摘 要

传统的构件描述与检索方式，由于缺乏构件的语义信息描述，用户难以精确检索到与需求匹配的构件资源，所以不能很好地实现资源共享和重用的目的。

本文针对传统的构件和构件库存在的问题引入本体的概念。引入本体概念后对构件知识进行分析，建模，使其明确化，在人、软件代理之间达成对构件知识的共同理解。

要引入本体首先要明确构件概念的范围，本文确定以水利领域构件为概念的范围。首先是针对水利这单个领域，构件数量不会太大，术语也不必考虑扩展问题；其次，单个领域构件的理解和抽象相对来说要简单容易得多。本文主要内容：

（1）参照 REBOOT、3C、青鸟构件库等模型，给出水利领域构件的刻面分类模型。

（2）以给出的水利领域构件的刻面分类模型为基础，参照七步法方法构建水利领域构件的本体，然后对构建的本体进行简单的优化、验证和评价。

（3）简要研究了常用的本体的复用方法，即简单的把整个源本体导入目标本体的 Import 复用和本体模块复用方法。

（4）构件的检索是构件复用的重要组成部分，运用 Jena 对本文构建的本体进行简单的构件检索，并对检索结果进行了简要的查全率和查准率分析。

本文所构建的水利领域构件本体能解决水利领域构件的异构构件库查询问题和语义查询问题，能提高构件的查全率和查准率。

关键字：领域构件，构件本体，本体构件库模型，构件刻面分类模型，本体复用

Abstract

In the traditional components description and retrieval, due to the lack of semantic information describing, users are difficult to accurately match the demand component resources from the component library, so it is not good to realize resource sharing and reuse purposes.

This paper, introduce the concept of ontology aim at the traditional components and component library problems above. With ontology, we could analysis, modeling the component and make it formal, we could also reach a common understanding of component between person and software agency.

The introduction of Ontology must firstly clear the scope of the component concept, here identified areas of water resources domain component. The first reason for this single water resources domain area is that the number of components will not be too great and there is no need to consider extending the term problem; Secondly, the components in the individual field is easy to abstract and understand. The main content is:

(1)Reference REBOOT, 3C, Jade Bird Component Library and other models, gives the component facets classification model of water resources domain.

(2) In the light of the seven-step method of constructing ontology to construct the water resources domain ontology, reference to the component facets classification model , carved face and term, then against the ontology for simple optimization, verification and evaluation.

(3) Research on the ontology reuse; for example, add all the source ontology into goal ontology by using the sentence "import"; divide source ontology into a lot of modules, and the goal ontology only use the needed modules.

(4) Components retrieve is the important part in the component reuse, we use HP's Jena tool to retrieve components from the ontology which constructed in this paper, after then simply analyses the results by recall rate and precision rate.

Water resources domain component ontology talked in this paper can solve the problem of heterogeneous query and semantic query, it can also improve the components' recall rate and precision rate in component retrieve.

Keywords: domain component, component ontology, ontology component library model, component facets classification model, ontology reuse

学位论文独创性声明：

本人所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同事对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。如不实，本人负全部责任。

论文作者（签名）：陈胜玉 2008 年 6 月 16 日

（注：手写亲笔签名）

学位论文使用授权说明

河海大学、中国科学技术信息研究所、国家图书馆、中国学术期刊（光盘版）电子杂志社有权保留本人所送交学位论文的复印件或电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅。论文全部或部分内容的公布(包括刊登)授权河海大学研究生院办理。

论文作者（签名）：陈胜玉 2008年 6 月 16 日

（注：手写亲笔签名）

第一章 绪论

1.1 研究背景及意义

1.1.1 研究背景

软件复用是在软件开发中避免重复劳动的解决方案，其出发点是应用系统的开发不再采用一切“从零开始”的模式，而是以已有的工作为基础，充分利用过去应用系统开发中积累的知识和经验，如：需求分析结果、设计方案、源代码、测试计划及测试案例等，从而将开发的重点集中于应用的特有构成成分。

通过软件复用，在应用系统开发中可以充分地利用已有的开发成果，消除了包括分析、设计、编码、测试等在内的许多重复劳动，从而提高了软件开发的效率，同时，通过复用高质量的已有开发成果，避免了重新开发可能引入的错误，从而提高了软件的质量[1]。

20 世纪 80 年代以来，软件重用已经成为现代软件工程的一个重要目标，面向对象技术的发展与应用，在提高软件可重用性方面起了积极的推动作用。面向对象技术把现实世界中的事物抽象为“对象”，把数据以及相关的方法(对数据的操作)封装在一起，使用继承的特性并利用已有对象的功能来构造新的对象，有效地提高了软件的开发效率。但是，面向对象的软件开发设计并没有发挥出它应有的最大力量。首先，它支持的软件重用是源代码级别的，某一面向对象语言的类只能用于同一语言中；其次，在开发大型软件系统时，它的粒度粗细难以有效控制；另外，不同软件开发商所提供的对象(指语言的类)不易交互合作，难以支持系统的集成。因此，从理论上讲，面向对象技术是应能支持软件的重用和集成，但在实际上，面向对象技术只是可以作为一种基础。

软件构件技术是从面向对象技术发展而来的，通过软件构件达到全面应用面向对象技术与概念，成为开发出高效、低成本、可重用软件系统的重要的现实途径。当今软件开发技术的主流已是基于软件构件技术[2]。

目前，我国水利领域已经建设了大量针对特定水利业务需求的软件构件，如水文预报方面、防汛指挥方面、水资源决策支持方面、水土保持决策支持方面的构件等。这些不同的水利业务之间存在着密切联系，需要相互协作与配合。但是，由于管理体制和信息技术水平的限制，这些系统基本上都是独立设计、单独开发，各个单位采用的构件库和和构件描述不同，从而不能实现构件的共享和利用，即存在构件语义描述和查询问题、构件的异构查询问题。因此，迫切需要利用现有

的技术将不同的水利构件集成，实现构件共享。

1.1.2 研究意义

随着软件构件技术的发展，构件应用的领域越来越广，构件的数量越来越大，由于缺乏对构件的准确描述和分类，缺乏构件的语义信息描述，用户难以精确检索到与需求匹配的构件资源，所以不能很好地实现资源共享和复用的目的。

针对上述问题，本文在构件的基础上引入了本体的概念，由于构件的种类和应用领域繁多，不同的领域有着不同的领域概念和术语，在单个领域中构件数量不会太大，对软件构件的理解相对来说要简单容易得多而且构件的描述也不需要考虑跨领域的问题，本文选择构建水利领域构件本体。

（1）本体能够在语义层次上描述构件，为自动搜索打下基础。现在对构件库中大量构件进行搜索查找，主要通过字符串匹配的方式，而利用本体，就可以消除自然语言理解中的歧义，明确概念涵义并根据相关概念进行推理，挖掘隐含信息。利用本体中概念和概念约束的明确规范说明，可以帮助系统在多个可能的意义中选择最适合的意义，从而实现在语义的层次上进行搜索匹配，提高了查全率和查准率。

（2）本体可以起到“理解构件含义”的桥梁的作用。因为需求不同，上下文环境不同，对于一个本质上相同的问题，不同人会有各种各样不同的观点和假设。他们各自使用不同的术语；他们各自所说的概念、结构、方法或者互相重叠，或者互不匹配。因而，对于相同的构件，不同的设计者可能采用不同描述格式，即不同的描述模型。同样，不同的使用者，对同一个构件也会得到不同的认识，产生不同的使用模型。本体能够在构件的创建者和使用者之间，架起一座“理解构件含义”的桥梁[14]。

（3）软件复用的成功与否在很大程度上取决于构件的表示与检索，研究合理高效的构件表示方法和检索机制对促进软件复用技术的发展具有重要的意义。

（4）引入了本体的概念后，对构件的分类和管理将起到很好的促进作用。

（5）结合本体的描述，讨论有效的领域本体建立方法，为其他领域构件的描述提供参考。

（6）基于水利领域构件的本体语义描述，可以共享水利领域知识，从而减少了用户的不同理解和表达差异。

1.2 研究现状

1.2.1 软件构件模型

构件模型是对构件本质特征的抽象描述。构件模型规定了构件接口的结构以及构件与软件构架、构件与构件之间的交互机制。构件模型通常还提供创建和实现构件的指导原则。一个被所有构件生产者和构件复用者所接受的构件模型实际上起到了构件标准化的作用。目前国际上已经出现了许多构件模型，有学术界的抽象程度较高的 REBOOT 模型、3C 模型，也有应用于工程实践的实现模型，其中最有影响的是 CORBA、COM 和 EJB，而国内最著名的就是青鸟构件库模型。

（1）REBOOT 模型

REBOOT 模型是在 REBOOT 项目中总结提出的，它实质上是一个刻面分类模型。该模型认为：可以用有限维信息空间的术语组合从若干个刻面的综合角度来刻画一个构件。REBOOT 项目中提出的刻面有以下 4 个：依赖(dependencies)、抽象(abstraction)、操作(operations)、操作对象(operation)。另外该模型还提出了用同义词典(thesaurus)来提供术语间语义关系的描述手段。

（2）3C 模型

3C 模型的命名主要来自该模型描述构件所采用的 3 个 C 特征，即概念(Concept)，内容(Content)和语境(Context)。概念是关于“构件做什么”的抽象描述，可以通过概念去理解构件的功能。概念包括接口规约和语义描述两个部分，语义描述和每个操作相关联，表示操作完成的功能，也包括对操作的约束条件。内容概念的具体实现，描述构件如何完成概念所刻划的功能，如算法、结构等等，它是概念的细化描述；描述构件和外围环境在概念级和内容级的关系。语境刻划构件的应用环境，为构件的选用和适应性修改提供指导。3C 模型给出了构件模型必须包含的一些内容，其本身并不适合应用于工程实践，而且也缺乏具体易懂的描述，因此仅仅具有一定的宏观指导意义[12]。

（3）CORBA 模型

CORBA 是 OMG(Object Management Group)设计的一个标准。CORBA 是 OMG 在对象管理结构(OMA)的基础上。以对象请求代理(ORB)为核心制定的分布对象标准，定义了对象之间通过 ORB 透明的发送请求和接受响应的机制。它允许开发人员设计在分布异构型网络环境中可互操作的软件对象。它通过定义了一个通用的接口定义语言(Interface Definition Language—IDL)从而允许跨网络的对象交换和方法调用。CORBA 主要的目标是实现重用性、移植性和分布的、异构的环境中的基于对象软件的互操作性。这些规格的一致性将可能开发出一种异构的、跨所有主要硬件平台和操作系统的应用环境。

（4）COM 模型

COM 是由 Microsoft 公司推出的构件接口标准，COM 构件是一种二进制标准的构件技术。在空间方面，接口编译以后生成的二进制代码的结构要满足一定的内存结构，它不依赖于任何语言，正是通过这种统一的二进制代码结构，不同语言之间的互操作和代码的共享才能得以实现。在时间方面，COM 在接口与实现之间建立了一种弱的、松散的耦合，在必要时这种弱的耦合能通过接口找到其相应的实现；并且在不影响所有客户使用的前提下可将编译好的二进制代码中的一个接口实现换为另一种接口实现，从而将客户和构件本身真正地分开，有利于以后版本升级和维护。COM 通过 DLL 机制实现了对构件生命期的管理并提供了与其它构件进行交互操作的接口，COM 还提供了无须知道构件中有无特定的接口的服务。COM 可以作为各种高级语言的互操作的中间桥梁：按 COM 标准实现的软件也可以被所有的语言环境所共享，这种机制会给软件开发带来诸多好处。

（5）EJB 模型

SUN 发布的文档中对 EJB 的定义是：EJB 是用于开发和部署多层结构的、分布的、面向对象的、跨平台的 Java 应用系统构件体系结构。EJB 公共草案规范允许在企业 bean 的调用上进行互操作，使得 J2EE 客户端同其它 EJB 容器的 EJB 之间能够通信，这些 J2EE 客户端包括 Java 服务器页面、servlet 和应用客户端。这些功能使部署在不同供应商提供的 J2EE 产品中的 EJB 调用能够正常工作，对不同构件之间互操作的支持包括事务传播、命名服务和安全性服务。这些服务通常使用各种不同产品和技术，实现在不同的平台上。EJB 提供了通用接口，能够容易地访问这些服务。遵从 EJB 规范说明开发的构件可在任何一个支持 EJB 的系统中运行[13]。

（6）青鸟构件库模型

北京大学青鸟工程以三个视角（形态、层次和表示）和九个方面定义构件模型。

三个视角：

1）构件形态(Form)被分为类(Class)、类树(Class Tree)、框架(Framework)、设计模式(Design Pattern)、体系结构(Architecture)五种。

2）构件层次被分为分析件(指系统需求规约和功能规约)、设计件(指系统体系结构和设计方案)、编码件(由具体程序设计语言编制的源代码构件)、测试件(测试计划和测试案例)四个层次。

3）构件的表示则与层次有关，不同层次的构件具有不同的表示媒介和手段，如图形、复合文档、正文、伪码、编程语言、目标码等。

九个方面：

1）概念：对构件功能的抽象描述。

2）操作规约：用来指称构件对外提供的、可被请求的服务。

3）接口：给出了构件的对外行为描述。

4）类型：用于定义“什么值可用作为操作参数”。

5）实现体：这是构件的具体实现部分，是实际完成被请求服务的系统。

6）构件复合：构件通过复合组成系统。

7）构件性质：指明构件的形态、层次和表示。

8）构件注释：描述和构件库相关的其他性质。

9）构件语景：描述构件的软、硬件使用环境和实现依赖。

青鸟构件模型更多地关心构件的易理解性、封装性及间关系，通过给构件提供明确的对外接口实现服务提供者和其服务请求者的分离，更多地关心构件及其使用者间的交互，特别是对构件使用者有意义的部分。

1.2.2 构件分类与检索

构件的表示与检索是构件研究的关键技术。针对不同的构件描述形式，研究人员已提出了许多相应的检索方法。

W.Frakes 从构件表示方法出发，将已发表的构件表示与检索方法分为人工智能（AI）方法、超文本方法和信息科学方法三类。

W.Frakes 重点讨论了应用信息科学的编目方法来实现构件的表示与检索的方法。在信息科学方法中以枚举、刻面、属性值、关键词和正文等分类编目与检索方法最为常用，其中又以关键词(Keyword)分类和刻面(Faceted)分类两种方法应用最为常见。

关键词分类方法原理简单，操作方便，易学易用。构件按一组与之相关联的关键词进行编目，查询者通过给出关键词来查找所需构件。关键词对相应构件的描述不够准确时，缺少上下文语境，容易产生歧义，因而查询效率较低。

刻面分类方法选择能准确描述构件本质特征的一组特定视角，并将这些特定视角称为构件的刻面(Facets)，然后在刻面中定义一组术语（即关键词），并据此对构件进行分类。各个刻面的术语共同构成结构化的术语空间。术语仅限于在给定的刻面之中取值（也就是取自受控的词汇表）。这一做法将关键词（术语）置于特定语境之中，从而提高构件的查询效率。刻面的选择和术语空间的建立主要取决于构件库的领域特征和软件复的需求。如 REBOOT 构件库中定义了抽象(Abstraction)、操作(Operations)、操作对象(Operates On)和依赖(Dependencies)四个刻面。

H.Mili 则按照构件表示的复杂度和检索效果的递增关系将已发表的构件表示与检索方法分为基于文本的、基于词法描述子的和基于规约的三类。

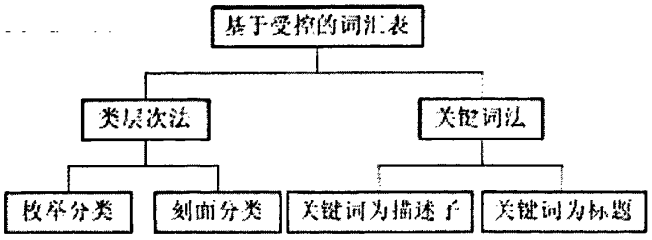

图 1.1 基于受控词汇表的表示方法

基于词法描述子的编码与信息科学编目方法中“基于受控词汇表”的表示方法类似。必须在构件生产与应用的各个环节参与者间建立和维护公共词汇表。在这一方法类中，最具特色是近似匹配算法。近似匹配算法将基于偏序关系的检索扩展为基于距离的检索，从而可以利用带权语义网、模糊逻辑触发器和带权的术语空间。

基于规约的编码和检索该方法基于形式化的构件规约，排除了一般检索可能存在的编码二义性，具有良好的理论背景。构件规约通常能够保证构件被准确找到和正确使用。

1.2.3 本体研究应用

国外本体的研究应用

（1）知识工程

本体的核心概念是知识共享，通过减少概念和术语上的歧义，本体描述统一了领域中的术语和概念，使得来自不同背景，持不同观点和目的的人员之间的理解和交流成为可能，并保持语义上的一致性。

（2）系统之间的互操作或信息系统的集成上的应用

应用程序使用本体论实现异构系统之间的互操作，即不同系统或是工具之间的数据传输。语义 Web 服务就属于此类。

（3）软件工程

在需求分析中，本体论通过对问题和任务的理解描述，提高明确性，减小分析代价同时，本体可进一步作为软件设计的基础，以(半)自动方式检查需求和设计的一致性，提高软件可靠性。本体还可以通过对系统内部各个功能模块和它们之间的联系的详细描述提高软件的重用性。作为系统分析方法，本体论应用于信息建模、面向对象分析和数据库设计等。本体建模过程澄清了领域知识的结构，为信息系统的分析和设计提供基础。

国内关于本体的研究应用

国内本体方面的研究应用整体上处于起步阶段，目前主要集中在知识工程、信息管理和语义 Web 三个方面。本体工程方面的研究比较有名的通用本体构建研

究包括中科院计算技术研究所的大规模知识系统研究和中科院数学研究所的常识知识库研究。信息管理方面讨论相对较多的主要有本体论与信息检索、本体论与数字图书馆、本体论与信息管理，此外还包括知识库系统、数据挖掘、电子商务、机器翻译、需求分析等。国内语义 Web 的方面研究主要集中在服务的发现上。目前国内基于本体论的构件的研究相对较少，注意力相对集中在语义 Web 和 Web 服务上。

1.3 主要研究内容和目标

主要研究内容：

（1）通过对构件、构件库和构件检索技术及相关理论的研究，把本体论的思想引入到构件和构件库的描述中。通过对水利领域构件的分析，提出了一种构件刻面分类模型，抽象出水利领域构件的刻面、各个刻面的术语空间及其相互关系。

（2）以提出的刻面分类模型为基础，参照本体的七步法构建方法，构建出水利领域构件本体，并对本体进行简单的优化、验证和评价。

（3）简要研究常见的本体复用方法。本体是在领域专家的参与下构建的，本体中的类，类的属性得到了大家一致的认可，本体相当于一个标准，把相同意义的概念（概念有不同的表示形式）映射成同一概念，采用统一的处理方法。

（4）运用 Jena 对本文构建的水利领域本体进行简单的构件检索应用，并对检索结果简要地进行了查全率和查准率分析。

研究目标：

（1）构建出水利领域构件的本体，解决水利领域构件的语义查询问题和构件库的异构查询问题。

（2）为了更好地了解本体和构建一个合理的本体，介绍现阶段主要的本体复用方法。

（3）针对本文构建的本体，给出水利领域构件本体的构件检索应用。

1.4 本文的组织结构

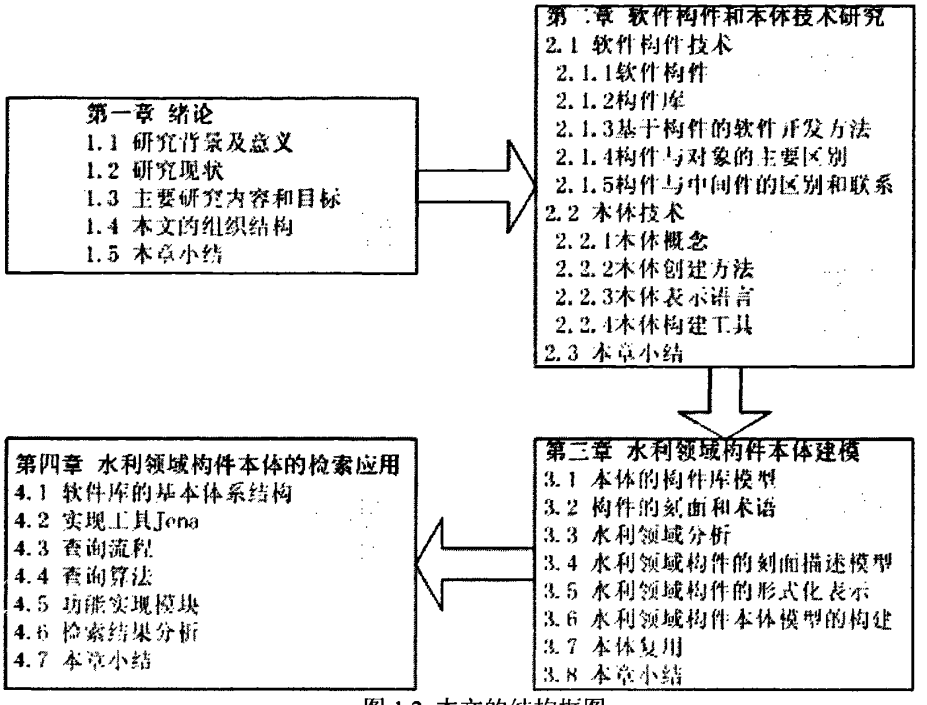

图 1.2 本文的结构框图

本文第二章介绍软件构件和本体技术，主要介绍了什么是软件构件和构件库，软件构件与对象、中间件的区别和联系，本体的概念，本体的创建方法，本体的表示语言，本体的构建工具。第三章提出了本体的构件库模型，参照 REBOOT 模型、3C 模型、青鸟构件库模型、参考论文[17][19][20]，分析水利领域特点，抽取出刻面和术语，参照刻面和术语的合并算法，提出了水利领域构件的刻面分类模型，以该模型为基础构建领域构件本体，并简要研究了本体的复用技术。第四章运用 Jena 对第三章构建的水利领域构件本体给出具体的构件检索应用。

1.5 本章小结

本章主要给出本文的研究背景和意义，针对研究背景中的问题，提出问题的解决方法，在构件中引入本体，用本体来描述构件；给出本文要研究内容的研究现状和本文的研究目标。

第二章 软件构件和本体技术研究

本文要构建的是水利领域的构件本体，首先对什么是构件作简要的介绍；说到构件就离不开构件库，构件库是构件的载体，所以紧接着介绍什么是构件库；介绍完构件、构件库后给出基于构件是怎样进行软件开发的。由于构件和对象、中间件的关系密切，简要介绍构件与对象、中间件的主要区别和联系。

2.1 软件构件技术

2.1.1 软件构件

构件(Component)是一个可独立开发和交付的软件单位，其设计和实现封装在一起，通过接口向外界（应用框架、其它构件或最终用户）提供服务。构件可以是被封装的对象类、类树、一些功能模块、软件框架、软件构架(或体系结构)、文档、分析件、设计模式等。软件构件技术是支持软件复用的核心技术，下面以青鸟构件库为例简要介绍构件。

青鸟构件的登记表格中有下列属性：

（1）名称：每个构件都必须有一个确定的名称，该名称必须完整地标识了该构件的本质。如 stack, resource manager 等。

（2）作者：即制作或提供该构件的单位或个人的名称，以及联络地址等相关信息。

（3）制作日期：即构件制作的完成日期。

（4）入库日期：即构件进入构件库的日期。

（5）版本号：即该构件在一组构件演化系列中相应的版本号。

（6）使用环境：即使用(包括理解/组装/修改)该构件时必须提供的硬件和软件平台。如所需的特定的硬件环境、操作系统、数据库平台和网络环境等。

（7）应用领域：即该构件原来或可能被使用到的应用领域(及其子领域)的名称。如 MIS,CAI 等。

（8）用途：即该构件在被应用的领域中所发挥的作用。

（9）功能：即该构件在原有或可能的软件系统中所提供的软件功能集。

（10）表示方法：即用来描述该构件内容的语言形式或媒体。如源代码构件所用的编程语言等。

（11）形态：即该构件的组成成分及其相互关系。如类、类树、框架、模块等。

（12）层次：即该构件相对于软件开发过程阶段的抽象层次。如分析、设计、编码等。

（13）上下文环境：即该构件在组装时系统所必须提供的程序级上下文环境。

（14）尺寸(size)：即该构件的大小。

（15）创作工具：即构件的制作者在制作该构件时所使用的软件工具。

2.1.1.1 构件的构造原则

一个构件系统不只是为某个软件的开发定制的，而是为多个软件的开发所共享。因此需要软件开发人员一开始就把重用性作为初始设计的一个目标。因此，从系统分析、设计到构件提取、描述、认证、测试、分类和入库，都必须围绕重用这个目的而进行，构件的构造应遵循下述的一些原则：

（1）增强构件的可重用性需要提高抽象的级别，应有一套有关名字，异常操作，结构的标准。

（2）可理解性，必须伴随有完整、正确、易读的文档，具有完整的说明，有利重用。

（3）构件具有抽象性，能提供一些所需的特定操作、属性、事件和方法接口。

（4）提高构件的重用程度，分离功能构件，将可变部分数据化、参数化，以适合不同的应用需求。

（5）构件的尺寸大小、复杂度适中。

（6）构件要易于演化，数据与其结构是封装在一起的，数据存放在数据构件对象中，能主动解释其结构[2]。

2.1.1.2 构件的主要研究内容

（1）构件获取，是从事有目的的构件生产的第一步，对已有系统进行分析和研究，从中挖掘提取有复用价值的构件。

（2）构件模型，研究构件的本质特征和内部组织结构及构件间的关系，指导构件及构件系统的实现，如 3C 模型。

（3）构件描述语言，以构件模型为基础，解决构件的精确描述、理解及组装问题，如构件描述语言、构件组装描述符等。

（4）构件分类与检索，研究构件分类策略、组织模式及检索策略，建立构件库系统，支持构件的有效管理。

（5）构件组装，在构件模型的基础上研究构件的组装机制，包括利用特定组装工具进行的静态组装和基于构件互操作性的运行级组装。

（6）标准化，包括构件模型的标准化和构件库系统的标准化。标准的建立和实施是构件技术得以充分发挥其优势的前提条件[3]。

2.1.2 构件库

构件库是领域工程和应用工程两个开发过程的桥梁。构件库系统当然是一类数据库管理系统，它具备数据库的基本特征和功能，为了向基于构件的应用系统开发提供构件，构件库管理系统必须能够存储构件和构件相关信息：

（1）构件的描述（Description）：描述构件功能和用途。

（2）构件的分类（Classification）：对构件相同特征的聚类。

（3）构件的形态（Form）：构件的类型，广义构件包括类、类树、框架、模块等；构件的技术环境，包括构件的开发工具、配置方法和部署环境等；构件的形式：包括源代码、二级制代码等。

（4）构件的状态（Status）：版本、历史等时间相关构件的属性。

为了能够管理和维护构件信息，构件库管理系统必须能够提供如下的操作：

（5）构件的添加（insertion）：主要是构件的测试和认证方法，只有合格的构件才能添加到构件之中。

（6）构件的检索（Retrieval）：也就构件的需求匹配，与一般的数据库不同，构件匹配通常不能得到完全满意的构件，对与需求相关或相近的构件，需要通过适配使之满足需求；由于构件的粒度不同，需求问题分解和构件合成是两种配合检索的方法。

其他构件库管理手段：构件的删除、备份、用户登记和存取控制、使用跟踪和统计分析、异构构件库的连接等。

2.1.3 基于构件的软件开发方法

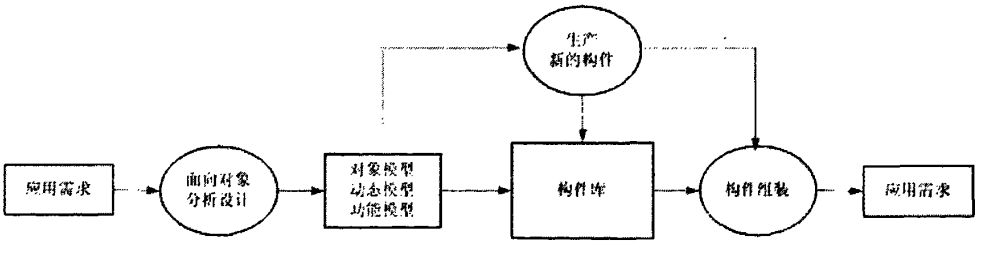

图 2.1 基于构件的软件开发过程

基于构件的软件开发(Component Based Soft ware Development， CBSD)的理论建立在软件工程、软件复用等基础之上，已经成为新一代的软件开发方法。而且 CBSD 方法在 CORBA、EJB 和 DCOM 等构件标准的支持下已经有了相当快

的发展。

在CBSD中，一个应用系统是由一些标准的构件(通用的和专用的)组装而成，这些构件可以通过商业采购、定制或自主开发获得。不像传统的软件开发，基于构件的软件开发重在软件的集成，而不是软件的编程。构件开发常常成为第三方的任务，他们定义了一组预制好的标准构件，以适应某些特殊需求。软件开发人员只需将这些构件搭建起来，构成一个应用系统。

2.1.4 构件与对象的主要区别

（1）纯面向对象设计中，对象(类)、封装和继承三者缺一不可，但对构件可以没有继承性，只要实现封装即可。因此，构件不是面向对象设计中的对象，而是与基于对象设计中的对象类似。

（2）从构件和对象的生成方式上，面向对象设计中的对象生成属于实例化的过程，比较单一，而生成构件的方式较多。

（3）构件是设计的概念，与具体编程语言无关，不像面向对象设计中的对象属于编程中的概念，要依赖于具体的编程语言。

（4）在对构件操作时不允许直接操作构件中的数据，数据真正被封装了。而对象的操作通过公共接口部分，这样数据是可能被访问操作的。

（5）面向对象设计中的对象对软件重用是通过继承实现的，构件对软件重用不仅可以通过继承还可以通过组装时的引用来实现。

（6）从抽象程度来看，面向对象技术已达到了类级重用(代码级)，它以类为封装的单位。但这样的重用粒度还太小，不足以解决异构互操作和效率更高的重用。构件将抽象的程度提到一个更高层次，它是对一组类的组合进行封装，并代表完成一个或多个功能的特定服务，也为用户提供了多个接口。整个构件隐藏了具体的实现，只用接口提供服务。这样，在不同层次上，构件均可将底层的多个逻辑组合成高层次上的粒度更大的新构件，甚至直接封装到一个系统，使模块的重用从代码级、对象级、架构级到系统级都可能实现，从而使软件像硬件一样，能任人装配定制而成。

2.1.5 构件与中间件的区别和联系

（1）中间件是一种独立的系统软件或服务程序，是位于平台（硬件和操作系统）和应用之间的通用服务。中间件在操作系统网络和数据库之上，应用软件之下，其主要功能是帮助用户灵活、高效地开发和集成复杂的应用软件。这些服务具有标准的程序接口和协议。针对不同的操作系统和硬件平台，它们可以有符合接口和协议规范的多种实现。因此，中间件是一类软件，而非一种软件。中间

件不仅仅实现互连，还要实现应用之间的互操作，中间件是基于分布式处理的软件，并特别强调了网络通讯功能。

（2）构件是可复用软件实体，是一个系统的功能单元，能够从一个项目转到另一个项目发挥作用。构件的思想更多地将重点从建模本身发展到对软件生产的考虑，即构件可以在应用领域的软件生产中作为零件纳入新的体系中被重用。因此，构件是面向对象思想的沿袭和扩展，认识事物的角度从对象个体本身上升到个体在群体中的作用。构件技术在最初更多是作为一种思想存在，逐步才在一些关键的环节上发展出解决问题的技术分支。构件的存在某种程度上极大地依赖于构架技术，或环境、基础设施、计算平台，只有在适当的构架中，软件才有可能被抽象和隔离，最终成为构件。因此，单独讨论构件是抽象而空洞的。构架不是操作系统、数据库或网络协议，也不完全是应用，而是在某种特定意义上的构件运行容器，层次上介于应用和基础设施之间。

（3）中间件，本质上是对分布式应用的抽象，它抛开了与应用相关的业务逻辑细节，保留了典型分布交互模式的关键特征。经过抽象，将纷繁复杂的分布式系统经过提炼和必要的隔离后，以统一的层面形式呈现给应用。应用在中间件提供的环境中可以更好地集中于业务逻辑，并以构件化的形式存在，最终自然而然地在异构环境中实现良好的协同工作。构件与中间件是不同的概念。中间件的目的是为了更好的达到基于构件的复用，消除构件对处理平台和开发语言的依赖。

（4）构件和构件之间的联系方式通过接口定义。中间件则是对多层结构中间那个剖面中符合一定规范构件集合的通称，换言之，中间件不是物理上的存在，而是一系列构件的逻辑组合。而中间件本身优劣取决于构成这个中间件剖面的构件功能、组合方式等因素，最终将中间件变成一个较高层次的大构件，也就是说把中间件整个剖面上的构件以及之间的联系看成一个构件，只暴露出若干接口让其他构件来连接[17]。

2.2 本体技术

由于构件的描述缺乏语义信息，可以用多种构件库系统存储构件，构件存在描述差异、表达差异问题和构件库的异构查询问题，用户难以检索或难以精确检索到与需求匹配的构件资源，所以不能很好地实现资源共享和复用的目的。本体能对共享概念进行明确的形式化表示，并且是在语义的层次进行描述，所以本文把本体引入到水利领域构件中，用本体来描述水利领域的构件。下面就对创建水利领域构件本体中将要用到的概念（本体、本体创建方法、本体表示语言和本体构件工具）作相关介绍。

2.2.1 本体概念

本体(Ontology)最早是一个哲学概念，与认识论相对，研究事物客观存在的本质，是对客观存在的系统的阐释和说明，描述的是客观存在的抽象本质。1993年，Gruber给出了Ontology的一个最为人们所熟知和认可的定义，“Ontology是概念模型的明确的规范化的说明”。后来，Borst在此基础上给出了Ontology的另外一种定义“Ontology是共享概念模型的形式化规范说明”。Studer等对上述两个概念进行了深入的研究，认为Ontology是共享概念模型的明确的形式化规范说明。这包含了4个层次的含义：

（1）概念模型(conceptualization)

概念模型是通过抽象出客观世界中的一些现象的概念而得到的模型，其所表现的含义独立于具体的环境状态。

（2）明确(explicit)

“明确”是指所使用的概念和概念约束都有明确的定义。

（3）形式化(formal)

“形式化”是指Ontology是计算机可读的，可被计算机理解和处理的。

（4）共享(share)

“共享”是指Ontology所体现的是共同认可的知识，是相关领域中公认的概念集。

Ontology的目标是捕获相关的领域知识，提供对该领域知识的共同理解，确定该领域内共同认可的词汇，并从不同层次的形式化模式上给出这些词汇(术语)和词汇之间相互关系的明确定义。

本体的建模元语：Perez等人用分类法组织了Ontology，归纳出5个基本的建模元语(Modeling Primitives)

（1）类(classes)或概念(concepts)

类的含义很广泛，指任何事务，如工作描述、功能、行为、策略和推理过程等等。从语义上讲，它表示的是对象的集合，其定义一般采用框架（frame）结构，包括概念的名称，与其它概念之间的关系的集合，以及用自然语言对概念的描述。

（2）关系(relations)

在领域中概念之间的交互作用，形式上定义为 n 维笛卡儿积的子集：R：C1×C2×…×Cn。如子类关系(subclass-of)。在语义上关系对应于对象元组的集合。基本的关系有下表的4种：

表 2.1 本体基本关系的种类

关系名	关系描述
Part-of	表达概念之间部分与整体的关系。
Kind-of	表达概念之间的继承关系，类似于面向对象中的父类与子类之间的关系。给出两个概念 C 和 D，记 C′ ={x | x 是 C 的实例}，D′ ={x | x 是 D 的实例}，如果对任意的 x 属于 D′ ，X 都属于 C′ ，则称 C 为 D 的父概念，D 为 C 的子概念。
Instance-of	表达概念的实例与概念之间的关系，类似于面向对象中的对象和类之间的关系。
Attribute-of	表达某个概念是另一个概念的属性。如概念“颜色”是概念“玫瑰花”的一个属性。

在实际建模过程中，不一定要严格地按照上述 5 类基本建模元语来创建 Ontology，概念之间的关系不限于上面列出的 4 种基本关系，可以根据领域的具体情况定义相应的关系，以满足应用的需要。

（3）函数（functions）

一类特殊的关系。该关系的前 n−1 个元素可以唯一决定第 n 个元素。形式化的定义为 F：C1×C2×…×Cn-1→Cn。如 Mother-of 就是一个函数，Mother-of(x,y)表示 y 是 x 的母亲。

（4）公理（axioms）

代表永真断言，如概念乙属于概念甲的范围。

（5）实例（instances）

代表元素。从语义上讲实例表示的就是对象[16]。

2.2.2 本体创建方法

目前本体工程这个思路虽然已经被大家所接受，但是并没有出现成熟的方法论作为支持。出于对各自学科领域和具体工程的不同考虑，创建本体的过程各不相同。目前尚没有一套标准的本体创建方法。一般认为， Gruber 在 1995 年提出的 5 条规则是比较有影响的：

（1）明确性和客观性：本体应该用自然语言对术语给出明确、客观的语义定义。

（2）完整性：所给出的定义是完整的，能表达特定术语的含义。

（3）一致性：知识推理产生的结论与术语本身的含义不会产生矛盾。

（4）最大单向可扩展性：向本体中添加通用或专用的术语时，通常不需要修改已有的内容。

（5）最少约束：对待建模对象应该尽可能少列出限定约束条件[18]。

2.2.2.1 IDEF-5 方法

IDEF 的概念是在 70 年代提出的，是在结构化分析方法的基础上发展起来的。在 1981 年美国空军公布的 ICAM(integrated computer aided manufacturing) 工程中首次用了名为“IDEF”的方法。IDEF 是 ICAM DEFinition method 的缩写，到目前为止它已经发展成了一个系列。IDEF5 是 KBSI(Knowledge Based Systems Inc.)开发的一套用于描述和获取企业本体的方法。IDEF5 通过使用图表语言和细化说明语言，获取关于客观存在的概念、属性和关系，并将它们形式化成本体。

2.2.2.2 Skeletal Methodolody 骨架法

Mike Ushold & Micheal Gruninger 的骨架法(Skeletal Methodology)，又称 ENTERPRISE 法，专门用来创建企业本体(ENTERPRISE ontology，是有关企业建模过程的本体)。该方法建立在企业本体基础之上，是相关商业企业间术语和定义的集合，只提供开发本体的指导方针。

2.2.2.3 Methodology 方法

Mariano Fernández & GOMEZ-PEREZ 等的 Methodology 方法是由西班牙 Madrid 理工大学 AI 实验室提出的。该方法是在结合了骨架法和 GOMEZ-PEREZ 方法后，提出的一种更为通用的本体建设方法。这个本体开发方法更接近软件工程开发方法。它将本体开发进程和本体生命周期两个方面区别开来，并使用不同的技术予以支持。Methodology 法，专用于创建化学本体，该方法已被马德里大学理工分校人工智能图书馆采用。

2.2.2.4 KACTUS 工程法

KACTUS 工程法(KAC96)是基于 KACTUS 项目而产生的，KACTUS 是指“关于多用途复杂技术系统的知识建模”工程(Modeling Knowledge About Complex Technical systems for multiple Use)，是欧洲 ESPRIT 框架下的研发项目之一，属于 ESPRIT-III 所支持的项目。

2.2.2.5 SENSUS 法

SENSUS 法是开发用于自然语言处理的 SENSUS 语言本体的方法路线，由美国 USC/ISI 研制开发。ISI 自然语言研究小组旨在为机器翻译提供广泛的概念结构。SENSUS 为机器翻译提供概念结构，用该方法开发的 SENSUS 本体系统用于自然语言程序，目前 SENSUS 语言本体共包括电子科学领域的 7 万多个概念。为了能在 SENSUS 基础上构造特定领域的本体，必须把不相关的术语从中剪除。

2.2.2.6 七步法

斯坦福大学医学院开发的七步法，主要用于领域本体的构建。七个步骤分别是：

第一步确定本体的专业领域和范畴

先要明确几个基本问题：

（1）所构建的本体将覆盖哪个专业领域？

（2）应用该本体的目的?是为了更好地挖掘本领域的深层信息。

（3）本体中的信息能回答哪些类型的问题?

（4）该本体的用户与系统维护者是哪些人?

这些问题的答案随着本体设计过程的深入可以随时调整，但是在任何特定的时间段里，他们对于限制模型的范畴都是有帮助的，所以需要相对稳定。

（5）确定领域本体可以“回答”的专业问题，即系统能力问题(Competency Questions)。

第二步考查复用现有本体的可能性

如果自己的系统需要和其它的应用平台进行互操作，而这个应用平台又与特定的本体或受控词表结合在一起，那么复用现有的本体就是最行之有效的方法。许多本体都有电子版本，而且可以输入到个人使用的本体开发系统中。即便一种知识表达系统不能直接以某种特殊的格式来工作，将本体由一种格式转换为另一种格式并不困难。在 Web 上可以找到很多现成的本体文库。

第三步列出本体中的重要术语

应该列出一份所有术语的清单，清单中的术语是需要解释给用户的。首先，需要一份最全的术语清单，此时暂不考虑概念间会有属性及表达上的重复。接下来的两个重要步骤是完善等级体系和定义概念属性(slots)，这两个步骤是密不可分、互相交织的，二者必须同时进行。这两个步骤在本体的设计进程中最为重要。

第四步定义类(Class)和类的等级体系(Hierarchy)

完善一个等级体系有几种可行的方法：

（1）自顶向下法：由某一领域中最大的概念开始，而后再将这些概念细化。

（2）自底向上法：由底层最小类的定义开始，它们是这个等级体系的细枝末节，然后将这些细化的类组织在更加综合的概念之下。

（3）综合法：综合上两种方法。首先定义大量重要的概念，然后分别将它们进行恰当地归纳和演绎。然后将它们与一些中级概念关联起来。

每位研究者要采取什么方法主要依赖于个人对这一专业领域的理解程度和观点。如果开发人员对某一专业领域具备一套自上而下的系统认识论，那么利用自顶向下的方法就会事半功倍。由于“中层概念”在领域的概念中应该更具代表性，所以综合法对许多本体的开发者而言最便捷。如果想要收集到更多更广泛的实例，那么自底向上的方法更加适合。最终，无论选择哪种方法，都要从“类”的定义开始。

第五步定义类的属性

只有类的体系根本不足以提供系统能力问题所需的答案信息。一旦定义好了一些类，就必须开始描绘概念间的内在结构。

首选从来自第三步的术语列表中选择好类。绝大多数剩下的术语可能是这些类的属性(properties)。通常，有几种对象属性的类型能够成为一个本体中的属性：

（1）“内在”属性("intrinsic" properties)，例如某种花卉的颜色。

（2）“外在”属性("extrinsic" properties)，例如某种花卉的产地。

（3）如果对象是结构化的，那么它的一部分，可以是既具体又抽象的元素。

（4）与其它个体的关系。此处的“关系”是指某个类中的个体成员与其它类之间的关系。任意一个类的所有下位类都会继承其上位类(父类)的属性。

第六步定义属性的分面(Facets)

一个属性可能由多个“分面”组成。一个属性的“分面”，就是属性取值的类型(Value Type)、容许的取值(Allowed Values)、取值个数(Cardinality 集的势，基数)和有关属性取值的其它特征。

第七步创建实例

定义某个类的下属实例需要：

（1）确定一个类，

（2）创建该类的一个实例，以及

（3）添加这个类的属性值。

2.2.3 本体表示语言

本体语言为用户编写领域模型提供了清晰的、形式化的概念描述，因此它应该满足以下要求：

（1）良好定义的语法(a well-defined syntax)

（2）良好定义的语义(a well-defined semantics)

（3）有效的推理支持(efficient reasoning support)

（4）充分的表达能力(sufficient expressive power)

（5）表达的方便性(convenience of expression)

本体语言不仅要有描述能力，同时也应具备推理能力，因此它一般都是基于某种逻辑语言的，目前开发的本体语言主要是基于一阶逻辑和掐述逻辑的。虽然高阶逻辑是所有已知逻辑中表达能力最强的，但是它没有好的计算性，虽存在真命题，但不可证明。因此，一般情况下如果不需要高阶语义，二阶逻辑可以转换为一阶逻辑。

本体表示语言可以简单地如下分类：

（1）基于 AI(Artificial Intelligence,人工智能)的本体描述语言：如 KIF，Ontoligua，CycL，Loom，Flogic。

（2）基于 Web 的本体描述语言:如 SHOE，XOL，RDF，RDFS，OIL，DAML+OIL，OWL。

下面重点介绍 XML、RDF/RDFS、DAML+OIL、OWL

2.2.3.1 XML

XML 是严格符合 SGML 的结构化语言，其实现了文档的显示和数据分离，这种结构化的数据易于使用、携带和传递，是 Web 数据交换的较好的语法格式。XML 提供 DTD、XML Schema 对文档结构进行有效性验证，通过描述/约束文档逻辑结构实现数据的语义。XML 对本体的描述，就是利用 DTD 或 XML Schema 对本体所表达的领域知识进行结构化定义，然后再利用 XML 文档结构与 XML 内容之间的关系对本体知识进行描述，从而提供对数据内容的语义描述。

但是 DTD 自身描述能力、数据类型的支持、约束定义的能力是有限的，无法对 XML 实例文档做出更细致的语义限制。因此，通过 DTD 表示的本体，无法表达概念间的继承关系，XML Schema 虽然解决了 DTD 存在的问题，例如定义了更为丰富的语法结构、可以定义元素类型、提供了包含和继承机制等，但是 DTD、XML Schema 为 XML 文档提供的约束机制只是用限定 XML 文档所用到的标记和这些标记之间的结构关系，通过 DTD 和 XML Schema 可以解决对数据的词汇和用途的说明，其语义仍然是隐含的。因此，XML 所表示的本体是轻量级的本体，只能保证人们是用相同的词汇，是一种较低层次的本体的应用，本体中不包含语义信息。

2.2.3.2 RDF 和 RDFS

W3C 的资源描述框架(Resource Description Framework, RDF)为基于元数据的语义表示提供了基础，RDF 为在 Web 上应用系统间进行机器可理解的信息交换提供了互操作能力。

RDF 对资源描述基于如下思想：利用当前现有的 Web 体系结构中的标识符 URIs 作为标识符系统来标识事物，用简单的属性(Property)以及属性值(Value) 来声明资源(Resource)，这里的资源指 Web 上任何可以被标识的事物，可以创建 URIs 来引用声明中需要被标识的任何资源，例如，一份电子文档、一个图片、一个声音文件等网络可访问资源；或者如人、公司、图书馆中的图书等非网络可访问资源；或者如“作者”、“主席”这样非物理存在的抽象概念。

RDF 的基本构造为陈述(或者叫做声明，statement)了一个资源-资源具有的属性-属性值(主体-属性-客体)的三元组。它表现的是一个数据模型，通俗的说一个陈述就是一个什么事物(资源)具有什么属性(属性值)，这个属性是怎样的属性(属性值)。为数据模型提供了简单的语义，这些数据模型能够用 XML 语法进行表达。

但是，RDF 只是提供了一个用于领域无关的机制来描述元数据，描述资源属性及其相关关系，没有提供按照类的机制描述信息资源、声明属性、描述属性语义及其与资源之间的关系。也就是说 RDF 不能描述领域相关的语义关系，如同义词、一词多义等，因此，提出了 RDFS。

RDFS 是 RDF 的扩展，在 RDF 基础上增加了许多语义原语。提供了一种机制来定义相关领域的资源的属性、类型及其关系，用来更进一步增加对资源的描述能力。如，核心类 rdfs: Resource、rdfs: Property、rdfs: Class；核心特性 rdf: Type、rdfs: subClassOf、rdfs: subPropertyOf；核心约束 rdfs: ConstraintResource、rdfs: ConstraintProperty、rdfs: domain、rdfs: range。

RDFS 虽然提供了简单的机器可理解语义模型，解决了 RDF 中存在的一些问题，如属性、概念间的继承关系，但是对语义描述的深度仍然不够，只是进一步提高了计算机处理的自动化程度，还无法表达概念间的合取、析取、不相关等关系。在 RDFS 建模的基础上，针对 RDFS 在语义方面表示的不足，一些研究团体提出了其他的本体表示语言 XOL、OIL、DAML，在此基础上，W3C 发布了 Web 标准本体表示语言 OWL，在机器间实现本体的共享和重用，实现对 Web 信息的智能化处理。

2.2.3.3 OWL

OWL 全称 Web Ontology Language，是 W3C 推荐的语义互联网中本体描述语言的标准。它是从欧美一些研究机构的一种结合性的描述语言 DAML+OIL 发

展起来的，其中 DAML 来自美国的提案 DAML-ONT，OIL 来自欧洲的一种本体描述语言。在 W3C 提出的本体语言栈中，OWL 处于最上层。

OWL 能够被用于清晰地表达词汇表中的词条(term)的含义以及这些词条之间的关系。而这种对词条和它们之间的关系的表达就称作 Ontology。OWL 相对 XML、RDF 和 RDFSchema 拥有更多的机制来表达语义，从而 OWL 超越了 XML、RDF 和 RDFSchema 仅仅能够表达网上机器可读的文档内容的能力。和 XML Schema 相比，OWL 语言是知识表示，不是信息表示格式；和 RDFS 相比，OWL 不仅可以用更复杂的方法描述类，如 disjoint，而且扩展了 RDFS 属性，允许表示属性的 transitive、symmetric 以及 functional 性质，表达了更强的概念语义信息，支持描述逻辑推理。OWL 语言提供了三种表达能力不同的子语言 OWL Lite、OWL DL、OWL Full，分别满足不同的需要[15]。

表 2.2 OWL 的三个子语言描述

子语言	描述	例子
OWL Lite	用于提供给那些只需要一个分类层次和简单的属性约束的用户。	支持基数（cardinality），只允许基数为 0 或 1。
OWL-DL	支持那些需要在推理系统上进行最大程度表达的用户，这里的推理系统能够保证计算完全性(computational completeness，即所有地结论都能够保证被计算出来)和可决定性(decidability，即所有的计算都在有限的时间内完成)。它包括了 OWL 语言的所有约束，但是可以被仅仅置于特定的约束下。	当一个类可以是多个类的一个子类时，它被约束不能是另外一个类的实例。
OWL Full	支持那些需要在没有计算保证的语法自由的 RDF 上进行最大程度表达的用户。它允许在一个 Ontology 在预定义的 (RDF、OWL)词汇表上增加词汇，从而任何推理软件均不能支持 OWL FULL 的所有 feature。	一个类可以被同时表达为许多个体的一个集合以及这个集合中的一个个体。

2.2.4 本体构建工具

经过近 10 年的发展，本体编辑工具已经比较成熟，目前存在 Ontolingua、Webonto 等数种有影响力的本体编辑工具。在这些工具中，被使用最广泛、最受关注的是斯坦福大学医学情报研究组开发的本体编辑工具——Protégé

(http://protege.stanford.edu)。该工具采用 Java 编写，可以免费下载，其界面与普通的 Windows 应用程序风格一致，由于其开放性和兼容性备受瞩目，成为目前本体编辑的首选工具[11]。

Protégé有很多其他编辑工具所不具备的优点：

（1）采用图形化界面,与 Windows 操作系统的风格一致，模块划分清晰。

（2）是一个开放资源，允许用户二次开发，目前拥有最多的注册用户。

（3）系统的可扩展性好，支持下载安装或者自行开发插件，拓展 protégé 的功能。

（4）能够以多种方式存储本体，包括多种数据库格式和纯文本格式。

（5）支持多种本体表示语言输出，包括 XML、RDFS、OWL 等。

（6）不需要掌握具体的本体表示语言，比较容易学习。

（7）拥有众多的可视化插件供用户选择，概念关系一目了然。

（8）支持中文编码。

当然，Protégé在设计上也存在着一定程度的不足：

（1）一次只能打开一个本体：有些大型本体的编辑需要参考和引用已经存在的本体，这样的方式会影响大型本体的工作效率。

（2）不支持协同开发：目前的 Protégé只有客户端，没有服务器端，不支持协同开发。而尤其是大型本体的开发，需要多人的合作，大大影响本体的编辑效率。

（3）不能进行批量处理：在输入实例的过程中，不能以文本的形式将同类型的内容一次性导入，输入比较繁琐。

（4）运行速度比较慢，编辑效率不高，启动需要占用大量的内存资源。

（5）部分图形化显示工具不支持中文本体的显示，如 OWLViz。

Protégé使用方法：

Protégé工具有三个主要功能标签(Tab)：类(Class)、属性(Property)、实例(Instance)，主要是定义类和类结构，属性和属性值约束，类之间的关系和关系的属性等。使用 Protégé编辑本体只需要经过简单的操作即可实现，本节以简单的动植物本体为例阐述 Protégé的使用。

（1）建立本体项目

建立 project 文件，类似软件项目开发中的项目文件，为要建立的本体选择描述语言，目前的 Protégé有 XML、RDF 以及 OWL 三种语言供用户选择。

（2）建立类

为动植物本体建立类，在 Protégé中可以对类进行设置，如不相交类。设置了动物和植物两个类；还能继续建立类的子类，如为动物建立食草动物和食肉动物。

（3）设置属性

为动植物本体设置属性主要是定义类之间的语义关系，以 OWL 语言为例，其中的属性有 Object 属性 、Datatype 属性和 Annotation 属性。其中最主要的是 Object 属性和 Datatype 属性，前者用来描述两个类之间的语义关系，定义属性的定义域(domain)和值域(range)，如定义“吃”属性用来连接动物和植物，表示动物吃植物；Datatype 属性用来描述类本身的特点，如动物编号。

（4）添加实例

为已经设置好的类添加实例，例如为食草动物建立长颈鹿实例。

以上是最基本的设置，在 OWL 语言中，还可以为本体设置更多的语义关系约束，如把属性设置为：

（1）逆反属性(inverseOf)

如果 P1 被声明为 P2 的逆反属性，那么如果 X 通过 P1 关联到 Y，那么 Y 通过 P1 关联到 X。如果 hasChild 是 hasParent 的逆反属性，Deborah hasParent Louise，那么我们就能够推理出 Louise hasChild Deborah。

（2）传递属性(TransitiveProperty)

如果(x,y)是传递属性 P 的一个实例，(y,z)也是传递属性 P 的一个实例，那么(x,z)是传递属性 P 的一个实例。如果 ancestor 被声明为传递的，(Sara,Louise)是它的一个实例，(Louise,Deborah)也是他的一个实例，那我们就能够推理出(Sara,Deborah)是他的一个实例。

（3）对称属性(SymmetricProperty)

如果(x,y)是对称属性 P 的一个实例，那么(y,x)也是它的一个实例。被声明为对称的属性不能有任意的 domain 和 range。Friend 可以被说成是一个对称属性，如果 Frank 是 Deborah 的 Friend，那我们可以推断出 Deborah 是 Frank 的 Friend。为属性添加约束(Restrictions)，包括

（1）allValuesFrom 约束

该约束将一个属性的取值和一个 class 相关。也就是说，如果一个 class 的实例通过这个属性和另外一个 individual 相关，那么后一个 individual 则能够被认为是该约束类的一个实例。

Class Person 有一个属性 hasOffspring，该属性被约束在 allValuesFrom 上取值为 Person 类。这就是说如果 Person 的一个实例 Louise 通过属性 hasOffspring 和另一个 individual Deborah 相关，从这一点我们能推断出 Deborah 是 Person 的一个实例。这种约束允许 hasOffspring 属性被其他 class 使用，例如被 class Cat 使用，从而做出相应的约束。

（2）someValuesFrom 约束

和上面类似，该约束也将一个属性的取值和一个 class 相关。只不过此时要

求该属性的取值至少有一个是该 class 类型的[15]。

2.3 本章小结

本章是对构件和本体的相关概述；首先介绍了软件构件的相关概念，构件库、基于构件的软件开发方法、构件与对象的主要区别、构件与中间件的区别和联系；接着介绍了创建本体要用到有关概念，本体创建方法、本体表示语言、本体构建工具。

第三章 水利领域构件本体建模

介绍完构件和构件的相关概念后，本章就开始采用本体来具体描述水利领域构件，首先给出本体构件库模型。刻面是从某个方面或视角对构件进行描述，抽象出水利领域构件的刻面分类模型就有了水利领域构件本体的初步的概念和概念层次；就可以以该刻面分类模型为基础，构建出水利领域构件的本体。

下面提到的构件指水利领域构件，构件本体指水利领域构件本体。

3.1 本体的构件库模型

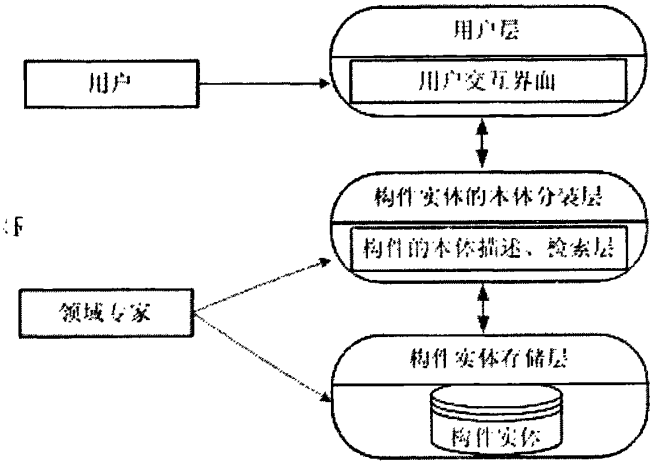

图 3.1 基于本体的构件库系统模型

基于 OWL 的构件描述信息的构件库分为三层，最底层为构件实体，是具体的构件实现；中间层为构件的本体描述和检索层，构件的本体描述是对最底层构件实体的本体封装，用 OWL 语言进行描述，可以存储在 OWL 文件中，也可存储在数据库中，构件的检索支持语义信息检索和语义推理；最外层为用户界面层，包括用户交互界面与用户检索语句的预处理过程。

给出了本体的构件库模型后，接下来就是根据模型构建具体的构件本体，本文的本体构建以构件刻面分类模型为基础，下面介绍什么是刻面、术语？怎样合并刻面和术语，分析水利领域，抽象出水利领域构件刻面分类模型。

3.2 构件的刻面和术语

刻面是一个单词或短语的固定集合，用于描述构件的某个方面或视角。术语就是一般性词语的集合，它们常常用来描述特定领域中令人感兴趣的对象，也可以将术语描述为“领域中特殊对象的技术词汇”。每个刻面具有一组术语，术语

之间有类层次关系和同义词关系而形成结构化的术语空间。不同的刻面（刻面，术语）构成分类中子集合中元素的满足条件。构件的描述术语仅限在给定的刻面之中选取，在术语空间中游历可以帮助复用者理解相关领域[20]。

3.2.1 刻面和术语设计要求

刻面分类策略尽管是一种比较理想的构件库分类策略，但在刻面设计上必须满足一定的要求，才能体现出刻面分类的优点。其设计要求主要在以下两个方面：

（1）如何定义刻面：构件的属性有很多，要从构件的属性中挑选出刻面是一件很棘手的工作。因为刻面有如下限制：

1）刻面是构件库的使用者查询时最感兴趣的构件属性，也是与构件的复用最相关的构件属性。

2）一个构件库的刻面数目不能太多，一般不超过 7 个。

3）该刻面是否对构件库的每一个构件都能适用，即构件库中每一个构件在该刻面的术语空间中是否有相应的术语。

4）任意两个刻面的术语空间必须正交，某个刻面的术语发生变化不会影响到其它刻面的术语空间。

5）刻面必须能适应构件库的变化。因为尽管刻面的术语空间随时间的演进会发生变化，但刻面相对于构件库来说是稳定的，即构件库的刻面一旦确定，就不能再改变。

（2）术语空间的建立：构件库的刻面确定好之后，为该刻面建立相应的术语空间也是一件非常困难的工作，这主要体现在刻面术语并没有确定的、为大家所公认的标准。究竟什么样的术语才是合法的术语?哪些术语具有同义词关系?哪些术语具有一般 - 特殊关系?要回答上面的问题，就必须要有一个刻面术语的标准，而这正是软件界所缺少的。软件术语的差异也是影响软件复用发展的重要因素之一[17]。

较少的刻面易于实现和维护，但会影响构件的检索效果，而过多的刻面可以更精确的描述构件，提高构件检索的效果，但会加剧刻面维护的工作量。同时，属于空间的维护以及术语同义词的维护与检索效果也应该有个平衡点。应该针对领域的大小和构件的复杂程度，确定合适的刻面数目，并确定术语和术语同义词的处理方案。

3.2.2 刻面和术语的表示[23]

（1）刻面的表示

定理 3.1：设 $\Psi|_{facet}$ 表示在刻面 facet 语境下的问题描述空间术语集

合，即：

facet=Space(facet)，Space 表示术语空间。

如果刻面的划分为 n 个，即 $facet_1$，$facet_2$，…，$facet_n$，简写为 f_1，f_2，…，f_n。

则有 $\Psi=\{\Psi|f_1, \Psi|f_2, \cdots, \Psi|f_n\}$

即：$\Psi|f_1 \cup \Psi|f_2 \cup \Psi|f_3 \cup \cdots \cup \Psi|f_n=\Psi$

在各个不同的刻面 facet 的问题空间的描述术语上不存在交集，即：

$\Psi|f_1 \cap \Psi|f_2 \cap \Psi|f_3 \cap \cdots \cap \Psi|f_n=\Phi$（$\Phi$表示空集）；

可知，同一个刻面模型中所有刻面所描述的术语空间不存在交集，因此，我们可以定义所有子类相互之间是不相交的（disjoint），即任意两个不同的刻面在描述术语上不存在交集。通过这个步骤，我们可以在后面使用推理工具自动检测类之间是否存在不一致的定义。

（2）术语的表示

相对于刻面来说，术语空间的表示要复杂许多。各种刻面模式在具体刻面数量、刻面名称虽不尽相同，但它们的刻面描述的问题空间是相同的，只是在具体的名称和表示上不同，或者是对一个问题划分空间颗粒度不同，对一个相同的问题空间，有些把它划分成一个刻面来表示，而有则把它划分成几个刻面表示。因此，不同构件库的刻面模式之间是可以转变的，可以根据刻面术语空间是否表达的是同一问题空间的为依据来确定不同刻面之间的关系。如 3C 模型中的“语境（context）”在青鸟构件模型中被划分为“构件注释（Component-Notes）”和“运行环境（Running-Environment）”两个刻面。

定理 3.2：有一相同的构件问题描述空间术语集合Ψ，对相同的Ψ有两组不同的刻面划分方法$\{f_1, f_2, f_3, \cdots, f_n\}$和$\{F_1, F_2, F_3, \cdots, F_m\}$，令：

$\Psi_i(f)$=Space（f_i），在$\{f_1, f_2, \cdots, f_n\}$刻面划分中的刻面 f_i 的术语集合；

$\Psi_j(F)$=Space（F_j），在$\{F_1, F_2, \cdots, F_m\}$刻面划分中的刻面 F_j 的术语集合；

则各个术语集合必存在如下关系：

$\Psi 1(f) \cap \Psi 2(f) \cap \Psi 3(f) \cap \cdots \cap \Psi n(f) = \Phi$

$\Psi 1(f) \cup \Psi 2(f) \cup \cdots \cup \Psi n(f) = \Psi$

$\Psi 1(F) \cap \Psi 2(F) \cap \Psi 3(F) \cap \cdots \cap \Psi m(F) = \Phi$

$\Psi 1(F) \cup \Psi 2(F) \cup \cdots \cup \Psi m(F) = \Psi$

也就是说，即使采用不同的刻面划分，但其术语集合表达的问题描述空间是一致的，这也是我们能进行刻面模式的转换的理论基础。

（3）术语的整合

为了把不同刻面分类模式中的术语空间统一起来，建立统一的刻面术语本体，我们需要对现有刻面的术语空间进行整合，同视刻面术语空间进行合并在构

造术语本体的过程中，相同视角的刻面要进行合并，合并后生成基于同视刻面的新的术语空间，以类的形式加入到术语本体论中。

定义 3.1：$F_i=<V_i, E_i>$，Fi 为某一刻面，$V_i=\{ v \mid v$ 是一个结点,v 的值是一个术语$\}$，$E = \{<v_1, v_2> \mid v_1, v_2 \in V_i, v_1$和 v_2代表的术语具有一般/特殊关系$\}$，术语空间中任意两个术语都不相同。

定义 3.2：若两个刻面 F1 和 F2 都是从分类视角 A 对构件进行分类，则称 F_1和 F_2 是基于分类视角 A 的同视刻面，记为：$[A]F_1 \approx F_2$。若 F_1 和 F_2 的分类视角完全相同或者一个刻面的分类视角包含另一个，则可以简单记为：$F_1 \approx F_2$。

下面给出刻面的合并算法：

设 $F1=<V_1, E_1>$，$F_2=<V_2, E_2>$是两个刻面，若 F_1 和 F_2 满足如下条件，则称 F_1 和 F_2 是可合并的：

(1)$F_1 \approx F_2$（F_1 与 F_2 是同视刻面）；

(2)一个刻面的名称是另一个刻面中的一个术语。

我们将刻面 F_1和 F_2的合并运算记为 $F = F_1 \cup F_2$。合并原则：合并后 F_1和 F_2 中的术语以及它们之间的一般/特殊关系保持不变。

在第一种情况下，$F= F_1 \cup F_2= F_1 \mid F_2$（|为或者关系）。

在第二中情况下，将两个可合并的刻面 F_1和 F_2合并为刻面 F，设 F_1的名称是 F_2中的一个术语，$F=<V, E>$。

```
Merge ( F1 ,F2 )
{
   V = {F2 的根和根的儿子};
   E = {e| e ∈ E2且 e = < F2 的根, v>};
   E' = E1 + E2 - E;
   Enew = E;
   While (E' != NULL )
   {
      For (所有 < u, v >∈ Enew)
      {
         Enew = Enew-<u, v>;
         For ( 所有<s, w>⊂E' ,s 和 v 的值相等)
         {
               E' = E' - <v, w>;
            if ( w 代表的术语不在 F 中)
            {
               V = V + {w};
```

```
                    E = E + {<v, w>};
                    E_new = E_new + {<v, w>};
                }
                Else //设<v', w'>∈ E, w 和 w' 的值相等
                if (s 和 v, 的值不相等)
                {
                    //设 E 中 x 和 s 的值相等，需要调整节点的层次
                    E = E - {<v' , w' >};
                    E = E + {<x, w' >};
                }
            }
        }
    }
}
```

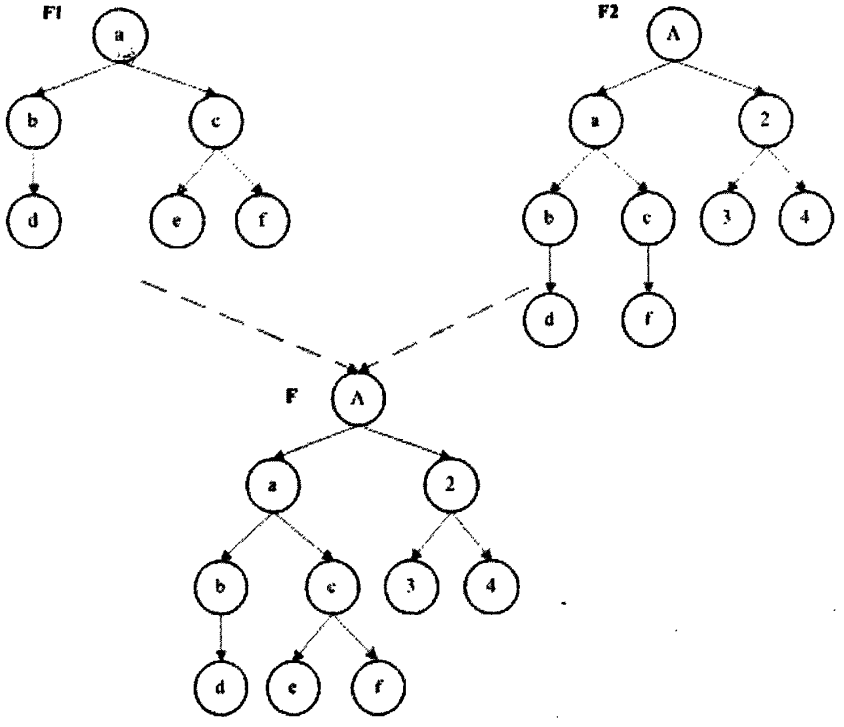

图 3.2 术语合并(刻面 F1 和 F2 合并为刻面 F)

根据上述算法，初始数据如下：

V = {A, a, 2}

E = {<A, a>, <A, 2>}

E_1 = {<a, b>, <b, d>, <a, c>, <c, e>, <c, f>}

E_2 = {<A, a>, <a, b>, <b, d>, <a, c> <c, f>, <A, 2>, <2, 3>, <2, 4>}

$E' = E_1 + E_2 - E = \{<a, b>, <b, d>, <a, c>, <c, e>, <c, f>, <2, 3>, <2, 4>\}$

$E_{new} = E = \{<A, a>, <A, 2>\}$

进行第一轮合并操作：

$<u, v> = <A, a>$

$E_{new} = E_{new} - <u, v> = \{<A, 2>\}$

$E' = E' - <v, w> = \{<b, d>, <a, c>, <c, e>, <c, f>, <2, 3>, <2, 4>\}$

$V = V + \{w\} = \{A, a, 2, b\}$

$E = E + \{<v, w>\} = \{<A, a>,<A, 2>, <a, b>\}$

$E_{new} = E_{new} + <v, w> = \{<A, 2>, <a, b>\}$

以此类推，进行循环操作，直到最终完成两个刻面的合并。

最后一轮的合并操作结果：

$V = \{A, a, 2, b, c, 3, 4, d, e, f\}$

$E = \{<A, a>, <A, 2>, <a, b>, <a, c>, <2, 3>, <2, 4>, <b, d>, <c, e>, <c, f>\}$

3.3 水利领域分析

水利领域应用具有许多不同于其他领域的特点。首先，大多数的水利应用业务都基于相同的数据。例如，防汛减灾、水量调度、水质监控等应用系统都要使用相同的水文数据、工情数据和空间数据。本质上，这些应用系统是对相同数据的不同业务逻辑处理。水利行业的软件一般是由各地各部门分别进行开发的，因此重复建设了许多功能相同或相近的系统，这些系统使用的编程语言、操作系统以及数据库系统都有很大差异，使得水利行业的应用系统呈现出重复性、复杂性和孤立性，造成系统间的集成和信息共享困难。而且，各种各样的系统在对同样的数据进行处理时容易产生数据不一致的问题。

水利领域的业务应用有以下特点：

地理性：水利业务与地理环境密切相关，它的处理对象多以天然流域为边界，但其服务对象却多以行政区划为约束。因此，水利业务系统需要在地理空间上实现数据的大规模共享。

分散性：水利业务系统管理的对象分布在广阔的地理空间之中，不可能在整个行政区划内或整个流域内建设一套集中的业务系统来处理本来就处于分散状态的对象。

重复性：由于水利业务管理是按行政区域分级进行的，从中央到地方均存在功能相同或相似的水利业务系统。

时效性：水利业务应用系统中有一部分是实时应用。如防汛减灾信息、水资源量与质的监控监测等方面，需要信息的快速采集、传递与分析处理，并具备对

其进行快速反应与决策支持的能力。

测控性：水利业务应用系统中，对水利信息的采集和对水利工程的监测监控是其重要的组成部分。

水利业务应用系统的主要业务逻辑包括信息采集、传输、存贮处理、服务与工程控制。其间，既有水利业务专业的逻辑，也有水利行政管理的逻辑。这些业务逻辑在一定范畴内具有相对的独立性同时它们有相互关联，如水调系统需要靠水量水质信息处理与服务来支撑，水资源评价系统也需要利用水量水质信息处理与服务。并且，数据处理的逻辑也是相同的。应用中还存在大量的子业务逻辑相同或相似，因此可以提取出公共的子业务逻辑和专有的子业务逻辑。同时，水利业务应用中存在许多松散的、低耦合的业务应用。例如，水量的调度应用，由于河流、水库等分布在各地，它的业务也分布在各处。每一个流域或地区的调度业务是一个独立的应用，同时所有这些业务可以共同组成全国的水量调度系统。地区与地区之间的业务应用就是一种松散的、低耦合的关系[31]。

3.4 水利领域构件的刻面分类模型

与通用构件库相比，水利领域的构件库针对水利这个特定的领域，在这个特定的领域中提取它们的公共特征，用这些具有相对稳定的公共特征描述领域构件。

当前通用构件库系统中的几种典型的构件的刻面分类模型，很少涉及到构件应用到的领域特征及其领域中专业方面的属性，为了提高复用者对构件库中的可复用构件的理解，更准确的检索到所需要的构件，本文针对构件的领域的特性，分析领域模型，定义领域范围。在参照 REBOOT 模型、3C 模型、青鸟构件库模型、参考论文[17][19][20]和水利领域分析的基础上定义水利领域构件的刻面分类模型。

水利领域构件的刻面分类模型的刻面描述定义为以下几个方面：

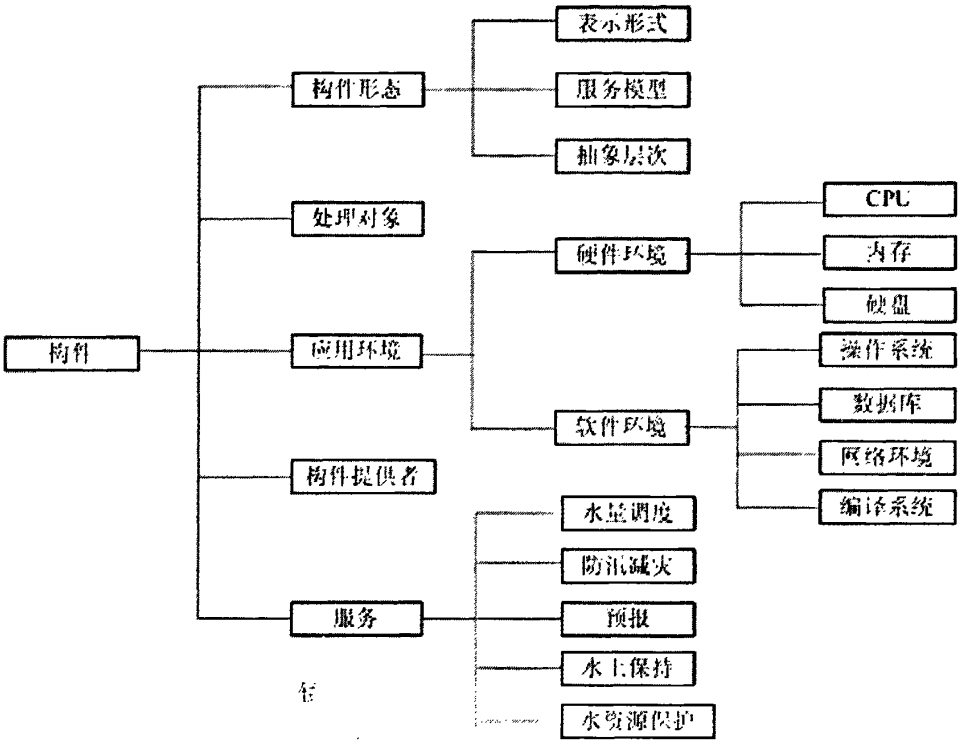

图 3.3 水利领域构件的描述刻面和子刻面

图 3.3-3.7 中 ▭ 表示刻面， ⬭ 表示术语。

3.4.1 构件形态刻面

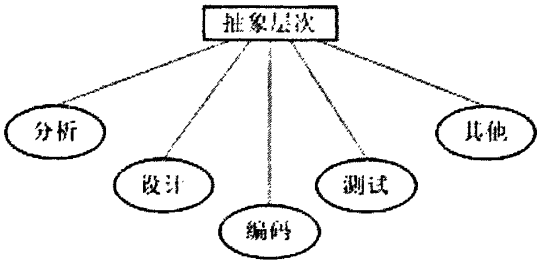

图 3.4 抽象层次子刻面及其术语

构件形态刻面是指构件的形式和状态等集合。

其中，表示形式子刻面是用来描述构件内容的语言形式或媒体，构件库中任何构件都以一定的形式存在，因而必然有其外在的表现形式。根据实际应用情况，将其术语空间指定为枚举值：DLL、EXE、源码、图形、其它。

服务模型子刻面指定为枚举值：COM\COM+、ActiveX(OCX)、.NET、VCL\CLX、JavaBean、 CORBA、ClassLibrary、Others 。

抽象层次子刻面是构件相对于软件开发过程阶段的抽象层次，根据实际应用情况，指定枚举值为：分析、设计、编码、测试、其他。

3.4.2 处理对象刻面

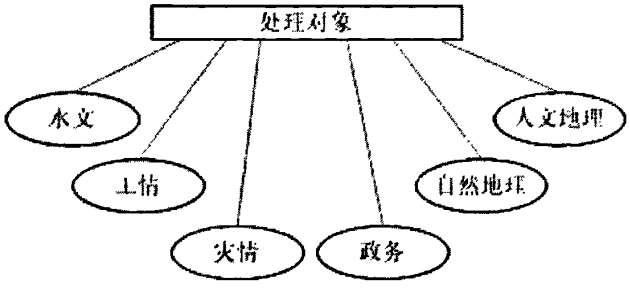

图 3.5 处理对象刻面及其术语

处理对象刻面的内容是水利业务的一个涉水信息，是和水资源领域密切相关的一个方面，当使用这样的描述方案来描述构件，构件的专业方面的属性得到了体现，可以是水文、工情、灾情、政务、自然地理、人文地理。

3.4.3 应用环境刻面

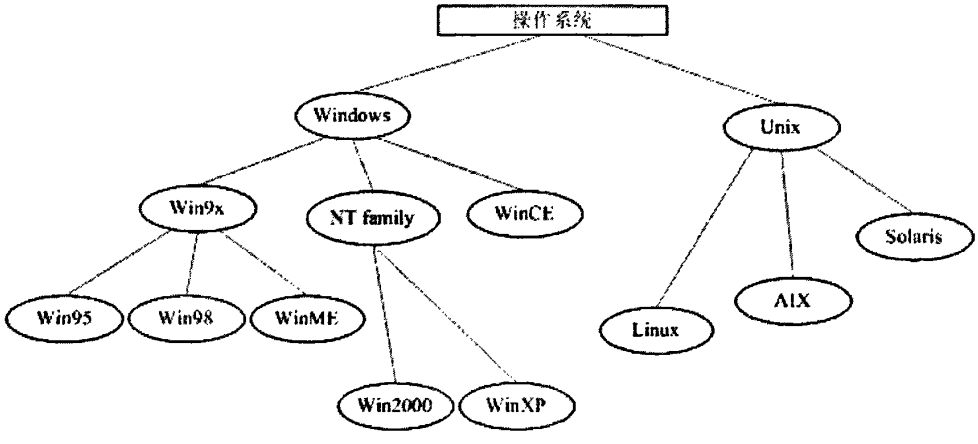

图 3.6 操作系统子刻面及其术语

应用环境刻面是使用(包括理解/组装/修改)该构件时必须提供的硬件和软件平台。如所需的特定的硬件环境、软件环境、CPU、内存、硬盘、操作系统、数据库平台、网络环境和编译系统等。构件库中任何构件都必须依赖于一定的使用环境才能得到复用。

3.4.4 构件提供者刻面

构件提供者刻面是对构件作者的信息描述，便于构件信息的反馈、更新及使用者与作者的交流等，故应对作者信息有较详细的描述。

3.4.5 服务刻面

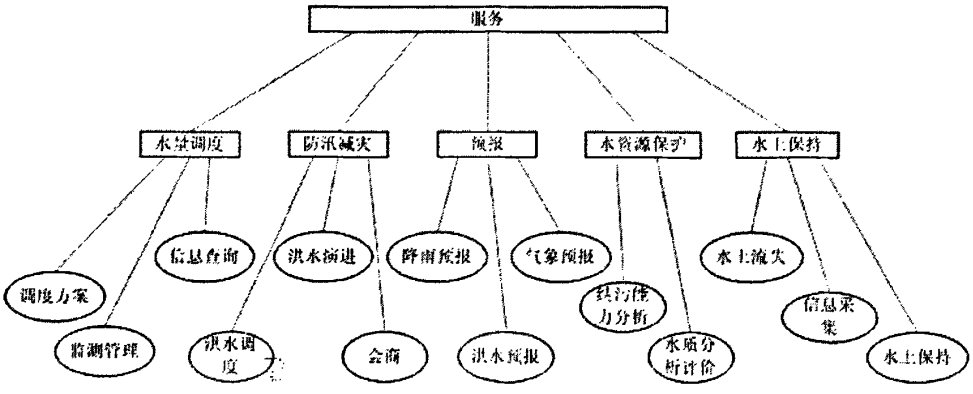

图 3.7 服务刻面、子刻面及其术语

服务刻面是描述构件在软件系统中所提供的软件功能集。构件库中任何构件都必须提供一种或多种功能。

以上五个刻面彼此间相互隔离，而且比较充分地体现了构件与复用相关的特性，能较好的适应构件库今后的发展。同时，设计中尽可能多的对元素术语指定枚举值，在一定程度上减少了软件术语差异产生的影响。

3.5 水利领域构件的形式化表示

参照 3.4 水利领域构件的描述刻面和术语，下面给出了水利领域构件的刻面结构模型。水利领域构件刻面结构模型约定了水利领域构件的组成，它是对水利领域构件实体的抽象。

在构件库系统中考虑到构件库管理及其它需要，还应为构件添加一些元素，本文为构件添加两个元素：

（1）构件名称：每个构件都必须有一个确定的名称，该名称必须完整地标识该构件的本质。

（2）构件地址：是构件实际存储的地址，可以是一个实际的物理地址，也可以是一个构件查询语句。

除了添加上面添加的两个元素外，可以根据需要添加其它的构件元素，比如

构件 ID，可以唯一地标识一个构件，这里统称这些信息为构件的基本信息。

基于语义的水利领域构件模型表示如下：

水利领域构件::=<构件基本信息><构件刻面信息>

构件基本信息::=<构件名称><构件地址>[<其它>]

构件刻面信息::=<构件形态><处理对象><应用环境>
<构件提供者><服务>

构件形态 ::=<表示形式><服务模型><抽象层次>

表示形式 ::=DLL|EXE|源码|图形|其它

服务模型 ::= COM\COM+|ActiveX(OCX)|.NET|VCL\CLX|JavaBean|CORBA|C1assLibrary|Others

抽象层次 ::=分析|设计|编码|测试|其他

处理对象 ::=水情|工情|灾情|政务|自然地理|人文地理

应用环境 ::=<硬件环境><软件环境>

硬件环境 ::=<CPU><内存><硬盘>

软件环境 ::=<操作系统><数据库平台><网络环境><编译系统>

构件提供者 ::=作者&地址&联系电话&E-mail

服务 ::=[<水量调度>|<防汛减灾>|<预报>|<水资源保护>|<水土保持>]$^{+}$

水量调度 ::=[<调度方案>|<监测管理>|<信息查询>]$^{+}$

防汛减灾 ::=[<洪水调度>|<洪水演进>|<会商>]$^{+}$

预报 ::=[<降雨预报>|<洪水预报>|<气象预报>]$^{+}$

水资源保护 ::=[<纳污能力分析>|<水质评价分析>]$^{+}$

水土保持 ::=[<水土流失>|<信息采集>|<水土保持>]$^{+}$

(| 表示或者关系，& 表示并且关系，$^{+}$ 表示至少存在一个)

领域构件语义模型的建立促进了领域构件的描述和领域构件组装的实施,为领域构件的检索、评价、剪裁和扩展奠定了坚实的基础。

3.6 水利领域构件本体模型的构建步骤

本文领域构件本体模型的构建主要通过以下几个步骤实现：

（1）水利领域构件刻面分类模型的确定：主要工作是获得水利领域构件本体模型开发所需的原始数据，分析数据以便进行本体抽取。通过 3.4 节已经完成。

（2）水利领域构件本体的构建：主要工作是从获得的水利领域构件描述刻面得到一个原始的水利领域构件本体模型。

（3）优化本体：主要工作是对原始的水利领域构件本体模型进行优化。

（4）评价并验证本体：主要工作是对水利领域构件本体模型进行评价并验

证。

3.6.1 水利领域构件本体的构建实现

领域构件本体的构建主要包括三个部分：定义原子概念和概念的层次、定义概念的属性和创建概念的实例。

构件的分类模型确定后相当于有了初步的概念和概念的等级体系，下面的工作就以确定的刻面和术语为基础，对初步的概念和概念的等级体系进行适当的调整，定义本体的类和类的等级体系，下面的主要步骤按照七步法中的第四步(定义类和类的层次)到第七步(创建类的实例)的操作进行(必要时可以进行局部调整)，在具体的操作过程中第四步和第五步(定义类的属性)在进行到第六步(定义属性的分面)前可以不断地进行调整，把类调整为属性可以简化关系的复杂性，把属性调整为类可以丰富类之间的关系。

定义 3.3：一个 Ontology 的逻辑结构被表示成一个五元组，O = {C, R, H, Rel, A}

①C ：概念元素的集合。

②R ：概念间关系的集合。

③H ：概念间的层次，是 C ×C 的一个子集。H (C1, C2)表示 C1 是 C2 的子概念。

④Rel ：函数，函数的定义域是 R，值域是 C ×C 的一个子集。即： Rel (R) = (C1, C2) 。

⑤A：公理集，包含了 ontology 所需的公理。使用适当的逻辑语言,例如一阶逻辑。

下图为以 3.6(1) “水利领域构件刻面分类模型的确定”为基础，实现 3.6(2) “水利领域构件本体的构建”的具体流程。

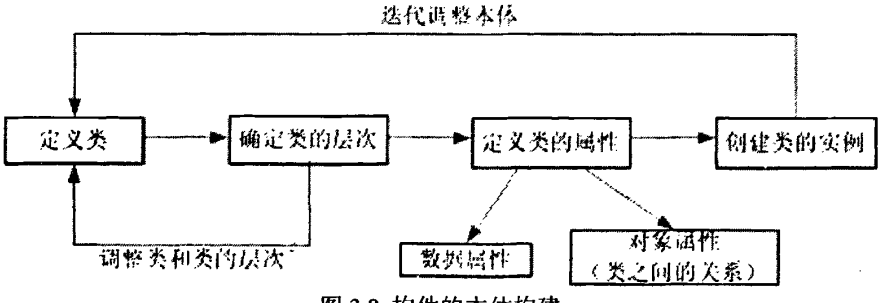

图 3.8 构件的本体构建

图 3.8 中把两者分开是为了强调两者有互动关系：如果类的层次定义得不合理可以考虑重新“定义类”，对类进行重新调整类。

3.6.1.1 定义类、确定类的层次

类也称概念、概念类，本文类、概念和概念类表示同一个意思。建立本体时，首先确定本体包含的类（ Class ）和类的层次，目前有三种常见方法来确定原子概念及其层次：

（1）自顶向下的开发过程：开始定义领域中最顶层的概念，然后逐步细化这些概念。

（2）自底向上的开发过程：开始定义最底层的概念，然后定义层次结构中的叶子结点，最后将这些概念分组为更一般的概念。

（3）组合的开发过程：是自顶向下和自底向上两种方法的组合。

本文采用自顶向下的方法，确定类和类的关系。

由于 Protégé的 OWLViz 图形化显示工具不支持中文本体的显示，而且用中文标志处理起来比较麻烦，所以本文类用英文表示。

图 3.9 定义类及类的层次

（1）首先确定 Component（构件）类：

<owl:Class rdf:ID="Component"/>。

（2）根据上面抽象的构件五个刻面，分别建立五个刻面类。

根据“构件形态刻面”抽象出 ComponentForm（构件形态）类：

<owl:Class rdf:ID="ComponentForm"/>，

根据“处理对象刻面”抽象出 TreatObject（处理对象）类：

<owl:Class rdf:ID="TreatObject"/>，

根据“应用环境刻面”抽象出 AppEnvironment（应用环境）类：

<owl:Class rdf:ID="AppEnvironment"/>，

根据“构件提供者刻面”抽象出 Author（构件提供者）类

<owl:Class rdf:ID="Author"/>，

根据“服务刻面”抽象出 Server (服务)类：

<owl:Class rdf:ID="Server"/>。

（3）刻面如果有子刻面，分别建立各自的子刻面类，如果子刻面还有子刻面则递归建立子刻面的子类（子刻面为刻面或上层子刻面的子类）。

“构件形态刻面”有“表示形式”、“服务模型”、“抽象层次”三个子刻面，分别建立 ComponentForm 的三个子类 Presentation、ServeModel、AbstractLevel。

同理，建立 AppEnvironment 的两个子类 Hardware、Software，然后分别递

归建立 Hardware 和 Software 的子类 CPU、Memory、HardDisk 和 OS、DB、NetEnvironment、CompileSystem；建立 Server 的子类 WaterVolDispatch、FloodPrevention、Forecast、WaterResourceProtection、WaterSoilMaintenance。

（4）根据各个刻面的非叶子术语建立刻面的子类，如果非叶子术语还有子术语并且不为叶子节点，则和第三步的刻面一样递归建立术语的子类。

以OS为例，建立OS的子类Windows、Unix，Windows的子类Win9x、NTFamily、WinCE。

（5）为概念添加同义词类。对某一概念有时有着不同的名称表示，如“Operation System”有时简写表示为“OS”；对同一概念不同的人有不同的表示名称，如对中文“操作系统”其英文名为“Operation System”，本体能实现资源共享，为了实现对同一概念的共同理解，架起语义理解的桥梁，必须为类增加同义词。为本文中的构件、构件的刻面、子刻面、术语增加中文名。

图3.10 Protégé创建的类

由于OWLViz图形化显示工具不支持中文本体的显示，所以图3.10显示的是去掉（5）中的类之后的内容。

3.6.1.2 定义类的属性

仅仅用类不能提供足够的信息，定义一些类之后，必须描述这些类的内部结

构，所以定义类的属性（Property），用属性来描述概念的特征。

（1）为类添加数据属性

例：“构件”类的数据属性

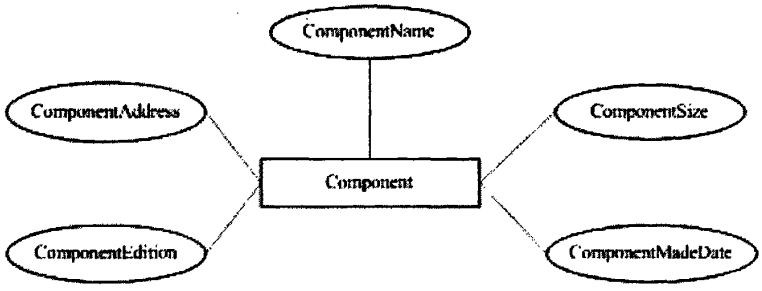

图 3.11 构件类的数据属性

为 Component 类添加 ComponentName（构件名称）、ComponentAddress（构件地址），ComponentSize（构件大小），ComponentMadeDate（构件制作日期），ComponentEdition（构件版本）5 个数据属性。

“构件名称”：

```
<owl:DatatypeProperty rdf:ID="componentName">
    <rdfs:domain rdf:resource="#Component"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

“构件地址”：

```
<owl:DatatypeProperty rdf:ID="componentAddress">
    <rdfs:domain rdf:resource="#Component"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

“构件大小”：

```
<owl:DatatypeProperty rdf:ID="ComponentSize">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#Component"/>
</owl:DatatypeProperty>
```

“构件制作日期”：

```
<owl:DatatypeProperty rdf:ID="ComponentMadeDate">
    <rdfs:domain rdf:resource="#Component"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
</owl:DatatypeProperty>
```

“构件版本”：

```
<owl:DatatypeProperty rdf:ID="ComponentEdition">
```

```
        <rdfs:domain rdf:resource="#Component"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    </owl:DatatypeProperty>
```

同理，为其它类添加类的数据属性。

（2）为概念类添加对象属性

protégé中类之间的关系是通过 ObjectProperty（对象属性）来体现的。

为 Component 类添加对象属性 hasForm(构件形态)、hasDomain（处理对象）、hasEnvironment（应用环境）、hasAuthor（提供者）、hasServer（服务）使 Component 和分别和 ComponentForm、TreatObject、AppEnvironment、Author、Server 类相关联：

例：“构件”到“构件形态”关系

```
<owl:ObjectProperty rdf:ID="hasForm">
  <rdfs:domain rdf:resource="#Component"/>
  <rdfs:range rdf:resource="#ComponentForm"/>
</owl:ObjectProperty>
```

a. “构件”到“表示形式”关系

```
<owl:ObjectProperty rdf:ID="hasPresentation">
    <rdfs:range rdf:resource="#Presentation"/>
    <rdfs:subPropertyOf>
        <owl:ObjectProperty rdf:about="#hasForm"/>
    </rdfs:subPropertyOf>
</owl:ObjectProperty>
```

b. “构件”到“服务模型”关系

```
<owl:ObjectProperty rdf:ID="hasModel">
    <rdfs:subPropertyOf>
        <owl:ObjectProperty rdf:ID="hasForm"/>
    </rdfs:subPropertyOf>
    <rdfs:range rdf:resource="#ServeModel"/>
</owl:ObjectProperty>
```

c. “构件”到“抽象层次”关系

```
<owl:ObjectProperty rdf:ID="hasAbstract">
    <rdfs:range rdf:resource="#AbstractLevel"/>
    <rdfs:subPropertyOf rdf:resource="#hasForm"/>
</owl:ObjectProperty>
```

构件之间有精化关系、版本关系、包含关系、协作关系，为 Component 添加

hasRefine、hasVersion、hasInclude、hasCollaboration 四个对象关系。

为 OS 类添加对象属性 hasfollow，说明操作系统间的版本关系。

由于“防汛减灾”要以“预报”为基础，根据预报做出相应的防汛减灾处理，所以为防汛减灾添加 hasForecast 对象关系，使两者联系起来。

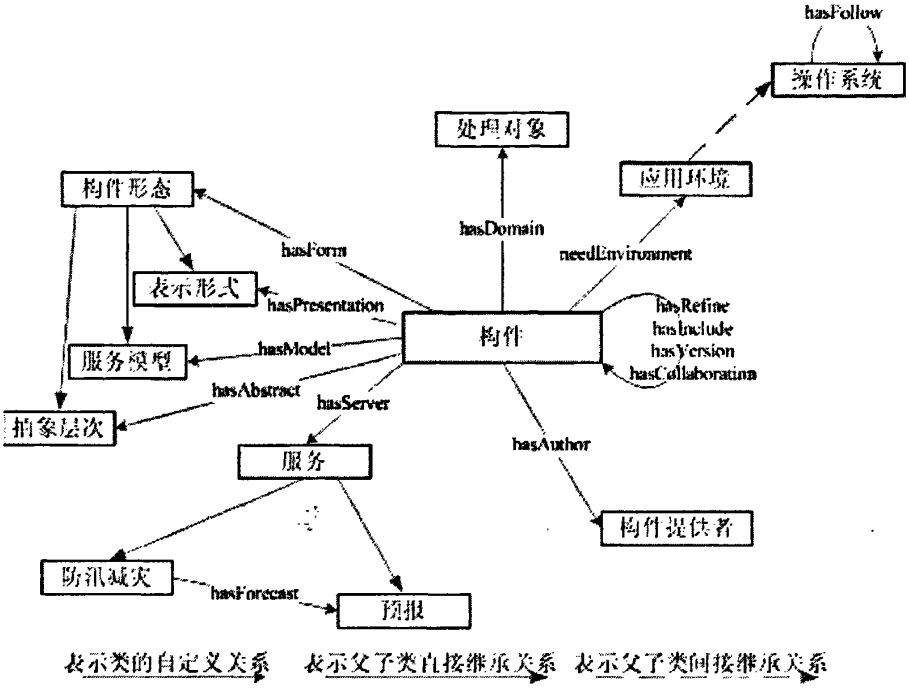

图 3.12 构件类的对象属性

上图中 hasPresentation，hasModel，hasAbstract 三个对象关系是从 hasForm 继承的子关系。

3.6.1.3 创建类的实例

本文中把 3.4 节中抽取的叶子术语作为刻面或父术语的实例来处理。

protégé中创建类的实例包括三步：选择一个类、创建该类的实例、填充实例的属性值。

例如为 Win9x 创建三个实例，win95，win98,winMe 并填充其属性值。

下图为构件类添加水情数据获取实例，填充 hasForm, hasServer, hasVersion, ComponentName, ComponentAddress 等属性的值，如果属性值不存在，则为空不填。

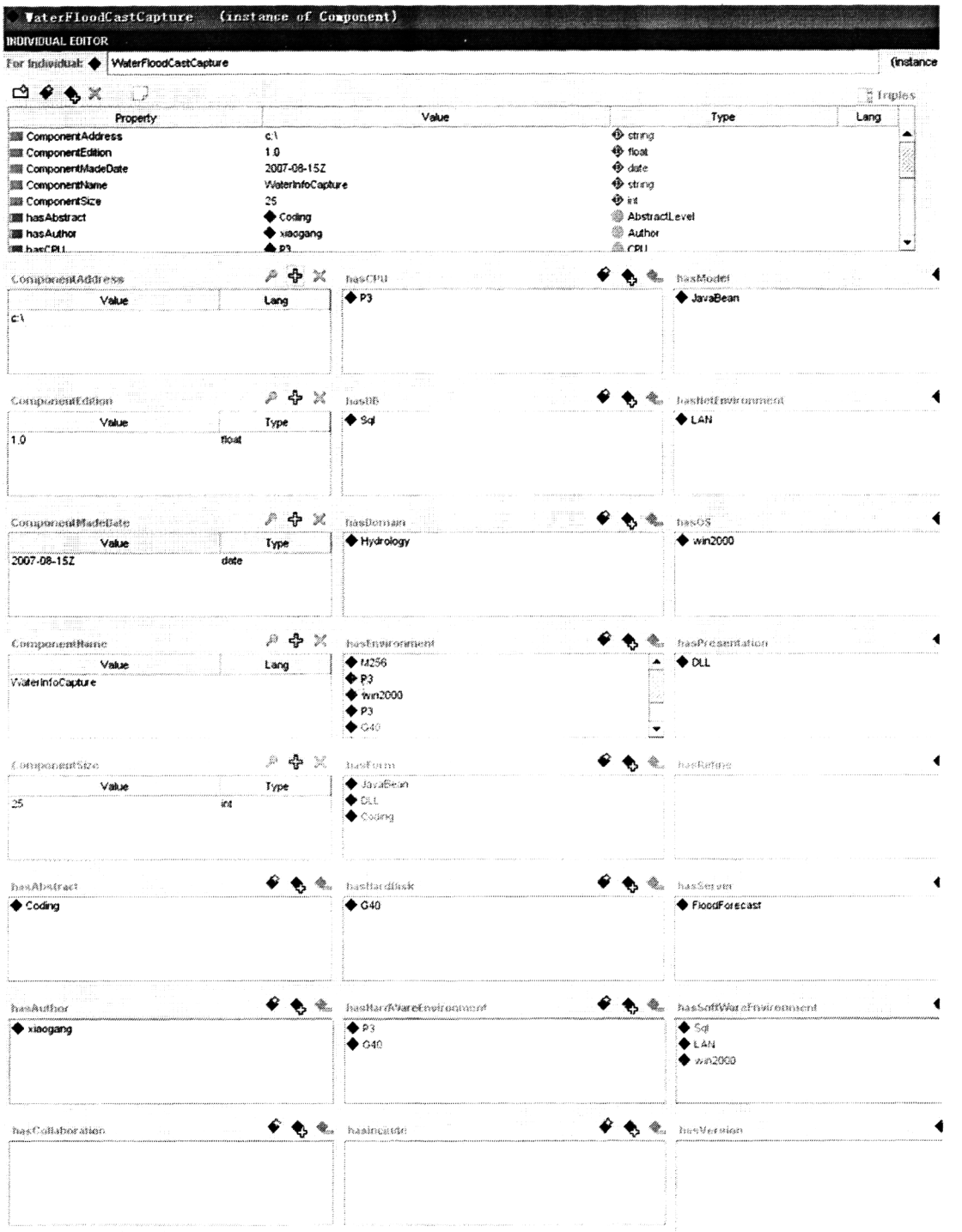

图 3.13 Protégé 添加类的实例

本文水利领域构件本体定义为：

O = {C, R, H, Rel, A}：

C={

Component, ComponentForm, AbstractLevel, Presentation, ServeModel, AppEnvironment, HardWare, CPU, HardDisk, Memory, Software, CompileSystem, DB, NetEnvironment, OS, Unix, Windows, NTFamily, Win9x, WinCE, Author, TreatObject, Server, FloodPrevention, Forecast, WaterResourceProtection,

WaterSoilMaintenance, WaterVolDispatch

};

R={

Subclass(Hardware,AppEnvironment),Subclass(CPU,Hardware),Subclass(HardDisk,Hardware),Subclass(Memory,Hardware),Subclass(Software,AppEnvironment),Subclass(CompileSystem,Software),Subclass(DB,Software),Subclass(NetEnvironment,Software),Subclassof(OS,Software),Subclass(Unix,OS),Subclass(Windows,OS),Subclass(NTFamily,Windows),Subclass(Win9x,Windows),Subclass(WinCE,Windows),Subclassof(AbstractLevel,ComponentForm),Subclassof(Presentation,ComponentForm),Subclassof(ServeModel,ComponentForm),Subclassof(FloodPrevention,Server),Subclassof(Forecast,Server),Subclassof(WaterResourceProtection,Server),Subclassof(WaterSoilMaintenance,Server),Subclassof(WaterVolDispatch,Server),Instanceof(win95,Win9x)

.....

};

H={

(Win9x,Windows),(Windows,OS),(OS,Software),(Software,AppEnvironment),

(CPU,Hardware),(Hardware,AppEnvironment),

(AbstractLevel,ComponentForm),

(FloodPrevention,Server),

.....

};

Rel={

hasAuthor(Component,Author),hasForm(Component,ComponentForm),

hasFollow(OS,OS),

.....

};

3.6.2 优化本体

构建领域构件本体后，接下来的工作就是优化本体。优化本体主要有两个方面：优化类和类的层次、优化属性。

（1）优化类和类的层次

优化类和类的层次主要是从以下几个方面进行优化：

1）控制类的数量

在构造良好的本体中，类的直接类的数量在 2－15 之间。有两条指导原则作

为参考：如果一个类只有一个直接子类，可能存在建模问题或者本体不完整；如果一个类多于 15 个子类，那么可能要增加中间类。

2）避免类环路

避免类层次结构中出现环路。例如：类 B 是类 A 的子类，同时又是类 A 的父类，那么在这个类层次中就出现了环路。如果出现这样的环路，那么说明类 A 中所有的实例都是类 B 的实例，类 B 中所有的实例又都是类 A 的实例。类环路会使得构件检索中出现死循环的结果，因此必须在开发中避免这种情况的发生。

3）处理好类层次中兄弟节点的层次关系

类层次结构中的兄弟节点指的是同一个类的直接子类。层次结构中的所有的兄弟（除根节点外）必须位于同一个层次中。一般化的概念比具体的概念处于层次的较高层。

4）子类不相交

因为类不能有共同的实例，所以它们是不相交的。

（2）优化属性

对于属性的优化，主要是从以下两个方面进行优化：

1）设置属性的缺省值

如果存在这样的现象：对于类的许多实例，实例的某个属性值都是相同的，那么我们可以把这个属性值定义为属性的缺省值。那么每次创建包含这个属性的实例时，系统就会自动填写缺省的属性值。缺省值不会强加任何约束。例如，如果大多数领域构件的“抽象层次”都是“源码”的话，那么我们就可以将“源码”作为属性“抽象层次”的缺省值。

2）设置属性的逆反属性

一个属性的值可能会依赖于另一个属性的值。例如，为 OS 的 hasFollow 对象属性添加 hasPrevious 逆反属性，如果在两个方向上记录这个信息是多余的，设置逆反属性就只需要记录一个方向上的信息。“win2000”的 hasFollow 是“winXp”，那么在设置了逆反属性的情况下，系统将自动地将“winXp”设置为“win2000”的“hasPrevious” ，并且在语义检索中还能实现双向的推理。

3.6.3 评价、验证领域构件本体

使用领域构件本体模型可以对领域构件进行语义检索，但是领域构件本体模型的质量决定检索的性能。如果存在错误的本体定义，那么就可能推导出错误的结论，所以需要领域专家对本体进行评价和验证。

本文主要从完整性、可读性、清晰性、一致性、可扩展性和持久性等方面进行评价和验证。

完整性：本文抽取了一些水利领域构件的相关概念，初步建立了构件的本体描述，可以在后续的工作中继续进行概念的细分和扩展。

可读性、清晰性：本文从水利领域构件的刻面为基础建立本体，概念和层次可读性和清晰性都比较强。

一致性：运用 RacerPro 推理机进行一致性检查，通过一致性错误检查。

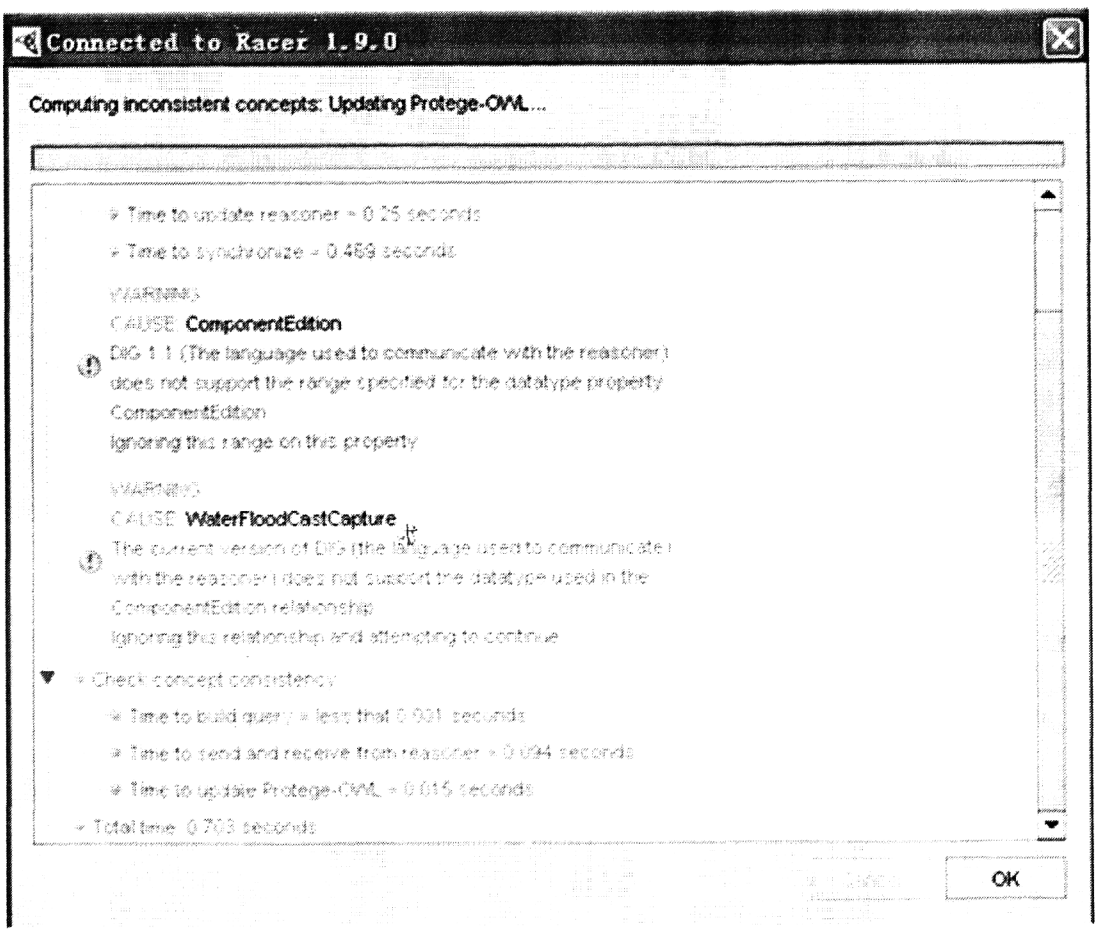

图 3.14 RacerPro 对构建的本体进行一致性检查

如上图 RacerPro 检查出现两处警告，都是说明 RecerPro 不支持数据属性的 Range 为 float，把 float 该成其它的类型（比如 int 类型），就不出现警告信息。但一般的版本信息都是表示成“***.***”格式，所以本文忽略警告信息，仍用 float 类型。

可扩展性：领域本体的概念层次树可以根据应用需求方便地进行扩充、进一步细化或者修改。

持久性：水利领域构件本体刻画了水利领域的相关知识，这些知识相对稳定。本体采用 OWL 文件存储或者将本体存入数据库。

3.7 本体复用

随着本体的广泛运用即越来越多的应用使用本体来表达语义信息，怎样支持本体复用变得越来越重要。

本文讨论的本体表示语言为 OWL。

最简单的本体复用采用 OWL：imports 方法，它列出了其它的本体，其内容被设定为当前本体的一部分，支持复制+粘贴一个本体到另一个本体，称之为语法导入。

例如在另一本体中用 OWL：imports 加入本文创建的本体：

在本体头中加入 <OWL:imports rdf:resource =

"http://www.owl-ontologies.com/OntologyComponent.owl/">;

OWL:imports 是一个传递性的属性，如果本体 A 引入了本体 B，本体 B 引入了本体 C，则本体 A 也引入了本体 C。

本体有三种复用类型：

（1）简单的把整个源本体导入目标本体，且仅用其中部分规则，而直接忽略其余规则，出现的问题是导入源本体中相冲突的规则易出现矛盾，效率不高，在仅需要复用源本体中部分对象时，也要导入整个本体。

（2）仅“复制+粘贴”源本体规则的某子集到目标本体为导入规则的子集，要用户细心找到想要的规则，否则会出现信息丢失，而本体的结构复杂异构，完全靠手工来完成变得很难实现。

（3）把源本体分解成不同的模块，根据相关标准，仅导入需要的模块（这和程序语言中的模块-构造是基于信息隐藏的想法类似：模块把某个功能提供给外部世界（通过模块的输出子句），但是引入了模块的模块本身不必关心这个功能是如何实现的）。

相比而言，第（3）种方法是最有发展前景的方法，目前的困难在于本体的模块化还不很成熟，只提供了本体类(称之为安全本体–能划分成模块的本体)的理论和算法。

W3C 标准定义的 OWL 本体由一系列规则组成，包括类规则、属性规则和实例规则。在如下导入相似性标注的帮助下，OWL 本体可以导入其他的 OWL 本体。

Annotation(imports <http: //www.Linux.org/linux#>)

Annotation(imports <http: //www.xxx.it/os#>)

导入标注含有把两个源本体导入目标本体的规则。把本体概念加以扩充，可以实现从源本体的类、属性和实例的语义导入。为了表述上的简化，这里仅考察抽象定义域，而非具体数据定义域[41]。

定义 3.4：(本体). 本体是元组 W_i=<i,M_i,S_i,O_i>其中：

1. i 是本体 W_i 的标识；

2. M_i(称为语法导入集)是一套本体标识符，由 W_i 按语法导入(用 owl: imports 实现)；

3. S_i(称为语义导入集)是类、属性和/或实例的符号集，由 W_i 按语义导入(用 owl:semanticImports)；

4. O_i(称为公理集)包含一组类、属性和实例的公理。

设符号 l: A, l:B，l:R，l:O_1，l:O_2∈S_2，l 是词典，A，B 是类，R 是关系，O_i 是公理。l:A 表示用词典 l 的词汇表示的类 A，l:A⊆l:B 表示用词典 l 表示的类 A 和类 B 的包含关系。

（1）类的层次结构的导入：如果 W_1 使规则 l: A⊆l: B 成立，那么 W_2 使 l: A⊆l: B 成立。

（2）互斥类导入：如果 W_1 使 l:A⊆「l:B 成立，那么 W_2 使规则 l:A⊆「l:B 成立。

（3）属性闭包导入：所有 l:O_1 的 l:R- successors 和 l: O_1 的 l:R- predecessor 也被导入。其中 R- successors 是后继，R- predecessor 是前驱。

（4）等价对象的导入：如果 W_1 使得 l: O_1≈l: O_2(或 l: O_1≠l: O_2) 成立，那么 W_2 也使规则 l: O_1≈l: O_2(或 l: O_1≠l: O_2) 成立。

例 1：构建一个交通工具的本体，可复用已有的本体，如 SUMO。车和马可作为交通工具，因此可利用如下的规则：

sumo: Horse⊆sumo: TransportationDevice (1)

sumo: Car⊆sumo: TransportationDevice (2)

和 SUMO 中的一些知识如：

sumo:car⊆∃sumo:capability.(sumo: Transportation) (3)

W_{ppl} =< h_{ppl};M_{ppl}; S_{ppl};O_{ppl}>标识一个本体，M_{ppl}= Φ(不是语法导入)，S_{ppl}={sumo:OS，sumo:DataBase,sumo:Teaching,sumo:Course,sumo:capability}(W_{ppl} 语义导入类 sumo:OS,sumo:DataBase,sumo:Teaching,sumo:Course 和属性 sumo:capability)并且 O_{ppl} 包含公理：

Sumo: DataBase ⊆∃sumo: capability.(sumo: Teaching) (4)

∃sumo: capability.(sumo: Teaching) ⊆∃sumo: Course (5)

ppl:Windows XP⊆sumo: OS (6)

ppl:MySql⊆sumo: DataBase (7)

(4)声明数据库可以教学；(5)数据库可以教学,所以数据库是课程；(6)断言 Windows XP 是操作系统；而(7) 声明 MySQL 是一种数据库。这些在 SUMO 中没有的规则,W_{ppl} 语义导入 sumo:OS, 即 Wppl 需要 SUMO 中 sumo: OS 的语义。总之,如果 W_{ppl} 定义域中的一个对象不在 SUMO 本体定义域中，那么该对象就不会是 sumo: OS 的定义域对象。所以语义导入器可定义为类、属性或实例集合的描述,实际上语义导入器也包含各组成部分的清单。

3.8 本章小结

本章开始给出了本体的构件库模型；然后对水利领域进行分析，给出水利领域的特点；紧接着给出领域构件的刻面分类模型，在刻面分类模型的基础上构建出水利领域构件本体；最后研究了本体的复用方法，重点介绍了基于本体模块的复用方法。

第四章 水利领域构件本体的检索应用

因为构件的检索技术是构件复用的重要内容，构建水利领域构件本体的最终目的是为了更方便、快捷、全面和准确地检索到水利领域的构件，所以本章主要讨论怎样在水利领域构件本体中检索构件，为软件复用提供最合适的构件。

4.1 构件库的基本体系结构

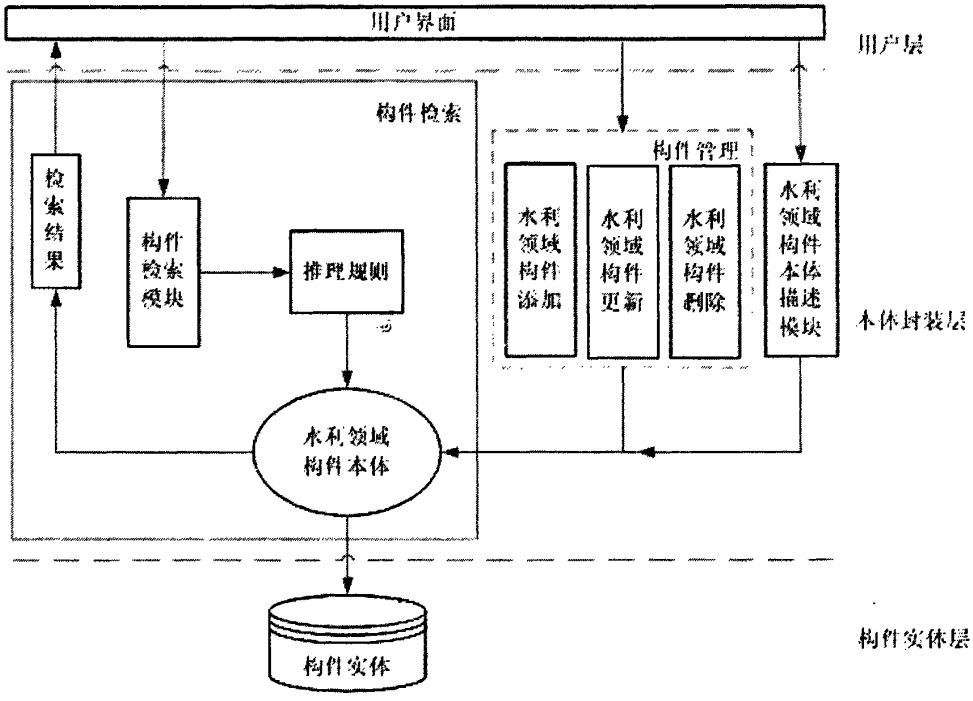

图 4.1 基于本体的构件库的体系结构

4.2 实现工具 Jena

Jena 是由 HP 实验室研究开发,用于构建语义 Web 应用的开发框架。该框架工具完全以 Java 开发的,它定义的类很容易用来进行语义 Web 开发,既可以方便地访问标准的 OWL 类及属性,也可以进行关联文件的推理、基于结构的推理、基于实例的推理等等。

Jena从最初的Jena1版本开始，经过不断的发展改进，目前的版本是Jena2.5 (2008-01-18 08:47)。Jena1的主要贡献是为处理RDF图提供了丰富的用于Model类的API。围绕着这些API，Jena1提供了大量的工具，包括对模型的多种方式的输入和输出I/O（RDF/XML，N3，N－triple），RDF查询语言RDQL，使用这些API，

用户可以选择将RDF图存储在内存或者是持久性存储（即文件和数据库方式）。Jena1还提供了附加的用于处理DAML＋OIL数据的API，但是Jena1并不支持OWL。Jena2支持基于RDFS和OWL等语义推理。Jena支持RDQL语义网查询语言。Jena2还拥有一种表现层接口是RDF WebAPI，它能够提供Web客户端查询RDF图，这种基于Web查询的数据获取方式当然也可以成为系统和应用程序员提供接口，是Jena以后的发展方向。

Jena 是一个具有三层架构和多种视图的语义网开发框架，提供给各种开发人员多种应用程序开发接口，具有很高的灵活性。

Jena2 的系统架构分为三层：Graph 层、EnhGraph 层和 Model（Ontology）层。各层的详细功能如下：

（1）Graph 层：用 RDF 三元组作为全局数据结构，包括内存和持久性存储方式，同时实现提供多种持久性存储三元组的方式，并且内置了基于 RDFS 和 OWL-Lite 的推理。

（2）Model 层：为应用开发人员提供视图，提供了大量的方法来操作 RDF 有向图（通过 Model 接口）和图中的节点（通过 Resource 接口）。

（3）EnhGraph 层：多种视图的同步，Model 层和 Graph 层之间的中间层，使得系统能够同步提供多种图或节点的方式。

Jena 主要由 6 部分组件构成，分别是 ARP、RDF API、持久性存储、推理子系统、Ontology API 和 RDQL。

（1）ARP

ARP 是 Jena 的一部分，它的功能是解析 RDF/XML 数据文件，主要用于 Jena 中的读取操作，但是也可以脱离 Jena 而使用在其他的 Java 程序中。ARP 是基于 Xerces（一种 xml 解析器）的，它遵循以下的规范：daml:collection，xml:lang，xml:base，URI，XML Names，International URI refs，Unicode Normal Form C，XML Literal，Relative Namespace URI references。

通常 ARP 被用在其他应用程序中，但是可以通过 NTriple 类的命令行方式来将一个 RDF/XML 文档转换成 N－Triple 形式，同时可以测试一个 RDF/XML 文档的健壮性。

（2）RDF API

RDF API 用来创建、解析、处理和查询 RDF 模型。Jena 定义了很多接口来处理和访问 RDF 数据。一个 RDF 图有多个三元组组成，而每个三元组又是由三个 RDFNode 构成，一个三元组可以用{ subject，predicate，object}来表示。

RDFNode 接口为所有的可以作为 RDF 三元组的元素提供了一个公共的基础，Resource 和 Literal 集成了 RDFNode 接口。Literal 接口表示文字值，提供了将文字值转换为 String、int 和 double 这样 Java 类型数据的方法。Resource 接

口提供了将RDF数据模型当作一个具有属性资源的集合来处理的方法。Property接口对象用来处理三元组中的<predicate>。Statement接口提供了将RDF数据模型当作一个RDF三元组集合来处理的方法。一个Statement表示一个三元组，或者一个三元组中的<object>。

（3）持久性存储

Jena2存储子系统扩充了Model类，可以透明的使用基于数据库引擎的持久性存储模型，支持的数据库有：MySQL、Oracle，PostgreSQL，而且支持Linux和Windows平台。

创建持久性模型主要采用工厂化（factory）方法。该方法分为三个步骤。第一步，加载JDBC驱动类，建立与数据库的连接。第二步，构造ModelMaker类，用来创建模型的持久性实例。第三步，调用ModelMaker类的方法创建新的模型或者打开已经存储的数据库持久性模型。

持久性存储子系统支持快速路径查询方法——RDQL查询，对于数据库Model，Jena内部动态的将RDQL转换成数据库支持的SQL查询。

（4）推理子系统

语义检索是基于概念及其概念之间的关系进行的语义层面的检索，其关键在于概念之间的推理。Jena提供基于规则的推理机(如RDFS Reasoner、OWL Reasoner等)，它包含了一般的推理功能，此外用户还可以根据需要自定义推理规则，也可以注册使用第三方推理引擎。

（5）Ontology API

Ontology API为语义网应用开发人员提供了一个持续稳定的开发接口，支持Ontology的各种语言，如RDFS、DAML+OIL以及OWL。该接口是完全采用Java语言描写的开发接口，提供了本体的建立、本体的解析、本体的推理等各种方法的类库，包括本体中类、实例以及属性的建立、读取和修改方法。

常用的类有OntModel、OntClass、OntResource、OntProperty、Individual等，其中OntModel类用于建立本体模型，包括指定描述语言以及推理机制等；OntClass类用来对本体主要部件——类的处理，如类的建立、读取等；OntResource定义了一个资源类，是本体中任何取值的超类，可以用于本体中各种类的存取；OntProperty主要对本体的各种属性进行处理，如建立、读取、删除属性操作；而Individual类主要是对本体的实例进行处理。

（6）RDQL

RDQL是Jena中针对RDF的查询语言，已经被许多基于RDF的系统采用。RDQL是从早期的RDFDB QL和SquishQL演变过来的，在Jena的很多RDFAPI中都实现了对RDQL的支持。这种查询语言将RDF视为三元组，也就是带有向边的图，图的节点是资源（Resource）或者文字（Literal）。RDQL提供了一种图匹配策略，

查询 RDF 中满足一定节点的图，查询结果返回匹配到的属性值。

RDQL 满足一定的范式，基本由 SELECT、FROM、WHERE 和 USING 字句组成。其中 SELECT 字句衔接的是提问中心，WHERE 字句衔接的是查询条件，即满足一定语义关系的用户查询组合，而 FROM 和 USING 字句分别衔接的 RDF 模型和本体的 URI 地址，由检索时设定[15]。

4.3 查询流程

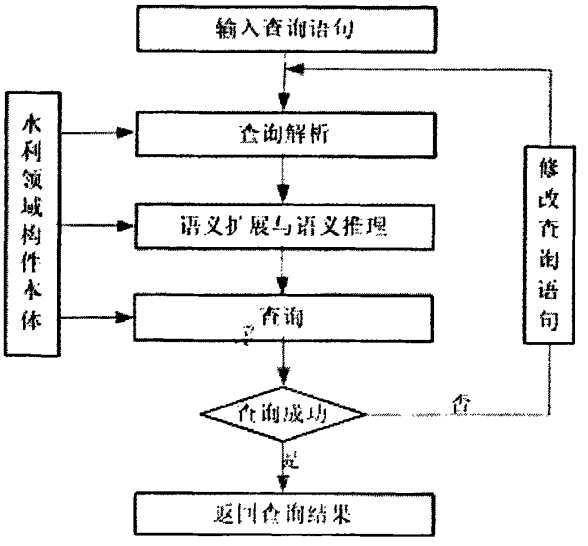

图 4.2 基于本体的构件查询流程

用户首先输入构件查询语句；参照水利领域构件本体对查询语句进行查询解析；参照水利领域构件本体对查询解析的结果进行扩展和语义推理；对语义扩展和语义推理后的结果在水利领域构件本体中进行查询；最后返回结果或修改查询语句重新进行查询。

4.4 查询算法

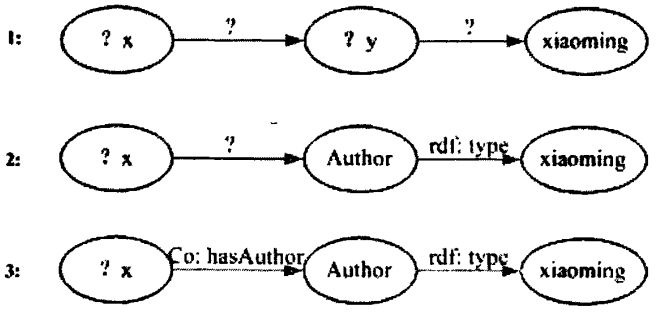

图 4.3 基于本体的构件查询图

（1）用户首先输入构件查询语句；对输入的查询语句进行预处理，得到查询关键字；

（2）参照本体对关键字进行解析，得到初始的实例，对实例进行扩展和语义推理，得到推理后的实例（本体三元组中对应 Object）；

（3）参照本体可以知道实例所属的类，知道实例的类后，根据实例类得到实例类和构件的关系（本体三元组中对应 Predicate）；

（4）根据本体的三元组 Subject、Predicate 、Object 关系，已知 Predicate 和 Object 从本体库中检索到要查询的构件（本体三元组中对应 Subject）。

见图 5.3，“构件提供者”为“xiaoming”为查询条件进行构件查询。

但（2）中每次判断关键字是不是实例时都要遍历本体，当本体中的类、实例较多会很费时。

查询输入语句一般是先给出类名关键字，类名关键字后紧接着的是类的实例（如“构件提供者为 xiaoming”，类名“构件提供者”后紧接着是“xiaoming”实例），所以上面的算法可以首先判断类，然后在类的实例中判断关键字是不是类的实例。

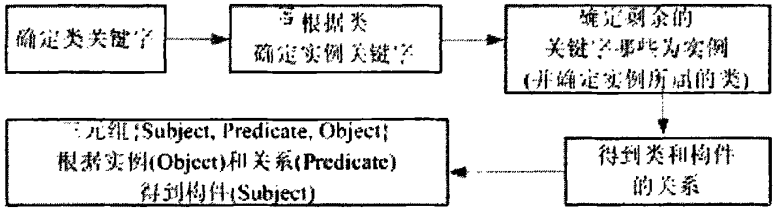

图 4.4 本体构件查询算法

假设输入关键词的集合为 C，当 C $\neq\Phi$ 时。

（1）C $\neq\Phi$，关键字映射到本体，判断关键字是不是类，若是类存入集合 A_c

（2）$C-A_c \neq \Phi$，$C-A_c$ 关键字映射到本体，其中关键字为（1）中确定的类的实例，存入和 A_c 中的类对应的集合 A_{il}

（3） $C-Ac-\sum A_{il} \neq \Phi$，参照本体对关键字进行解析，得到实例存入对应的集合 A_{il}，根据实例得到实例所属的类，存入集合 A_c。

（4）参照本体，根据集合 A_c 中的类确定和构件的关系，关系存入和 A_c 中的类对应的集合 A_{iP}

（5）根据 A_c 中类的关系和 A_c 中类的实例，对类中的实例集合 A_{il} 进行推理和扩展。

假设 P 为 A_{iP} 中属性，O 为 A_{il} 中概念，SPARQL 查询语句的生成：

If $A_{il} = \Phi$，then 返回　//要求用户重新考虑查询提问。

//根据 Predicate，Object 查询本体库中的 Subject

if $A_{IP} \neq \Phi$, then

取出 A_{IP} 和 A_{II} 中相对应的 P、I，得到 SPARQL 查询语句{?S, P, I}，提交给 Jena 查询，从查询结果中获得查询结果“?S”。

举例分析：用户输入构查询语句：查询构件，构件提供者为 xiaoming。

（1）首先对用户输入的语句进行分析得到关键字“构件提供者”和“xiaoming”，对“构件提供者”关键子进行本体映射知道“构件提供者”对应本体中的类“Author”，存入 A_C:{Author}；

（2）对“xiaoming”关键字进行本体映射，知道“xiaoming”是“Author”的实例，存入 A_{II}：{xiaoming}。

（3）有实例“xiaoming”的类 Author 和构件类有 hasAuthor 关系，得到 A_{IP}:{hasAuthor}。

（4）有三元组 Subject、Predicate(hasAuthor) 、Object(xiaoming)关系，可得出具体的 Subject(Component)。

4.5 功能实现

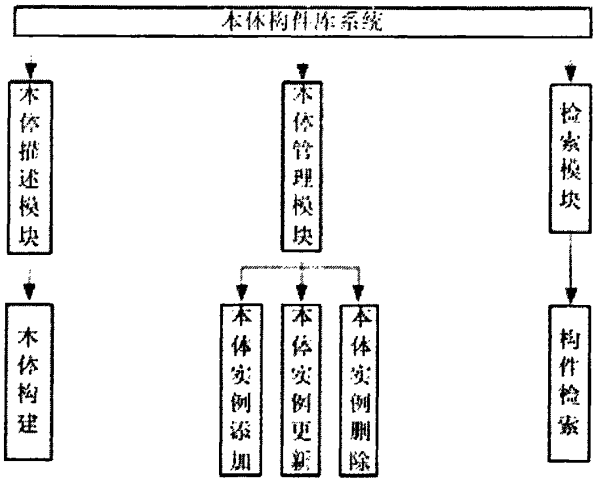

图 4.5 本体构件库功能模块

其中本体描述模块的本体构建在第三章中通过 Protégé工具实现；本体管理模块的本体实例添加、本体实例更新、本体实例删除可以通过在 Protégé工具中操作实现，也可通过 Jena 编程实现；检索模块是先对用户输入的检索内容进行处理，分析出用户的语义然后再通过推理检索查找相关的信息，下面主要介绍构件的检索。

4.5.1 功能实现关键代码

4.5.1.1 重要的 Jena 包

在使用 Jena 进行语义网应用时，最重要的包是 com.hp.hpl.mesa.rdf.jena.model，这个包包含了用于表达 model、resource、property、literal、statements 以及其他 RDF 的关键的接口。

com.hp.hpl.jena.ontology.OntModel 是专门处理本体（Ontology）的，它是 com.hp.hpl.jena.rdf.model.Model 的子接口，具有 Model 的全部功能，同时还有一些 Model 没有的功能，例如 listClasses()、listObjectProperties()，因为只有在本体里才有“类”和“属性”的概念。

com.hp.hpl.jena.query，通过 com.hp.hpl.jena.query 包中的类，使用 Jena 来创建和执行 SPARQL 查询。

4.5.1.2 关键代码

本文创建的本体文件名为 OntologyComponent，存储在 C:\Program Files\Protege_3.2.1 文件夹中。

（1）创建本体模型

```
OntModel m = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM, null );
```

（2）读取本体模型

```
String filePath = "C:\\Program Files\\Protege_3.2.1\\OntologyComponent.owl";
m.read(new FileInputStream(filePath), "");
```

（3）列出本体中的类

```
for(Iterator i=m.listClasses();i.hasNext();)
{
    j++; //用来统计类的个数
    OntClass c = (OntClass)i.next();
    System.out.println(j+c.getLocalName());
}
```

（4）读取类的等同术语

```
String camNS = "http://www.owl-ontologies.com/Ontology1209647070.owl#"
OntClass SOntClass = m.getOntClass( camNS + "OS" );
for(Iterator i = SOntClass.listSameAs(); i.hasNext(); )
{
   OntResource OntClass=(OntResource) i.next();
```

```
        System.out.print(OntClass.getLocalName());
    }
```

（5）列出类的指定属性和属性值类型（下面代码列出的是以 has 开头的属性）

```
    for(Iterator y=c.listDeclaredProperties(true);y.hasNext();)
    {
        OntProperty property=(OntProperty)y.next();
        String PropertyName=property.getModel().usePrefix(property.getURI());
        String strRange=property.getRange().toString();
        String strRangeName=property.getModel().usePrefix(strRange);
        //show just the "has" Properties
       if(strPropertyName.substring(1).substring(0,3).equals("has"))
        {
            System.out.print("  ");
            System.out.print(strPropertyName.substring(1));
            System.out.print(":");
            System.out.println(strRangeName.substring(1));
        }
    }
```

（6）列出类的实例

```
    for(Iterator i=m.listClasses();i.hasNext();)
    {
        OntClass c = (OntClass)i.next();
        if(c.getLocalName().equals(className))  //指定类的实例
        {
            System.out.println(className);
       for (Iterator Instans = c.listInstances(); Instans.hasNext();)
            {
                OntResource individual = (OntResource)Instans.next();
                System.out.println("   "+individual.getLocalName());
            }
        }
    }
```

（7）查询哪些是“小刚”提供的构件

```
    String Indiv="xiaogang";  //查询的实例名
```

```
String queryString =
"PREFIX rdf: <http://www.owl-ontologies.com/Ontology1206595540.owl#> "+
"SELECT ?a "+
"WHERE{"+
"?a rdf:hasMadeBy rdf:"+Indiv+ "."+
" }";
//创建查询
Query query = QueryFactory.create(queryString);
//执行查询，获得结果
QueryExecution qe = QueryExecutionFactory.create(query,m);
ResultSet results = qe.execSelect();
for(;results.hasNext();)
{
    QuerySolution solution = results.nextSolution();
    Resource r=solution.getResource("a");
    System.out.println(r.getLocalName());
}
```

4.5.2 检索功能

本功能实现根据用户的输入查询语句，从本体中推理检索出满足用户需求的领域构件，实现对特定功能领域构件的语义检索。实现为 IBM 的 NetBeans 5.0+HP 的 Jena-2.5.4。

构件查询：查询构件，构件提供者为 xiaogang。

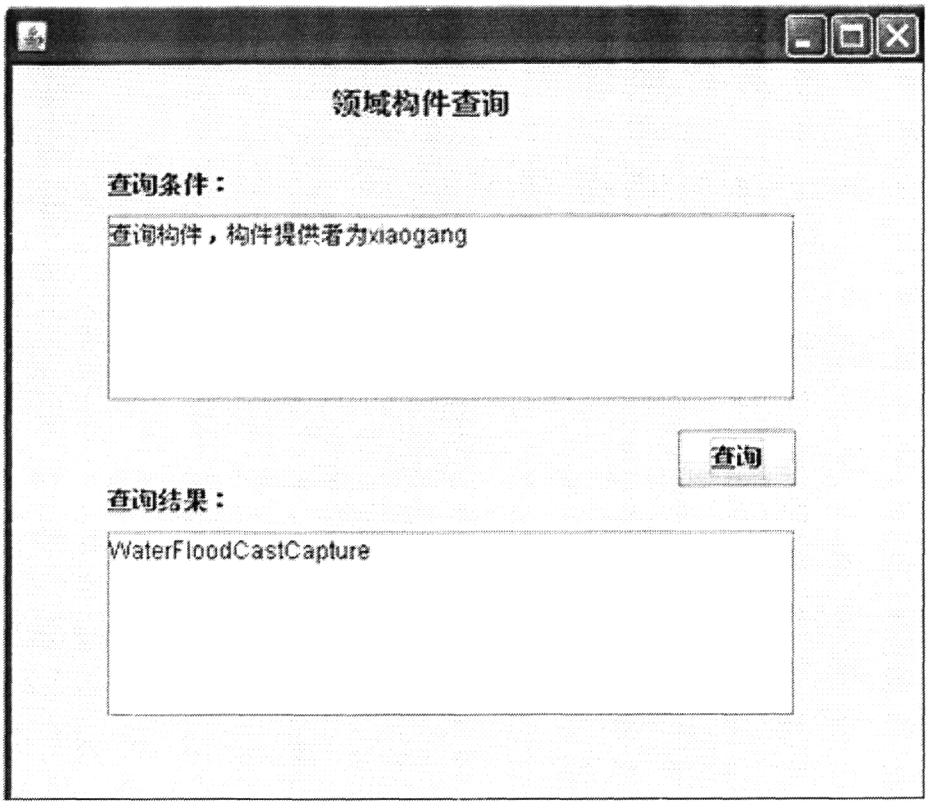

图 4.6 构件提供者为 xiaogang 的查询结果

4.6 检索结果分析

为了验证特定领域构件的刻面检索的效率以及在实践中的可行性，在本体构件库系统原型中做检索实验。假设查询者熟悉本体构件库系统，构件的检索效率用查准率和查全率两个指标来评价：

查全率=检索到的相关构件集合/库中所有相关构件集合；

查准率=检索到的相关构件集合/检索到的所有构件集合。

查询条件：

（1）查询构件，构件的服务为 FloodForecast，操作系统为 winMe。

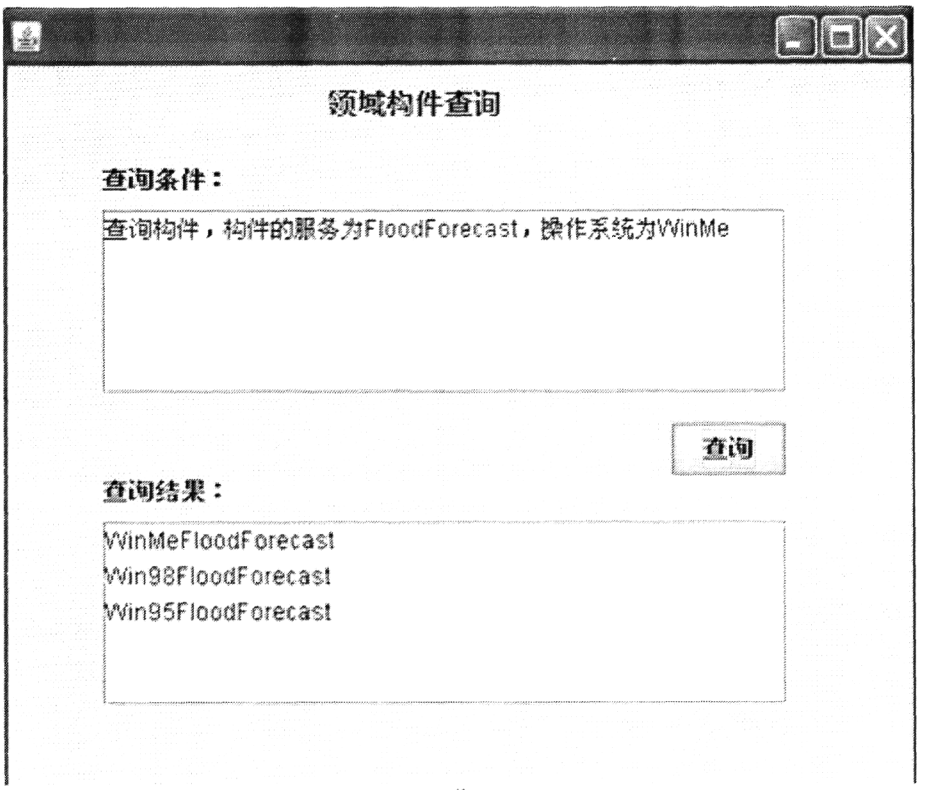

图 4.7 服务为洪水预报、操作系统为 WinMe 的查询结果

1）系统首先预处理得到 Ac：{ Server, OS}、A_{1I}：{ FloodForecast }，A_{2I}：{ WinME }；

2）Server 和 Component 有关系 A_{1P}：{ hasServer }，OS 和 Component 有关系 A_{2P}：{ hasOS }；根据根据 OS 的 hasFollow 关系对 OS：{ WinME }进行语义扩展得到 OS:{ WinME、Win98、Win95 }；

3）查询{*， hasOS，{WinME ∪ Win98 ∪ Win95} } ∩ {*， hasServer，FloodForecast }得到三个查询结果{ WinMeFloodForecast, Win98FloodForecast, Win95FloodForecast }；

4）返回结果。

检索结果分析：

本体构件库中存在 3 个洪水预报构件。

使用传统的构件库查询只能查询到“WinMeFloodForecast”构件，查全率为 1/3=0.33，查准率为 1；使用本体构件库能查询到上面显示全部的 3 个构件，查全率为 1，查准率为 1。

（2）查询构件，构件的处理对象为 Hydrology，表示形式为 Code。

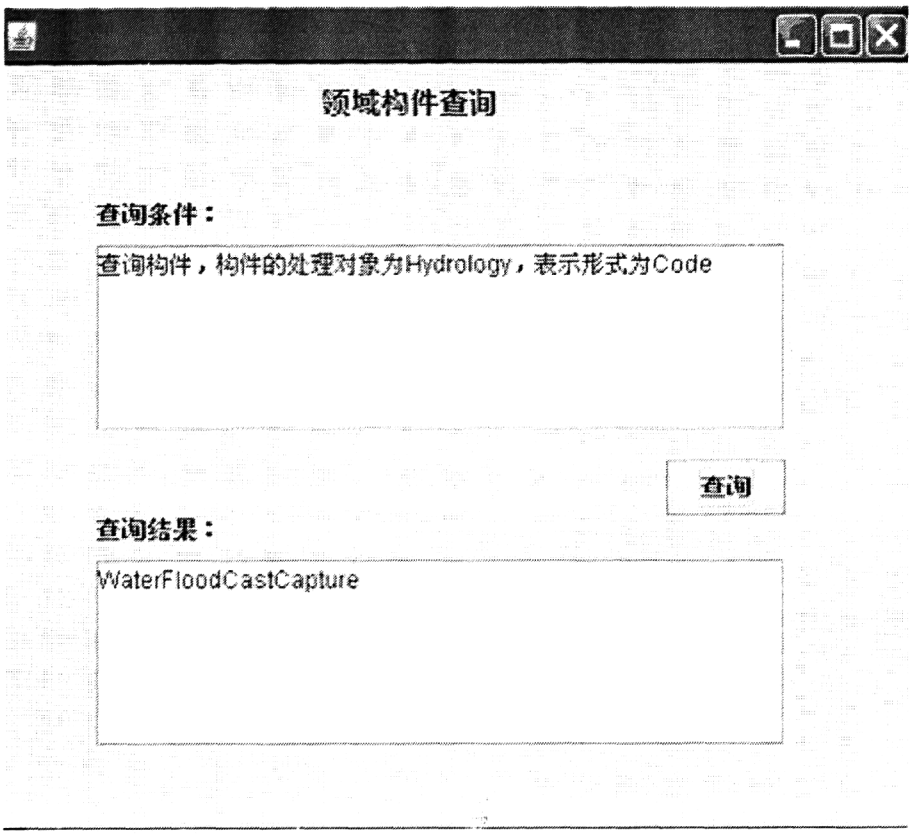

图 4.8 处理对象为 Hydrology、表示形式为 Code 的查询结果

1）系统首先预处理得到 Ac：{ObjectTreat, Presentation}、A_{1I}：{ Hydrology }，A_{2I}：{ Code }；

2）ObjectTreat 和 Component 有关系 A_{1P}：{ hasDomain }，Presentation 和 Component 有关系 A_{2P}：{ hasPresentation }；

3）查询{ *,hasDomain, Hydrology } ∩{ *，hasPresentation，Code }得到查询结果{ WaterFloodCastCapture }；

4）返回结果。

检索结果分析：

本体构件库中存有 4 个水情构件，表示形式分别为 Dll、Exe、Code、Graph。

使用传统的构件库查全率为 1，查准率为 1/4=0.25（传统的构件库中没有“处理对象”这个刻面，所以对“处理对象”这项是忽略的，实际只有“表示形式”为 Code 这一查询条件）；使用本体构件库查全率为 1，查准率为 1。

通过更多的查询比较分析，得出本体构件库比传统的构件库在查全率上有很大的提高，查准率在有处理对象参与检索的情况下也有很大的提高，但更一般的查询本体构件库比传统的构件库查准率有了一定程度的下降（本文的本体是针对水利领域构件构建的，用户的查询一般都有处理对象的条件，所以总体来说查准率还是比传统的构件库高），这一问题可以通过构件的排序实现，即进行语义扩展后，原始查询条件的查询结果，其排序最靠前，和原始查询条件关系越密切的

越靠前。

4.7 本章小结

本章对领域构件本体进行检索运用，通过 NetBeans 5.0 和 Jena 2.5.4 实现；给出本体中构件检索的查询流程和查询算法，并对检索结果进行简要的分析。

第五章 总结与展望

5.1 主要工作总结

概括来说，本论文的主要研究内容包括：

针对水利领域的构件，充分分析与挖掘了构件的语义信息，建立了本体构件库模型和水利领域构件的语义模型，参照七步法构建本体的方法，构建出水利领域构件的本体，然后使用 Protégé工具实现本体的 OWL 表示和存储，然后简要介绍了怎样复用本体，利用 HP 的 Jena 实现对构建的水利领域构件本体的构件查询实现。

具体来说有以下几点：

（1）通过对大量的相关科技文献资料的阅读与学习，对构件和本体的理论进行了系统的学习和总结。

（2）总结了国内外学者对人工智能与知识表示领域中本体的概念，本体的构成要素和本体的类型。对本体的作用和本体在水利领域应用的意义进行了研究。

（3）在进行领域构件本体创建时，充分参考已有文献描述的领域概念、属性和关系，尽量减少了领域专家参与的程度。

（4）对本体的创建方法、创建工具和本体表示语言进行了详细的学习。

（5）通过对本体理论研究及领域构件本体的创建，总结出了一套以刻面为基础创建领域构件本体的方法。

（6）选用水利领域构件为研究对象，初步创建了该领域的本体概念，为进一步实践打下了基础。

（7)给出了怎样利用 Jena 实现对 OWL 表示的本体的具体应用和操作方法。

5.2 工作展望

本文的研究仅仅是一个起点，关于领域构件本体的构建，后续要进行的工作还有很多。主要有以下几个方面：

（1）本文对水利领域构件的本体概念抽取的粒度有些偏大，下一阶段可以研究把本文的本体概念继续细分，提取出更多更细的概念，表达的概念间的关系可以更多更丰富。可以通过本体复用在其他本体中复用本文创建的构件类，实现概念类的复用，概念类的创建不用再重头开始。

（2）目前本文创建本体的方法是基于手工的，下一阶段可以向机器自动构

建的方向努力。

（3）可以把本文所建的水利领域构件本体和水利领域本体相结合，从而让构件本体得到更广泛的应用。

（4）由于本体具有形式化表示、共享和表示概念间语义关系的优势，可以参照本文构建其他领域的构件的本体，实现领域构件的形式化表示和语义共享，从而解决人机交互问题，用户的语义理解偏差和语义表述不一致问题。

（5）可以通过本体复用,复用该构件本体，使本体的创建不用从头开始。

致谢

在本论文完成之际，我要衷心的感谢我的导师陈金水老师在我的课题研究以及论文写作过程中所给予的悉心指导和帮助。本文从选题、理论研究、实验分析到论文撰写都得到了陈老师的悉心指导和无私帮助。陈老师在繁忙的工作中抽出大量时间关心我的课题进展，并对本论文进行了严格的审查和细致的修改。陈老师严谨求实的治学态度，忘我的工作和平易近人的师者风范将让我终生难忘，终生受益。特向陈老师表示衷心的感谢！

在三年的研究生学习中，得到了龚晓燕、沙婷婷、韩冰冰、洪姗姗、赵夏阳、劳莹莹、李煜、秦焕香、王兆勇、倪月芳、刘云、元绍华、纪诚、季叶飞、贺会景等众位学友的帮助和支持。在此，表示诚挚的感谢！

感谢舍友程志华，刘灏，王炜，三年来在对我生活中的帮助和学习中的指导。

感谢计信院05级软件班全体同学的关怀与帮助，让我感受到了集体的友爱。

感谢所有关心和帮助过我的计信院的老师们！

感谢我的父母，他们为我付出了很多，给了我很多很多无私的关怀和帮助。他们的关心和支持是我的坚强后盾，他们对我无私的爱永远是我积极生活、认真完成学业的动力！

感谢在百忙之中抽出时间审阅论文和参加答辩的各位老师，谢谢你们提出的宝贵建议。

参考文献：

[1] 杨芙清，梅宏，李克勤. 软件复用与软件构件技术〔J〕. 电子学报,1999,(2).

[2] 李延春，晏敏. 软件构件技术的现状与未来. 计算机工程与应用,2003 年 31 期.

[3] 杨斐，高芹. 计算机软件构件认知. 黄石理工学院学报,2005 年 8 月第 21 卷第 4 期.

[4] 贺秋芳. 基于构件的软件复用技术. 广东轻工职业技术学院学报,2005 年 03 期.

[5] 肖强华，李明东. 基于构件的软件开发技术和方法. 宜宾学院学报,2005 年 06 期.

[6] 刘绍华，魏峻. 软件构件化迁移. 软件世界,2005 年 08 期.

[7] 张海藩. 软件工程[M]. 北京:人民邮电出版社, 2002.

[8] 朱三元，钱乐秋，宿为民. 软件工程技术概论〔M〕. 北京:科学出版社,2001.

[9] 蔡梦. 软件构件技术〔M〕. 西安:西北工业出版社,2002.

[10] 闵道辉，李强，赵正文. 基于 UML 的软件构件技术. 西南民族大学学报(自然科学版),2005 年 03 期.

[11] 曹春萍，龚崇栋. 基于可复用构件的软件开发过程. 上海水产大学学报 ,2005 年 03 期.

[12] 唐彦. 基于本体的构件描述. 河海大学硕士论文,2005 年 6 月.

[13] 陈海林,潘孝铭. 软件构件技术研究. 福建电脑,2006 年第 8 期.

[14] 刘赟. 基于本体论的软构件语义检索研究. 华侨大学硕士学位论文,2007 年 10 月.

[15] 何琳. 古农学本体的半自动构建及检索研究.南京农业大学博士论文，2007 年 6 月.

[16] 陈建. 领域本体的创建和应用研究. 对外经济贸易大学硕士论文,2006 年 4 月.

[17] 李宗平. 基于本体论的构件表示与检索研究. 重庆大学硕士学位论文,2006 年 4 月.

[18] 李景. 本体理论及在农业文献检索系统中的应用研究——以花卉学本体建模为例. 中国科学院博士学位论文,2004 年 7 月.

[19] 陈颖，沈军. 基于本体的构件描述与检索. 计算机应用与软件,2007 年 7 月.

[20] 袁冬娟. 基于刻面描述的水资源领域的构件检索方法. 河海大学硕士学位论文,2007 年 3 月.

[21] 王渊峰，张涌，任洪敏，朱三元，钱乐秋. 基于刻面描述的构件检索. 软件学报,2002,13(8).

[22] 李岳蒙. 军用飞机领域本体的扩展和改进研究. 南京理工大学硕士学位论文,2006 年 6 月.

[23] 郭红艳. 基于本体论的多构件库检索技术研究. 华侨大学硕士学位论文,2006 年 12 月.

[24] 胡珉. 基于领域本体的知识获取和重用技术研究. 北京化工大学硕士学位论文,2006 年 5 月.

[25] Liu Quan, Jin Xinjuan, Long Yihong. Research on Ontology-based Representation and Retrieval of Components. Eighth ACIS International Conference on Volume 1, July 30 2007-Aug. 1 2007 Page(s):494 - 499.

[26] 王梅. OWL 领域本体构件方法研究. 图书情报工作,2006 年 12 月.

[27] Dagobert Soergel, Boris Lauser, Anita Liang, Frehiwot Fisseha, Johannes Keizer and Stephen Katz. Reengineering Thesauri for New Applications: the AGROVOC Example. http://jodi.tamu.edu/Articles/v04/i04/Soergel/ (access:May,12,2007).

[28] Jian Qin,Stephen Paling. Converting a controlled vocabulary into an ontology: the case of GEM. http://informationr.net/ir/6-2/paper94a.html (access:July,20,2007).

[29] Junmei Sun, Huaikou Miao, Xiaoxia Cao. A Domain Formal Ontology and the Application in Service Component Retrieval. Software Engineering Advances, International Conference on Oct. 2006 Page(s):33 – 33.

[30] 杨建伟, 张志平. 叙词表的 OWL 表示方法研究. 情报杂志,2007 年第 3 期.

[31] 费玉奎. Web 构件模型研究及其在水利领域中的应用. 河海大学博士学位论文,2005 年 3 月.

[32] Nanda J., Thevenot H.J., Simpson T.W.. Product family design knowledge representation, integration, and reuse. Information Reuse and Integration, Conf, 2005. IRI -2005 IEEE International Conference on.15-17 Aug. 2005 Page(s):32 – 37.

[33] Jeff Z. Pan, Luciano Serafini, Yuting Zhao. Semantic Import : An Approach for Partial Ontology Reuse. http://tcc.itc.it/projects/ontotext/Publications/WoMO-iswc06-serafini.pdf, access:Nov.10,2007.

[34] 彭鑫, 赵文耘, 钱乐秋. 基于领域特征本体的构件语义描述和组装. 电子学报,2006 年 12 月.

[35] 边小凡, 夏华轩, 尹胜斌. 基于领域本体的构件自动化组装原形系统. 计算机工程与设计,2006 年 11 月.

[36] 冯兰萍, 朱礼军, 张继国. 一种本体驱动的 Web 信息检索模型及实现. 情报学报,2006 年 6 月.

[37] 王青, 张春海. 基于构件的本体扩展工作流的构架研究与应用. 计算机工程与应用,2006 年 25 期.

[38] 杨明华, 钱乐秋, 赵文耘, 彭鑫. 特定领域本体的构造方法研究. 计算机工程,2006 年 11 期.

[39] 聂卉. 基于本体的查询扩展与规范. 现代图书情报技术,2007 年第 3 期.

[40] 李盛欣, 陈杰. 本体应用的开发研究. 湘南学院学报,2006 年 10 月.

[41] 王体春, 韩永国. 本体复用方法语义导入的研究与设计. 自动化与仪表,2008 年 01 期.

[42] Ma Yutao, He Keqing, Liu Wei, Tian Jingbai. A Grid-Oriented Platform for Software Component Repository Based on Domain Ontology. Services Computing, 2007. SCC 2007. IEEE International Conference on 9-13 July 2007 Page(s):628 – 635.

[43] Paik I., Akimoto H.. Ontology querying and its software component for design support in

supply chains. Information Technology and Applications, 2005. ICITA 2005. Third International Conference on Volume 1, 4-7 July 2005 Page(s):179 - 184 vol.1.

[44] 万捷，滕至阳. 本体论在基于内容信息检索中的应用. 计算机工程,2003 年 3 月.

[45] 廖明宏. 本体论与信息检索. 计算机工程,2000 年 2 月.

[46] 刘强. 构件库之构件检索与理论. 西北工业大学硕士学位论文,2003 年 3 月.

[47] Peng Xin, Zhao Wenyun. An Incremental and FCA-Based Ontology Construction Method for Semantics-Based Component Retrieval. Seventh International Conference on 11-12 Oct. 2007 Page(s):309 – 315.

[48] da Silva Jacinto A., Parente de Oliveira J.M.. Structuring Semantic Web ITS. Advanced Learning Technologies, 2006. Sixth International Conference on 05-07 July 2006 Page(s):920 - 922

附录：

1. OWL 语言表示的水利领域构件本体：

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns="http://www.owl-ontologies.com/Ontology1209647070.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.owl-ontologies.com/Ontology1209647070.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="服务">
    <owl:equivalentClass>
      <owl:Class rdf:ID="Server"/>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>
  <owl:Class rdf:ID="Component">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <owl:equivalentClass>
      <owl:Class rdf:ID="构件"/>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="ComponentForm">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <owl:equivalentClass>
      <owl:Class rdf:ID="构件形态"/>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="Presentation">
    <rdfs:subClassOf rdf:resource="#ComponentForm"/>
  </owl:Class>
  <owl:Class rdf:ID="Software">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="AppEnvironment"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#构件形态">
    <owl:equivalentClass rdf:resource="#ComponentForm"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>
  <owl:Class rdf:ID="Unix">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="OS"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="应用环境">
    <owl:equivalentClass>
      <owl:Class rdf:about="#AppEnvironment"/>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>
  <owl:Class rdf:ID="WaterResourceProtection">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Server"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Win9x">
    <rdfs:subClassOf>
```

```
      <owl:Class rdf:ID="Windows"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="作者">
    <owl:equivalentClass>
      <owl:Class rdf:ID="Author"/>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>
  <owl:Class rdf:ID="WaterVolDispatch">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Server"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="AbstractLevel">
    <rdfs:subClassOf rdf:resource="#ComponentForm"/>
  </owl:Class>
  <owl:Class rdf:about="#OS">
    <rdfs:subClassOf rdf:resource="#Software"/>
  </owl:Class>
  <owl:Class rdf:ID="HardDisk">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Hardware"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="DB">
    <rdfs:subClassOf rdf:resource="#Software"/>
  </owl:Class>
  <owl:Class rdf:ID="TreatObject">
    <owl:equivalentClass>
      <owl:Class rdf:ID="处理对象"/>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>
  <owl:Class rdf:about="#AppEnvironment">
    <owl:equivalentClass rdf:resource="#应用环境"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>
  <owl:Class rdf:ID="Memory">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Hardware"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="WinCE">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Windows"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="FloodPrevention">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Server"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#Author">
    <owl:equivalentClass rdf:resource="#作者"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>
  <owl:Class rdf:about="#Hardware">
    <rdfs:subClassOf rdf:resource="#AppEnvironment"/>
  </owl:Class>
  <owl:Class rdf:ID="ServeModel">
    <rdfs:subClassOf rdf:resource="#ComponentForm"/>
  </owl:Class>
```

```
<owl:Class rdf:about="#Windows">
  <rdfs:subClassOf rdf:resource="#OS"/>
</owl:Class>
<owl:Class rdf:ID="CompileSystem">
  <rdfs:subClassOf rdf:resource="#Software"/>
</owl:Class>
<owl:Class rdf:about="#构件">
  <owl:equivalentClass rdf:resource="#Component"/>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:ID="CPU">
  <rdfs:subClassOf rdf:resource="#Hardware"/>
</owl:Class>
<owl:Class rdf:ID="WaterSoilMaintenance">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Server"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="NetEnvironment">
  <rdfs:subClassOf rdf:resource="#Software"/>
</owl:Class>
<owl:Class rdf:ID="NTFamily">
  <rdfs:subClassOf rdf:resource="#Windows"/>
</owl:Class>
<owl:Class rdf:ID="Forecast">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Server"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Server">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <owl:equivalentClass rdf:resource="#服务"/>
</owl:Class>
<owl:Class rdf:about="#处理对象">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <owl:equivalentClass rdf:resource="#TreatObject"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasAuthor">
  <rdfs:range rdf:resource="#Author"/>
  <rdfs:domain rdf:resource="#Component"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasDB">
  <rdfs:range rdf:resource="#DB"/>
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:ID="hasSoftWareEnvironment"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasModel">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:ID="hasForm"/>
  </rdfs:subPropertyOf>
  <rdfs:range rdf:resource="#ServeModel"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasHardWareEnvironment">
  <rdfs:range rdf:resource="#Hardware"/>
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:ID="hasEnvironment"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasServer">
  <rdfs:domain rdf:resource="#Component"/>
  <rdfs:range rdf:resource="#Server"/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="hasCPU">
  <rdfs:subPropertyOf rdf:resource="#hasHardWareEnvironment"/>
  <rdfs:range rdf:resource="#CPU"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasDomain">
  <rdfs:domain rdf:resource="#Component"/>
  <rdfs:range rdf:resource="#TreatObject"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasPresentation">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#hasForm"/>
  </rdfs:subPropertyOf>
  <rdfs:range rdf:resource="#Presentation"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasNetEnvironment">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#hasSoftWareEnvironment"/>
  </rdfs:subPropertyOf>
  <rdfs:range rdf:resource="#NetEnvironment"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasEnvironment">
  <rdfs:range rdf:resource="#AppEnvironment"/>
  <rdfs:domain rdf:resource="#Component"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasComplie">
  <rdfs:range rdf:resource="#CompileSystem"/>
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#hasSoftWareEnvironment"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasHardDisk">
  <rdfs:range rdf:resource="#HardDisk"/>
  <rdfs:subPropertyOf rdf:resource="#hasHardWareEnvironment"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasSoftWareEnvironment">
  <rdfs:range rdf:resource="#Software"/>
  <rdfs:subPropertyOf rdf:resource="#hasEnvironment"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasMemory">
  <rdfs:range rdf:resource="#Memory"/>
  <rdfs:subPropertyOf rdf:resource="#hasHardWareEnvironment"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasForm">
  <rdfs:domain rdf:resource="#Component"/>
  <rdfs:range rdf:resource="#ComponentForm"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasOS">
  <rdfs:subPropertyOf rdf:resource="#hasSoftWareEnvironment"/>
  <rdfs:range rdf:resource="#OS"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasForecast">
  <rdfs:range rdf:resource="#Forecast"/>
  <rdfs:domain rdf:resource="#FloodPrevention"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasAbstract">
  <rdfs:range rdf:resource="#AbstractLevel"/>
  <rdfs:subPropertyOf rdf:resource="#hasForm"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="ComponentSize">
  <rdfs:domain rdf:resource="#Component"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="AuthorEmail">
```

```
    <rdfs:domain rdf:resource="#Author"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="AuthorName">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Author"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="ServerFunction">
    <rdfs:domain rdf:resource="#Server"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="ComponentEdition">
    <rdfs:domain rdf:resource="#Component"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="AuthorPhone">
    <rdfs:domain rdf:resource="#Author"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="ComponentMadeDate">
    <rdfs:domain rdf:resource="#Component"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="ServerProperty">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Server"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="ComponentAddress">
    <rdfs:domain rdf:resource="#Component"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="AuthorAddress">
    <rdfs:domain rdf:resource="#Author"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="ComponentName">
    <rdfs:domain rdf:resource="#Component"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="ServerKeyWords">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Server"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="ServerDescription">
    <rdfs:domain rdf:resource="#Server"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:TransitiveProperty rdf:ID="hasFollow">
    <owl:inverseOf>
      <owl:TransitiveProperty rdf:ID="hasPrevious"/>
    </owl:inverseOf>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:range rdf:resource="#OS"/>
    <rdfs:domain rdf:resource="#OS"/>
  </owl:TransitiveProperty>
  <owl:TransitiveProperty rdf:ID="hasVersion">
    <rdfs:domain rdf:resource="#Component"/>
    <rdfs:range rdf:resource="#Component"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:TransitiveProperty>
  <owl:TransitiveProperty rdf:ID="hasInclude">
    <rdfs:domain rdf:resource="#Component"/>
    <rdfs:range rdf:resource="#Component"/>
```

```
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:TransitiveProperty>
  <owl:TransitiveProperty rdf:ID="hasCollaboration">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:range rdf:resource="#Component"/>
    <rdfs:domain rdf:resource="#Component"/>
  </owl:TransitiveProperty>
  <owl:TransitiveProperty rdf:ID="hasRefine">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:range rdf:resource="#Component"/>
    <rdfs:domain rdf:resource="#Component"/>
  </owl:TransitiveProperty>
  <owl:TransitiveProperty rdf:about="#hasPrevious">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:range rdf:resource="#OS"/>
    <rdfs:domain rdf:resource="#OS"/>
    <owl:inverseOf rdf:resource="#hasFollow"/>
  </owl:TransitiveProperty>
  <NetEnvironment rdf:ID="Internet"/>
  <NTFamily rdf:ID="winXp">
    <hasFollow>
      <NTFamily rdf:ID="win2000">
        <hasFollow>
          <Win9x rdf:ID="WinMe">
            <hasFollow>
              <Win9x rdf:ID="Win98">
                <hasFollow>
                  <Win9x rdf:ID="Win95"/>
                </hasFollow>
              </Win9x>
            </hasFollow>
          </Win9x>
        </hasFollow>
      </NTFamily>
    </hasFollow>
  </NTFamily>
  <Memory rdf:ID="M256"/>
  <ServeModel rdf:ID="ClassLibrary"/>
  <WaterSoilMaintenance rdf:ID="WaterSoilDrains"/>
  <TreatObject rdf:ID="HumGeography"/>
  <WaterSoilMaintenance rdf:ID="InformationCollection"/>
  <Presentation rdf:ID="Graphi"/>
  <ServeModel rdf:ID="COM"/>
  <AbstractLevel rdf:ID="Others"/>
  <FloodPrevention rdf:ID="FloodDispatch"/>
  <Presentation rdf:ID="EXE"/>
  <AbstractLevel rdf:ID="Analyze"/>
  <AbstractLevel rdf:ID="Test"/>
  <WaterResourceProtection rdf:ID="MonitorManage"/>
  <Component rdf:ID="WaterFloodDispatch"/>
  <WaterResourceProtection rdf:ID="PollutionAcceptCapabilityAnalysis"/>
  <Component rdf:ID="Win95FloodForecast">
    <hasServer>
      <Forecast rdf:ID="FloodForecast"/>
    </hasServer>
    <hasOS rdf:resource="#Win95"/>
  </Component>
  <HardDisk rdf:ID="G40"/>
  <CPU rdf:ID="P4"/>
  <Presentation rdf:ID="Code"/>
  <AbstractLevel rdf:ID="Design"/>
  <DB rdf:ID="Oracle"/>
  <Author rdf:ID="xiaoming">
```

```
    <AuthorName rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >xiaoming</AuthorName>
    <AuthorEmail rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >xxx@163.com</AuthorEmail>
    <AuthorPhone rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >000-00000000</AuthorPhone>
    <AuthorAddress rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >nanjing,jiangsu,china</AuthorAddress>
  </Author>
  <FloodPrevention rdf:ID="FloodEvolution"/>
  <Author rdf:ID="xiaogang">
    <AuthorEmail rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >g@163.com</AuthorEmail>
    <AuthorPhone rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >025-88888888</AuthorPhone>
    <AuthorName rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >xiaogang</AuthorName>
    <AuthorAddress rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Hohai University</AuthorAddress>
  </Author>
  <ServeModel rdf:ID="JavaBean"/>
  <WaterResourceProtection rdf:ID="AnalysisAppraisal"/>
  <AbstractLevel rdf:ID="Coding"/>
  <CPU rdf:ID="P3"/>
  <ServeModel rdf:ID="DotNET"/>
  <DB rdf:ID="Sql"/>
  <CPU rdf:ID="P2"/>
  <Forecast rdf:ID="RainForecast"/>
  <FloodPrevention rdf:ID="InterLocution"/>
  <TreatObject rdf:ID="Hydrology"/>
  <HardDisk rdf:ID="G20"/>
  <HardDisk rdf:ID="G80"/>
  <WaterVolDispatch rdf:ID="WaterResourceMonitorManage"/>
  <Component rdf:ID="Win98FloodForecast">
    <hasServer rdf:resource="#FloodForecast"/>
    <hasOS rdf:resource="#Win98"/>
  </Component>
  <ServeModel rdf:ID="ActiveX"/>
  <Memory rdf:ID="G1"/>
  <WaterVolDispatch rdf:ID="DispatchPattern"/>
  <Presentation rdf:ID="Other"/>
  <Component rdf:ID="WinMeFloodForecast">
    <hasServer rdf:resource="#FloodForecast"/>
    <hasOS rdf:resource="#WinMe"/>
  </Component>
  <Component rdf:ID="WaterFloodCastCapture">
    <hasServer rdf:resource="#FloodForecast"/>
    <ComponentMadeDate rdf:datatype="http://www.w3.org/2001/XMLSchema#date"
    >2007-08-15Z</ComponentMadeDate>
    <ComponentName rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >WaterInfoCapture</ComponentName>
    <hasModel rdf:resource="#JavaBean"/>
    <hasEnvironment rdf:resource="#M256"/>
    <ComponentAddress rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >c:\</ComponentAddress>
    <hasHardDisk rdf:resource="#G40"/>
    <ComponentSize rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >25</ComponentSize>
    <hasOS rdf:resource="#win2000"/>
    <hasAuthor rdf:resource="#xiaogang"/>
    <hasAbstract rdf:resource="#Coding"/>
    <hasNetEnvironment>
      <NetEnvironment rdf:ID="LAN"/>
```

```
    </hasNetEnvironment>
    <hasPresentation rdf:resource="#Code"/>
    <hasEnvironment rdf:resource="#win2000"/>
    <ComponentEdition rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
    >1.0</ComponentEdition>
    <hasEnvironment rdf:resource="#P3"/>
    <hasDomain rdf:resource="#Hydrology"/>
    <hasDB rdf:resource="#Sql"/>
    <hasCPU rdf:resource="#P3"/>
  </Component>
  <Memory rdf:ID="M512"/>
  <Forecast rdf:ID="MeteorologyForecast"/>
  <TreatObject rdf:ID="WorkStatus"/>
  <ServeModel rdf:ID="CORBA"/>
  <TreatObject rdf:ID="DisaStatus"/>
  <WaterVolDispatch rdf:ID="InfoResearch"/>
  <Presentation rdf:ID="DLL"/>
  <WaterSoilMaintenance rdf:ID="WaterSoilMaintan"/>
  <TreatObject rdf:ID="NatGeography"/>
  <TreatObject rdf:ID="GovAffair"/>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.2.1, Build 365)  http://protege.stanford.edu -->
```

2. Jena 查询代码

①定义类的变量 m，下面的代码放在构造函数中，创建本体模型

```
m = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM, null );
String filePath="C:\\Program Files\\Protege_3.2.1\\OntologyComponent.owl";
try
{
    m.read(new FileInputStream(filePath), "");
}
catch(IOException ioe)
{
    System.err.println(ioe.toString());
}
```

②

```
public void getIndividuals()
    //获得实例，判断输入的查询语句中哪些词是实例
    {
        String result;
        String DealStr=this.jTextArea2.getText();
        String[] strLine=DealStr.split(", ");
        for(int i=0;i<strLine.length;i++)
        {
            for(int j=0;j<strLine[i].length();j++)
            {
                for(int k=strLine[i].length();k>j;k--)
                {
                    //jTextArea3.append(strLine[i].substring(j,k)+"\n");
                    if(m.getIndividual(OntologyUrl+strLine[i].substring(j,k))!=null)//得到类
和实例
                    {

if(m.getIndividual(OntologyUrl+strLine[i].substring(j,k)).isIndividual())
                        {
                            if(Indiv_str==null)
                            {

Indiv_str=m.getIndividual(OntologyUrl+strLine[i].substring(j,k)).getLocalName();
                            }
                            else
                            {
                                Indiv_str+="
```

```
"+m.getIndividual(OntologyUrl+strLine[i].substring(j,k)).getLocalName();
                                }
                            }
                        }
                    }
                }
            }
        }

③ public OntClass IndivToConcept(String IndivName)
    //通过实例字符串关键字找到实例所属的类
    {
        for(Iterator i=m.listClasses();i.hasNext();)
        {
            OntClass c = (OntClass)i.next();
            for (Iterator Instans = c.listInstances(); Instans.hasNext();)
            {
                OntResource individual = (OntResource)Instans.next();
            if(individual.getLocalName().equals(IndivName))
            {
                    return c;
            }
            }
        }
        return null;
    }

④public OntClass getNameToClass(String OntClassName)
    //通过类名找到类
    {
        OntClass singleOntClass = m.getOntClass(OntologyUrl+OntClassName);
    return singleOntClass;
    }

⑤public void getSemanticExtend(Resource os)
//进行语义扩展，例输入 win2000，得到 win2000 的兼容系统 winMe，win98，win95
{
        Resource resParm=null;
        if(os!=null)
        {
            for(StmtIterator y=os.listProperties();y.hasNext();)
            {
                Statement state=y.nextStatement();
                Property pro=state.getPredicate();
                Resource res=state.getSubject();
                if(pro.getLocalName().substring(0,3).equals("has"))
                {
                    String queryString = "PREFIX rdf:
<http://www.owl-ontologies.com/Ontology1209647070.owl#> "+
                                        "SELECT ?r "+
                                        "WHERE{"+
                                            "rdf:"+res.getLocalName()+"
rdf:"+pro.getLocalName()+" ?r" +
                                                " }" ;
                    Query query = QueryFactory.create(queryString);
                    QueryExecution qe = QueryExecutionFactory.create(query,m);
                    ResultSet results = qe.execSelect();
                    for(;results.hasNext();)
                    {
                        QuerySolution solution = results.nextSolution();
                        Resource rs=solution.getResource("r");
                        //semanticExtStr 为类的成员变量
                        semanticExtStr+=",";
```

```
                        semanticExtStr+=rs.getLocalName();
                        //System.out.println(rs.getLocalName());

                        resParm=rs;
                    }
                }
            }
        getSemanticExtend(resParm);
        }
    }

    ⑥public void getRelations()
     //得到全部类之间的关系
     {
        OntClass ComponentClass=getNameToClass("Component");
        String[] Individuals=Indiv_str.split(", ");
        for(int i=0;i<Individuals.length;i++)
        {
            relationShip(ComponentClass,IndivToConcept(Individuals[i]));
        }
     }

    ⑦public void relationShip(OntClass c1,OntClass c2)
     //得到单个类之间的关系
     {
        OntClass OntTemp=c2;
        int Exit_flag=0;
        while(OntTemp!=null)
        {
            for(Iterator y=c1.listDeclaredProperties(true);y.hasNext();)
            {
                OntProperty property=(OntProperty)y.next();
                OntResource rangClass=property.getRange();
                if(rangClass.equals(OntTemp))
                {
                    if(Relation_str==null)
                    {
                        Relation_str=property.getLocalName();
                    }
                    else
                    {
                        Relation_str+=", "+property.getLocalName();
                    }
                    Exit_flag=1;
                    break;
                }
            }
            if(Exit_flag==1)
            {
                break;
            }
            OntTemp=OntTemp.getSuperClass();
        }
    }

⑧public ResultSet ResearchComponent(String[] Indiv,String[] relat)   //查询构件
    {
        //创建查询语句
        String queryString = "PREFIX rdf:
<http://www.owl-ontologies.com/Ontology1209647070.owl#> "+
        "SELECT ?a "+
        "WHERE{";
        for(int i=0;i<Indiv.length;i++)
```

```
        {
            queryString += "?a rdf:"+relat[i]+" rdf:"+Indiv[i]+ "." ;
        }
        queryString += " }";
        //创建查询
        Query query = QueryFactory.create(queryString);
        //执行查询，获得结果
        QueryExecution qe = QueryExecutionFactory.create(query,m);
        ResultSet results = qe.execSelect();
        return results;
    }
```