基于视图的访问控制语言 VPL 及其应用*)

姚 前 陈 舜 谢 立
(南京大学计算机系 南京210008)

摘 要 CORBA 中间件的安全管理是一个烦琐而重要的工作。本文针对 CORBA 安全服务规范在安全管理方面的不足,介绍了一种新的访问控制管理语言,即视图策略语言(VPL),并用一个具体实例说明了 VPL 如何对访问控制策略进行静态和动态管理,体现了 VPL 在策略管理方面带来的好处。

关键词 CORBA,视图策略语言(VPL),访问控制策略管理

Access Control Management Language VPL and its Application

YAO Qian CHEN Shun XIE Li
(Department of Computer Science, Nanjing University, Nanjing 210008)

Abstract The security control of the CORBA middleware is a complicated and important work. Focusing on the deficiency of the security service standard for security control, this article introduces a new language of access control management, namely VPL, and uses a specific example to explain how VPL manages access control strategy in both static state and dynamic state, which shows the advantage VPL brings to the strategy management.

Keywords CORBA, VPL, Access control strategy management

1 引言

作为重要的分布式系统模型,OMG 组织指定的 CORBA (Common Object Request Broker Architecture)[1]较好地解决了跨平台、跨操作系统、跨语言、跨协议和跨版本等软件兼容问题,实现了分布式软件集成[2],而基于 CORBA 的应用领域越来越多。随着现实应用需求的不断变化,对 CORBA 的安全性也提出了越来越高的需求。对于 CORBA 这样的开放分布式系统所固有的异构性,仅用操作系统固定的安全机制是无法满足应用安全需求的,因此,必须由 CORBA 提供与平台无关的安全服务。

为了解决 CORBA 分布式系统对象之间的安全问题,OMG 提出了 CORBA 安全服务规范(Security Service Specificaiton)[1]。这一标准提供了主体鉴别,特权委托,访问控制等功能,为 CORBA 的安全性提供了较好的支持。然而,CORBA 安全服务规范所定义的访问控制模型无法对访问控制管理提供足够的支持。

针对 CORBA 安全服务规范给安全策略管理带来的不便,本文介绍了一种基于视图(view)的访问控制语言——视图策略语言(View Policy Language, VPL)。安全管理人员可以用 VPL 对 CORBA 的安全策略进行行之有效的管理,并可以进行静态的类型检测以保证规范的一致性,同时可以充分描述策略的动态变化。本文还用一个具体实例,说明了如何用 VPL 来管理访问控制策略和动态权限的变换。

2 CORBA 安全服务标准

为了对 CORBA 的安全服务进行管理,OMG 提出了安全服务规范。安全服务规范定义了一个访问控制的参考模型,把真正的访问执行和访问控制决策分离开来。OMG 为 CORBA 的安全服务定义了一个被称为 DomainAccessPolicy 的标准访问模型,安全策略通过域来实施,单个策略实际上是用 DomainAccessPolicy 对象来表示的[1]。安全服务的授权模型是基于访问控制矩阵的[3],对 CORBA 访问控制模型中访问控制矩阵语义的详细描述可参考文[4]。矩阵的行表示主体,列表示对象(客体),矩阵中的元素表示元素所在行的主体对所在列对象的访问权限。安全服务定义的权限都在权族 corba 中,它包含四个普通权限:*g*, *s*, *m*, *u*,分别表示 *get*, *set manage* 和 *use*。系统授权的检测是针对单个权限来进行的。CORBA 访问模型并没有明确地对操作进行分组。另外,最小特权(least privilege)是安全机制设计所必须遵循的机制之一[5],而四个普通权限的组合只能表示有限的授权情况,所以实现最小特权原则是非常困难的[6]。

尽管安全服务规范为 CORBA 提供了一定的安全支持,但是在安全策略管理上,它存在着如下一些不足。首先,安全服务规范的域访问策略无法为细粒度的访问策略提供足够的支持,如果一个策略要求对某一对象类型的不同实例进行不同的处理,则必须为每一次处理建立不同的域,这样使得策略的管理复杂化。第二,因为用 CORBA 权限概念可表达的授权需求常常是一致的,所以模型无法随着对象和接口数的增加来相应地扩充系统,从而缺少灵活性。第三,由于 corba 权族表达的权限集合数量有限,所以无法实现最小特权原则。第四,访问控制模型没有定义访问控制概念之间的任何结构关系,因此不便于策略的管理。针对 CORBA 安全服务规范在策略管理方面的不足,有必要采用一种新的方法来对 CORBA 的安全策略进行有效的管理。

*)基金项目:国家十五科技攻关项目(金融示范工程),课题编号:2001BA102A04。姚 前,陈 舜 博士生,主要研究方向为分布式系统和计算机安全。谢 立 教授,博士生导师,主要研究方向为分布式计算、并行处理、先进操作系统等。

3 用 VPL 管理访问控制策略

针对 CORBA 安全服务规范在访问控制策略管理方面的不足，Gerald[6~8]提出了视图(view)的概念，并用视图策略语言(VPL)来管理访问控制策略。与数据库中视图不同的是，这里的视图表示一个对象类型的操作集。在基于视图的访问控制矩阵模型中，主体对对象的授权不是以单个权限，如普通权限，为单位，而是以视图为单位，表1是基于视图的访问控制矩阵，其中，r 是角色，s 是主体，o 是对象，t 是对象类型，Managing，Reading 等都是视图。基于视图的访问控制决策过程是：当一个主体(或角色)试图通过其拥有的操作访问一个对象时，如果主体(或角色)所拥有的视图包含了这一操作，且操作对象属于视图的控制对象类型时，访问被允许，否则被拒绝。

本节我们用一个实例来说明 VPL 是如何进行策略管理的。

表1 访问控制矩阵

	O_1	…	O_n	t_1	…	t_n
r_1			{Reading}			{Creating}
…						
r_i			{Updating}			
s_1						
…						
S_j		{Managing}				

我们给出的实例是一个电子会议主席的例子，通过该系统可以进行网上征集论文，网上投稿和网上论文评审。应用系统的用例如图1所示。

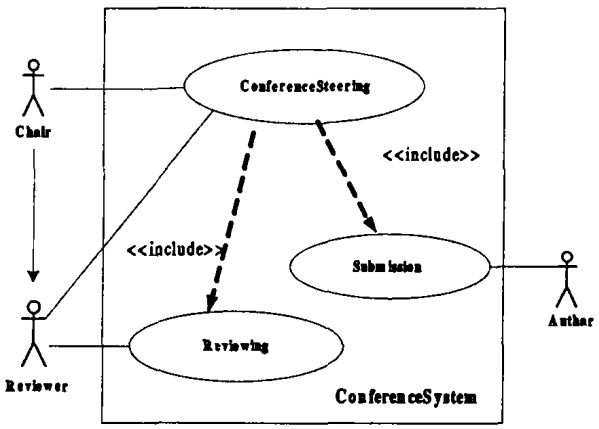

图1 会议系统的参与者和用例

系统工作流程如下：

1. 当会议主席(角色 Chair)开始征集会议论文时，ConferenceSteering 用例启动。

2. 会议主席启动 Submission 用例，宣布提交论文阶段开始。

(a)论文作者(角色 Author)向会议注册论文，并接收到论文编号；

(b)作者用会议指定的论文编号向会议投稿。

3. 会议主席宣布论文投稿截止日期已到，从而中止了 Submission 用例。

4. 会议主席为每篇论文指定一定数量的评审人(角色 Reviewer)，并把相关论文发给他们。

5. 论文评审阶段开始，即 Reviewing 用例开始。

(a)评审人评审论文，并提交评审意见；

(b) 提交了一篇论文的评审意见以后，评审人可以阅读其他评审人对同一篇论文的评审意见，也可以修改他自己的评审意见。

6. 会议主席宣布论文评审阶段结束。

图2用 CORBA IDL 列出了会议应用接口。

```
Interface ConferenceManagement {
   Submission Management getSubmissionManagement();
   Readonly attribute Document callForPapers;
   Void issueCallForPapers(in string cfp);
   Void beginSubmission();
   Void deadlineReached();
   Void makeDecision();
};
interface SubmissionManagement{
   Paper registerPaper(in string author_name, in string title)
         raises(AlreadyRegistered,IncompleteInformataion);
   PaperIdSeq listPapers();
   Paper getPaper(in long paperNumber);
};
interface Paper:Document{
   readonly attribute long number;
   void submit();
   Review submitReview(in string review, in long reviewerNumber);
   longSequence listReviews();
   Document getReview(in long receiverNumber);
};
interface Review: Document {
   readonly ReviewerId reviewer;
};
```

图2 会议应用的 IDL 接口

安全策略的设计只关注应用级安全策略。访问控制策略的设计从两方面来考虑。第一是访问权的初始配置。访问权的初始配置可以用角色，角色约束和在系统启动时角色所拥有的视图来定义。第二，系统在不同阶段时主体和角色的授权的变化情况，在 VPL 中是由模式(schema)和由这些模式指派和吊销的视图来体现的。

3.1 角色，约束和初始视图

对于这个例子的角色，角色约束和初始 view 的定义如图3所示。

```
roles
  Author
      holds SubmissionAccessing on ConferenceManagement
      holds Registering on SubmissionManagement
  Reviewer
      holds Member on SubmissionManagement
      holds SubmissionAccessing on ConferenceManagement
      holds PaperReviewing on Paper
  Chair:Reviewer
      holds Steering on ConferenceManagement
      holds PaperReading on Paper
      maxcard 1
      excludes Author
```

图3 角色声明，约束，和初始视图

图1中用例确定的参与者直接对应图3的角色。图3确定了三个角色，即 Author，Reviewer 和 Chair，给出了这些角色的声明，约束，以及初始拥有的 views。其中，角色 Author 拥有控制 SubmissionManagement 接口的视图 Registering 和控制 ConferenceManagement 接口的视图 SubmissionAccessing。Chair 继承了 Reviewer 角色，除拥有 Reviewer 的视图外，还拥有 Steering 和 PaperReviewer 视图，分别控制接口 ConferenceManagement 和 Paper。角色 Chair 声明的最后两行表示担任 Chair 角色的只能有一个主体，且不能是 Author 角色。

图4给出了一些初始 views。

```
view SubmissionAccessing
  controls ConferenceManagement{
   allow
      getSubmissionManagement
      callForPapers
```

```
}
view Registering
  controls SubmissionManagement
  requires SubmissionPhase
{
  allow
    registerPaper
}
view PaperReviewing: PaperReading
    controls Paper
     restricted_to Reviewer
 required ReviewingPhase
{
  allow
  listReview
       submitReview
}
view Steering:SubmissionAccessing
    required ReviewingPhase
    restricted_to Chair
{
  allow
    issueCallForPapers
    beginSubmission
    deadlineReached
    makeDecision
}
view Reading
    controls Document
{
  allow
    read
    title
}
view PaperReading: Reading
     controls Paper
{
  allow
   number
   author
   listReviews
}
```

图4　初始 view 定义

关键字 view 表示视图定义，冒号":"表示视图继承关系；controls 子句表示视图控制的接口类型，为了能表示细粒度的访问控制，也可以是单个对象类型；restricted_to 子句表示视图只能指派的角色；requires 子句表示条件视图，即主体要拥有该视图必须预先拥有 requires 子句后的视图，通过使用条件视图以及后面介绍的模式(schema)，VPL 可以描述动态授权的变化；视图定义体中，allow 表示该视图允许的操作，视图不允许的操作用关键字 deny 定义。在图4中，Author 角色初始拥有的 Registering 是一个条件视图，这个视图在初始阶段可以不考虑，因为它需要另一个视图 SubmissionPhase，而此时 Author 还未拥有这个视图。同样 PaperReviewing 继承了 PaperReading 视图，它控制的是 Paper 接口，只有担任 Reviewer 角色的主体可以使用该视图，该视图是一个条件视图，它的执行需要另一个视图 ReviewingPhase。实际上 SummissionPhase 和 ReviewingPhase 都定义成了虚拟视图，这在后面会提到。

3.2　动态权限的改变

电子主席系统的处理过程主要分成四个阶段，第一阶段是初始化，会议主席向外界征求论文；第二阶段是论文提交阶段，这个阶段 Author 角色的用户可以向会议提交论文；第三个阶段是论文评审阶段，这个阶段将不再允许投稿，而是由会议主席指定的论文评审人评审论文，并提交评审意见；第四阶段是论文最终决策阶段，这时论文评审已经结束，决定是否录用论文。为了描述的方便，其它的一些处理细节在这里忽略。在从一个处理阶段到下一个处理阶段过渡过程中，系统的角色所拥有的视图必然会发生变化。

```
virtual view SubmissionPhase
virtual view ReviewingPhase
schema Steering observes ConferenceManagement
{
   issueCallForPapers
     assigns Reading on this. callForPapers
          to Author, Chair, Reviewer
   beginSubmission
     assigns SubmissionPhase on Object to Author
   deadlineReached
     assigns ReviewingPhase on Object to Reviewer
     removes SubmissionPhase on Object form Author
   makeDecision
      removes ReviewingPhase on Object from Reviewer
}
```

图5　Steering 模式和虚拟视图

图5首先定义了两个虚拟视图 SubmissionPhase 和 ReviewingPhase。虚拟视图使用关键字 virtual 来定义。一个虚拟视图不引用任何操作，它的作用仅是让一个授权与其它的视图组合在一起。由于虚拟视图不定义操作，因此它不需要控制任何对象类型。比如图5中虚拟视图就没有 controls 子句。一般，默认的控制类型是 CORBA::Object。虚拟 view 一般可以起到控制处理阶段转换的作用。

为了描述系统授权的变化情况，VPL 提供了模式(schema)概念，它定义了对控制对象的操作所引起的视图的指派和吊销情况。模式用关键字 schema 定义，关键字 observes 表示模式控制的接口。当用例被激活和禁止时，许多视图要么被指派或激活，要么被吊销或禁止，这样可以更有效地管理论文提交阶段和评审阶段主体权限的改变。电子会议处理阶段的转换用模式 Steering 来建模的，图5描述了 ConferenceManagement 接口中操作所引起的访问权的变化。会议主席调用 beginSubmission()操作，通过 assigns 语句把虚拟视图 SubmissionPhase 指派给了 Author 角色。当会议主席调用 deadlineReached()时，通过 removes 吊销 Author 角色的 SubmissionPahse，同时用 assigns 语句把 ReviewingPhase 指派给 Reviewer 角色。从而完成了从论文提交阶段到论文评审阶段的转变。同样，当会议主席调用 makeDecision()时，系统吊销了 Reviewer 角色的 ReviewingPhase，从而结束了论文评审阶段。由于许多 view 的执行都依赖于虚拟 view，所以用 Steering 模式就实现了细粒度的授权的变化。

为了更详细地描述会议实例，我们讨论论文提交阶段(Submission)和论文评审阶段(Reviewing)的用例。

Submission 用例

在虚拟视图 SubmissionPhase 被指派给 Author 角色以后，论文作者就可以用他们的初始视图 Registering(如图4)，通过调用对 SubmissionManagement 对象的 registerPaper()操作注册论文。为了对与注册和提交论文有关的更细粒度的授权改变建模，我们为论文提交阶段定义一个新模式 Submission：

```
schema Submission observes SubmissionManagement
  {
     registerPaper
        assigns PaperSubmitting, Modifying on result to caller
}
```

Submission 模式表示作者注册了论文，获得注册号以后，可以提交和修改论文。保留字"result"和"caller"表示操作的结果和调用操作的主体，这里的操作是 registerPaper()。

```
view PaperSubmitting: PaperReading
  controls Paper
  restricted_to Author
  required SubmissionPhase
  {
     allow
```

```
        submit
    }
view Modifying : Reading
  controls Document
  {
      allow
        update
        write
        append
}
```

图6 Submission 模式

为了控制对 Paper 对象的操作引起的访问权的改变，进一步定义了 PaperSchema 模式(图7)。当论文提交截止期限到来时，Submission 用例结束，会议主席调用 deadlineReached()操作，从而不再允许投稿(图5)。Steering 模式从 Author 角色吊销 SubmissionPhase。然而，吊销 Registering 并不足以阻止继续提交此时已经注册的论文，因为提交是由视图 PaperSubmitting 控制的。所以 PaperSubmitting 也定义成条件视图，以 SubmissionPhase 为前提。这样，和 Registering 一样，当 SubmissionPhase 被吊销时，Author 也不能再用视图 PaperSubmitting 提交论文了。

```
schema PaperSchema observes Paper
{
    submit
        assigns PaperReading on this to Reviewer
        removes Modifying on this from caller
    submitReview
        assigns Modifying on result to caller
        assigns ReviewReading on Result to Reviewer
        assigns ReviewAccess on this to caller
        assigns NoMorePaperReviewing on this to caller
}
```

图7 Paper 模式

Reviewing 用例

由图5知，会议主席调用 deadlingReached()操作，吊销 Author 角色用户的 SubmissionPhase 视图，而把虚拟视图 ReviewingPhase 指派给 Reviewer 角色，从而激活 Reviewing 用例，系统处理进入了论文评审阶段。这时，论文评审人员(Reviewer 角色)可以使用 PaperReviewing，提交论文评审意见。图7定义了对 Paper 对象的操作引起的视图的变化。应用策略规定，评审人一旦提交了一篇论文的评审意见以后不能再提交同一篇论文的评审意见，但是他可以阅读别的评审人对同一篇论文的评审意见。模式 PaperSchema 定义了 submitReview()操作引起的视图授权变化，相关视图定义如图8所示。

```
    view ReviewAccess
      controls Paper
      restricted_to Reviewer
{
  allow
     getReview
}
view ReviewReading:Reading
    controls Review
    {
    allow
     receiver // attribute
}
view NoMorePaperReviewing
  controls Paper
{
  deny
     submitReview
}
```

图8 论文评审阶段的 views

评审人执行 submitReview()操作，递交评审意见以后，可以通过 Modifying 视图更新其提交的评审意见。因为只有 ReviewAccess 视图允许 gerReview()操作，而这个视图又是在调用者调用 submitReview()操作后才被指派给他的，所以，恰好是那些能够使用 ReviewReading 视图的人已经提交了他们自己的评审意见，也就是说，只有提交了评审意见的人才能阅读别人关于同一篇文章的评审意见，这正是策略所规定的。另外，策略还规定，评审人提交了评审意见后，不能再对同一篇论文提交另一份评审意见。但是我们不能通过吊销论文评审人的 PaperReviewing 视图来禁止其进一步提交评审意见，因为，PaperReviewing 视图是指派给整个 Reviewer 角色，而不是单个评审人，吊销它将禁止其它评审人提交评审意见，同时还会吊销调用 listReview()操作的权限。所以这里重新定义了一个视图 NoMorePaperReveiwing，通过关键字 deny，禁止了 submitReview()操作。

当会议主席调用 makeDecision()操作时，Steering 模式吊销 Reviewer 角色的 PaperReviewing 视图，论文评审阶段结束。

总结 本文针对 CORBA 标准安全服务规范在访问控制策略管理方面的不足，通过一个具体实例，介绍了一种新的访问控制语言 VPL，较好地解决了 CORBA 访问控制的管理问题。从这个实例可以看出，系统开发人员和安全管理人员可以在一个抽象的语言层次上规范和管理 CORBA 对象的访问控制策略，既可以静态地配置访问策略，也能描述策略的动态变化。这种方法不但可以应用在 CORBA 访问控制策略的管理上，对于应用级和中间件安全策略的管理的研究也有重要的借鉴意义。

参考文献

1 OMG. The Common Object Request Broker: Architecture and Specification, Revision 2.3, June 1999

2 刘晖，沈钧毅，林欣. 用 CORBA 创建电子商务系统. 北京希望电子出版社，2000

3 Lampson B. Protection. ACM Operating Systems Review. Jan. 1974,8(1):18～24

4 Karjoth G. Authorization in CORBA Security. Proc. ESORICS'98, LNCS 1485, Springer (1998) 143-158

5 Saltzer J, Schroeder M. The Protection of Information in Computer Systems. In: Proc. of the IEEE 63(9), Sep. 1975. 1278～1308

6 Brose G. A Typed Access Control Model for CORBA. Procs. ERORICS 2000, Toulouse, France. Springer LNCS 1895. 88～105

7 Brose G. A View-Based Access Control Model for CORBA. Security Internet Programming: Security Issues for Mobile and Distributed Objects. LNCS 1603, Spring 1999. 237～252

8 Brose G. Manageable Access Control for CORBA. Journal of Computer Security, 2002,4:301～337