CN43-1258/TP
ISSN 1007-130X
计算机工程与科学
COMPUTER ENGINEERING & SCIENCE
2007 年第 29 卷第 9 期
Vol. 29, No. 9, 2007

文章编号:1007-130X(2007)09-0037-04

角色访问控制中基于描述逻辑的角色互斥实现*

Mutual Exclusion of Roles and Its Ontological Implementation in Role-Based Access Control Systems Based on Description Logic

孙小林,卢正鼎,李瑞轩,王治刚,陈新华
SUN Xiao-lin, LU Zheng-ding, LI Rui-xuan, WANG Zhi-gang, CHEN Xin-hua
(华中科技大学计算机科学与技术学院,湖北 武汉 430074)
(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

摘 要:本文基于描述逻辑的本体技术在不同的系统中利用相同的分布式的词汇表来实现知识的共享,从而达到不同自治域的角色权限的统一。此外,由于角色互斥是角色访问控制中至关重要的限制之一,本文利用描述逻辑的语法完成了角色互斥的本体实现。

Abstract: In this paper, knowledge sharing is realized by the same distributed vocabulary, which is utilized by the ontology based on description logic in different systems. It leads to the unification of the roles and authorities in multi- autonomy domains. Besides, because the mutual exclusion of roles is one of the most important restrictions in role-based access control systems, we achieve the ontological mutual exclusion of roles by the grammar of description logic.

关键词:角色访问控制;本体;角色互斥;描述逻辑
Key words: role-based access control; ontology; mutual exclusion of roles; description logical
中图分类号:TP309 **文献标识码**:A

1 引言

起源于哲学的本体论(Ontology)近年来受到信息科学领域的广泛关注[1,2],其重要性也已在许多方面表现出来并得到广泛认同[1,3,4]。然而本体描述语言要走向通用,还需解决一些重要问题,如正规和充足的语义表示机制,以及标准化问题。这将依靠下述基于描述逻辑的本体语言的发展。描述逻辑(Description Logics,简称 DL)[5]是近 20 多年来人工智能领域研究和开发的一个相当重要的知识表示语言,目前正被积极应用于本体描述,或者作为其他本体描述语言的基础。2002 年 7 月,W3C 在提交的 DAM+OIL 基础上发展 OWL 语言,以使其成为国际通用的标准本体描述语言。OWL 建立在 XML/ RDF 等已有标准基础上,通过添加大量的基于描述逻辑的语义原语来描述和构建各种本体。

随着网络技术的发展,大型网络应用系统或数据管理系统所面临的一个难题就是日益复杂的数据资源的安全管理,常用的自主访问控制(Discretionary Access Control)和强制访问控制(Mandatory Access Control)方法都是由主体和访问权限直接发生关系,根据主体/客体的所属关系或安全级别来决定主体是否有对客体的访问权。然而大型网络应用系统的用户往往种类繁多、数量巨大,并且访问权限动态变化,使得用传统的访问控制方法进行安全管理变得非常困难且不合实际。基于角色的访问控制 RBAC(Role-Based Access Control,简称 RBAC)方法引入角色这个中介,安全管理人员根据需要定义各种角色,并设置合适的访问权限,而用户根据其责任和资历再被指派为不同的角色。这样,整个访问控制过程就分成了两部分,即访问权限与角色相关联,角色与用户相关联,从而实现了用户与访问权限的逻辑分离。正是由于用户和访问权限的逻辑分离使得权限管理变得非常方便。同时描述角色层次关系的能力加

* 收稿日期:2006-06-01;修订日期:2006-10-23
基金项目:国家自然科学基金资助项目(60403027)
作者简介:孙小林(1980-),男,湖北黄石人,博士生,研究方向为数据挖掘、本体论与语义网等。
通讯地址:430074 湖北省武汉市华中科技大学计算机科学与技术学院 卢正鼎;Tel:(027)87544285;E-mail:arthursun.2006@gmail.com
Address: School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, P. R. China

强，有利于更好地实现最小权限(Least Privilege)原则。

然而在实际的大型应用系统中，系统安全策略要求用户不能超出他的职位应有的访问权限，而且要尽量减少同一个用户同时分配两个角色后产生欺骗行为的可能。

2 RBAC中的角色互斥

RBAC可追溯到早期的存取控制系统。从外表看来，它很像存取系统中用户组的概念，但在角色与用户组织间存在着很大的差别[6]。

2.1 RBAC角色概念

RBAC中的角色与实际的角色概念有所不同。在一个RBAC模型中，一个用户可以被赋予多个角色，一个角色可以对应多个用户，这些角色是根据系统的具体实现来定义的；同样，一个角色可以拥有多个权限，一个权限可以被多个角色所拥有。RBAC中角色的基本概念有两方面的含义：用户获得一个角色以及赋予角色以特权和许可。在实际应用中，一些特殊情况导致了角色的分层、基数限制和互斥限制等很多情况，我们在接下来的部分进行较详细的讨论。

2.2 RBAC模型族

RBAC模型族包括四个概念模型，即RBAC0、RBAC1、RBAC2及RBAC3[7]。它们各自具有不同的功能及特点，其关系图如图1所示。

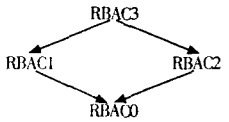

图1 RBAC模型之间的关系

2.2.1 基本模型RBAC0

RBAC0是基本模型，处于模型的最底层，是对任何宣称支持RBAC模型的系统的最低要求。RBAC0模型由如下部分组成：

(1)U、R、P和S(用户集、角色集、权限集和会话集)；

(2)PA⊆P×R，多对多的权限到角色的分配关系；

(3)UA⊆U×R，多对多的用户到角色的分配关系；

(4)user：S→U，一个函数映射，每个会话si对应唯一的用户user(si)；

(5)roles：S→2^R，一个函数映射，每个会话si对应一个角色集，roles(si)⊆{r|(user(si),r) UA}，会话si所拥有的权限为$\bigcup r\in roles(si)\{p|(p,r)\in PA\}$。

2.2.2 角色层次RBAC1

RBAC1和RBAC2都包含了RBAC0，各自在RBAC0的基础上增加了一些高级特性，RBAC1增加了角色的层次关系，RBAC2增加了各部件之间的约束关系。两者之间是不可比的。

RBAC1模型由如下部分组成：

(1)U、R、P、S、PA、UA和user与RBAC0中的定义相同；

(2)RH⊆R×R，是R集上的偏序关系，称为角色层次关系，用≤表示；

(3)roles：S→2R，修改为roles(si){r′|∃r′≤r且(user(si)，r)∈UA}。

我们使用一个例子来说明模型RBAC1的特征。项目组由四种角色组成，即项目成员(Project Member)、测试工程师(Test Engineer)、程序员(Programmer)和项目主管(Project Supervisor)，其层次关系如图2所示。

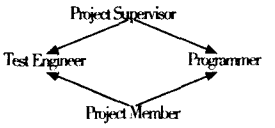

图2 项目组角色之间的关系

由RBAC1的定义可知，项目主管继承了程序员、测试工程师和项目成员的权限，隐式地成为了他们的成员。

假设程序员不想项目主管看到还未编写完成的程序，这种想法应该是合乎情理的，但由于主管拥有程序员的所有权限，使得这种想法得不到实现。可以采取如下方法解决这个问题：定义一个新的Programmer0角色，让它继承Programmer角色的所有权限，并在Programmer0角色里增加Programmer的私有权限，并将Programmer用户分配到Programmer0角色。由于Programmer0角色与Project Supervisor角色不可比，所以Project Supervisor角色不能继承Programmer0的私有权限，从而达到目标。如图3所示。

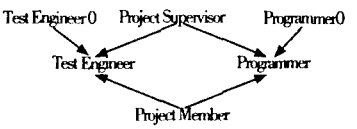

图3 项目组角色的层次关系

2.2.3 限制约束RBAC2

RBAC2除了继承RBAC0的特征外，还增加了一个限制集。这些限制决定RBAC0中部件值的变化是否可接受，只有可以接受的变化才被允许。

这些限制应由应用系统根据实际给出，下面列举了一些常用的限制：

(1)互斥限制(静态互斥)：一个用户最多只能被指派到一组互斥角色中的一个角色，特定的权限不能同时被指派到互斥角色。

(2)基数限制：一个角色的用户成员数目受限，一个用户所属的角色数目也受限，一个权限能指派的角色数目受限，一个角色对应的权限数目也受限。

(3)先决条件限制：一个用户可以被指派到某角色仅当他已是另一角色的成员；一个权限可以指派给某角色仅当该角色已拥有另一权限。

(4)运行时的互斥限制：可以允许一个用户同时具有两角色的成员资格，但在系统运行时不可同时激活这两个角色。

2.2.4 统一模型RBAC3

作为前三者的合并，RBAC3包括了RBAC1和RBAC2，由传递性，也包括了RBAC0。RBAC3结合了RBAC1的层次特性和RBAC2的限制特性。角色层次必须是偏序关系，这是RBAC1的内在限制，由RBAC模型图

中可以看出，外在限制(RBAC2)也施加在角色层次上，这样可以限制给定角色的上层角色和下层角色的数目，可以限制两个或更多的角色没有上层(或下层)角色。

层次和限制的结合带来一些微妙的相互影响。假设例子中的测试工程师和程序员被配置为互斥角色，而项目主管继承了两者的权限，也就违反了这种互斥限制。有时候这种情况是可以接受的，有时是不能接受的，这要看具体情况而定。基数限制也有类似情况。假设测试工程师角色被限制为最多只能有一个用户，同样由于项目主管角色继承了它的权限，也就违反了这种限制。换句话说，基数限制是只对直接成员起作用还是对整个角色层次上起作用需要不同的具体情况分析。

2.3 角色互斥

RBAC尽管是策略中性的模型，它仍然支持以下三条著名的安全原则：最小权力原则、职责分离原则和数据抽象原则。前面我们提到RBAC2模型引进了约束概念，其中最基本的约束成为“角色互斥”。由于角色之间相互排斥，一个用户最多只能分配到这两个角色中的一个。例如，一个项目中用户不能既是测试人员又是编程人员。我们利用角色互斥来判断系统是否违反了职责分离原则，定义安全状态，为完成任何一个关键任务所需的权限都不能被一个用户所拥有。

这里给出静态互斥和动态互斥之间的关系：静态互斥意味着互斥的角色不能分配给同一个用户；动态互斥意味着互斥的角色可以分配给同一个用户，但该用户在一个会话中不能同时激活这些互斥角色。换句话说，静态互斥是在安全管理人员给用户分配角色时采用互斥规则；动态互斥是在用户建立会话之后选择角色时采用互斥规则。

3 基于描述逻辑的角色互斥及其本体表示与推理

RBAC的形式化描述方式多种多样，这里我们基于描述逻辑语言给出角色互斥的形式化描述及其推理过程。

3.1 基于描述逻辑的表达方式

本体是为说明某语言词汇表的内在意义而设计的一套逻辑公理。给定一个语言L(我们这里使用的是描述逻辑语言)和本体承约K，语言L的本体就是基于以下目的而设计的逻辑公理集合：其自身模型尽可能地接近依照K的、由语言L描述的所有预定模型。所以本体是与其描述语言密切相关的，这里我们利用基于描述逻辑语言的本体来对RBAC中的角色互斥进行描述和推理。

描述逻辑的本体描述中，一个知识库是由两个部分组成的，即术语库(TBox)和断言库(ABox)。TBox存放本体中所有的公理，包括所有概念的声明和关系的定义；ABox中包含所有实例的断言，用来描述实例的从属和实例间的关系。描述逻辑具有以下主要特点：

(1)定义良好的语义和表示能力；

(2)基于逻辑的推理能力；

(3)保证计算复杂性和可判定性；

(4)明确的推理算法，如知名的基于Tableaux的算法[8]；

(5)现有工具的有力支持，如高度优化的推理器FaCT[9]、RACER[10]等。

3.1.1 描述逻辑的基本语法

描述逻辑语言来源自AL(Attribute Language，简称AL)，其描述元素格式与说明如下：

A：atomic concept；

$\top$：universal concept；

$\bot$：bottom concept；

$\neg A$：atomic negation；

$C \cap D$：intersection；

$\forall R.C$：value restriction；

$\exists R.T$：limited existential quantification。

但是，随着需要表达的语义日渐丰富与复杂，DL增加了两种描述方法来加强其表述能力：

$\exists R.C$：full existential quantification；

$\geqslant nR$ or $\leqslant nR$：number restrictions。

其中，A表示原子概念，是用来组合成其它复杂概念最基本的基石，本体系统在构造的时候自动承认原子概念及其属性。令I为概念到实例集合的映射，则表示的实例集合为A^I；$\top$是通用概念，指所有概念的父概念，即在应用域中任何情况下都成立的概念，描述集合中的全集$\triangle^I$；$\bot$描述与$\top$相反的概念，即在应用域中任何情况下都不成立的概念，类似于集合中的空集$\varnothing$；$\neg A$表示与A相离且与之相并为通用概念的概念，称之取反，用集合的方法位表示为$\triangle^I / A^I$；$C \cap D$表示C、D概念的交集；$\forall R.C$描述当二元谓词R宾语属于C概念时其主语的集合，表示为$\{a \in \triangle^I \mid \forall b.(a,b) \in R^I \rightarrow b \in C^I\}$；$\exists R.T$描述应用域中所有满足关系$R$的主语的集合，表示为$\{a \in \triangle^I \mid \exists b.(a,b) \in R^I\}$；$\exists R.C$考虑了对$\exists R.T$中宾语的限制，表示为$\{a \in \triangle^I \mid \exists b.(a,b) \in R^I \cap b \in C\}$；值限制$\geqslant nR(\leqslant nR)$对满足关系$R$的实例个数进行了限制规定，其形式化描述为$(\geqslant nR)^I = \{a \in \triangle^I \mid |\{b \mid (a,b) \in R\}| \geqslant n\}$及$(\leqslant nR)^I = \{a \in \triangle^I \mid |\{b \mid (a,b) \in R\}| \leqslant n\}$。

3.1.2 RBAC系统本体知识库的构建

基于描述逻辑的本体知识库的构建最重要的是TBox和ABox的构建。以一个RBAC系统中描述上文提到的例子的片断为例说明知识库的构建方法，我们需要描述：

一个项目组中有四个角色组成：项目成员、测试工程师、程序员、项目主管。项目成员可以使用程序完成的结果；测试工程师继承项目成员的权限并可以对程序进行调试，然后确认程序是否完成；程序员与测试工程师一样继承项目成员的权限并可以在任何时候修改程序，但不希望项目主管看到自己未完成的程序；项目主管继承所有权限，但无法继承测试工程师与程序员的私有权限，从而满足程序员的要求。其基数限制有：项目主管角色的用户不得超过五个；其角色互斥定义有：一个用户不可能同时拥有程序员、测试工程师与项目主管中任意两个角色。

TBox公理集原子概念为(Human，Program，Project)，原子属性有(Edit，Test，Supervise，hasMember，hasbeenFinishedby，Belongsto)，其中原子属性的语义依次

为编辑、测试、监督管理、拥有用户、被完成及工作范围属于。TBox 的内容(公理列表)如下：

Project Program≡*Program*⋂∃*Belonsto. Project*

Finished Project Program≡*Project Program*⋂∃*hasbeenFinishedby. Human*

Unfinished Project Program≡*Project Program*⋂¬*Finished Project Program*

ProjectMember≡*Human*⋂∃*Belongsto. Project*

TestEngineer0≡*ProjectMember*⋂∃*Test. Project Program*

TestEngineer≡*TestEngineer0*⋂∃*Test. Finshed Project Program*

Programmer0≡*ProjectMember*⋂∃*Edit. Project Program*

Programmer≡*Programmer0*⋂∃*Edit. Finshed Project Program*

ProjectSupervisor≡*ProjectMember*⋂∃*Supervise. ProjectMember*

ABox 中加入实例的断言，其格式如 *ProjectSupervisor*(Andy)，就意味着 Andy 成为 *ProjectSupervisor* 的用户实例，这里就不作详细阐述。

3.1.3 角色互斥基于描述逻辑的形式化描述

由于需要描述的本体有两个限制，我们使用描述逻辑对基数限制和角色互斥进行描述，即在 TBox 中修改并增加其中的公理来达到目的。

首先修改 *ProjectSupervisor* 的定义来完成基数限制：

ProjectSupervisor ≡ *ProjectMember* ⋂ ∃*Supervise. ProjectMember*⋂≤5*hasMember*

又由于：*Programmer0*⋂*TestEngineer0*≡⊥

⇒ *ProjectMember* ⋂ (∃*Edit. Project Program* ⋂ ∃*Test. Project Program*)≡⊥

由于 *ProjectMember*≠⊥，可以得到新的公理：

∃*Edit. Project Program* ⋂ ∃*Test. Project Program* ≡⊥

用原子概念描述为：

∃*Edit.* (*Program*⋂∃*Belongsto. Project*)⋂∃*Test.* (*Progam*⋂∃*Belongsto. Project*)≡⊥

同理，根据角色互斥条件有以下公理成立：

∃*Edit.* (*Program*⋂∃*Belongsto. Project*)⋂∃*Supervise.* (*Progam*⋂∃*Belongsto. Project*)≡⊥

∃*Test.* (*Program*⋂∃*Belongsto. Project*)⋂∃*Supervise.* (*Progam*⋂∃*Belongsto. Project*)≡⊥

这样，就将各种限制在公理集中描述出来了，表示为 *ProjectSupervisor* 的成员数不得超过五个；并且，*ProjectSupervisor*、*Programmer0* 和 *TestEngineer0* 不可能拥有共同的用户。

3.2 基于描述逻辑的推理方式

描述逻辑的推理功能十分强大，推理算法也十分纯熟。用户可以利用推理工具进行语义查询、实例检测、包含检测、概念冲突检测等许多应用。由于推理内容的多样性，我们这里仅仅讨论推理概念 *C*、*D* 是否静态互斥。

3.2.1 角色概念静态互斥的检测

在我们的例子中可以知道，角色 *TestEngineer0* 与角色 *Programmer0* 是静态互斥的，我们利用推理算法检测概念 *TestEngineer* 与 *Programmer* 之间的互斥性。检测概念 *TestEngineer* 与 *Programmer* 是否互斥的实质就是计算 *TestEngineerProgrammer* 的结果。如果 *TestEngineer*⋂*Programmer*=⊥，则很明显两个概念是互斥的。检测公理中概念定义是否互斥的算法步骤如下：

(1)将公理化为最简形式，即用原子概念代入等式右边表示代测公理。以下为转化步骤：

TestEnginer ⋂ *Programmer* = (*TestEngineer0*) ⋂ ∃*Test. Finished ProjectProgram*) ⋂ (*Programmer0* ⋂ ∃*Edit. Finished ProjectProgram*)=

((*ProjectMember* ⋂ ∃*Test. ProjectProgram*) ⋂ ∃*Test. Finished ProjectProgram*)⋂

((*ProjectMember* ⋂ ∃*Edit. ProjectProgram*) ⋂ ∃*Edit. Finshed ProjectProgram*)=

((*Human*∃*Belongsto. Project*)⋂∃*Test.* (*Program*⋂∃*Belongsto. Project*)⋂∃*Test.* ((*Program*⋂∃*Belongsto. Project*) ⋂ ∃*hasbeenFinishedby. Human*) ⋂ (*Human* ⋂ ∃*Belongsto. Project*) ⋂ *Edit.* (*Program* ⋂ ∃*Belongsto. Project*) ⋂ *Edit.* ((*Program* ⋂ ∃*Belongsto. Project*) ⋂ ∃*hasbeenFinishedby. Human*)

(2)此时等式右边全部元素为原子概念，则开始化简：

TestEngineer⋂*Programmer*=((*Human*⋂∃*Belongso. Project*) ⋂ ∃*Test.* ((*Program*⋂∃*Belongsto. Project*) ⋂ ∃*hasbeenFinishedby. Human*) ⋂ ∃*Edit.* ((*Program*⋂∃*Belongsto. Project*)⋂∃*hasbeenFinishedby. Human*))=

((*Human*⋂∃*Belongsto. Project*))⋂(∃*Test.* (*Program*⋂∃*Belogsto. Project*)⋂∃*Edit.* (*Program*⋂∃*Belongsto. Project*))⋂(∃*Test.* (∃*hasbeenFinishedby. Human*)⋂∃*Edit.* (∃*hasbeenFinishedby. Human*)))

(3)由于存在公理：

∃*Edit.* (*Program*⋂∃*Belongsto. Project*)⋂∃*Test.* (*Program*)⋂∃*Belongsto. Project*)≡⊥

则可得：

TestEngineer⋂*Programmer*=((*Human*⋂∃*Belongsto. Project*)⋂(∃*Test.* (∃*hasbeenFinishedbyHuman*)⋂∃*Edit.* (∃*hasbeenFinishedby. Human*))⋂⊥)=⊥

所以我们得出结论：概念 *TestEngineer* 与 *Programmer* 静态互斥。

3.2.2 基于 Tableaux 算法的包含检测

在 TBox 中除了需要检测概念的一致性，有时还需要检测两个概念之间的包含关系。我们基于 Tableaux 的算法思想进行包含关系的一致性检测。其根本思想是将检测公理 *C*⊑*D* 转化为检测 *C*⋂¬*D* 的可满足性：如果 *C*⋂¬*D*=⊥，则 *C*⊑*D* 成立；反之，*C*⊑*D* 不成立。

仍然以我们例子中的角色概念为例：

假设需要判断 *Programmer* 与 *Programmer0* 是否是包含关系，则根据 Tableaux 的思想，需要采用 3.2.1 节中同样的步骤来计算 *Programmer*⋂¬*Programmer0* 的结果。篇幅所限，这里不列出详细推导过程，仅给出最后几步简化过程：

Programmer⋂¬*Programmer0* = ((*ProjectMember* ⋂ ∃*Edit. Project Program*)⋂∃*Edit. Finish Project Program*)⋂¬(*ProjectMember*⋂∃*Edit. Project Program*)=((*ProjectMember*⋂∃*Edit. Project Program*)⋂¬(*ProjectMember*⋂∃*Edit. Project Program*)⋂∃*Edit. Finish*

(下转第 44 页)

[3] Christensen E, Curbera F, Meredith G, et al. Web Service Description Language (WSDL) 1.1 [EB/OL]. http://www.w3.org/TR/wsdl, 2001-11.

[4] IBM. Business Process Execution Language for Web Services. Version 1.1 [EB/OL]. ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf, 2003-10.

[5] The OWL Services Coalition. Semantic Markup for Web Services (OWL-s). Version 1.0[S]. 2004.

[6] UDDI Org. UDDI Spec TC, Version 3.0.2[S]. 2004.

[7] Massimo P, Kakahiro K, Terry R P, et al. Importing the Semantic Web in UDDI[A]. Proc of Web Services, E-business and Semantic Web Workshop[C]. 2002. 225-236.

[8] Sivashanmugam K, Verma K, Mulye R, et al. SpeedR: Semantic P2P Environment for Diverse Web Services[Z]. Final Presention, Department of Computer Science, University of Georgia, 2002.

[9] Hu Jianqiang, et al. Web Services Peer-to-peer Discovery Service for Automatic Web Service Composition[A]. Proc of the Int'l Conf on Computer Networks and Mobile Computing [C]. 2005. 509-518.

[10] Wombacher A, Fankhauser P, Neuhold E J. Transforming BPEL into Annotated Deterministic Finite State Automata for Service Discovery[A]. Proc of the Int'l Conf on Web Services[C]. 2004. 316-323.

[11] LTSA[EB/OL]. http://www.doc.ic.ac.uk/ltsa/bpel4ws/ltsaeclipse031.zip, 2005-03.

(上接第 40 页)

Project Program = ⊥

因此我们可以得出 *Programmer* ∩ ¬ *Programmer*0。

3.3 OWL 文档实现

下面是 OWL 代码中用来描述例子中几个角色的片断:

```
⋮
<owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Test_Engineer0">
    <rdfs:subClassof>
      <owl:Class rdf:ID="Test_Engineer"/>
    </rdfs:subClassof>
    <owl:disjointWith>
      <owl:Class rdf:ID="Programmer0"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="Project_Supervisor"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:ID="Project_Member"/>
  <owl:Class rdf:about="#Project_Supervisor">
    <rdfs:subClassof>
      <owl:Class rdf:about="#Test_Engineer"/>
    </rdfs:subClassof>
    <owl:disjointWith>
      <owl:Class rdf:ID="#Programmer0"/>
    </owl:disjointWith>
    <rdfs:subClassof>
      <owl:Class rdf:ID="Programmer"/>
    </rdfs:subClassof>
    <owl:disjointWith rdf:resource="#Test_Engineer0"/>
    <rdfs:subClassof>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Has_Member"/>
        </owl:onProperty>
        <owl:maxCardinality rdf:datatype="Http://www.w3.org/2001/XMLSchema#int"
          >5</owl:maxCardinality>
      </owl:Restriction>
    <rdfs:subClassof>
  </owl:Class>
⋮
```

由于 RBAC 系统的本体是基于描述逻辑编写的,所以其文档格式采用 OWL。OWL 根据表示和推理能力分为三类:OWL Full 与 RDF 保持最大程度的兼容,具有最大的表示能力,但不能保证计算性能;OWL DL 是以描述逻辑为基础,在不失掉计算完全性和可判定性条件下支持最大的表示能力;OWLLite 则局限于对概念(类)的层次分类和简单的约束等进行描述。

4 结束语

本文利用描述逻辑的强大描述能力,为 RBAC 系统中的角色互斥策略建立了公理集合,并在此基础上对角色互斥的逻辑进行描述。本文以一个程序开发管理机制为例,设计并实现了 RBAC2 中基于本体的角色互斥访问控制。虽然仅仅针对该具体的系统设计本体,但可以利用本体的高度抽象将具体应用设为通用概念的子类。完成通用的本体构建并不是遥不可及的事情。最后结合 Tableaux 算法实现了概念之间的包含检测,并利用 TBox 中公理的推导进行了证明。

参考文献:

[1] Guarino N. Formal Ontology and Information Systems[A]. Proc of the 1st Int'l Conf on Formal Ontology in Information Systems[C]. 1998. 13-15.

[2] Uschold M, Gruninger M. Ontologies: Principles, Methods, and Applications[J]. Knowledge Engineering Review, 1996, 11(2): 93-98.

[3] Berners-Lee T, Hendler J, Lassila O. The Semantic Web1 [J]. Scientific American, 2001, 284(5): 34-43.

[4] Berners-Lee T. Semantic Web Road Map[EB/OL]. http://www1w3org/DesignIssues/Semantic1html, 1998-09.

[5] Baader F, Calvanese D, McGuinness D, et al. The Description Logic Handbook: Theory, Implementation and Applications[M]. Cambridge: Cambridge University Press, 2003.

[6] Sandhu R S. Role-Based Access Control[A]. Proc of the 2nd ACM Workshop on Role-Based Acess Control[C]. 1997.

[7] Ravi S, Edward C, Hal L F, et al. Role-Based Access Control Models[J]. IEEE Computer, 1996, 29(2): 38-47.

[8] Schwind C. A Tableaux-Based Theorem Prover for a Decidable Subset of Default Logic[A]. Proc of the 10th Int'l Conf on Automated Deduction[C]. 1990. 528-542.

[9] Horrocks I. Using an Expressive Description Logic: FaCT or fiction? [A]. Proc of the 6th Int'l Conf on the Principles of Knowledge Representation and Reasoning[C]. 1998.

[10] Haarslev V, Moller R. RACER System Description[A]. Proc of the Int'l Joint Conf on Automated Reasoning[C]. 2001.