RBAC模型中条件时态的研究与设计*)

欧阳凯[1] 周敬利[2] 董理君[2]

(武汉科技大学计算机学院 武汉 430081)[1] (华中科技大学计算机学院系统结构系 武汉 430074)[2]

摘 要 具有时态特征的角色访问控制(RBAC:Role Based Access Control)模型能够为RBAC控制机制提供动态的时间控制因素,是目前安全模型领域的研究热点。基于对周期理论和时态RBAC模型的研究,本文认为时态不仅能够为模型提供时间维的控制因素,而且模型中的约束也能作用于时间维形成条件时态平面的控制因素,从而能够进一步提高模型控制的灵活性和多样性。为此,本文提出了条件周期表达式和条件时态的概念,形式化描述了条件时态语义;并通过条件周期事件和角色状态在条件周期下的断言详细论述了条件时态。

关键词 角色访问控制,条件,时态,约束

A Design Study for Conditional Temporal in RBAC

OUYANG Kai[1] ZHOU Jing-Li[2] DOGN Li-Jun[2]

(College of Computer Science & Technology, Wuhan University of Science & Technology, Wuhan 430081)[1]

(Department of System Architecture, College of Computer Science & Technology, Huazhong University of Science & Technology, Wuhan 430074)[2]

Abstract The temporal characteristic could provide the dynamic control factor for the RBAC (Role Based Access Control) model, which is the research hotspot in the security model field. Based on the research of the periodic time theory andthe Temporal Role Based Access Control (TRBAC) model, we find that the temporal aspect not only could provide the time dimension factor for the model's access control, but also the model's constrains could impact the time dimension to form the conditional temporal plane. Hence, we put forward the conditional periodic expression and the conditional temporal concept so as to improve the flexibility and the variety of the access control. In this paper, we formally describe the definition of the conditional periodic express and detail the conditional periodic event and the conditional temporal predicate of the role status.

Keywords Role based access control, Conditional, Temporal, Constrain

1 引言与背景

安全模型是安全操作系统抽象、无歧义的描述,是安全操作系统研究的理论基础。基于角色的访问控制模型(RBAC)是目前安全模型领域的研究热点,被广泛地应用在操作系统、数据库和网络控制等方面。从Ferraiolo等人提出RBAC模型[1],RBAC经历了许多专家学者的改进和完善。根据Ferraiolo等人2001年提出的RBAC标准[2],最终于2004年形成了NIST标准,并给出了完整的形式描述。NIST将RBAC分成4个组件:核心RBAC(Core RBAC)、层次RBAC(Hierarchical RBAC)、静态职责分割(SSD:Static Separation of Duty)和动态职责分割(DSD:Dynamic Separation of Duty)。RBAC核心思想是通过角色集的用户指派和许可指派,在用户会话期间建立访问控制机制,从而避免了用户集和许可指派集的直接关联,降低了管理的复杂度;通过角色集和许可指派集关联,增强了许可指派的逻辑可读性。

另一方面,时间描述是满足安全模型时态控制的基础,但是它的困难在于如何通过有限表达式来描述无限时间范畴[3]。为此,Niezette等人[4]提出通过周期表达式(Periodic Expressions)来形式化描述时间,从而解决有限表达式描述无限时间范畴的问题。周期表达式的基础概念是:历法(Calendars)和片段(Slices)。历法是指具有逻辑意义的时间结构表达(比如年、月、日、星期和学期等);片段是历法的组合方式,通过该组合可以对复杂的时间信息做出准确的表达。基于周期表达式,Bertino和Bonatti将周期约束(Periodicity Constraints)和瞬时原因(Temporal Reasoning)引入RBAC模型,扩展为时态RBAC(TRBAC:Temporal RBAC)[5],使得访问控制的约束具有时态特征。J. B. D. Joshi等人[6]在2005年进一步提出一种普遍适用的时态RBAC(A Generalized Temporal RBAC)模型,将时态从约束的影响因素扩展为RBAC模型的影响因素(不仅使得约束具有时态性,而且使得用户指派、许可指派等操作都具有时态性)。

基于上述研究分析,可以得知时态控制日益成为控制机制的重要因素,已经渗透到RBAC的各种逻辑操作和逻辑事务中。本文认为时态不仅能够是约束的时间特征,而且约束也可以是时态的条件。为此提出了条件时态的概念,将时间-约束的二维关系扩展为条件-时间-约束的三维关系,为RBAC提供多因素、多视角的控制机制,进一步提高访问控制的灵活性和多样性。为了准确无歧义地描述条件时态特征,本文提出了条件周期表达式,并通过条件周期事件和角色约

*)国家自然科学基金(编号60373088)。**欧阳凯** 博士,讲师,研究方向为安全操作系统、网络安全控制技术和网络存储技术;**周敬利** 教授,博士生导师,研究方向为高性能网络存储技术及多媒体技术;**董理君** 博士研究生,研究方向为网络安全结构与控制技术。

束的描述详细地论述了条件时态。

2 条件时态基本语义

条件时态形式化描述的基础是条件周期表达式，本节通过对条件周期表达式的定义，详细阐述条件时态的基本语义。

2.1 周期表达式

Bertino 等人在周期理论基础上提出的周期表达式如定义 1 所示。

定义 1(周期表达式) 给定历法 $C_d, C_1, \cdots, C_n$，周期表达式的定义为

$$P=\sum_{i=1}^{n} O_i.\ C_i \triangleright r.\ C_d$$

其中 $O_1=all, O\in 2^N \cup \{all\}, C_i \subseteq C_{i-1}, i=2,\cdots,n$，以及 $r\in N$。O 是整数或者整数集；all 表示任意时间偏移；r 代表时间间隔。符号 $\triangleright$ 是分割符，将周期表达式 P 分成两部分：前面部分表明时间间隔的起点(偏移)集合；后面部分是历法 C_d 时间间隔的持续时间。符号 $\subseteq$ 代表子历法关系，即 C_i 是 C_{i-1} 的子历法(比如：日是月的子历法；星期不是月的子历法)。表 1 的示例展示了周期表达式如何描述时间(当 r 为 1，持续时间是历法 C_d 单位时间时，后面部分可以省略)。

表 1 周期表示法示例

时间	周期表达式
周一和周五	$weeks+\{2,6\}.\ days$
每个月的第二十天	$months+20.\ days$
夏天	$years+7.\ months 3.\ months$
工作日	$weeks+\{2,\cdots,6\}.\ days$
工作日的上午九点到下午一点	$weeks+\{2,\cdots,6\}$ $days+10.\ hours \triangleright 4.\ hours$

2.2 条件周期表达式

定义 2(条件周期表达式) 具有约束条件集(简称条件集，记为 COND)限制的周期表达式称之为条件周期表达式，记为 $\Im$，其定义是：$\Im = Logic(cond; COND).\ P$。其中函数 $Logic(cond; COND)$ 代表对条件集的元素进行逻辑运算，返回布尔值(TRUE/FALSE)。

示例 1：当条件集为空(记为 nil)时，其逻辑运算结果为 TRUE，即当条件集为空时，$\Im=P$。

示例 2：Niezette 等人[4]提出的时刻集合 $Sol(\langle [begin, end], P\rangle)$ 代表了在时间 $begin$ 和 end 之间的周期表达式；根据条件周期表达式的定义，可以将限制时间 $[begin, end]$ 转换为条件集元素 $c_1=CondFun1([begin, end], t)$，$t$ 是执行时的当前时间，则时刻集合的条件周期表达式是：$\Im=\{c_1\}.\ P$。

条件集的元素以函数的形式存在，返回值是布尔值；函数 $Logic$ 对条件集元素的返回值按照预先设定的逻辑关系进行逻辑运算。根据 RBAC 模型的最小权限原则[7]，当函数 $Logic$ 执行结果为 TRUE 时，落在周期 P 中的各种约束具有控制意义，能够为模型提供控制依据；当执行结果为 FALSE 时，落在周期 P 中的各种约束只能为模型提供否定的(Negative)的控制依据。函数 $Logic$ 包含了 3 类逻辑运算：逻辑乘 $\wedge$、逻辑和 $\vee$、逻辑非 $\neg$，其运算示例如图 1 所示。假设条件集 $COND=\{Cond_1, Cond_2, Cond_3, Cond_4, Cond_5\}$，函数 $Logic$ 首先分别对 $Cond_1$ 逻辑非、$Cond_2$ 和 $Cond_3$ 逻辑和、$Cond_4$ 和 $Cond_5$ 逻辑乘；然后将 3 者的结果逻辑和，最后得出布尔值输出。

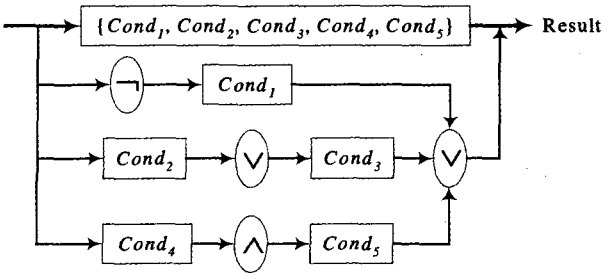


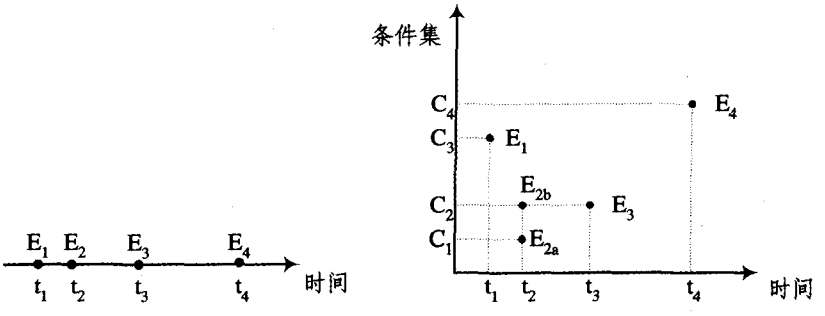
图 1 条件集逻辑运算示例

3 条件周期事件

时态 RBAC 模型中的周期事件以 $(I, P, \rho: E)$ 的形式表达，I 是时间间隔、P 是周期表达式、$\rho: E$ 是具有优先次序的事件表达式，且 $\rho \prec T$ (T 代表优先次序的全序集合上线边界)。根据定义 2，条件周期事件定义如下：

定义 3(条件周期事件) 条件周期事件(CPE：Conditional Periodic Event)以 $(\Im, p: E)$ 的形式表达，$\Im$ 是条件周期表达式，$\rho: E$ 是具有优先次序的事件表达式；优先次序的定义和时态 RBAC 的定义是一致的。

因此，时态 RBAC 的周期事件只是条件周期事件的一个子集，即 $(I, P, \rho: E)$ 可以表达为 $(\{Cond([begin, end], t)\}.\ P, \rho: E)$。条件周期事件表达式将周期事件表达式从一维坐标扩展到二维平面。假设模型系统中有 4 个事件集 (E_1, E_2, E_3, E_4)，分别在周期时间 t_1、t_2、t_3 和 t_4 发生，则其周期事件表达如图 2(a)所示；进而假设事件集 E_1 的发生满足约束条件集 $Cond_1$，事件集 E_2 的发生满足约束条件集 $Cond_1$ 或者 $Cond_2$，事件集 E_3 的发生满足约束条件集 $Cond_3$，事件集 E_4 的发生满足约束条件集 $Cond_4$，则其条件周期表达式如图 2(b)所示。



(a) 周期事件示例　　(b) 条件周期事件示例

图 2 周期事件与条件周期事件对比示意

示例 3：在网络访问控制环境中，假设用户接入 VPN 时，必须在 10min 内完成认证事件，其中用户接入条件以函数 *SinkCond*()表示，*High* 代表事件的优先级，用户认证事件以函数 Authentication 描述，则其条件周期表达式是

$CPE=(\{SinkCond()\}.(all \triangleright 10.\ minutes), High: Authentication)$

示例 4：以示例 3 为基础，进一步假设该用户的认证是从 2005 年 6 月 1 日开始有效，且只能在工作日接入 VPN，则条件周期表达式是

$CPE=(\{SinkCond(), Cond([5/10/2006, \infty], t)\}.(weeks+\{2,\cdots,6\}.days \triangleright 10.\ minutes), High: Authentication)$

4 角色约束的条件时态性

一个角色可以有 3 种状态：失效的(disabled)、有效的(enabled)和活性的(active)，其状态变迁如图 3 所示。失效角色表明该角色不能用于任何用户会话中，比如一个用户不能获得该角色的任何许可指派；失效角色能够被生效。有效角色表明该角色的指派用户/授权用户能够激活该角色。当用户激活角色后，该角色处于活性状态。一旦角色处于活性状态，再次激活该角色的行为对该角色没有任何改变。当对一个活性角色进行去活(deactivation)操作时，处于该操作执行会话中的该角色转变为有效状态，而该角色在其它会话中仍然处于活性状态。对一个角色进行失效操作时，当前所有会话中的该角色无论其状态是有效还是活性，都会转为失效状态。

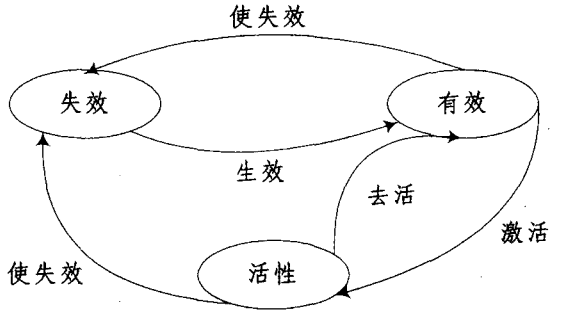


图 3 角色状态变迁

根据 J. B. D. Joshi 等人[6]提出的状态断言(predicate)定义，假设 $r \in ROLES$，$u \in USERS$，$p \in PRMS$，$s \in SESSIONS$，$\xi \in \Im$，具有条件时态特性的角色状态断言描述如表 2 所示。

如图 4 所示，以条件时态下角色 r 的用户指派为例进一步阐述具有条件时态特征的状态断言。在时间和条件集平面上，$enabled(r,\xi)$是(t_2,t_3)时段且其条件集是 $COND_1$ 和(t_4,t_6)时段且其条件集是 $COND_2$；在会话和时间平面上，用户 u_1 在 t_1 开始会话 s_1，到 t_6 终止会话；用户 u_2 在 t_2 开始会话，到 t_8 终止会话。假设与 r 有指派关系的用户 u_1 和 u_2 都需要在会话中 $activeUS(r,u,s)$，C_d 是时间维度的基本历法。

表 2 状态断言

基本状态	条件周期状态	语义[在条件周期 ξ 下]
$enabled(r)$	$enabled(r,\xi)$	有效的角色 r
$disabled(r)$	$disabled(r,\xi)$	失效的角色 r
$actived(r)$	$actived(r,\xi)$	角色 r 至少在一个会话中是活性的
$u\_actived(r,u)$	$u\_actived(r,u,\xi)$	角色 r 在用户 u 的会话中是活性的
$s\_actived(r,u,s)$	$s\_actived(r,u,s,\xi)$	角色 r 在用户 u 的会话 s 中是活性的
$u\_assgiened(u,r)$	$u\_assgiened(u,r,\xi)$	角色 r 已经指派的用户 u
$p\_assgiened(p,r)$	$p\_assgiened(p,r,\xi)$	角色 r 已经指派的许可 p
$acquired(u,r,p)$	$acquired(u,r,p,\xi)$	用户 u 获得角色 r 的许可 p

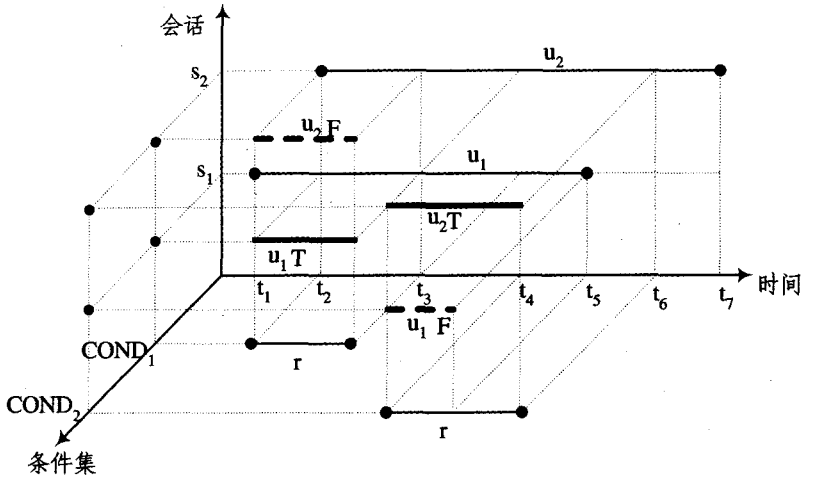


图 4 角色 r 的用户指派示例

用户 u_1 的会话 s_1 在 t_1 开始，但是在(t_1,t_2)时间段，角色 r 是 $disabled(r,nil.(t_2-t_1).C_d)$，因此 u_1 不能激活 r；在(t_2,t_3)期间，角色 r 是 $enabled(r,COND_1.(t_3-t_2).C_d)$，$u_1$ 通过条件集 $COND_1$ 的执行(TRUE)，能够激活角色 r，因此角色 r 在该会话中的状态是 $s\_actived(r,u_1,s_1,COND_1.(t_3-t_2).$

(下转第 289 页)

的办法惟有在 finally 块中使用清除操作时不抛出异常，如 dispose()，close()。Java 语言没有所谓的“析构函数”(destructor)，因此在 Java 语言中不存在自动资源回收功能。只能使用 finally 子句通过人工设置代码回收极少数必须手工回收的资源。

4 再论检查型异常和非检查型异常

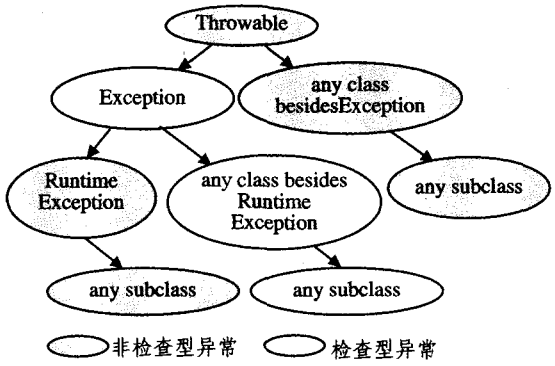


图 2 非检查型异常与检查型异常

异常处理的重要准则就是：如果你不知道如何处理这个异常，就不要去捕捉它。异常处理的目标就是把处理异常的代码同正常流程的代码区别开来，这样程序就不会被异常处理这样的枝节问题纠缠，更易于理解和维护。

但是，Java 的检查型异常强迫客户程序必须在 try 子句后加上 catch 子句捕捉异常，如果仅仅是静态类型的问题检查，检查型异常也许是必要的，可是开发人员出于多一事不如少一事的心理，有时只是简单地打印栈轨迹应付了事，这样编译就通过了，可是异常就完全被忽略掉了。在 C++ 和 Python 语言中，所有异常都是非检查型的，异常一旦被抛出，你可以捕捉它进行处理，也可以根本不用去理睬它，程序似乎也不受影响。

异常的作用应该体现在：(1)提供一个标准统一的机制报告错误；(2)允许客户程序不必关心异常处理。而 Java 提供的检查型异常似乎违反了这样的准则，因为太多的客户程序利用 catch 子句忽略掉了许多的异常，这和检查型异常设置的初衷是相违背的。开发人员习惯于在编译时发现异常，处理异常，并且认为这是可靠、安全的方式，一旦异常出现在运行时，总认为是不可靠的，其实这是个误解。

非检查型异常允许客户程序根据需要决定是否捕捉它，系统总会捕捉非检查型异常，这使得正常工作的代码可以写得更为清晰和有条理，而不是跟异常处理代码纠缠在一起。下面是一个将检查型异常转换成非检查型异常 RuntimeException 的例子：

```
import java. io. * ;
class ExceptionAdapter extends
RuntimeException{
    private final String stackTrace;
    public Exception originalException;
    public ExceptionAdapter(Exception e){
      super(e. toString());
      originalException=e;
      StringWriter sw = new StringWriter();
      e. printStackTrace(new
PrintWriter(sw));
      stackTrace = sw. toString();
    }
    public void printStackTrace(){
      printStackTrace(System. err);
    }
    public void rethrow(){throw
originalException;}
}
    ......
    catch(ExceptionAdapter ea){
      try {
        ea. rethrow();
      } catch(IllegalArgumentException e)
{
        // ...
      } catch(FileNotFoundException e){
        // ...
      }
      // etc.
    }
```

从例子中开发人员仍然能够捕捉特定的异常，但不必在程序的任何地方设置 try 和 catch 子句捕捉异常，甚至如果忘了捕捉，异常也会被提交到更高层次的场合进行处理。

(下转第 292 页)

(上接第 285 页)

C_d)，如图中的粗直线 u_1T 所示。而角色 r 在 t_3 时刻失效，故而在(t_3，t_4)时段是 $disabled(r, nil.(t_4-t_3).C_d)$；$u_1$ 和 r 之间的指派关系也失效了。从 t_4 开始，角色 r 再次有效 $enable(r)$，u_1 能够再次激活 r；但是 u_1 不能通过条件集 $COND_2$ 的执行(FALSE)来激活角色 r，对于用户 u_1 来说，角色 r 只是有效状态，如图中的粗虚线 u_1F 所示。直到会话 s_1 结束(t_5 时刻)，u_1 都没有机会激活角色 r。

用户 u_2 的会话 s_2 在 t_2 开始，虽然在(t_2，t_3)时段，角色 r 是 $enabled(r, COND_1.(t_3-t_2).C_d)$，但是 u_2 不能通过条件集 $COND_1$ 的执行(FALSE)激活角色 r，如图中的粗虚线 u_2F 所示。同样的角色 r 在(t_3，t_4)时段是 $disabled(r, nil.(t_4-t_3).C_d)$，$u_2$ 没有权利激活角色 r。在(t_4，t_6)时段，角色 r 是 $enabled(r, COND_2.(t_6-t_4).C_d)$，且 u_2 能够通过条件集 $COND_2$ 的执行(TRUE)，因此角色 r 在该会话中的状态是 $s\_actived(r, u_2, s_2, COND_2.(t_6-t_4).C_d)$，如图中的粗直线 u_2T 所示。在 t_6 时刻，角色 r 再次失效，u_2 和 r 之间的指派关系也随之失效，直至该会话 s_2 结束(t_7 时刻)，u_2 都不能激活角色 r。

结束语 本文在研究周期理论和时态 RBAC 模型的基础上，提出了条件周期表达式和条件时态的概念，将模型的时间维控制因子扩展为条件时间平面的控制因子，从而提高了模型控制的灵活性和多样性。通过对条件周期表达式、条件周期事件和条件时态下的角色状态断言论述，形式化地描述了条件时态的原理和控制机制。

参 考 文 献

1 Ferraiolo D, Cugini J, Kuhn D R. Role Based Access Control (RBAC): Features and Motivations. In: Proc. 1995 Computer Security Applications Conference, December 1995. 241～248
2 Ferraiolo D, Sandhu R, Gavrila S, et al. A Proposed Standard for Role Based Access Control. ACM Transactions on Information and System Security, August 2001. 224～274
3 St'evenne J-M. A model-checking approach to temporal reasoning. In: the Second Bar-Ilan Symposium on Foundation of Artificial Intelligence, January 1991
4 Ni'ezette M, St'evenne J-M. An efficient symbolic representation of periodic time. In: International Conference on Information and Knowledge Management, 1992
5 Bertino E, Bonatti P A, Ferrari E. TRBAC: A temporal role-based access control model. ACM Trans. on Information and System Security, 2001, 4(3): 191～233
6 Joshi J B D, Bertino E, Latif U, et al. Generalized Temporal Role-Based Access Control Model. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(1): 4～23
7 Sandhu R, Coyne E, Feinstein H, et al. Role-Based Access Control Models. Computer, 1996, 29(2): 38～47