文章编号:1001 - 9081(2010)02 - 0402 - 04

一种支持软件知识共享的本体模型研究

鲁 强[1,2], 陈 超[1], 王智广[1]
(1. 中国石油大学(北京) 计算机科学与技术系,北京 102249; 2. 中国石油大学(北京) 地球探测与信息技术北京市重点实验室,北京 102249)
(luqiang@ cup. edu. cn)

摘 要:为了支持分布式环境下项目团队成员之间软件开发知识有效的共享,需要对软件开发知识以及它们之间的关系进行分析和定义。根据软件开发知识的内容、特点以及它们之间的关系,对软件开发知识进行了形式化的表示,并创建了软件开发知识本体和软件开发知识本体规则。通过 protégé 和 Jena 实现对此本体的创建、存储和对此本体规则的 SPARQL 形式转换,据此形成知识本体库来支持软件开发知识共享。

关键词:知识共享;本体;软件开发知识
中图分类号: TP311.5 **文献标志码**:A

Research on ontology-based model for supporting software knowledge sharing

LU Qiang[1,2], CHEN Chao[1], WANG Zhi-guang[1]
(1. *Department of Computer Science and Technology, China University of Petroleum, Beijing* 102249, *China*;
2. *Beijing Key Laboratory of Earth Prospecting and Information Technology, China University of Petroleum, Beijing* 102249, *China*)

Abstract: For supporting Software Development Knowledge (SDK) sharing among members in the distributed team, SDK and its relationship must be analyzed and defined. In the paper, the formalized representation of SDK was shown based on SDK's content, characteristics and relationship. Then, the SDK ontology and its rules were described. The ontology database in which SDK ontology and its rules were created, transformed and implemented by using protégé, SPARQL and Jena, was built in order to support SDK sharing.

Key words: knowledge sharing; ontology; Software Development Knowledge (SDK)

0 引言

随着软件规模的增长和网络社区的出现,软件开发活动日益呈现全球化。一个大型公司(组织)下的软件项目经常需要分布在不同地方的软件开发团队成员合作来进行分布式地开发。由于不同开发团队(成员)的知识结构不同,在针对某个问题的理解和实现方法会有所不同,从而导致已经开发的软件产品模块和代码很难被其他人理解和重用。虽然可以通过 E-mail、MSN、视频会议等通信方式来进行沟通,但是由于这种分布开发方式下团队成员的分布性和流动性,使得这种知识交流方式很难有效地、快速地开展。

为了解决软件重用问题,产生了领域工程[1],以领域模型和领域体系结构为基础来识别、组织领域中可重用的软件知识[2]。为此产生了多种领域工程方法,如 FODA[3]、Software Product Line[4]、青鸟领域工程方法[5]等。然而,这些方法关注的是如何从领域应用工程里面抽象出通用的领域模型以供以后重用,而不是关注如果更加有效地组织和共享已经形成的软件制品及其中的知识。为了增强团队知识共享,诸葛海先后提出知识流模型[6]和知识空间模型[7]来收集、整理和组织分布式环境下的成员知识。我们通过建立团队知识共享模型和操作[8-9]以支持分布式团队成员之间的知识共享。当将此模型应用到软件团队中进行软件开发知识共享时,发现以前建立的本体模型只是简单地描述了项目知识本体和其他辅助类型知识本体,并通过标注来描述它们之间的静态关系,并没在进一步对软件开发知识进行分类细化,也没有对它们之间的关系、规则进行细化,因此不能够很好地满足软件知识共享要求,这是由于软件开发知识具有易变性、相关性、依赖性、适用性、离散性等特点。

易变性 由于系统需求和功能要求经常会发生变化,因此软件开发知识也要作出相应的变化。

相关性 由于软件系统是一个整体,其中涉及到的开发知识必定会通过一定的业务逻辑整合起来,因此这些软件开发知识之间是业务逻辑相关的。

依赖性 软件开发是系统工程,各个步骤之间的知识具有依赖的特性。例如,要进行代码的编写,需要有软件功能需求。

适用性 软件开发知识都具有一定的适用环境,离开其操作环境,并不能发挥应有的作用。例如设计模式中的抽象工厂模式可以指导创建对象的方法,但是不能描述对象之间的相互调用和功能组合。

离散性 一条软件开发知识可能涉及到软件系统中不同部分的内容,它是由离散的各个部分内容组合而成的。

为了支持软件开发知识的共享,需要根据软件开发知识以上特性对软件开发知识作出形式化描述,使得分布式开发环境下的开发人员能够共享其开发软件的知识。本文首先在分析软件知识的种类和特点的基础上,来建立对这些知识的形式化描述,然后以此为基础建立相应的软件开发知识本体及其规则,最后通过 protégé 和 Jena 进行了本体库的实现。

1 软件开发知识

将软件开发知识(Software Development Knowledge, SDK)定义为软件开发过程中系统分析、设计和测试人员所用到的

收稿日期:2009 - 07 - 24;**修回日期**:2009 - 09 - 08。 **基金项目**:国家自然科学基金资助项目(60072006)。

作者简介:鲁强(1977 -),男,河北唐山人,副教授,博士,CCF 会员,主要研究方向:Semantic Web、知识工程、分布式系统; 陈超(1987 -),男,江西南昌人,硕士研究生,主要研究方向:知识工程; 王智广(1964 -),男,内蒙古通辽人,教授,CCF 高级会员,主要研究方向:开放式分布计算。

与软件系统有关的显性知识。这些知识应该以文件或数据的形式进行保存。

1.1 软件开发知识内容

软件开发过程大体包括系统分析阶段、系统设计开发阶段和系统测试阶段。每个阶段涉及的知识内容是不同的,特点也是不同的。这些知识的具体表述如下:

1)业务知识(BK)。指的是开发指定软件系统需要涉及到的某种工作岗位的业务流程和业务标准,包括工作流程手册、规章制度、行业标准、协议标准等文件。

2)需求知识(RK)。由系统分析人员根据用户要求来进行构建,用来描述设计软件必须满足的业务要求知识。需求知识的建立依赖于对业务知识的了解,其以需求文档的形式存在。

3)系统环境知识(SK)。是指软件系统开发和运行的环境知识,包括软/硬件使用说明书、网页、影音文件等文件。系统环境知识在软件开发中是基础性知识,它被其他类型的知识所使用。

4)设计知识(DK)。是指设计人员进行软件系统设计时使用和产生的知识。使用的知识称为设计元知识(Design Meta Knowledge,DMK),是实际系统设计的抽象。产生的知识称为设计案例知识(Design Case Knowledge,DCK),是具体系统的设计内容。DMK 描述的是设计抽象的设计方案,不随设计项目的更换而改变,例如设计模式、软件体系结构等。DCK 以软件项目设计文档的形式存在,它依赖于需求知识、业务知识、系统环境知识和 DMK 等。

5)程序知识(PK)。指的是软件程序编写方面的知识,它包括程序设计语言、程序算法、数据结构、程序设计方法等方面的知识。程序知识以多种形式的文档格式存在,具有较强的通用性,不随软件项目的更换而改变。它是软件开发知识中的基础性知识。

6)代码知识(CK)。是根据软件项目设计知识和程序知识而最终形成的代码文件。代码知识是以上述各种知识为基础综合作用形成的。

7)测试知识(TK)。是针对设计阶段形成的代码知识进行测试的知识。测试知识分为两种类型:一种为针对具体软件开发项目的测试方案知识(Test Case Knowledge,TCK),包括测试方案和测试结果等形式的文档;另一种为测试元知识(Test Meta Knowledge,TMK),包括各种通用的测试方法、理论和模型等。

1.2 软件开发知识之间关系

根据上面的 SDK 的内容描述,它们之间的关系如图 1 所示。

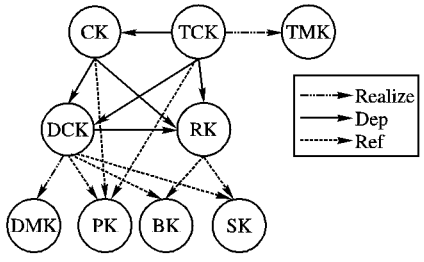

图 1　知识关系图

其中,Ref 关系表示知识之间的引用,Dep 关系表示知识之间的依赖,Realize 关系表示知识之间的实现,这些关系都具有方向性。Ref 与 Dep 关系虽然都表示一种知识使用了另一种知识,但是它们表示的知识之间关系强度是不同的。Ref 表示被引用知识只是对引用知识内容的一种间接影响,没有被引用知识也有可能不会影响引用知识内容,而 Dep 表示被依赖的知识是依赖知识出现或发生作用的前提条件,没有被依赖的内容一定不会有依赖的内容。Realize 关系表示知识之间的实现关系类似于面向对象方法中的"isA"关系,例如一个项目中对象创建方案与设计模式中抽象工厂模式之间是"isA"关系。

2 软件开发知识的形式化表示

2.1 软件知识形式化分析

通过对 SDK 的分析,SDK 可以分为两种类型:一种是相对比较稳定的知识,包括业务知识、程序知识、系统环境知识、设计元知识和测试元知识,这种类型的知识是形成具体软件系统的基础知识,称为软件开发基础知识(Software Development Base Knowledge,SDBK);另一种是根据项目不同而发生变化的知识,包括需求知识、设计方案知识、代码知识和测试方案知识,这种类型的知识是软件开发过程中最终形成的知识成果,称为软件成果知识(Software Development Product Knowledge,SDPK)。将 SDK 定义为:

$$SDK = \{SDBK,\ SDPK\}$$

$$SDBK = \{BK,\ SK,\ PK,\ DMK,\ TMK\}$$

$$SDPK = \{RK,\ DCK,\ TCK,\ CK\}$$

SDBK 类型的知识可以细化为两种类型:一种是具有"普适"的知识内容,这种类型的知识作为知识库中的基本知识内容被其他知识无条件地引用或使用,类似于辞典中的知识条目,不受引用知识所处的条件、环境等因素影响。这种类型的知识包括 *BK*、*PK* 和 *SK*。将这种类型的知识内容成为知识条目 *kitem*,定义为:

$$kitem = (kitemId,\ kname,\ ktype,\ kcontent,\ kfile,\ kpos);$$

$$ktype \in \{BK, PK, SK\}$$

其中:*kitemId* 表示知识条目的 ID 号,*kname* 表示知识名称,*ktype* 表示某种知识类型,*kcontent* 表示知识条目内容,*kfile* 和 *kpos* 分别表示知识所在的文件及其在文件中位置。

另一种是具体指定适应环境的知识内容,这种类型的知识只在特定的条件或环境中发生作用,包括 *DMK* 和 *TMK*。将这种类型知识称为 *kpattern*。根据设计模式的描述方式和文献[10]中的模式本体,将其表示为:

$$kpattern = (kpatternId,\ kname,\ kdtype,\ kcontext,\ problem,\ forces,\ solution,\ example,\ knowcase,\ relatedpatterns,\ kfile,\ kpos);$$

$$kdtype \in \{DMK,\ TMK\},\ relatedpatterns = kpatterm^{+}$$

其中:*kcontext* 表示知识上下文环境,*problem* 表示知识所能够解决的问题,*forces* 表示应用知识后所产生的效果,*solution* 表示知识具体的描述内容,*example* 表示实现例子代码,*knowcase* 表示此类型知识应用于实际中知识实例,*relatedpatterns* 表示同此类型知识相关的知识。

SDPK 中的知识通过具体项目联系在一起,它们存在于项目产品的各个文档中。将 SDPK 类型的知识定义为:

$$pitem = (pid, kid,\ ptype,\ kptype,\ kname,\ kcontent,\ kfile,\ kpos, kver)\ kptype \in SDPK$$

其中:*pid* 表示项目的标识,*ptype* 表示项目类型,*kptype* 表示 *SDPK* 的知识类型,*kver* 表示知识的版本。

SDBK 中的知识反映的是领域知识,这些知识按照领域中知识体系框架进行分类。针对软件知识,选择 ACM(Association for Computing Machinery)的 ACM-CCS 来进行知

识分类,如图 2 所示。这些领域知识基本上是按照目录结构来进行分类的,表示领域知识概念的逐层细化,描述的是知识概念之间的相似关系。将这种关系表示为 $isA(a,b)$,表示 a 是 b 的子类知识,其中 $a, b \in SDBK$。isA 具有传递性。如果一个知识 a 在 isA 的传递性质得到知识 b,则称 a 和 b 具有直接 isA 关系,否则是间接 isA 关系。

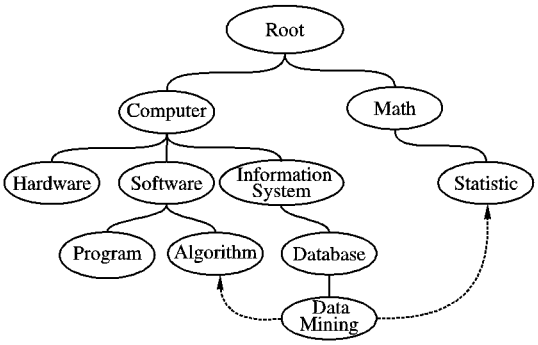

图 2 SDBK 关系图

由于 SDBK 中知识之间是相互联系的,某些联系并不能用 isA 关系来表示,例如数据挖掘需要算法和数理统计相关的内容,但数据挖掘同算法和数理统计之间并不具有直接的 isA 关系。将这种不具有直接 isA 关系的联系称为相关关系,即 $corr(a,b)$,用来表示知识 a 同知识 b 具有相关的联系,$a, b \in SDBK$。相关关系具有传递性。

SDPK 中的知识通过具体项目联系在一起,它们之间具有依赖关系,即 $dep(a,b)$ 表示知识 a 依赖于知识 b,其中 $a,b \in SDPK$。依赖关系具有传递性。

$SDPK$ 与 $SDPK$ 类型知识之间的关系有 ref 和 $realize$ 两种关系。其中 $ref(a,b)$ 表示知识 a 引用知识 b 的内容,其中 $a \in SDBK$, $b \in SDPK$。$realize(a,b)$ 表示知识 a 实现知识 b,其中 $a \in DCK$ or TCK, $b \in DMK$ or TMK。由此将软件知识中的关系定义为:

$$rel = \{isA, corr, dep, ref, realize\}$$

$$isA(a,b) = \{(a,b) \mid a \text{ is subconcept } b,\ a,b \in SDBK\}$$

$$corr(a,b) = \{(a,b) \mid a \text{ has } a \text{ some relation about } b,\ a,b \in SDBK\}$$

$$dep(a,b) = \{(a,b) \mid \text{if } b \text{ does not exist, } a \text{ can not bing, } a.pid = b.pid,\ a,b \in SDPK\}$$

$$ref(a,b) = \{(a,b) \mid b \text{ is appear in } a,\ a \in SDBK,\ b \in SDPK\}$$

$$realize(a,b) = \{(a,b) \mid a \text{ realizes meta-knowledge } b \text{ in project } a, a \in DCK \text{ or } TCK,\ b \in DMK \text{ or } TMK\}$$

基于以上关系运算可以定义为:

$$rel.k' = \{k \mid rel(k,k'), rel \in Rel\}$$

$$rel^-.k = \{k' \mid rel(k',k), rel \in Rel\}$$

其中,rel^- 为 rel 关系的逆运算。

2.2 软件开发知识本体

为了有效地支持知识的查找和共享,使用本体将上述知识及其关系进行形式化表示,如图 3 所示,其中圆形表示的是含有属性的 class,长方形表示指定数值类型的 class,它们之间的连线表示本体属性。其中,将 $kpattern$ 中属性 $knowcase$ 转换为关系 $realize$ 来进行表示,属性 $relatedpattern$ 可使用作用于 $SDBK$ 中的 $corr$ 关系来进行替代。

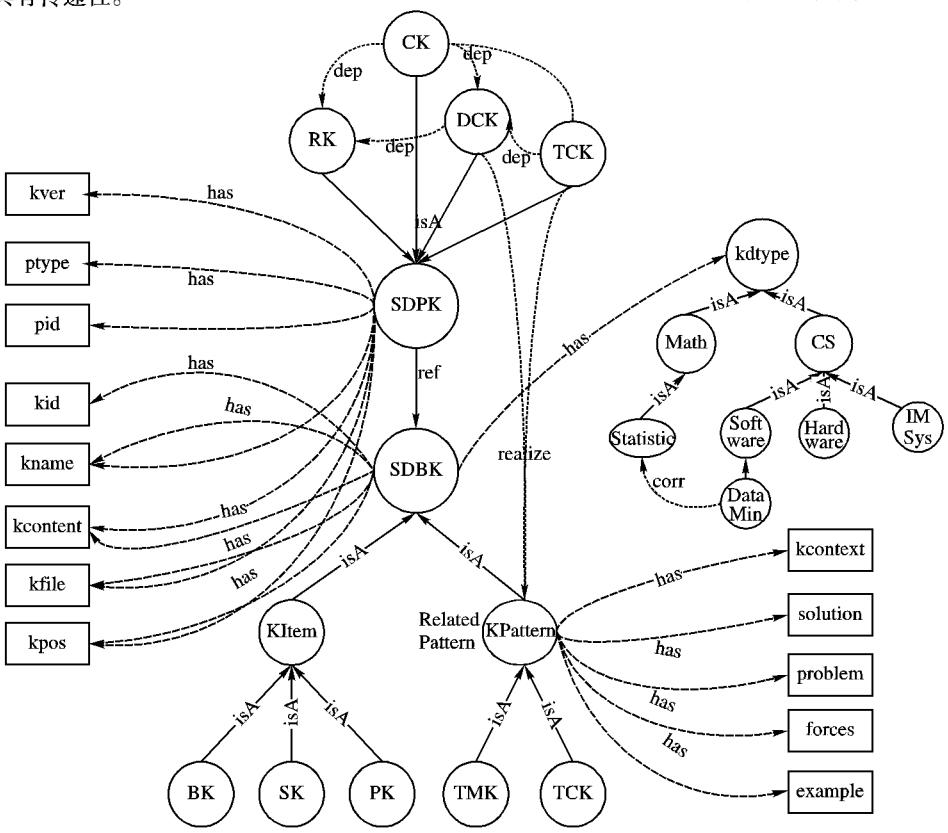

图 3 软件知识本体关系图

2.3 规则

为了明确和约束软件知识本体之间的关系,需要定义规则来增强使用软件开发知识 Agent 之间的相关协作。根据软件开发活动的不同,将规则包括以下两种类型基本关系规则(BRule)和知识共享规则(SRule)。

$$Rule = \{BRule, SRule\}$$

2.3.1 BRule

SDPK 中的知识内容是以项目来组织在一起的,因此同一项目中的知识具有相同的 pid,即 $Rl_{dep\text{-}id}$。如果知识的 kid 相同,则表示的是同一个软件开发知识内容,它们之间可以通过

版本或者知识类型等属性进行区分,即 $Rl_{same\text{-}pid}$。如果项目中被依赖的知识内容发生变化,则依赖的知识内容也要发生变化,即规则 $Rl_{dep\text{-}change}$。对于具有传递性质的关系,为了规定传递的范围定义了规则 Rl_t。即

$$BRule = \{Rl_{dep\text{-}id}, l_{same\text{-}pid}, Rl_{dep\text{-}change}, Rl_t\}$$

其中:$Rl_{dep\text{-}id}$ 表示 $dep(a,b) \rightarrow a.pid = b.pid$;$Rl_{same\text{-}pid}$ 表示 $a, kid = b.kid \rightarrow a.pid = b.pid$;$Rl_{dep\text{-}change}$ 表示 $change(b), dep(a,b) \rightarrow change(a)$;$Rl_t$ 表示 $k' = R^l.k$ or $k' = R^{-l}.k$, $l \leqslant n$, $R \in Rel$, $R^l.k = \underbrace{R.(R\cdots(R.k))}$。

2.3.2　SRule

为了得到同指定知识相关的知识,需要规定知识关系运算的规则。使用"{}"表示运算优先级,处在"{}"内部的运算内容具有较高优先级,Rl_{union} 表示多个规定得到结果的并集运算,$Rl_{intersection}$ 表示多个规则得到结果的交集运算,Rl_{co} 描述多个规则同时作用在一个知识上的运算,Rl_{seq} 表示多个规则按顺序依次作用于一个知识的运算。即:

$$SRule = \{"\{\}", Rl_{union}, Rl_{intersection}, Rl_{co}, Rl_{seq}\}$$

其中:$Rlunion$ 表示 $rl_1(k_1) \cup rl_2(k_2) \cup \cdots \cup rl_n(k_n)$, $rl_n \in Rule$, $k_n \in SDK$;$Rlintersection$ 表示 $rl_1(k_1) \cap rl_2(k_2) \cap \cdots \cap rl_n(k_n)$, $rl_n \in Rule$, $k_n \in SDK$;Rl_{co} 表示 $rl_1 \circ rl_2 \circ \cdots \circ rl_t(k) = rl_1(k) \cap rl_2(k) \cap \cdots \cap rl_{2n}(k)$, $rl_n \in Rule$, $k \in SDK$;$Rlseq$ 表示 $rl_1 \times rl_2 \times \cdots \times rl_t(k) = rl_1.\{rl_2.\{\cdots\{rl_t(k)\}\}\}$, $rl_n \in Rule$, $k \in SDK$。

运算符"∪","∩"和"∘"具有交换性,例如 $rl_1 \circ rl_2(k) = rl_2 \circ rl_1(k)$,而"×"不具有交换性。

3　实现

在实现过程中,使用 Protégé 工具来建立 OWL-DL 标准的 SDK 本体,如图 4 所示。通过 Jena 将此本体文件导入到文献[8]的知识共享模块本体库中,以扩充原来知识本体的类别和关系。由于 OWL-DL 限制,以上定义的规则形式不能被表示成本体的公理。因此,对于 BRule 类型的规则可以之间转换为函数来进行实现。例如 Rl_t. 规则对应于下面的函数。

```
Func Rl_t. (a,,Rl, l){                    //Rl 是表示关系的函数
  If( l = = 0 ) Return k;
  else{ k = Rl(a);l --; Rl_t. (a,,Rl, l);}
  }
```

对于 SRule 规则可以被转换为 SPARQL,并用 Jena 中的 ARQ 引擎来实现。即:

Rl_{union} →select $?k_1$, $?k_2$, ⋯, $?k_n$ where
{ GroupGraphPattern $\_k_1$ (' UNION' GroupGraphPattern $\_k_n$) * }

$Rl_{intersection}$ →select $?k_1$, $?k_2$, ⋯, $?k_n$ where
'{' TriplesBlock $\_k_1$? ((GraphPatternNotTriples $\_k_1$ | Filter $\_k_1$) '.'? TriplesBlock $\_k_n$?) * '}'

Rl_{co} → select $?k$ where
'{' TriplesBlock $\_k$? ((GraphPatternNotTriples $\_k$ | Filter $\_k$) '.'? TriplesBlock $\_k$?) * '}'

Rl_{seq} →select $?k_n$ where from where
{ $?k_1$ PropertyListNotEmpty_1 $?x$.
$?k_2$ PropertyListNotEmpty_2 $?k_1$ ⋯
$?k_n$ PropertyListNotEmpty_n $?k_{n-1}$ }

其中,GroupGraphPattern, TriplesBlock, GraphPatternNotTriples, PropertyListNotEmpty 等形式定义分别参见 SPARQL 的语法规则 20, 25 和 33。

根据上面描述的方法来进行软件开发知识本体构建及其上的推理规则实现,能够在描述软件开发知识基础上进行关系运算,以满足软件开发知识的易变性、离散性、依赖性等特点。例如,如果一个软件开发知识 a 发生了变化,能够根据 Func Rl_t. 函数得到所有受影响的知识,并返回给用户,使得用户能够马上判断修改所带来的影响。在此本体实现框架基础上,能够满足关系扩充的需求,如果增加一个关系,只需增加与此关系对应的规则运算函数即可。

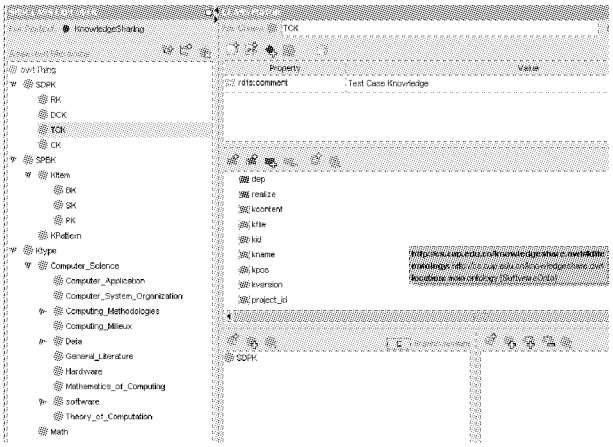

图 4　Protégé 下 OWL 格式的知识本体

4　结语

在本文中详细分析了软件开发知识中各种知识类别及其之间的关系,并定义了它们的形式化表示,据此建立了软件开发知识本体以及本体上的基本关系规则和共享规则。通过 protégé 本体构建工具和 Jena 库来构建文献[8]中分布式开发团队的软件开发知识本体库,以支持分布式开发环境下的软件开发知识共享。

在将来的工作中,要研究软件知识本体同软件开发方法之间的关系、规则以及实现,使得知识共享系统能够根据项目团队选择的开发软件方法,对软件开发知识进行约束,并向项目成员提供适当的软件开发知识。

参考文献:

[1] ARANGO G. Domain analysis: from art form to engineering discipline[J]. ACM SIGSOFT Software Engineering Notes, 1989, 14(3):152 - 159.

[2] 梅宏, 张伟. 领域工程——实现软件复用的有效途径[J]. 中国计算机学会通讯, 2007,3(11): 17 - 25.

[3] KANG K C, COHEN S G, HESS J A, *et al*. Feature-oriented domain analysis (FODA) feasibility study[EB/OL]. [2009 - 06 - 20]. http://wwwiti.cs.uni-magdeburg.de/iti_db/lehre/epmd/2008/bib/foda.pdf.

[4] CLEMENTS P, NORTHROP L. Software product lines: Practices and patterns[M]. Massachusetts: Addison-Wesley, 2001.

[5] 李克勤, 面向对象的领域工程方法研究[D]. 北京: 北京大学, 2000.

[6] ZHUGE HAI. Knowledge flow management for distributed team software development[J]. Knowledge-Based Systems, 2002, 15(8): 465 - 471.

[7] ZHUGE HAI. A knowledge grid model and platform for global knowledge sharing[J]. Expert Systems with Application, 2002, 22(4):313 - 320.

[8] 鲁强, 陈明. 一种基于本体的团队开发知识模型[J]. 计算机工程, 2006, 32(3):193 - 195.

[9] 陈明, 鲁强. 基于团队知识共享模型的知识订阅和发布机制研究[J]. 计算机工程, 2007, 33(16):32 - 34.

[10] GIRARDI R, LINDOSO A N. An ontology-based knowledge base for the representation and reuse of software patterns[J]. ACM SIGSOFT Software Engineering Notes, 2006, 31(1):1 - 6.