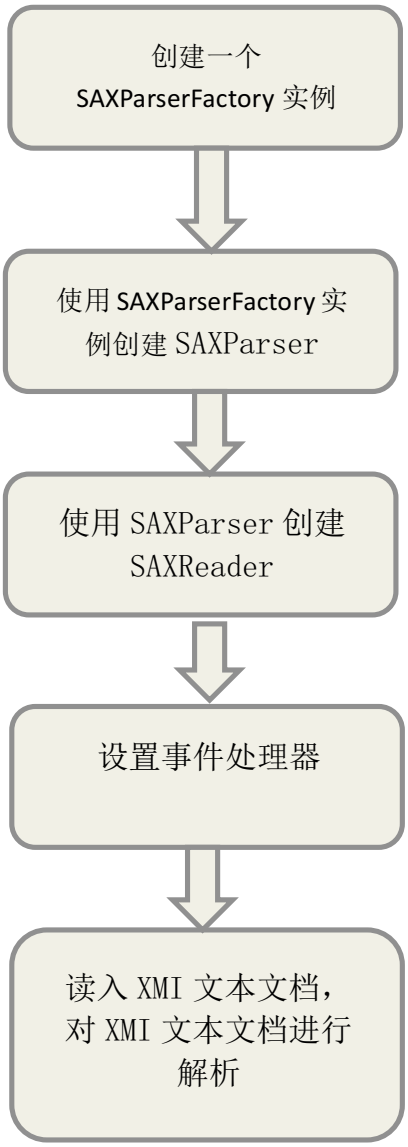


图 3-2 SAX 解析模型流程图

我们在对 XMI 文件进行解析的过程中，想要处理不同类型的模型元素，只需要编写不同的事件处理函数即可。

本节主要以类、属性、方法、实现关系、继承关系和依赖关系等几种典型的模型元素为例，介绍了使用事件处理程序对 UML 模型元素信息进行抽取的过程。

首先对模型元素进行分类，大致可以分成两类：一类为类元模型元素[16]；一类是类间的关系模型元素[16]。

3.1.2.1 类元模型元素的事件处理程序

要编写类元模型元素的事件处理程序首先必须分析类元的文本结构特征根据文本结构的特征来获取节点信息，实现类元的结构信息抽取。

类元是类，接口和数据类型的超类，类元模型元素主要包含了类名，类的修饰符，类的属性和类的操作这四部分，类，接口和数据类型具有相似的语法，在 UML 的文本文件中的结构大致相同，因而可以使用相同的算法对其进行信息的抽取，其 UML 中的类，接口，数据类型的图形如下：

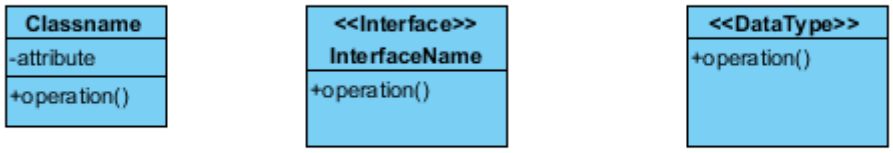


图 3-3 类元的类图显示

类在 UML 文件中的结构如下：

```
UML:Class
     UML:Classifier.feature
          UML:Attribute
          UML:Operation
```

在 UML 的文本文件中类 Class 的特征通常如下：

```
<UML:Class xmi.id = '-84-26-16--102--58cd4864:13e1b82a8fb:-8000:0000000000000866'
  name = 'Machine' visibility = 'public' isSpecification = 'false' isRoot = 'false'
  isLeaf = 'false' isAbstract = 'false' isActive = 'false'>
```

图 3-4 类的文本结构

```
<UML:Attribute xmi.id = '-84-26-16--102--58cd4864:13e1b82a8fb:-8000:0000000000000867'
  name = 'kind' visibility = 'private' isSpecification = 'false' ownerScope = 'instance'
  changeability = 'changeable' targetScope = 'instance'>
```

图 3-5 类的属性的文本结构

```
<UML:Operation xmi.id = '-84-26-16--102--58cd4864:13e1b82a8fb:-8000:000000000000086A'
  name = 'perform' visibility = 'public' isSpecification = 'false' ownerScope = 'instance'
  isQuery = 'false' concurrency = 'sequential' isRoot = 'false' isLeaf = 'false'
  isAbstract = 'false'>
```

图 3-6 类的方法的文本结构

接口的结构特征与类的结构特征相似，只是关键词由 UML：Class 改为 UML:Interface，在此不再赘述。

类元的事件处理程序的实现以接口为例详见附录 E。

3.1.2.2 类间的关系模型元素的事件处理程序

类间的关系用于描述两个类的之间的关系，主要的类的关系有Association 关联关系，Abstraction 实现关系，Dependency 依赖关系，Generalization 继承关系[16]。多个类间关系可以用来描述整个类图的结构，而且其在 UML 文本文件中的结构比较简单，主要的特征值有关系的 ID，关系的名称，关系连接的两个类的 ID 等主要信息，在信息抽取的过程中较为容易实现。要编写类间的关系模型元素的事件处理程序首先必须分析类间关系的文本结构特征根据文本结构的特征来获取节点信息。

Abstraction 实现关系的 UML 文本文件结构由以下几个主要的部分组成：UML:Dependency.client 为实现接口的类， UML:Dependency.supplier 为被实现的接口，xmi.id 是 Abstraction 实现关系的 id，xmi.idref 是实现的接口类和被实现接口类的 id。

```
<UML:Abstraction xmi.id = '-84-26-16--102--29efeb7f:13e3aa46f47:-8000:0000000000000A76'
  name = 'A1' isSpecification = 'false'>
  <UML:ModelElement.stereotype>
    <UML:Stereotype xmi.idref = '-84-26-16--102--29efeb7f:13e3aa46f47:-8000:0000000000000A77'/>
  </UML:ModelElement.stereotype>
  <UML:Dependency.client>
    <UML:Class xmi.idref = '-84-26-16--102--29efeb7f:13e3aa46f47:-8000:0000000000000A75'/>
  </UML:Dependency.client>
  <UML:Dependency.supplier>
    <UML:Interface xmi.idref = '-84-26-16--102--29efeb7f:13e3aa46f47:-8000:0000000000000A72'/>
  </UML:Dependency.supplier>
</UML:Abstraction>
```

图 3-7 实现关系的文本结构

Dependency 依赖关系的 UML 文本文件结构由以下几个主要的部分组成：UML:Dependency.client 描述的是依赖的类，UML:Dependency.supplier 描述的是被依赖的类，xmi.id 是 Dependency 依赖关系的 id，xmi.idref 是依赖类和被依赖类的 id。

```
    <UML:Dependency xmi.idref = '-84-26-16--102--11b5cbbc:13e3b18a403:-8000:0000000000000A8C'/>
  </UML:ModelElement.clientDependency>
  <UML:Namespace.ownedElement>
    <UML:Dependency xmi.id = '-84-26-16--102--11b5cbbc:13e3b18a403:-8000:0000000000000A8C'
      name = 'D1' isSpecification = 'false'>
      <UML:Dependency.client>
        <UML:Class xmi.idref = '-84-26-16--102--11b5cbbc:13e3b18a403:-8000:0000000000000A86'/>
      </UML:Dependency.client>
      <UML:Dependency.supplier>
        <UML:Class xmi.idref = '-84-26-16--102--58cd4864:13e1b82a8fb:-8000:0000000000000866'/>
      </UML:Dependency.supplier>
    </UML:Dependency>
```

图 3-8 依赖关系的文本结构

Generalization 继承关系的 UML 文本文件结构由以下几个主要的部分组成：UML:Generalization.child 为子类或子接口，UML:Generalization.parent 为父类或者父接口，xmi.id 是 Generalization 继承关系的 id，xmi.idref 是类和接口的 id。

```
<UML:Generalization xmi.id = '-84-26-16--102--29efeb7f:13e3aa46f47:-8000:0000000000000A86'
  isSpecification = 'false'>
  <UML:Generalization.child>
    <UML:Class xmi.idref = '-84-26-16--102--29efeb7f:13e3aa46f47:-8000:0000000000000A85'/>
  </UML:Generalization.child>
  <UML:Generalization.parent>
    <UML:Class xmi.idref = '-84-26-16--102--58cd4864:13e1b82a8fb:-8000:0000000000000866'/>
  </UML:Generalization.parent>
</UML:Generalization>
```

图 3-9 继承关系的文本结构

Association 关联关系的 UML 文本文件结构由以下几个主要的部分组成：其中 UML:AssociationEnd 是关联关系中的两个端，其中的属性：aggregation 表示聚集，isNavigable 属性是指是否可导航。UML:AssociationEnd.participant 描述的是两端的类元元素的 ID。

```
<UML:Association xmi.id = '127-0-0-1--41eca54c:13e58e5de9b:-8000:0000000000000A8D'
  name = 'ASS1' isSpecification = 'false' isRoot = 'false' isLeaf = 'false'
  isAbstract = 'false'>
  <UML:Association.connection>
    <UML:AssociationEnd xmi.id = '127-0-0-1--41eca54c:13e58e5de9b:-8000:0000000000000A8E'
      visibility = 'public' isSpecification = 'false' isNavigable = 'true' ordering = 'unordered'
      aggregation = 'none' targetScope = 'instance' changeability = 'changeable'>
      <UML:AssociationEnd.participant>
        <UML:Class xmi.idref = '-84-26-16--102--58cd4864:13e1b82a8fb:-8000:0000000000000866'/>
      </UML:AssociationEnd.participant>
    </UML:AssociationEnd>
    <UML:AssociationEnd xmi.id = '127-0-0-1--41eca54c:13e58e5de9b:-8000:0000000000000A8F'
      visibility = 'public' isSpecification = 'false' isNavigable = 'true' ordering = 'unordered'
      aggregation = 'none' targetScope = 'instance' changeability = 'changeable'>
      <UML:AssociationEnd.participant>
        <UML:Interface xmi.idref = '-84-26-16--102--29efeb7f:13e3aa46f47:-8000:0000000000000A72'/>
      </UML:AssociationEnd.participant>
    </UML:AssociationEnd>
  </UML:Association.connection>
</UML:Association>
```

图 3-10 关联关系的文本结构

以下以依赖关系为例说明类间关系的信息抽取算法的代码实现，详情见附录 F。

3.2 UML 模型匹配

为新设计的模型在模型库中寻找已经存在的模型做支撑，实现代码复用，软件维护,是设计模式识别的主要作用之一。为了在模型库中找到与新设计的模型相匹配的设计模式我们设计如下算法来处理 UML 文件，以实现 UML 模型文件的匹

配。

首先对抽取出来的元素信息节点进行遍历以得到一个深度优先的队列，然后以这个队列的顺序为序进行模型元素的匹配，直到找到匹配的设计模式或已经遍历所有元素仍未找到相匹配的模式结束匹配。

整个 UML 模型的匹配算法可以分为四部分：

1、输入模型即输入设计模式模型的预处理，通过对输入模型的解析得到一个匹配序列，按照匹配序列进行模型元素在模型库中的匹配。

2、类元的匹配，按照类的基本信息和类所包含的属性以及方法进行匹配。

3、类间关系的匹配，按照类间关系的基本信息以及关系两端的类元进行匹配。

4、按照深度优先遍历算法对输入模型进行深度匹配

3.2.1 输入模型的预处理

对输入模型预处理需要对新设计的 UML 模型的进行遍历，以获得一个包含所有元素的匹配序列。本文使用深度优先的遍历[13]，使用一个栈来生成遍历的序列，算法描述如下：

1、获取一个类元元素进栈。

2、模型元素不为空或模型为空栈不为空则获取栈顶元素，若为不满足则跳转到 4。

3、将栈顶元素出栈放入遍历序列，然后在模型中寻找所有与出栈元素关联的元素进栈，同时将关联的元素从模型中移除，若没有与之关联的模型元素则是叶节点，直接跳转 2。

4、得到一个深度优先遍历序列

算法伪代码如下：

```
GetsortedQueryPatternList (queryPattern)
1    Push (firstClass);                Δ将第一个类元元素进栈
2    while  queryPattern or stack is not empty
3         do element←Pop();
4            list[i++]←element;        Δ将栈顶元素加入序列
5            if  element is Classifier
6              then push all relationship connected element into stack;
7            else if  element is Dependency
8                then push all classifier connected element into stack;
9            else if  element is Abstraction
10               then push all classifier connected element into stack;
11           else if  element is Generalization
12               then push all classifier connected element into stack;
13           else if  element is Association
14               then push all classifier connected element into stack;
```

3.2.2 类元的匹配

类元的匹配比较简单，首先对类元的一些特征进行匹配主要有 xmi.id ，name ,visibility ,isSpecification ,isRoot,isLeaf,isAbstract,isActive。其中 xmi.id ， name 可以使用字符串比较的方式进行匹配，visibility 是枚举类型，isSpecification ， isRoot，isLeaf，isAbstract，isActive 是 boolean 类型都可以直接比较，类元的属性 Attribute 的特征有 xmi.id， name，visibility，isSpecification，ownerScope，changeability ， targetScope，

类元的方法的特征有 xmi.id ， name ，visibility ，isSpecification ，ownerScope ， isQuery ， concurrency ， isRoot ， isLeaf ， isAbstract，属性和方法除了 xmi.id ， name 需要调用字符串匹配的方式进行匹配，其他都可以直接进行比较。

类元匹配的伪代码如下

```
MatchClass (userclass, patternclass)
1  If Property values between userclass and patternclass is not match
2     then  return false;                Δ类元的属性值的匹配
3  if !MatchAttribute (userclass, patternclass) || !MatchOperation
4           (userclass, patternclass)
5     then  return false;   Δ类元的 Attribute 和 Operation 的匹配
6  return true
```

类元的 Attribute 的匹配算法的伪代码描述如下：

```
MatchAttribute (userclass, patternclass)
1  for each  Attribute in userclass
2      do  Attributematch←false;
3          for each Attribute in patternclass
4              do if   Attribute is match
5                    then  Attributematch←true;
6                          break;
7          if Attributematch ==false
8             then  return false;
9  return true
```

类元的Operation的匹配算法的伪代码与类元的Attribute的匹配算法类似在此不再赘述。

3.2.3 类间关系的匹配

由于类间的关系的文本文件的结构比较简单，因而在匹配过程中只要比较类间关系的两端元素是否匹配就可以判断此类间关系是否与模型库中的类间关系相匹配，由于使用了深度优先的遍历序列，而且第一个元素必定为类元，因而再比较类间关系时只有一端元素已经匹配和两端元素都已经匹配两种情况，这两种情况都可以认为类间关系已经匹配，若另一端不匹配则另一端会回退到上一个节点重新寻找匹配的类。

由于类间关系的结构和语法相似，以下以依赖关系为例说明算法的伪代码实现：

```
Dependencymatch( element)
1   if  two sides of element have been visited
2      then  if element exists in modelbase
3               match(matchResult, matchedIndex+1);
4            else
5               return;
6   else if only one side of element has been visited
7      then  if element exists in modelbase
8               then  add mapping between two IDs;
9                     match(matchResult,matchedIndex+1);
10                    remove mapping between two IDs;
11  else
12       return;     Δ没有找到匹配的关系
```

详细代码实现见附录 G。

3.2.4 整个模型的匹配

本文使用回溯法对整个模型进行匹配具体算法描述如下：

根据之前生成的匹配序列，按照序列进行模型元素的匹配，匹配方法使用 2 和 3 中的类元匹配方法和类间关系匹配方法，若在模型元素在模型库中找到匹配元素，则建立匹配元素之间的 ID 映射，同时深度加 1，进行下一个模型元素的匹配，直到深度等于新设计模型的元素个数表明已完成匹配，输出匹配的节点 ID 映射；若在模型元素在模型库中找不到匹配元素则回退到上一层，同时将之前建立的节点映射删除，若回退到第一层且遍历完第一层所有找到的匹配元素仍未找到匹配的模型映射则匹配失败。

算法的伪代码描述如下：

```
match(matchResult, matchedIndex)
 1   if matchedIndex equals modelsize
 2      then   print matchResult; Δ输出节点映射关系
 3             return;
 4   for each element in model
 5       do if element is Classifier
 6             then if element has not been visited
 7                     then for each matched element in model
 8                        do add mapping between two IDs;
 9                           match(matchResult,matchedIndex+1);
10                           remove mapping between two IDs;
11                  else for each matched element in model
12                         do match(matchResult, matchedIndex+1);
13                           remove mapping between two IDs;
14          else if element is Abstraction Δ调用实现关系匹配函数
15                  then  Abstractionmatch(element);
16          else if element is Dependency  Δ调用依赖关系匹配函数
17                  then  Dependencymatch(element);
18        else if element is Generalization Δ调用继承关系匹配函数
19                 then   Generalizationmatch(element);
20          else if element is Association  Δ调用关联关系匹配函数
21                  then  Associationmatch (element);
```

详细代码实现见附录 G。

3.3 本章总结

本章详细介绍了基于结构匹配的模型查询技术的运行原理和运行流程。其中

主要介绍了 UML 模型文件的结构信息抽取算法，该算法是深度优先的算法，该算法可以抽取模型文件中的类元信息和类间关系信息，并将这些信息以信息节点的方式储存在内存中以用于之后的匹配。此外，还主要介绍了 UML 模型文件的匹配算法，该算法包括类元匹配，关系匹配和整个模型的匹配三部分，详细地阐述了如何在模型库中寻找与用户输入模型相匹配的模型结构。

4. 模型的匹配算法改进

本章将针对基于结构查询的模型匹配算法中的一些问题和不足之处进行改进，以提高算法的匹配效率，减少程序运行的时间。具体通过逆向开源程序库的方式进行试验，从而分析 UML 模型自身的特点，然后根据分析结果对模型匹配算法做出改进，有效降低了 UML 模型查找过程的复杂度。

4.1 匹配算法分析

基于结构匹配的模型查询技术中的匹配算法实际上是子图同构问题（Subgraph isomorphism problem）[24]，子图同构问题（Subgraph isomorphism problem）是 NP 完全性问题[25]，其问题具有相当高的问题复杂度，为了便于我们匹配算法的分析，我们需要假定一些条件来简化这个 NP 完全性问题才能够对该匹配算法进行复杂度分析。

我们假定要匹配的模型是一条具有 k 个节点的路径，同时假设模型库与该路径中的节点类型相同的节点共有 n 个。

该匹配算法使用深度优先的算法，在匹配算法的运行过程中，由于路径的节点和边的选取是随机的，在模型库中选取与路径的节点和边相匹配的节点和边同样也是随机的，因而我们考虑匹配的最坏情况，即在路径每个节点和边的匹配过程中，每个节点都要经过 n 次查询比较才能够找到完全匹配的节点，因而第一个节点总计要搜索比较 n 次，第二个节点要总计比较n^2次，第三个节点要总计比较n^3次，第 k 个节点要总计比较n^k次，由此最坏条件下我们要搜索比较$\sum_1^k n^i$次才能找到匹配的模型子图，时间复杂度达到 O（n^k），这种选取匹配节点完全随机的匹配算法的时间复杂度无疑是很高的。

4.2 UML 模型元素分析

通过上一节的分析我们知道单纯的数学上的子图匹配算法的时间复杂度很高，达到了 O（n^k）。但是本文讨论的并不是单纯的数学问题，而是针对 UML 图这一特定领域进行的模型匹配。从 UML 图中抽取出来的节点包含有大量的 UML

的结构信息，通过对 UML 语义进行研究我们可以找到 UML 图中模型元素的一些规律，并根据这些规律我们可以对匹配算法做定向的优化。

为了找出 UML 图中模型元素的一些规律，我们对一些开源软件工程进行了模型元素和元素的各个属性的属性值进行了统计，其中用 Java 编写的 BitTorrent 客户端 VUZE 共拥有 9MB 的源代码，代码量较大，得出的结果比较的具有代表性。其模型元素和模型元素的各个属性值分布表见附录 A。

通过对模型元素和模型元素的各个属性的属性值进行统计我们可以发现，在 UML 模型中各种不同的模型元素，模型元素的各个属性的属性值出现的频率是不同的，最典型的例子就是类和接口的 name 这一属性值。例如对类和接口中的属性 isAbstract 这一枚举类型的值，不同的值得出现频率差别很大，在统计的 3271 个类元中只有 50 个类元的 isAbstract 属性为 TRUE，只占总数的 1.5%。

4.3 基于 UML 的匹配算法改进

在进行模型匹配的过程实际上是在模型库中排除子图的过程，比如说在寻找匹配第一个节点时有 m 个匹配的节点，由此产生了拥有 m 张相符的子图的集合，当选取与第一个节点相连的边的匹配过程中由于只有部分的边相符因而减少了 s 张符合的子图，子图的集合因此缩小了，如此进行下去直到找到匹配子图为止。如果一开始我们就选取比较稀有的节点进行匹配，那么一开始所产生的匹配的子图集合的规模就小了许多，在之后进行匹配的过程中还可以不断优先选取稀有的节点和边进行匹配，这可以不断的缩小子图集合的匹配规模，直到找到匹配的子图为止。因而输入队列对于匹配结果的影响很大。

通过上一节的分析，我们可以知道受到 UML 语义的影响，UML 模型元素和模型元素的各个属性的属性值出现的频率有很大的差别，这为我们根据 UML 模型元素和模型元素的各个属性的属性值出现规律进行匹配算法的定向优化改进提供了可能。

针对匹配算法的特征和 UML 模型元素的分布情况我们可以采用改变预处理后的输入序列排序的方法减少匹配算法的搜索匹配次数。本文通过对 UML 模型元素及其属性值进行优先级划分，并在对 UML 模型元素节点进行预处理时采取优先级低的先进栈优先级高的后进栈的策略，使得产生的预处理输入序列优先级高的

在序列的前端，优先级低的在序列后端，使得在递归搜索匹配时优先级高的被先调用访问进行匹配搜索，如果存在匹配就可以大大的减少递归搜索探查的次数。

以下是对 UML 模型元素进行优先级划分

优先级	模型元素
1	类间关系： 绑定依赖，授权依赖，抽象依赖，实现依赖，使用依赖，组合关系，聚合关系，关联关系，继承关系
2	类元：接口，类
3	属性
4	操作

表 4-1 UML 模型元素的优先级

以下是对 UML 模型元素的属性值进行优先级划分

优先级	模型元素属性
1	name ， initialvalue
2	Operation.isAbstract=True Operation.isQuery=True，Operation.Concurrency=guarded， Class.isAbstract=True
3	Operation.visibility=private ， Operation.visibility=package ， Class.visibility=private ， Class.visibility=protected ， Attribute.visibility=protected ， Attribute.visibility=package
4	Operation.ownerScope=classifier， Attribute.ownerScope=classifier
5	Operation.visibility=protected ； Operation.visibility=public ，

	Attribute.visibility=public ， Attribute.visibility=private， Class.visibility=public， Class.visibility=package

表 4-2 UML 模型元素属性值的优先级

详细的序列生成算法见附录 C 输入队列预处理算法代码实现部分。

4.4 改进匹配算法分析

改进的匹配算法同原匹配算法一样都是 np 完全性问题，因此同分析原匹配算法时一样，接下来我们需要假定一些条件来简化 np 完全性问题，以实现对改进匹配算法的时间复杂度分析。

我们假定要匹配的模型是一条具有 k 个节点的路径，定义稀有度为该节点在模型库中可以找到的匹配节点数，k 个节点的稀有度分别为x_i，假设模型库与该路径中的节点类型相同的节点共有 n 个，$x_i \leq n$。

首先我们将路径节点按照稀有度来排序遍历，可以得到一个按节点稀有度递增的遍历序列，序列中第 i 个节点的稀有度为y_i。因而在最差情况下，如果路径在模型库中存在，那么寻找第一个节点要比较 n 次，第二个节点要$\mathrm{n} * y_1$次，第 s 个节点要第二个节点要$\mathrm{n} * \prod_1^{s-1} y_i$ 次，遍历过程中比较次数为$\mathrm{n} * (1 + \sum_1^{k-1} \prod_1^{j} y_i)$。如果使用未改进的匹配算法进行匹配，那么遍历过程中查找比较次数为$\mathrm{n} * (1 + \sum_1^{k-1} \prod_1^{j} x_i)$，由于$\{y_i\}$序列是$\{x_i\}$序列的递增排序因此$\mathrm{y}_1 * y_2 * \ldots * y_s \leq \mathrm{x}_1 * x_2 * \ldots * x_s$总是成立的，因而路径上每一个节点的搜索次数都比为改进前要少，因此搜索匹配的效率比改进前高。改进算法本质上是一种贪心算法[26].

在实际的运行环境中在对 4 个节点的路径进行匹配，由 100 个类和接口，140 个类间关系组成的数据库中找到匹配模型。未使用改进算法的匹配过程耗时 8221 毫秒，使用改进算法后的匹配过程耗时 743 毫秒，比原来的算法快了近 10 倍。

4.5 本章总结

本章首先详细分析了原有算法的时间复杂度，指出了原有算法时间复杂度高，进行对模型库中的模型匹配效率低下的问题。然后我们对 UML 模型中的模型元素以及其属性值进行了统计分析。接着本章提出了一种利用 UML 类图中的元素和元素属性出现频率不同的性质进行优先级划分，并使用这种优先级对输入队列进行排序遍历处理，从而提高算法匹配效率的改进算法。最后本章对改进的匹配算法进行了时间复杂度的分析，将改进算法的时间复杂度与原算法的时间复杂度做比较，从而展示出改进算法的优越性。

5.设计模式的分析和识别

设计模式[4]被广泛的应用于现代软件工程的方方面面，为软件的开发提供了一套比较完善的系统方案，而且由于设计模式大致可以分为创建型的设计模式，行为驱动的设计模式，结构驱动的设计模式，语言相关的设计模式，特定领域相关的设计模式等几类。由于设计模式的类别比较有限，因而我们可以对设计模式进行识别已达到代码复用，增强软件可维护性的目的。下面本文将选取设计模式中一些结构特征明显，使用广泛的设计模式进行详尽地分析和说明，并详细的阐述如何使用基于结构匹配的模型查询技术进行设计模式的识别，

5.1 设计模式的分析

在设计模式中，由于适配器（变压器）模式（Adapter）结构特征明显，易于进行分析和匹配，而装饰模式（Decorator），桥梁模式（Bridge），门面模式（Façade)这三种模式在实际生产中使用广泛而且涵盖了模型识别中的各种模型元素，比较的有代表性，因而本文选取这四种模式作为分析的对象。

5.1.1 Adapter—适配器（变压器）模式

概念

GoF 在他们的《设计模式》这本书中是这样定义适配器模式的：“将一个类的接口转换成客户希望的另外一个接口。Adapter 模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作”[18]

UML 图

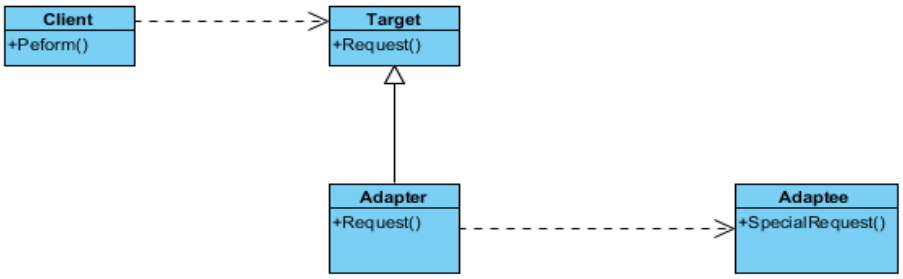


图 5-1 适配器模式 UML 模型图

- Target：定义 Client 使用的特定的接口。
- Client：与符合 Target 接口的对象协同。
- Adaptee：定义需要适配的一个已经存在的接口。
- Adapter：对 Adaptee 和 Target 进行适配，将 Adaptee 转换成 Target。

类间关系：Client 和 Target 之间是依赖关系，Adapter 和 Adaptee 之间是依赖关系，Target 和 Adapter 之间是继承关系。

实例展示

将笔记本电脑连接液晶电视来展示信息，但是大多数笔记本电脑只有 VAG 接口，而液晶电视只有 HDMI 接口因而需要一个 VAG 接口转换成 HDMI 接口的适配器，UML 图如下：

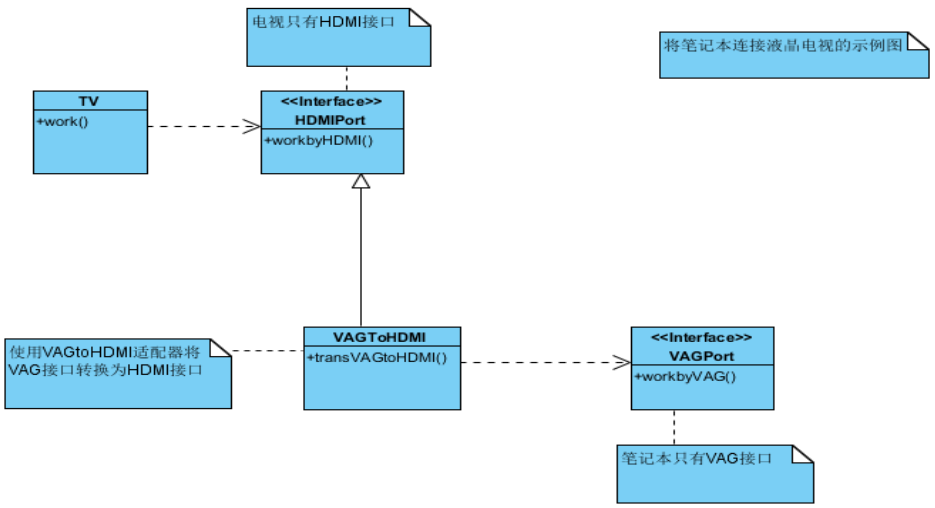


图 5-2 适配器模式实例 UML 模型图

5.1.2 Bridge—桥梁模式

概念

GoF 在他们的《设计模式》这本书中是这样定义桥梁模式的：“将抽象部分与它的现实部分分离，使它们都可以独立的变化”[19]也就是说在一个软件系统中将抽象部分和现实部分使用组合聚合关系来连接而不是使用继承关系来连接。

UML 图

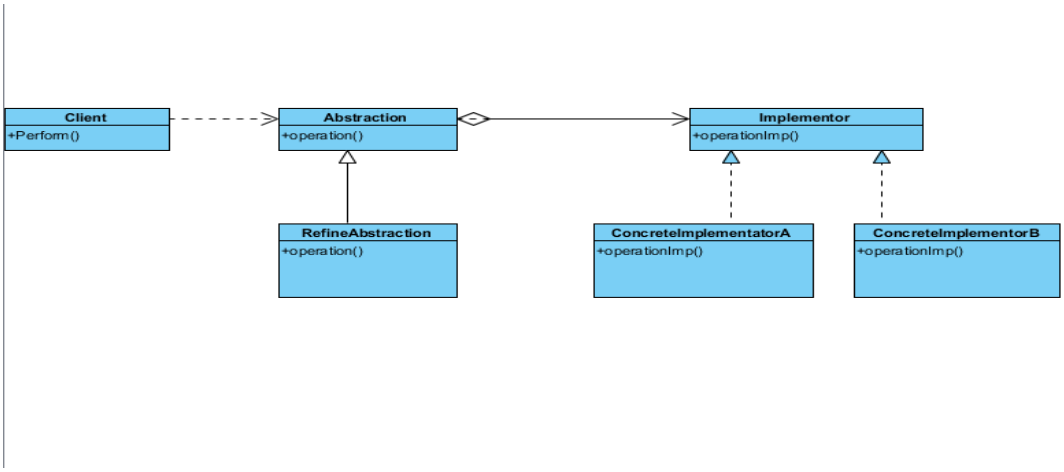


图 5-3 桥梁模式 UML 模型图

• Abstraction：对给出的定义抽象化。

• Refined Abstraction：对已经抽象化的角色进行扩展，对父类进行修改和修正。

• Implementor：现实化角色的接口。

• Concrete Implementor：给出现实化角色接口的具体实现

类间关系:Client 和 Abstraction 之间是依赖关系,Abstraction 和 Refined Abstraction 之间是继承关系,Abstraction 和 Implementor 之间是聚合/组合关系，Implementor 和 Concrete Implementor 是实现关系。

实例展示

手机的品牌有 3 种 samsung,apple,sony,手机软件有 5 种为,addressbook,

dictionary，message，email，game，每种手机有这五种软件的若干种，用桥梁模式表示的 UML 图如下：

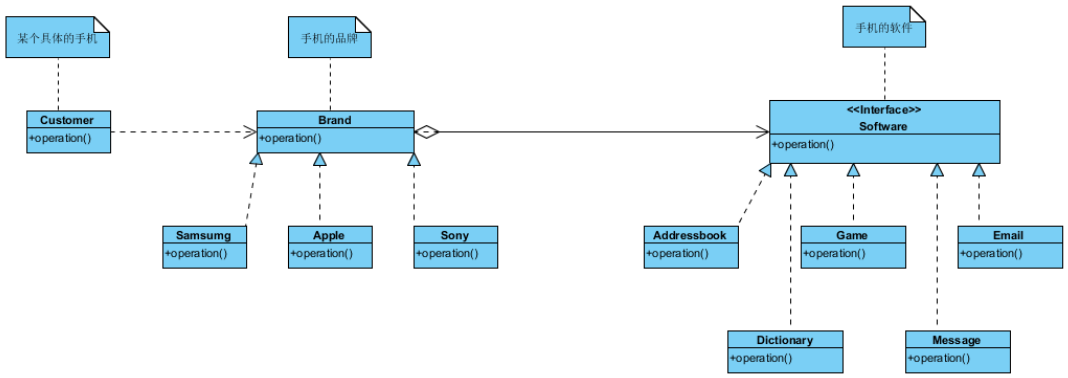


图 5-4 桥梁模式实例 UML 模型图

5.1.3 Facade—门面模式

概念

GoF 在他们的《设计模式》这本书中是这样定义门面模式的：“为子系统中的一组接口提供一个一致的界面，Facade 模式定义了一个高层接口，这个接口使得这一子系统更加容易使用”[20]也就是说每个子系统都可以有一个门面，一个软件系统可以有多个门面。

UML 图

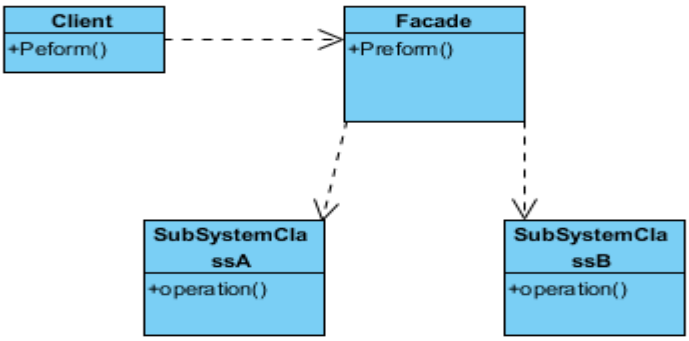


图 5-5 门面模式 UML 模型图

- Client：外部角色对系统的访问。
- Façade：实现对系统的访问门面类。
- SubSystemClassA：子系统的类 A 。
- SubSystemClassB：子系统的类 B 。

类间关系：所有类之间的关系都是依赖关系。

实例展示

在医院里，一个病人要看病要进行一下活动：挂号、门诊、划价、收费、取药等，我们设置一个接待台和一个门诊部，接待台负责挂号、划价、收费、取药，而门诊部则有医生负责门诊，门面模式的 UML 图如下：

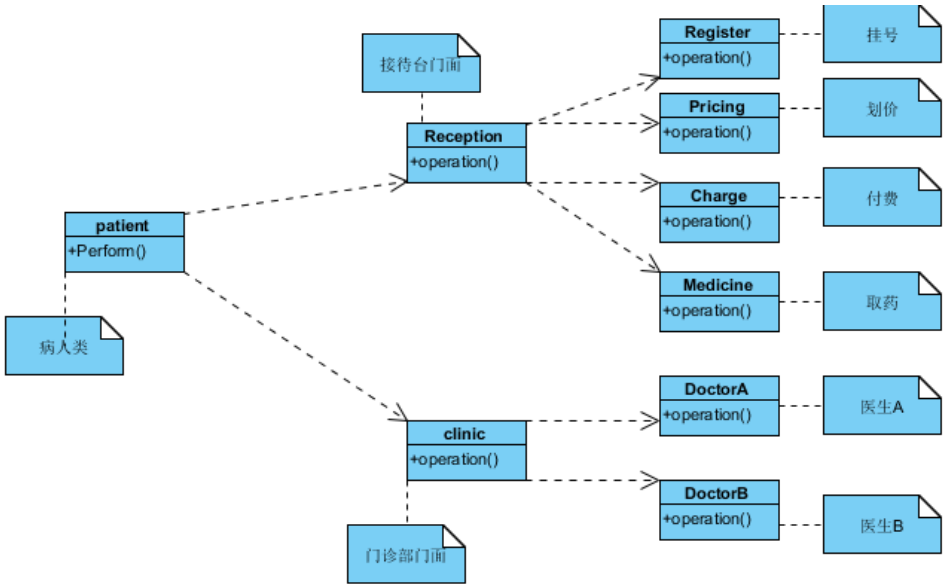


图 5-6 门面模式实例 UML 模型图

5.1.4 Decorator—装饰模式

概念

GoF 在他们的《设计模式》这本书中是这样定义装饰模式的：“动态地给一个对象添加一些额外的职责。就增加功能来说 Decorator 模式相比生成子类更为灵活”[21]。

UML 图

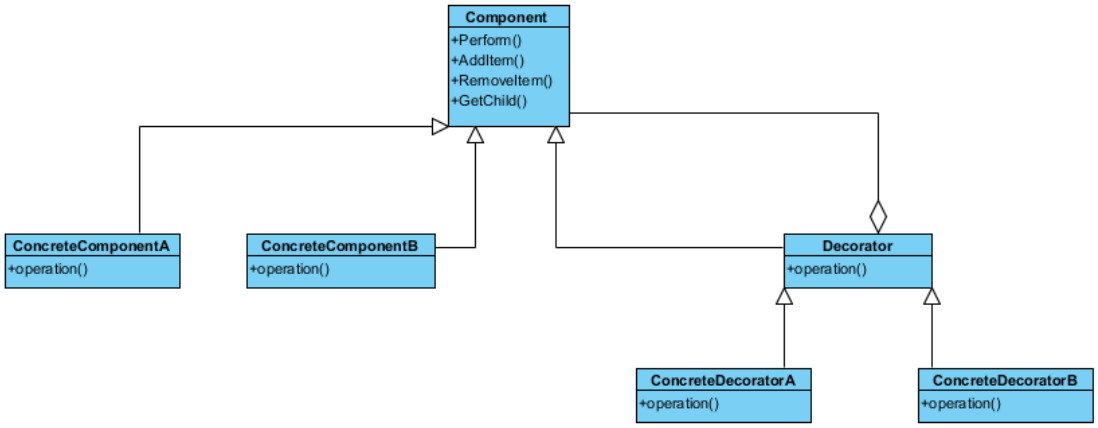


图 5-7 装饰模式 UML 模型图

- Component：角色的抽象类或接口。
- ConcreteComponentA：角色 A 的具体实现类。
- ConcreteComponentB：角色 B 的具体实现类。
- Decorator：装饰类的抽象类。
- Concrete DecoratorA：装饰类的具体实现 A。
- Concrete DecoratorB：装饰类的具体实现 B。

类间关系：Component 和 ConcreteComponentA，ConcreteComponentB 之间都是实现关系，Component 和 Decorator 之间是聚合/组合关系同时 Decorator 是对 Component 的一种实现，Concrete DecoratorA，Concrete DecoratorB 和 Decorator 之间是实现关系。

实例展示

一个 RPG 游戏，里面的人物可以选人类，精灵，恶魔，人物的装备有剑，刀，枪，铠甲，草鞋等，装饰模式的 UML 图如下：

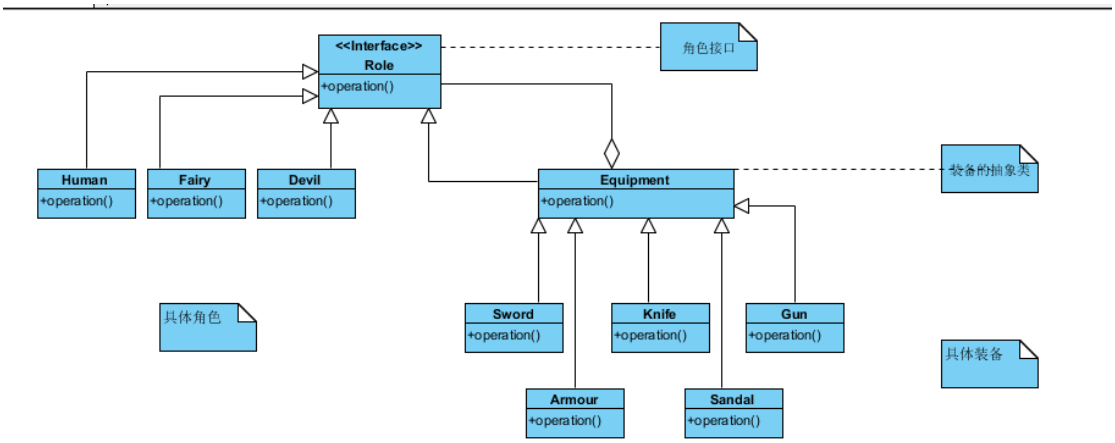


图 5-8 装饰模式实例 UML 模型图

5.2 设计模式的识别

要对设计模式进行识别首先必须要建立一个完备的模型库，本文使用一个开源的软件项目的设计模型作为模型库使用。在建立模型库时，对于那些拥有完整的设计模式模型图如 UML 类图等静态图的开源项目，我们将其设计模式的模型图直接作为模型库使用，对于那些只有代码，没有相应的 UML 设计模式模型图的开源软件项目，我们可以使用一些工具如 ArgoUML 让代码自动逆向生成其对应的设计模式的模型图作为模型库使用。在匹配时将会把模型库中的元素信息抽取出来和用户模型进行比较匹配。

对于用户需要识别的设计模式模型，由于我们需要在实际的项目中识别出用户指定的设计模式，因此这些输入的设计模式模型不能带有精确的信息也就只能是设计模式的抽象结构表示，因此我们首先要对设计模式的 UML 类图做模糊化处理，将其中一些具体的信息如类元的名称，关系的名称等去除，留下一个设计模式的 UML 类图的结构框架，然后对这个结构框架进行结构信息的抽取，将信息保存成一个个的节点，再对节点根据事先设定好的优先级进行排序处理得到一个匹配序列与模型库中的模型进行匹配。

设计模型识别的流程如下：

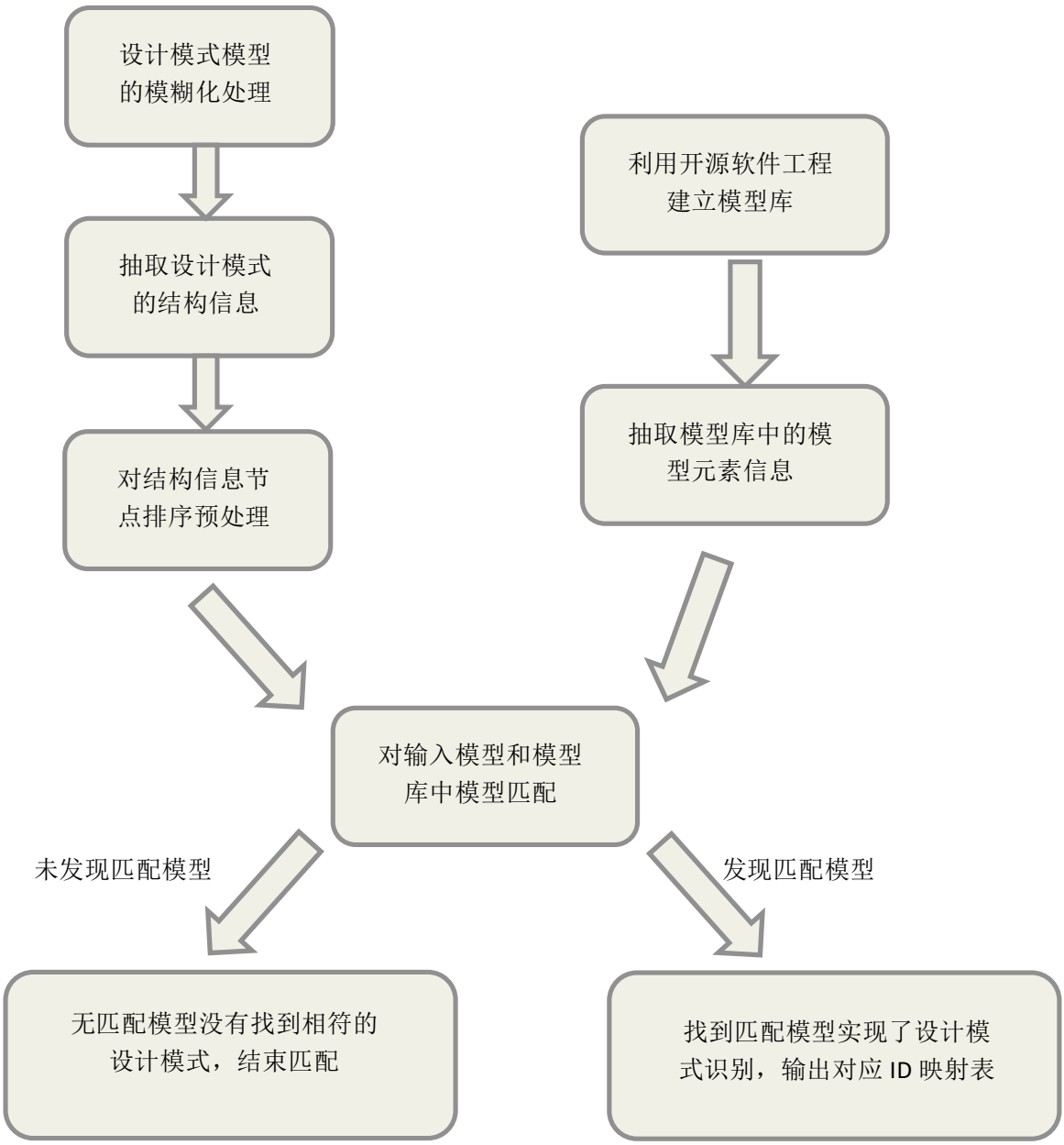


图 5-9 设计模式识别流程图

设计模式的识别算法如下：

```
RecognizeDesign(userfile, libfile)
1  list←Fileextractor(userfile);          Δ对用户 UML 图进行信息抽取
2  sortlist←sortedQueryPattern(list);     Δ对模型元素进行优先级排序
3  match(list, libfile);                  Δ对用户模型进行匹配
```

其中的优先级排序算法如下

```
sortedQueryPattern (queryPattern)
1   Push (firstClass);                Δ将优先级最高类元元素进栈
2   while  queryPattern or stack is not empty
3        do element←Pop();
```

```
4            list[i++]←element;        Δ将栈顶元素加入序列
5            if  element is Classifier
6              then push all relationship connected element into stack
7                   by Priority; Δ将关系按照优先级由低到高进栈
8           else if  element is Dependency
9              then push all classifier connected element into stack
10                  by Priority; Δ将类元按优先级由低到高进栈
11           else if  element is Abstraction
12             then push all classifier connected element into stack
13                  by Priority; Δ将类元按优先级由低到高进栈
14           else if  element is Generalization
15             then push all classifier connected element into stack
16                  by Priority; Δ将类元按优先级由低到高进栈
17           else if  element is Association
18             then push all classifier connected element into stack
19                  by Priority; Δ将类元按优先级由低到高进栈
```

详细代码实现参见附录 D

5.3 本章总结

本章首先着重对各种的设计模式进行了分析，并选取了结构特征明显的适配器(变压器)模式(Adapter)，使用广泛并且比较有代表性的装饰模式(Decorator)，桥梁模式（Bridge），门面模式（Facade）这四种模式进行了典型分析，从而可以详尽地阐述设计模式所有的一些特征。

然后本章节还介绍了设计模式识别的识别流程和设计模式的识别算法，其中识别流程主要包括了对用户输入的设计模式模型进行模糊化，结构信息抽取，输入队列排序等预处理，对模型库的生成以及用户设计模式模型与模型库中模型匹配这三个部分。而本章节的最后我们给出了设计模式识别的识别算法。

6. 实例研究

本章节将使用一个具体的实例来详细的展示，如何使用本文提出的设计模式识别技术对一些主流的设计模式进行识别。

6.1 实例研究

在使用本文提出的设计模式识别方法进行实例测试前，我们需要准备如下材料：

1．开源项目模型(模型库)

2．用户输入设计模式模型结构

本章之后的部分将会详细地叙述如何使用本文提出的设计模式识别方法进行设计模式的识别。

首先，假定用户需要在开源项目中对适配器模式进行设计模式的识别，要识别的适配器模式大致可以用如下的用户输入的设计模式的 UML 类图文件来表示。

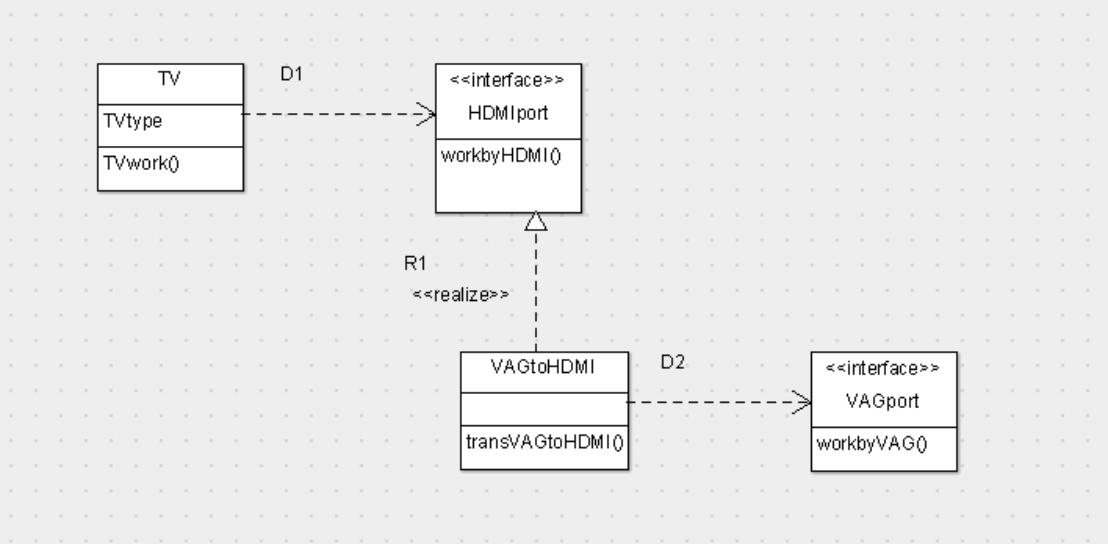


图 6-1 用户输入的设计模式模型的 UML 类图

要进行用户输入的设计模式模型的识别，第一步是对模型进行模糊化处理，对其中的具体的信息如类名，类间关系名等去除后，得到了一个模糊处理过后的只有框架结构信息的模型如图所示：

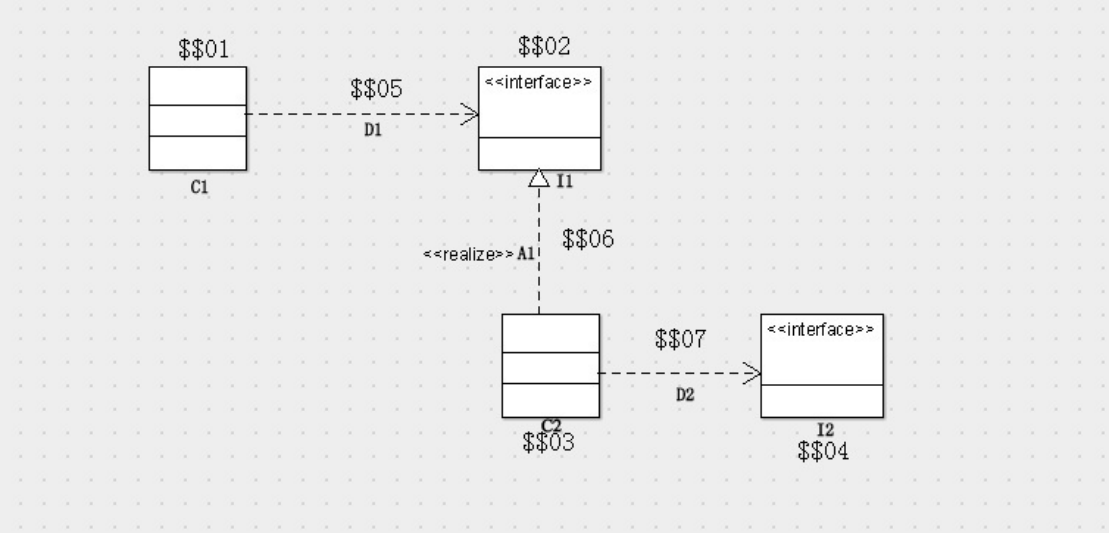


图 6-2 用户设计模式模型模糊化后的 UML 类图

其中类元模型元素共有 4 个记为 C1，C2，I1，I2，C1 的编号为$$01，C2 的编号为$$03，I1 的编号为$$02，I2 的编号为$$04,依赖关系的模型元素共有 2 个记为 D1，D2，D1 是 C1 和 I1 之间的关系，编号为$$05，D2 是 C2 和 I2 之间的关系，编号为$$07，实现关系的模型元素有 1 个，记为 A1，A1 是 C2 和 I1 之间的关系，编号为$$06。

对用户设计模式模型进行模糊化处理后，第二步是对使用 SAX 的 UML 解析工具对适配器设计模式的 UML 类图进行结构化的信息抽取可以得到以下信息节点：

```
模型包含的模型元素如下：

Class:
xmi.id: $$01
name:   null
visibility:     public
isAbstract:     false
fullName:       null
type:   UML:Class

Class:
xmi.id: $$02
name:   null
visibility:     public
isAbstract:     false
fullName:       null
type:   UML:Interface

Class:
xmi.id: $$03
```

图 6-3 用户设计模式模型元素的信息节点

抽取了用户设计模式的结构信息后第三步是对节点进行预处理得到一个信息节点的输入序列如图所示：

```
输入的节点序列如下：

Class:
xmi.id: $$01
name:   null
visibility:     public
isAbstract:     false
fullName:       null
type:   UML:Class

Dependency:
xmi.id: $$05
name:
client: $$01
clienttype:     null
supplier:       $$02
suppliertype:   null


Class:
xmi.id: $$02
name:   null
visibility:     public
isAbstract:     false
fullName:       null
type:   UML:Interface

Abstraction
xmi.id: $$06
```

图 6-4 信息节点的输入序列

得到的输入序列为：$$01→$$05→$$02→$$06→$$03→$$07→$$04

图形上的数字编号即为其模型 ID，将该序列拿到模型库中进行匹配可以得到以下匹配，模型元素 ID 之间的映射如下：

```
Mapping Result:
$$01->-84-26-16--102-1889f578:13eb645fd22:-8000:0000000000000B62
$$02->-84-26-16--102-1889f578:13eb645fd22:-8000:0000000000000B67
$$03->-84-26-16--102-1889f578:13eb645fd22:-8000:0000000000000B6A
$$04->-84-26-16--102-1889f578:13eb645fd22:-8000:0000000000000B6F
$$05->-84-26-16--102-1889f578:13eb645fd22:-8000:0000000000000B72
$$06->-84-26-16--102-1889f578:13eb645fd22:-8000:0000000000000B6B
$$07->-84-26-16--102-1889f578:13eb645fd22:-8000:0000000000000B73
```

图 6-5 设计模式识别后节点映射图

其 UML 类图对比如下：

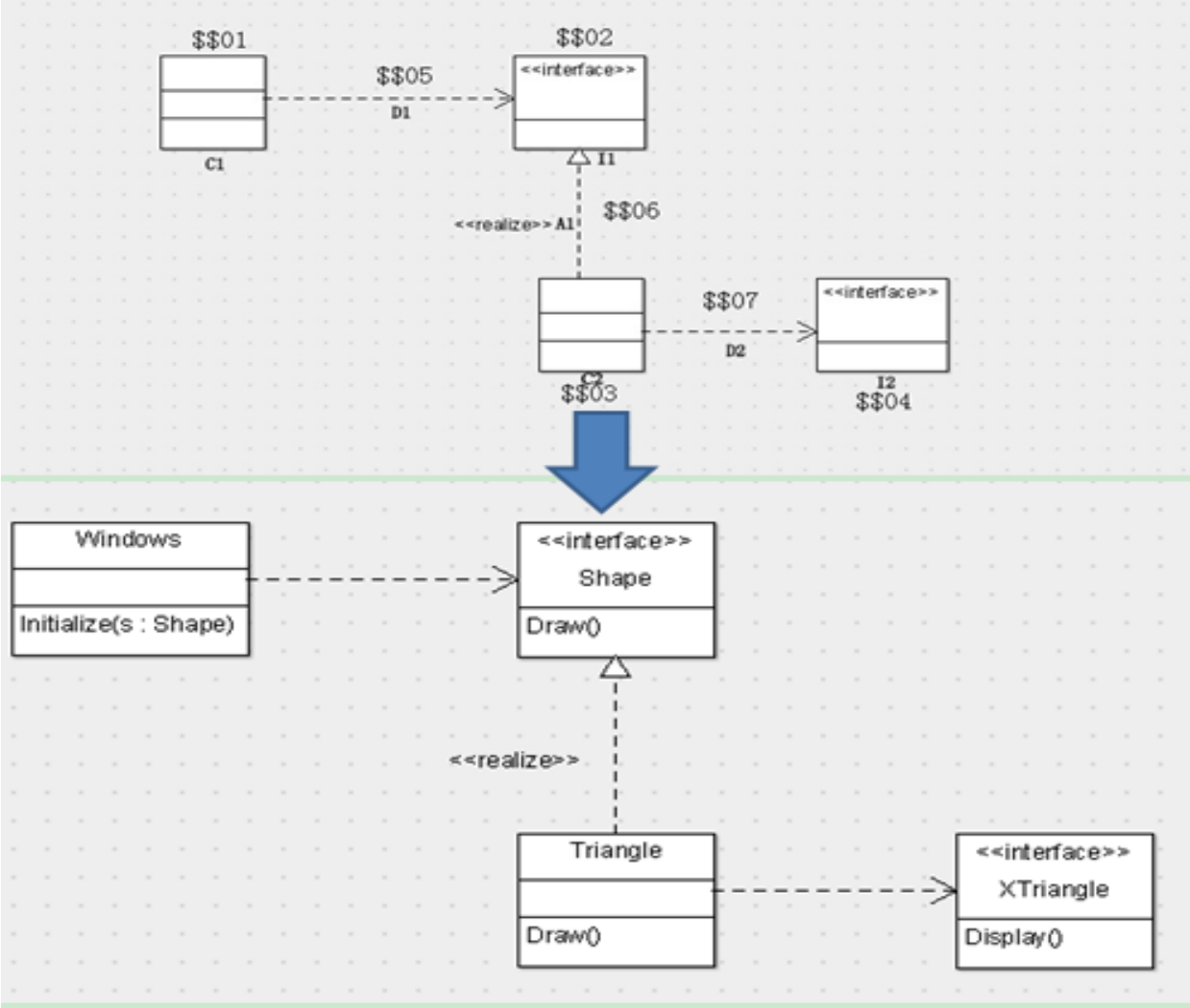


图 6-6 UML 类图对比

由于各个信息节点的 ID 在模型库中是唯一的，我们可以通过节点 ID 来定位模型库中的模型元素。我们可以看到$$01 节点映射到 Windows 类，$$02 节点映射到 Shape 接口，$$03 节点映射到 Triangle 类，$$04 节点映射到 XTriangle 接口，$$05 节点映射到 Windows 类和 Shape 接口之间的依赖关系，$$06 节点映射到 Triangle 类和 Shape 接口之间的继承关系，$$07 节点映射到 Triangle 类和 XTriangle 接口之间的依赖关系。通过以上操作我们便实现了设计模式的识别。

6.2 本章总结

本章通过对适配器模式的识别过程的详细展示和阐述，较好的说明了如何使用本文提出的设计模式识别技术在一个开源的软件工程中实现设计模式的识别。

7. 总结

本章首先对现有的相关工作进行比较，可以看出我们的工作相对于其他同类工作而言，具有更加直观、灵活和易用性的特点，并且可以对已有的设计模式结构进行快速扩展，从而适用于更多的模式。最后，我们也对现有工作进行了展望，给出下一步工作计划。

7.1 与相关工作的比较

传统的设计模式识别方法通常采用相似度匹配[22]，模板匹配[30]，DNIT (Depth-Node-Input Table)[23] 匹配以及利用控制流（CFG）图进行匹配[28][29]等方法，这些方法有的将设计模式的模型图转换为矩阵，通过计算矩阵的相似度的方法对设计模式进行识别，有的利用改变匹配图中的节点来计算匹配图的相似性来实现设计模式的识别，还有的利用基于基本块的数据流分析方法进行设计模式的识别。

相似度匹配[22]直接利用图的矩阵表示法对图的节点进行相似度计算，其缺陷在于这种设计模式的识别方法只注重各个节点之间的相似程度，而忽略了整张模型图的相似程度，这样只能保证各个节点的匹配而不能够保证整幅模型图的匹配，因而准确度不够理想。

模板匹配[30]则是根据不同的关系来构造多个关系矩阵然后将关系矩阵利用特征值合并归一的方式得到一个特征矩阵，然后对两个图的特征矩阵使用例如NCC 算法 (Normal Cross Correlation)[27][30]等相似性计算算法来计算两个图的相似性，然而这种方式虽然准确度较相似度匹配高，但是它仍然注重于节点相似而不是全图相似只是其携带的结构信息更多。

DNIT (Depth-Node-Input Table)[23] 匹配也是一种图匹配算法，它将图匹配的过程分解成为 k 步，k 的值为两幅图中的一副经过变换若干个节点可以变得相同的节点个数。首先利用一个由祖先节点的数目，子女节点的数目，兄弟节点的数目组成的三元组(t1,t2,t3)来表示模型图中的一个节点，然后再根据三元组使用一些算法来计算出每一个节点的特征值，这样便将整张模型图的节点信息转换成了该图的节点特征值表，然后再计算设计模式的模型图和用户系统模型图对

应的特征值表之间的距离，得到一个距离矩阵 P，然后使用迭代的方法迭代 K 步，每次迭代的过程都找出 P 中每行的最小值，最终获得到两幅图的相似度，借此来实现设计模式的识别。

利用控制流（CFG）图进行设计模式识别 [28][29]的方法是利用分析数据流的方式来分析系统的行为模式，得到一个个的基本块，将这些分析得到的基本块串联起来便得到了控制流图，利用控制流图对设计模式进行识别。

传统的设计模式识别的方法计算矩阵相似度过程非常的繁琐，对于设计模式的识别准确度也不高，而且将设计模式的模型转化成数字矩阵难以给人一个直观的识别过程的认识。文本提出的这种基于结构驱动的模型查询技术的设计模式的识别方法很好的克服了以上这些问题，识别过程清晰明了，识别的准确性也比较高。

7.2 总结和展望

本文结合一种基于结构驱动的 UML 模型查询技术提出了一种设计模式的识别方法。并且本文还针对模型查询技术中匹配算法由于使用递归而导致的时间开销大的问题作出了基于 UML 的定向优化改进。使用本文提出的设计模式识别方法，能够更高效灵活的理解代码，达到提高代码复用效率的目标，这样可以有效地提高软件的生产效率，降低软件的生产成本。

由于本文提出的算法时间复杂度虽然较之前的算法有所改进，但时间复杂度仍然比较高，还有很大的改进空间。下一步我们还可以尝试对算法进行进一步的优化，使得算法的运行效率更高，同时我们还打算将该工作完全移植到 EMF 框架下。

参考文献

[1]Yang Jian，Zhang Xiaoling，Zhou Shaoyun.Software Crisis and It’s Solution Measures. COMPUTER KNOWLEDGE AND TECHNOLOGY,pp135-136.2006

[2]Roger S.Pressman. Software Engineering : A Practitioner’s Approach, Sixth Edition,pp.1-13,2005.

[3] Liu Haiyan,Liang Jianlong,Suo Zhihai,Lu Qing. Design Pattern and Their Applications to Software Design,39(10),pp1043-1047,2005.

[4] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software,pp.54-231, 1995.

[5] 杜育根.软件工程教程：IBM RUP 方法实践,pp40-50,2005.

[6] W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition),2013. http://www.w3.org/TR/REC-xml/

[7] Lu BO ,Chai Yueting. On Unified Modeling Language—UML.COMPUTER ENGINEERING AND SCIENCE,22(4),pp.58-60,2000.

[8]WANG Fang , LI Zheng2fan. The Realization Method of Parsing XML Document by SAX. Journal of East China Jiaotong University,21(1), pp.84-86,2004.

[9] 李勇军 ,冀汶莉,马光思. 用 DOM 解析 XML 文档, Computer Applications, 21(8),pp103-105,2001.

[10] W3C. Document Object Model (DOM),2013. http://www.w3.org/DOM/.

[11] Oracle. Simple API for XML. Java API for XML Processing (JAXP) Tutorial, 2013. http://www.oracle.com/technetwork/java/sax-138988.html

[12] Thomas H.Cormen,Charles E.Leiserson,Ronald L.Rivest,Clifford Stein. Introduction to Algorithms,Second Edition.,pp330-331,2006.

[13]殷人昆. 数据结构：面向对象方法与 C++语言描述. pp364-365,2007.

[14] Wikipedia. Document Type Definition,2013. http://en.wikipedia.org /wiki/Document_Type_Definition .

[15] W3school. XML 系列教程,2013. http://www.w3school.com.cn/x.asp.

[16] Shao Wei-Zhong and Mei Hong. REVIEW OF THE UNIFIED MODELING LANGUAGE(UML). JOURNAL OF COMPUTER RESEARCH AND DEVELOPMENT, 36 (4),1999.

[17] 殷人昆. 数据结构：面向对象方法与 C++语言描述. Pp101-109,2007.

[18] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software , pp.92-99, 1995.

[19] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software,pp.100-107, 1995.

[20] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software , pp.121-128, 1995.

[21] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software , pp.108-121, 1995.

[22] Jing Dong, Yongtao Sun, Yajing Zhao. Design pattern detection by template matching. ACM Symposium on Applied Computing - SAC , pp.765-766, 2008.

[23] A. Pandel, Manjari Gupta, A.K.Tripathi. DNIT—A new approach for design pattern detection. International Conference on Computer and Communication Technology - ICCCT,2010.

[24] Wikipedia. Subgraph isomorphism problem,2013. http://en.wikipedia.org/wiki/Subgraph_isomorphism_problem

[25] Thomas H.Cormen,Charles E.Leiserson,Ronald L.Rivest,Clifford Stein. Introduction to Algorithms,Second Edition.,pp597-627,2006.

[26] Thomas H.Cormen,Charles E.Leiserson,Ronald L.Rivest,Clifford Stein. Introduction to Algorithms,Second Edition.,pp222-239,2006.

[27] J. P. Lewis.Fast Template Matching. Vision Interface. pp120-123, 1995.

[28]Manjari Gupta, R. Singh Rao, A. K. Tripathi. Design Pattern Detection

using inexact graph matching. Proceedings of the International Conference on Communication and Computational Intelligence. pp.211-217, 2010.

[29]Nija Shi, Ronald A. Olsson. Reverse Engineering of Design Patterns from Java Source Code. Automated Software Engineering - ASE. pp.123-134, 2006.

[30] Jing Dong, Yongtao Sun, Yajing Zhao. Design pattern detection by template matching. ACM Symposium on Applied Computing - SAC , pp.766-769, 2008.

致谢

经过 12 周的努力，我终于完成了这篇毕业设计。在毕业设计的完成的过程中我遇到了许多的问题和困难，但是在老师的精心指导下，在学长和同学的积极帮助下，我最终克服了一个又一个的问题和困难。

在此我首先要感谢我的毕业设计指导老师张天副教授，他从选题到毕业设计的进行再到最后的毕业论文的撰写都以严谨求实的态度要求我，这才使我能够完成一篇优秀的毕业论文。

其次我要感谢吴小兵老师和张学林学长，吴小兵老师在匹配算法这一方面给了我很大的帮助，通过吴小兵老师的指导我才能够对匹配算法进行合理的分析。而张学林学长在每当我遇到毕业设计中的难题时都积极的为我解惑，在我撰写毕业论文的过程中，张学林学长也给了我很多很好的毕业设计论文撰写的建议。

最后我要感谢本篇论文参考资料的诸位专家和学者，没有你们的研究成果本篇论文将难以完成。

附录 A 模型元素及其属性值分布表

编号	模型元素	属性名称	值类型	类别	类别数目	比例（%）
1	Classifier	Name	字符串	N/A	1	0.0306%
2	Classifier	Visibility	枚举类型	Public	1169	35.74%
				Private	46	1.41%
				Protected	129	3.94%
				Package	1927	58.91%
3	Classifier	isAbstract	枚举类型	true	50	1.53%
				flase	3221	98.47%
4	Attribute	Name	字符串	N/A	1	0.0127%
5	Attribute	InitialValue	字符串	N/A	1	0.0127%
6	Attribute	Visibility	枚举类型	Protected	468	5.92%
				Public	1495	18.92%
				Private	5017	63.51%
				Package	920	11.65%
7	Attribute	ownerScope	枚举类型	Classifier	1916	24.25%
				Instance	5984	75.75%
8	Attribute	targetScope	枚举类型	Instance	7900	100%
9	Attribute	multiplicity. range.lower	数值类型	1	7900	100%
10	Attribute	Multiplicity .range.upper	数值类型	1	7900	100%
11	Operation	Name	字符串	N/A	1	0.0068%
12	Operation	Visibility	枚举类型	Protected	3035	20.79%
				Public	10627	72.78%
				Private	715	4.90%
				Package	224	1.53%
13	Operation	ownerScope	枚举类型	Classifier	1006	6.89%
				instance	13595	93.11%
14	Operation	isQuery	枚举类型	true	0	0%
				false	14601	100%
15	Operation	Concurrency	枚举类型	Sequential	14508	99.36%
				Guarded	93	0.64%
16	Operation	IsAbstract	枚举类型	true	142	0.97%
				false	14459	99.03%
17	Abstraction	Name	枚举类型	UML：Interface	840	100%
18	Generalization	child.type	枚举类型	UML：Class	277	87.93%
				UML：Interface	38	12.07%

19	Generalization	parent.type	枚举类型	UML：Class	277	87.93%
				UML：Interface	38	12.07%
20	Dependency	Client.type	枚举类型	null	3016	100%
21	Dependency	Supplier.type	枚举类型	UML：Interface	1336	44.30%
				UML：Class	1434	47.55%
				UML：Package	246	8.15%

附录 B 用户设计模式模型信息抽取算法实现

Fileextractor.java

```
package designpatternrecognize;
import java.util.ArrayList;
import java.util.List;
import model.query.argouml.util.MAbstraction;
import model.query.argouml.util.MAttribute;
import model.query.argouml.util.MAssociation;
import model.query.argouml.util.MClassifier;
import model.query.argouml.util.MDependency;
import model.query.argouml.util.MGeneralization;
import model.query.argouml.util.MOperation;
import model.query.simple.AbstractionHandlerExtractor;
import model.query.simple.AttributeHandlerExtractor;
import model.query.simple. AssociationHandlerExtractor;
import model.query.simple.ClassifierHandlerExtractor;
import model.query.simple.DependencyHandlerExtractor;
import model.query.simple.GeneralizationHandlerExtractor;
import model.query.simple.OperationHandlerExtractor;

public class Fileextractor {
      private String sourceFileNamex;
      private List queryPattern = new ArrayList();
      Fileextractor(String path){this.sourceFileNamex=path;}
       public void setsourceFileNamex(String path){ this.sourceFileNamex=path;}
       public String getsourceFileNamex() {return sourceFileNamex;}
       public List getqueryPattern(){return queryPattern;}
      public    void infoext()
      {ClassifierHandlerExtractor classifierHandlerExtractor = new ClassifierHandlerExtractor
(sourceFileNamex);
            classifierHandlerExtractor.processExtract();
            List<MClassifier> list = classifierHandlerExtractor.PInfoHandler.listClassifier;
            for(int k = 0; k < list.size(); k++){
                  String classifierID = list.get(k).getId();
                  AttributeHandlerExtractor attributeHandlerExtractor = new
AttributeHandlerExtractor(sourceFileNamex);
                  attributeHandlerExtractor.processExtract();
                  List<MAttribute> listAttributes =
attributeHandlerExtractor.PInfoHandler.listAttributes;
                  for (int i = 0; i < listAttributes.size(); i++) {
                        if (((MAttribute) listAttributes.get(i)).getOwnerClassifier()
```

```
                        .getId().trim().equals(classifierID.trim()))
                {listAttributes.get(i).setId("$$"+listAttributes.get(i).getId());
                    list.get(k).addAttribute((MAttribute) listAttributes.get(i));
                }
            }
            // 获取方法的信息
            OperationHandlerExtractor operationHandlerExtractor = new
OperationHandlerExtractor(sourceFileNamex);
            operationHandlerExtractor.processExtract();
            List<MOperation> listOperations =
operationHandlerExtractor.PInfoHandler.listOperations;
            for (int i = 0; i < listOperations.size(); i++) {
                if (((MOperation) listOperations.get(i)).getOwnerClassifier()
                        .getId().trim().equals(classifierID.trim()))
                {listOperations.get(i).setId("$$"+listOperations.get(i).getId());
                    list.get(k).addOperation((MOperation) listOperations.get(i));}
            }
        }
        for (int i = 0; i < list.size(); i++) {
            list.get(i).setId("$$"+list.get(i).getId());
            queryPattern.add(list.get(i));}
        //获取类之间关系
        GeneralizationHandlerExtractor generalizationHandlerExtractor = new
GeneralizationHandlerExtractor(sourceFileNamex);
        generalizationHandlerExtractor.processExtract();
        List<MGeneralization> genList =
generalizationHandlerExtractor.PInfoHandler.listGeneralizations;
        for (int i = 0; i < genList.size(); i++) {
            genList.get(i).setId("$$"+genList.get(i).getId());
            genList.get(i).setChildxmiidref("$$"+genList.get(i).getChildxmiidref());
            genList.get(i).setParentxmiidref("$$"+genList.get(i).getParentxmiidref());
            queryPattern.add(genList.get(i));}
        AbstractionHandlerExtractor abstractionHandlerExtractor = new
AbstractionHandlerExtractor(sourceFileNamex);
        abstractionHandlerExtractor.processExtract();
        List<MAbstraction> abList =
abstractionHandlerExtractor.PInfoHandler.listAbstractions;
        for (int i = 0; i < abList.size(); i++) {
            abList.get(i).setId("$$"+abList.get(i).getId());
            abList.get(i).setClient("$$"+abList.get(i).getClient());
            abList.get(i).setSupplier("$$"+abList.get(i).getSupplier());
            queryPattern.add(abList.get(i));}
        DependencyHandlerExtractor dependencyHandlerExtractor = new
DependencyHandlerExtractor(sourceFileNamex);
```

```
            dependencyHandlerExtractor.processExtract();
            List<MDependency>                    deList                    =
dependencyHandlerExtractor.PInfoHandler.listDependencies;
            for (int i = 0; i < deList.size(); i++) {
                deList.get(i).setId("$$"+deList.get(i).getId());
                deList.get(i).setClient("$$"+deList.get(i).getClient());
                deList.get(i).setSupplier("$$"+deList.get(i).getSupplier());
                queryPattern.add(deList.get(i));}
            AssociationHandlerExtractor    associationHandlerExtractor    =    new
AssociationHandlerExtractor(sourceFileNamex);
            associationHandlerExtractor.processExtract();
            List<MAssociation>                   assList                   =
associationHandlerExtractor.PInfoHandler.listAssociations;
            for (int i = 0; i < assList.size(); i++) {
                assList.get(i).setId("$$"+assList.get(i).getId());
                assList.get(i).setClient("$$"+assList.get(i).getClient());
                assList.get(i).setSupplier("$$"+assList.get(i).getSupplier());
                queryPattern.add(assList.get(i));}
        }
    }
```

附录 C 输入队列预处理算法代码实现

Getsortedlist.java

```
package designpatternrecognize;
import java.util.ArrayList;
import java.util.List;
import java.util.Stack;
import model.query.argouml.util.MAbstraction;
import model.query.argouml.util.MClassifier;
import model.query.argouml.util.MDependency;
import model.query.argouml.util.MGeneralization;
public class Getsortedlist {
      private List sortedQueryPattern = new ArrayList();
       public List getsortedQueryPattern()
       {return sortedQueryPattern; }
       public List sortbyasspriority(List inlist,List queryPattern,Object obj )
       {     List outlist = new ArrayList();
             List templist = new ArrayList();
             for (int i = 0; i < inlist.size(); i++) {
                       if (inlist.get(i) instanceof MDependency) {
                            for(int k=0;k<queryPattern.size();k++){
                                 if (queryPattern.get(k) instanceof MClassifier) {
                                 if(((MDependency)inlist.get(i)).getSupplier()
                                           .equals(((MClassifier)queryPattern.get(k)).getId())&&
                                           !((MClassifier) obj).getId()
                                           .equals(((MClassifier)queryPattern.get(k)).getId()))
                                 { templist.add(queryPattern.get(k)); }
                                 }
                            }
                       }
             }
             templist=sortbypriority(templist);
             for(int k=0;k<templist.size();k++)
                 {        if (templist.get(k) instanceof MClassifier) {
                            for (int i = 0; i < inlist.size(); i++) {
                                 if (inlist.get(i) instanceof MDependency) {
                                      if(((MDependency)inlist.get(i)).getSupplier()
                                                .equals(((MClassifier)templist.get(k)).getId()))
                                      {outlist.add(inlist.get(i));
                                           inlist.remove(i);
                                           i--;
                                      }
```

```
                    }
                }
            }
        }
    while(!templist.isEmpty())
        templist.remove(0);
    for (int i = 0; i < inlist.size(); i++) {
            if (inlist.get(i) instanceof MDependency) {
                for(int k=0;k<queryPattern.size();k++)
                {   if (queryPattern.get(k) instanceof MClassifier) {
                    if(((MDependency)inlist.get(i)).getClient()
                            .equals(((MClassifier)queryPattern.get(k)).getId())&&
                            !((MClassifier) obj).getId()
                            .equals(((MClassifier)queryPattern.get(k)).getId()))
                    {templist.add(queryPattern.get(k)); }
                    }
                }
            }
    }
    templist=sortbypriority(templist);
    for(int k=0;k<templist.size();k++)
        {   if (templist.get(k) instanceof MClassifier) {
                for (int i = 0; i < inlist.size(); i++) {
                    if (inlist.get(i) instanceof MDependency) {
                        if(((MDependency)inlist.get(i)).getClient()
                                .equals(((MClassifier)templist.get(k)).getId()))
                        {   outlist.add(inlist.get(i));
                            inlist.remove(i);
                            i--;
                        }
                    }
                }
            }
        }
    while(!templist.isEmpty())
        templist.remove(0);
     for (int i = 0; i < inlist.size(); i++) {
            if (inlist.get(i) instanceof MAssociation) {
                for(int k=0;k<queryPattern.size();k++)
                { if (queryPattern.get(k) instanceof MClassifier) {
                    if(((MAssociation)inlist.get(i)).getClient()
                            .equals(((MClassifier)queryPattern.get(k)).getId())&&
                            !((MClassifier) obj).getId()
                            .equals(((MClassifier)queryPattern.get(k)).getId()))
```

```
                        {templist.add(queryPattern.get(k)); }
                        }
                    }
                }
            }
            templist=sortbypriority(templist);
            for(int k=0;k<templist.size();k++)
                {if (templist.get(k) instanceof MClassifier) {
                    for (int i = 0; i < inlist.size(); i++) {
                        if (inlist.get(i) instanceof MAssociation) {
                            if(((MAssociation)inlist.get(i)).getClient()
                                    .equals(((MClassifier)templist.get(k)).getId()))
                            {outlist.add(inlist.get(i));
                                inlist.remove(i);
                                i--;
                            }
                        }
                    }
                }
            }
            while(!templist.isEmpty())
                templist.remove(0);
            for (int i = 0; i < inlist.size(); i++) {
                if (inlist.get(i) instanceof MAbstraction) {
                    for(int k=0;k<queryPattern.size();k++)
                    {if (queryPattern.get(k) instanceof MClassifier) {
                        if(((MAbstraction)inlist.get(i)).getSupplier()
                                .equals(((MClassifier)queryPattern.get(k)).getId())&&
                                !((MClassifier) obj).getId()
                                .equals(((MClassifier)queryPattern.get(k)).getId()))
                        {templist.add(queryPattern.get(k)); }
                        }
                    }
                }
            }
            templist=sortbypriority(templist);
            for(int k=0;k<templist.size();k++)
                {   if (templist.get(k) instanceof MClassifier) {
                    for (int i = 0; i < inlist.size(); i++) {
                        if (inlist.get(i) instanceof MAbstraction) {
                            if(((MAbstraction)inlist.get(i)).getSupplier()
                                    .equals(((MClassifier)templist.get(k)).getId()))
                            {   outlist.add(inlist.get(i));
                                inlist.remove(i);
```

```
                                    i--;
                                }
                            }
                        }
                    }
                }
            while(!templist.isEmpty())
                templist.remove(0);
            for (int i = 0; i < inlist.size(); i++) {
                    if (inlist.get(i) instanceof MAbstraction) {
                        for(int k=0;k<queryPattern.size();k++)
                        {if (queryPattern.get(k) instanceof MClassifier) {
                            if(((MAbstraction)inlist.get(i)).getClient()
                                    .equals(((MClassifier)queryPattern.get(k)).getId())&&
                                    !((MClassifier) obj).getId()
                                    .equals(((MClassifier)queryPattern.get(k)).getId()))
                            {templist.add(queryPattern.get(k)); }
                            }
                        }
                    }
            }
            templist=sortbypriority(templist);
            for(int k=0;k<templist.size();k++)
                {if (templist.get(k) instanceof MClassifier) {
                        for (int i = 0; i < inlist.size(); i++) {
                            if (inlist.get(i) instanceof MAbstraction) {
                                if(((MAbstraction)inlist.get(i)).getClient()
                                        .equals(((MClassifier)templist.get(k)).getId()))
                                {   outlist.add(inlist.get(i));
                                    inlist.remove(i);
                                    i--;}
                            }
                        }
                    }
                }
            while(!templist.isEmpty())
                templist.remove(0);
            for (int i = 0; i < inlist.size(); i++) {
                    if (inlist.get(i) instanceof MGeneralization) {
                        for(int k=0;k<queryPattern.size();k++)
                        {if (queryPattern.get(k) instanceof MClassifier) {
                            if(((MGeneralization)inlist.get(i)).getParentxmiidref()
                                    .equals(((MClassifier)queryPattern.get(k)).getId())&&
                                    !((MClassifier) obj).getId()
```

```
                                .equals(((MClassifier)queryPattern.get(k)).getId()))
                        {templist.add(queryPattern.get(k)); }
                        }
                }
        }
}
templist=sortbypriority(templist);
for(int k=0;k<templist.size();k++)
    {
        if (templist.get(k) instanceof MClassifier) {
            for (int i = 0; i < inlist.size(); i++) {
                if (inlist.get(i) instanceof MGeneralization) {
                    if(((MGeneralization)inlist.get(i)).getParentxmiidref()
                            .equals(((MClassifier)templist.get(k)).getId()))
                    {   outlist.add(inlist.get(i));
                        inlist.remove(i);
                        i--;}
                }
            }
        }
    }
while(!templist.isEmpty())
    templist.remove(0);
for (int i = 0; i < inlist.size(); i++) {
        if (inlist.get(i) instanceof MGeneralization) {
            for(int k=0;k<queryPattern.size();k++)
            {if (queryPattern.get(k) instanceof MClassifier) {
                if(((MGeneralization)inlist.get(i)).getChildxmiidref()
                        .equals(((MClassifier)queryPattern.get(k)).getId())&&
                        !((MClassifier) obj).getId()
                        .equals(((MClassifier)queryPattern.get(k)).getId()))
                {templist.add(queryPattern.get(k)); }
                }
            }
        }
}
templist=sortbypriority(templist);
for(int k=0;k<templist.size();k++)
    {if (templist.get(k) instanceof MClassifier) {
            for (int i = 0; i < inlist.size(); i++) {
                if (inlist.get(i) instanceof MGeneralization) {
                    if(((MGeneralization)inlist.get(i)).getChildxmiidref()
                            .equals(((MClassifier)templist.get(k)).getId()))
                    {outlist.add(inlist.get(i));
```

```
                                        inlist.remove(i);
                                        i--;
                                    }
                                }
                            }
                        }
                    }
                while(!templist.isEmpty())
                    templist.remove(0);
                while(!inlist.isEmpty())
                {outlist.add(inlist.get(0));
                    inlist.remove(0); }
                return outlist;
            }
            public List sortbypriority(List inlist)
            {
                List outlist = new ArrayList();
                for (int i = 0; i < inlist.size(); i++) {
                        if (inlist.get(i) instanceof MClassifier) {
                            if(((MClassifier)inlist.get(i)).getName()!=null)
                            {outlist.add(inlist.get(i));
                            inlist.remove(i);
                            i--;}
                        }
                }
                for (int i = 0; i < inlist.size(); i++) {
                        if (inlist.get(i) instanceof MClassifier) {
                            if(((MClassifier)inlist.get(i)).getIsAbstract().equals("true"))
                            {outlist.add(inlist.get(i));
                            inlist.remove(i);
                            i--;}
                        }
                }
                for (int i = 0; i < inlist.size(); i++) {
                        if (inlist.get(i) instanceof MClassifier) {
                            if(((MClassifier)inlist.get(i)).getVisibility().equals("private"))
                            {outlist.add(inlist.get(i));
                            inlist.remove(i);
                            i--;}
                        }
                }
                for (int i = 0; i < inlist.size(); i++) {
                        if (inlist.get(i) instanceof MClassifier) {
                            if(((MClassifier)inlist.get(i)).getVisibility().equals("protected"))
```

```
                    {outlist.add(inlist.get(i));
                    inlist.remove(i);
                    i--;}
                }
            }
            for (int i = 0; i < inlist.size(); i++) {
                if (inlist.get(i) instanceof MClassifier) {
                    if(((MClassifier)inlist.get(i)).getListAttributes()!=null)
                    {outlist.add(inlist.get(i));
                    inlist.remove(i);
                    i--;}
                }
            }
            for (int i = 0; i < inlist.size(); i++) {
                if (inlist.get(i) instanceof MClassifier) {
                    if(((MClassifier)inlist.get(i)).getListOperations()!=null)
                    {outlist.add(inlist.get(i));
                    inlist.remove(i);
                    i--;}
                }
            }
                for (int i = 0; i < inlist.size(); i++) {
                if (inlist.get(i) instanceof MAssociation) {
                    outlist.add(inlist.get(i));
                    inlist.remove(i);
                    i--;}
            }
            for (int i = 0; i < inlist.size(); i++) {
                if (inlist.get(i) instanceof MDependency) {
                    outlist.add(inlist.get(i));
                    inlist.remove(i);
                    i--;}
            }
            for (int i = 0; i < inlist.size(); i++) {
                if (inlist.get(i) instanceof MGeneralization) {
                    outlist.add(inlist.get(i));
                    inlist.remove(i);
                    i--;}
            }
            for (int i = 0; i < inlist.size(); i++) {
                if (inlist.get(i) instanceof MAbstraction) {
                    outlist.add(inlist.get(i));
                    inlist.remove(i);
                    i--;}
```

```
        }
        while(!inlist.isEmpty())
        { outlist.add(inlist.get(0));
            inlist.remove(0); }
        return outlist;
    }
public void transtosortlist(List queryPattern)
{
    queryPattern=sortbypriority(queryPattern);
    Stack ST = new Stack();
    List templist= new ArrayList();
    for (int i = 0; i < queryPattern.size(); i++) {
        if (queryPattern.get(i) instanceof MClassifier) {
            templist.add(queryPattern.get(i)); }
    }
    List outlist=null;
    outlist=sortbypriority(templist);
    ST.push(outlist.get(0));
    queryPattern.remove(outlist.get(0));
    while(!templist.isEmpty())
        templist.remove(0);
    while(!outlist.isEmpty())
        outlist.remove(0);
    while (!queryPattern.isEmpty()||!ST.isEmpty()) {
        Object obj = ST.pop();
        sortedQueryPattern.add(obj);
        if (obj instanceof MClassifier) {
            for (int i = 0; i < queryPattern.size(); i++) {
                if (queryPattern.get(i) instanceof MDependency) {
                    if (((MDependency) queryPattern.get(i)).getClient()
                            .equals(((MClassifier) obj).getId())) {
                        templist.add(queryPattern.get(i));
                        queryPattern.remove(i);
                        i--;
                    } else if (((MDependency) queryPattern.get(i))
                            .getSupplier().equals(
                                    ((MClassifier) obj).getId())) {
                        templist.add(queryPattern.get(i));
                        queryPattern.remove(i);
                        i--;}
                } else if (queryPattern.get(i) instanceof MGeneralization) {
                    if (((MGeneralization) queryPattern.get(i))
                            .getChildxmiidref().equals(
                                    ((MClassifier) obj).getId())) {
```

```
                templist.add(queryPattern.get(i));
                queryPattern.remove(i);
                i--;
            } else if (((MGeneralization) queryPattern.get(i))
                    .getParentxmiidref().equals(
                            ((MClassifier) obj).getId())) {
                templist.add(queryPattern.get(i));
                queryPattern.remove(i);
                i--;}
        } else if (queryPattern.get(i) instanceof MAbstraction) {
            if (((MAbstraction) queryPattern.get(i)).getClient()
                    .equals(((MClassifier) obj).getId())) {
                templist.add(queryPattern.get(i));
                queryPattern.remove(i);
                i--;
            } else if (((MAbstraction) queryPattern.get(i))
                    .getSupplier().equals(
                            ((MClassifier) obj).getId())) {
                templist.add(queryPattern.get(i));
                queryPattern.remove(i);
                i--;}
        }
    }
    outlist=null;
    outlist=sortbyasspriority(templist,queryPattern,obj );
    for(int h=outlist.size()-1;h>=0;h--)
        ST.push(outlist.get(h));
    while(!templist.isEmpty())
        templist.remove(0);
    while(!outlist.isEmpty())
        outlist.remove(0);
} else if (obj instanceof MDependency) {
    for (int i = 0; i < queryPattern.size(); i++) {
        if (queryPattern.get(i) instanceof MClassifier) {
            if (((MDependency) obj).getClient().equals(
                    ((MClassifier) queryPattern.get(i)).getId())) {
                templist.add(queryPattern.get(i));
                queryPattern.remove(i);
                i--;
            } else if (((MDependency) obj).getSupplier().equals(
                    ((MClassifier) queryPattern.get(i)).getId())) {
                templist.add(queryPattern.get(i));
                queryPattern.remove(i);
                i--;}
```

```
                }
            }
            outlist=null;
            outlist=sortbypriority(templist);
            for(int h=outlist.size()-1;h>=0;h--)
                ST.push(outlist.get(h));
            while(!templist.isEmpty())
                templist.remove(0);
            while(!outlist.isEmpty())
                outlist.remove(0);
        } else if (obj instanceof MAssociation) {
            for (int i = 0; i < queryPattern.size(); i++) {
                if (queryPattern.get(i) instanceof MClassifier) {
                    if (((MAssociation) obj).getClient().equals(
                            ((MClassifier) queryPattern.get(i)).getId())) {
                        templist.add(queryPattern.get(i));
                        queryPattern.remove(i);
                        i--;
                    } else if (((MAssociation) obj).getSupplier().equals(
                            ((MClassifier) queryPattern.get(i)).getId())) {
                        templist.add(queryPattern.get(i));
                        queryPattern.remove(i);
                        i--;}
                }
            }
            outlist=null;
            outlist=sortbypriority(templist);
            for(int h=outlist.size()-1;h>=0;h--)
                ST.push(outlist.get(h));
            while(!templist.isEmpty())
                templist.remove(0);
            while(!outlist.isEmpty())
                outlist.remove(0);
        } else if (obj instanceof MGeneralization) {
            for (int i = 0; i < queryPattern.size(); i++) {
                if (queryPattern.get(i) instanceof MClassifier) {
                    if (((MGeneralization) obj).getChildxmiidref().equals(
                            ((MClassifier) queryPattern.get(i)).getId())) {
                        templist.add(queryPattern.get(i));
                        queryPattern.remove(i);
                        i--;
                    } else if (((MGeneralization) obj).getParentxmiidref()
                            .equals(((MClassifier) queryPattern.get(i))
                                    .getId())) {
```

```
                templist.add(queryPattern.get(i));
                queryPattern.remove(i);
                i--;}
            }
        }
        outlist=null;
        outlist=sortbypriority(templist);
        for(int h=outlist.size()-1;h>=0;h--)
            ST.push(outlist.get(h));
        while(!templist.isEmpty())
            templist.remove(0);
        while(!outlist.isEmpty())
            outlist.remove(0);
    } else if (obj instanceof MAbstraction) {
        for (int i = 0; i < queryPattern.size(); i++) {
            if (queryPattern.get(i) instanceof MClassifier) {
                if (((MAbstraction) obj).getClient().equals(
                        ((MClassifier) queryPattern.get(i)).getId())) {
                    templist.add(queryPattern.get(i));
                    queryPattern.remove(i);
                    i--;
                } else if (((MAbstraction) obj).getSupplier().equals(
                        ((MClassifier) queryPattern.get(i)).getId())) {
                    templist.add(queryPattern.get(i));
                    queryPattern.remove(i);
                    i--;}
            }
        }
        outlist=null;
        outlist=sortbypriority(templist);
        for(int h=outlist.size()-1;h>=0;h--)
            ST.push(outlist.get(h));
        while(!templist.isEmpty())
            templist.remove(0);
        while(!outlist.isEmpty())
            outlist.remove(0);
    } else {System.err.println("Error: Unknown Query Type "
                + obj.getClass() + " .");}
    }
  }
}
```

附录 D 设计模式识别算法实现

designfinder.java

```
package designpatternrecognize;
import java.util.List;
import model.query.simple.Matcher;
import model.query.simple.Matcher.MatchResult;
public class designfinder {
	public void finder(String userpath,String libpath)
	{
		Fileextractor testf=new Fileextractor(userpath);
		  testf.infoext();
		  Getsortedlist glist=new Getsortedlist();
		  List testlist=testf.getqueryPattern();
		  glist.transtosortlist(testlist);
		  List sortedQueryPattern=glist.getsortedQueryPattern();
		  MatchResult matchResult = new MatchResult(sortedQueryPattern);
		Integer matchedIndex = 0;
		Matcher m=new Matcher();
		m.SourceFileName=libpath;
		m.match(matchResult, matchedIndex);
	}
}
```

附录 E 类元的事件处理程序

```
1、  Function  StartElement(String uri, String localName, String qName,
2、  Attributes attributes)
3、     if (qName.equals("UML:Interface"))then
4、        %当前的访问节点是否为类的节点若是则对其进行处理并访问下一
5、        个节点%
6、         ParentInterface = CurrentInterface;
7、        CurrentInterface = new MInterface(…);
8、         %节点创建成功，将其加入接口的列表%
9、             if (CurrentInterface.getid()!=NULL &&
10、                 CurrentInterface.getname()!=NULL) then
11、              InterfaceList.add(CurrentInterface);
12、             Endif
13、    Endif
14、 Endfunction
```

```
1、  Function EndElement(String uri, String localName, String qName)
2、     if (qName.equals("UML:Interface")) then
3、         %若当前节点是接口的处理%
4、         CurrentInterface = ParentInterface;
5、         if (ParentInterface == NULL) then
6、             ParentInterface = NULL;
7、         else
8、             %将父节点设为父节点的父节点 %
9、            ParentInterface=ParentInterface.GetInterfaceparent();
10、        Endif
11、    Endif
12、 Endfunction
```

附录 F 类间关系的事件处理程序

```
1、  Function startElement(String uri, String localName, String qName,
2、  Attributes attributes)
3、      if (qName.equals("UML:Dependency")) then
4、          %进入 Dependency 节点 %
5、          CurrentDependency = new MDependency(id, name);
6、          Dependencyflag = true;
7、      else if (qName.equals("UML:Dependency.client")) then
8、                  %进入依赖关系节点的一端 %
9、                 if (Dependencyflag==true) then
10、                    DependencyClientflag = true;
11、                Endif
12、           else if (qName.equals("UML:Dependency.supplier")) then
13、                   %进入依赖关系节点的另一端 %
14、                   if (Dependencyflag==true)then
15、                      DependencySupplierflag = true;
16、                   Endif
17、                else if (qName.equals("UML:Class") ||
18、                         qName.equals("UML:Interface")) then
19、                        if (DependencyClientflag==true) then
20、                        %设置依赖的节点 ID%
21、                        CurrentDependency.setClient(xmi.idref);
22、                        Endif
23、                        if (DependencySupplierflag==true) then
24、                       %设置被依赖的节点 ID%
25、                       CurrentDependency.setSupplier(xmi.idref);
26、                         Endif
27、                    Endif
28、                Endif
29、            Endif
30、     Endif
31、 Endfunction
```

```
1、  Function endElement(String uri, String localName, String qName)
2、      if (qName.equals("UML:Dependency")) then
3、          %退出 Dependency 节点%
4、          if (CurrentDependency.getClient() != null
5、              && CurrentDependency.getSupplier() != null) then
6、              %如果节点合法则加入依赖节点列表%
7、              Dependencylist.add(CurrentDependency);
8、          Endif
9、          CurrentDependency = null;
```

```
10、        Dependencyflag = false;
11、    else if (qName.equals("UML:Dependency.client")) then
12、            %退出 Dependency.client 节点%
13、            DependencyClientflag = false;
14、        else if (qName.equals("UML:Dependency.supplier"))then
15、                %退出 Dependency.supplier 节点%
16、                DependencySupplierflag = false;
17、            Endif
18、        Endif
19、    Endif
20、Endfunction
```

附录 G 模型匹配的代码实现

match.java

```
public static void match(MatchResult matchResult, Integer matchedIndex) {
            if(matchflag==true) return;
          if (matchedIndex >= matchResult.queryPattern.size() - 1) {
               end=System.currentTimeMillis();
               System.err.println("cost time:"+(end-start)+"\n");
               matchflag=true;
               System.out.println(matchResult);
               return;   }
          if (matchResult.queryPattern.get(matchedIndex) instanceof MClassifier) {
               MClassifier classifierPattern = (MClassifier) matchResult.queryPattern
                         .get(matchedIndex);
               if (classifierPattern.getId().startsWith("$$")) {
                    String[] classifierIDS = ClassifierMatch
                              .find(classifierPattern);
                    for (int i = 0; i < classifierIDS.length; i++) {
                         String oldID = classifierPattern.getId();
                         String newID = classifierIDS[i];
                         if (matchResult.addMapping(oldID, newID)) {
                              match(matchResult, matchedIndex + 1);
                              matchResult.removeMapping(oldID, newID);}
                    }
               } else { String oldID = classifierPattern.getId();
                    String[] classifierIDS = ClassifierMatch
                              .find(classifierPattern);
                    for (int i = 0; i < classifierIDS.length; i++) {
                         if (classifierIDS[i].trim().equals(oldID.trim()))
                              match(matchResult, matchedIndex + 1); }
                    classifierPattern.setId(oldID); }
          }    else    if    (matchResult.queryPattern.get(matchedIndex)    instanceof
MGeneralization) {
               MGeneralization    generalizationPattern    =    (MGeneralization)
matchResult.queryPattern
                         .get(matchedIndex);
               if ((!generalizationPattern.getChildxmiidref().startsWith("$$"))
                         && (!generalizationPattern.getParentxmiidref().startsWith(
                                   "$$"))) {
                    MGeneralization pattern = new MGeneralization(null);
                    pattern.setParentxmiidref(generalizationPattern
                              .getParentxmiidref());
```

```
            pattern.setChildxmiidref(generalizationPattern
                    .getChildxmiidref());
            MGeneralization[] generalizations = GeneralizationMatch
                    .findMGeneralizations(pattern);
            if (generalizations.length > 0) {
                match(matchResult, matchedIndex + 1);
            } else {return; }
        } else if (generalizationPattern.getId().startsWith("$$")
                && generalizationPattern.getChildxmiidref()
                        .startsWith("$$")) {
                    MGeneralization pattern = new MGeneralization(null);
            pattern.setParentxmiidref(generalizationPattern
                    .getParentxmiidref());
            MGeneralization[] generalizations = GeneralizationMatch
                    .findMGeneralizations(pattern);
            for (int i = 0; i < generalizations.length; i++) {
                String oldID = generalizationPattern.getId();
                String newID = generalizations[i].getId();
                String oldChildID = generalizationPattern
                        .getChildxmiidref();
                String newChildID = generalizations[i].getChildxmiidref();
                if (matchResult.addMapping(oldID, newID)) {
                    if (matchResult.addMapping(oldChildID, newChildID)) {
                        match(matchResult, matchedIndex + 1);
                        matchResult.removeMapping(oldChildID, newChildID);
                    }
                    matchResult.removeMapping(oldID, newID);}
            }
        } else if (generalizationPattern.getId().startsWith("$$")
                && generalizationPattern.getParentxmiidref().startsWith(
                        "$$")) {
            MGeneralization pattern = new MGeneralization(null);
            pattern.setChildxmiidref(generalizationPattern
                    .getChildxmiidref());
            MGeneralization[] generalizations = GeneralizationMatch
                    .findMGeneralizations(pattern);
            for (int i = 0; i < generalizations.length; i++) {
                String oldID = generalizationPattern.getId();
                String newID = generalizations[i].getId();
                String oldParentID = generalizationPattern
                        .getParentxmiidref();
                String newParentID = generalizations[i].getParentxmiidref();
                if (matchResult.addMapping(oldID, newID)) {
                    if (matchResult.addMapping(oldParentID, newParentID)) {
```

```
                                match(matchResult, matchedIndex + 1);
                                matchResult.removeMapping(oldParentID,
newParentID);
                            }
                            matchResult.removeMapping(oldID, newID);
                        }
                    }
                } else {
                    System.err.println("Generalization Match Error!");
                }
            } else if (matchResult.queryPattern.get(matchedIndex) instanceof
MDependency) {
                MDependency dependencyPattern = (MDependency)
matchResult.queryPattern
                        .get(matchedIndex);
                if ((!dependencyPattern.getClient().startsWith("$$"))
                        && (!dependencyPattern.getSupplier().startsWith("$$"))) {
                    MDependency pattern = new MDependency("", null);
                    pattern.setSupplier(dependencyPattern.getSupplier());
                    pattern.setClient(dependencyPattern.getClient());
                    MDependency[] dependencies = DependencyMatch
                            .findMDependencies(pattern);
                    if (dependencies.length > 0) {
                        match(matchResult, matchedIndex + 1);
                    } else {return;}
                } else if (dependencyPattern.getId().startsWith("$$")
                        && dependencyPattern.getClient().startsWith("$$")) {
                    MDependency pattern = new MDependency("", null);
                    pattern.setSupplier(dependencyPattern.getSupplier());
                    MDependency[] dependencies = DependencyMatch
                            .findMDependencies(pattern);
                    for (int i = 0; i < dependencies.length; i++) {
                        String oldID = dependencyPattern.getId();
                        String newID = dependencies[i].getId();
                        String oldClientID = dependencyPattern.getClient();
                        String newClientID = dependencies[i].getClient();
                        if (matchResult.addMapping(oldID, newID)) {
                            if (matchResult.addMapping(oldClientID, newClientID)) {
                                match(matchResult, matchedIndex + 1);
                                matchResult.removeMapping(oldClientID,
newClientID);}
                            matchResult.removeMapping(oldID, newID);
                        }
                    }
```

```
            } else if (dependencyPattern.getId().startsWith("$$")
                    && dependencyPattern.getSupplier().startsWith("$$")) {
                MDependency pattern = new MDependency("", null);
                pattern.setClient(dependencyPattern.getClient());
                MDependency[] dependencies = DependencyMatch
                        .findMDependencies(pattern);
                for (int i = 0; i < dependencies.length; i++) {
                    String oldID = dependencyPattern.getId();
                    String newID = dependencies[i].getId();
                    String oldSupplierID = dependencyPattern.getSupplier();
                    String newSupplierID = dependencies[i].getSupplier();
                    if (matchResult.addMapping(oldID, newID)) {
                        if (matchResult
                                .addMapping(oldSupplierID, newSupplierID)) {
                            match(matchResult, matchedIndex + 1);
                            matchResult.removeMapping(oldSupplierID,
                                    newSupplierID);
                        }
                        matchResult.removeMapping(oldID, newID);}
                }
            } else {System.err.println("Dependence Match Error!");}
        } else if (matchResult.queryPattern.get(matchedIndex) instanceof
MAbstraction) {
            MAbstraction abstractionPattern = (MAbstraction)
matchResult.queryPattern
                    .get(matchedIndex);
            if ((!abstractionPattern.getClient().startsWith("$$"))
                    && (!abstractionPattern.getSupplier().startsWith("$$"))) {
                MAbstraction pattern = new MAbstraction("", null);
                pattern.setSupplier(abstractionPattern.getSupplier());
                pattern.setClient(abstractionPattern.getClient());
                MAbstraction[] abstractions = AbstractionMatch
                        .findMAbstractions(pattern);
                if (abstractions.length > 0) {
                    match(matchResult, matchedIndex + 1);
                } else {
                    return;
                }
            } else if (abstractionPattern.getId().startsWith("$$")
                    && abstractionPattern.getClient().startsWith("$$")) {
                MAbstraction pattern = new MAbstraction("", null);
                pattern.setSupplier(abstractionPattern.getSupplier());
                MAbstraction[] abstractions = AbstractionMatch
                        .findMAbstractions(pattern);
```

```
                    for (int i = 0; i < abstractions.length; i++) {
                        String oldID = abstractionPattern.getId();
                        String newID = abstractions[i].getId();
                        String oldClientID = abstractionPattern.getClient();
                        String newClientID = abstractions[i].getClient();
                        if (matchResult.addMapping(oldID, newID)) {
                            if (matchResult.addMapping(oldClientID, newClientID)) {
                                match(matchResult, matchedIndex + 1);
                                matchResult.removeMapping(oldClientID,
newClientID);
                            }
                            matchResult.removeMapping(oldID, newID);
                        }
                    }
                } else if (abstractionPattern.getId().startsWith("$$")
                        && abstractionPattern.getSupplier().startsWith("$$")) {
                    MAbstraction pattern = new MAbstraction("", null);
                    pattern.setClient(abstractionPattern.getClient());
                    MAbstraction[] abstractions = AbstractionMatch
                            .findMAbstractions(pattern);
                    for (int i = 0; i < abstractions.length; i++) {
                        String oldID = abstractionPattern.getId();
                        String newID = abstractions[i].getId();
                        String oldSupplierID = abstractionPattern.getSupplier();
                        String newSupplierID = abstractions[i].getSupplier();
                        if (matchResult.addMapping(oldID, newID)) {
                            if (matchResult
                                    .addMapping(oldSupplierID, newSupplierID)) {
                                match(matchResult, matchedIndex + 1);
                                matchResult.removeMapping(oldSupplierID,
                                        newSupplierID);
                            }
                            matchResult.removeMapping(oldID, newID);
                        }
                    }
                } else {System.err.println("Dependence Match Error!");}
            }
        }
```