摘 要

ANSI RBAC(基于角色的访问控制)规范是目前广泛使用的存取控制模型，对该规范中访问控制模型的描述形式灵活多样，如数据库表关系形式、普通文本形式、XML 文本形式等。但上述描述形式只限定于特定的应用环境，因为缺少通用性和交互性而无法在分布式系统的自治区域中共享相关访问控制信息，这不但增大了维护成本，而且降低了系统的交互性和可移植性。

XACML（可扩展访问控制标记语言）是一个 OASIS 标准，是一种通用的用于保护资源的策略语言和一种基于 XML 的标准的访问控制语言，用于为不同的设备和应用编写访问控制策略，能够解决上述几种描述形式的不足。但 XACML 语言的灵活性和强大的表达力带来的代价是复杂和冗长，这样很难直接操作策略语言和策略文件。目前尚无成熟的商业产品或开源项目，可以方便的操作、编辑 XACML 策略，一般用户很难直接使用基于 XACML 的访问控制系统。

为解决上述问题，本文以 XACML 为访问控制策略描述语言，参考 RBAC97、RBAC99 框架和 ANSI RBAC 功能规范，实现了基于 XACML 的 RBAC 访问控制及管理系统 XRBAC，并分析其在不同应用规模下的系统性能，取得了较好的效果。

本文主要工作如下：

1、 深入分析 ANSI RBAC 的理论模型和功能需求，研究 XACML 对 RBCA 的语言特性支持，提出了基于 XACML 的 RBAC 访问控制应用模型（XRBAC），并详细描述如何将理论模型转化为具体实现。

2、 详细分析和设计了 XRBAC 应用系统，实现了部分 RBAC 管理、浏览及性能测试功能，详细描述了策略决策点 PDP、策略信息点 PIP 模块的设计和实现。

3、 详细描述了实验方法和过程，通过对 PDP 性能测试所获取的实验数据，分析系统性能，验证系统可行性。

【关键字】 访问控制、XACML、RBAC、角色、权限

Abstract

RBAC (Role-based Access Control) model is widely used, there are all kinds of ways to describe the RBAC model including Database table, plain text, xml text, and so on. All of these ways are limited in special conditions, the policy language lack of general and interactive, which make distributed system can not share the same policy, so it increase the cost of maintenance and destroy the interaction and compatibility.

XACML is an OASIS (Organization for the Advancement of Structured Information Standards) standard that is a policy language based on xml schema to describe access control decision and policies for different devices and applications, which can resolve the problems described above. The cost of flexible and expressive of XACML is more complex and prolix, which make it difficult to maintain and edit XACML policy. As to now, there are not mature commercial products and open source projects to make it easy to maintain and edit XACML policy. The access control system based on XACML is difficult to use for us. In this study, we intend to develop a new XRBAC (XACML-based RBAC) system architecture by referring the architecture of ARBAC97, ARBAC99 and ANSI RBAC Functional Specification, to implement XRBAC access control and management system and analysis the performance under different application dimensions.

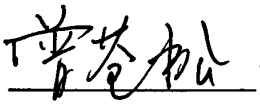
The main work of this thesis concentrated on the following parts:

1. Analysis the ANSI RBAC theory model and functional demands, study how to use the XACML to meet the requirements of role based access control, propose a XACML-based RBAC model, and describe how to implement the XRBAC.
2. Analysis and design the XRBAC application system, implement a part of functions of management, review and performance analysis, and describe how to design and implement the Policy Decision Point (PDP) and Policy Information Point (PIP) model.
3. Describe the way and process of PDP performance test, analysis the experiment data, and conclude the XRBAC is flexible and feasible.

【Keywords】 Access Control, XACML, RBAC, Role, Permission

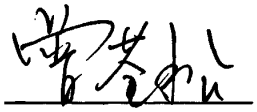
学位论文独创性声明：

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）：  2008年 6 月18日

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（含万方数据库）、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）：  2008年 6 月18日

第一章 绪论

本章介绍课题研究的背景和意义，讨论了国内外研究现状，提出了研究的必要性和价值，并阐述了本文的主要工作，最后介绍本文的组织结构。

1.1 引言

计算机网络和信息系统在当今社会扮演重要角色，信息技术带来的便捷让人愉悦。然而信息安全问题却一直困扰我们，如数据访问控制。网络环境中的信息需要受到保护，因此长时间以来计算机科学中的授权和访问控制一直处在研究当中，并且人们提出了许多访问控制机制，这些机制主要关注如何定义用户的权限，不违反访问控制策略。

访问控制技术[1][2][3]，是通过某种方式明确地准许或限制访问能力及范围的一种方法。通过访问控制服务，可以限制对关键资源的访问，防止非法用户的侵入或者因合法用户的不慎操作所造成的破坏。传统的访问控制方式包括自主访问控制 DAC（Discretionary Access Control）、MAC 强制访问控制（Mandatory Access Control）、基于角色的访问控制 RBAC（Role Based Access Control）[4][5][6][7]等。每种访问控制方式都有广泛应用，但应用的范围有所区别。

RBAC 引入角色的概念，并以角色作为授权管理的中介。系统安全管理员可根据需要定义各种角色，并为其设置合适的访问权限，然后根据用户所担任的工作职责或级别分配相应的角色，从而使用户获得相关的权限集。角色的引入使得用户的授权变得更加灵活，易于维护。RBAC 在访问控制机制中并非新概念，在商业领域有很成熟的应用。RBAC 以其特有的灵活和细粒度授权模式已被广泛采纳。

在某些应用系统中，权限都被集中管理和分配，因此对于 RBAC 的描述不需要和其他系统进行交互，只需要被本系统理解即可。而在分布式系统中，权限由各系统单独定义，不同的访问控制描述将使得系统间无法共享相关的访问控制信息，人们越来越倾向于使用独立的访问控制系统，统一的访问控制描述语言描述 RBAC 模型，以增强系统的交互性和可移植性。OASIS(Organization for the Advancement of Structured Information Standards) 于 2003 年推出 XACML（eXtensible Access Control Markup Language）作为访问控制的标准描述语言，明确了对 RBAC 的语言支持[8]，使得人们可以使用标准的访问控制策略描述 RBAC 模型，以改善分布式访问控制系统中的交互性和可移植性。

1.2 技术背景

RBAC 的主要思想就是将授权和角色联系在一起，使用户和权限分离，用户的权限被分配给合适的角色，可简化权限授权的管理。RBAC 模型在不同的系统配置下可以显示不同的安全控制功能，可以构造具备自主存取类型的系统，也可以构造成强制存取类型的系统，可通过角色权限粒度控制用户权限粒度，比较灵活。基于角色的访问控制以其特有的优势，正被广泛应用于各个系统中[9][10]。

目前基于 RBAC 的访问控制策略，通常以下几种形式描述：

（1）数据库表形式。这是目前相当一部分系统采用的访问控制策略描述方式。用户、角色、权限之间的关系，通过建立用户-角色表、角色-权限表等来描述。这种策略描述方式，适合大规模用户，且用户权限相对固定，权限适合集中管理。策略评估方式由系统自定义，系统的交互性和可移植性差。

（2）普通文本形式。在用户数量相对较少，角色关系简单，权限粒度大的应用系统中较为普遍，如 Solaris 操作系统的权限描述。其特点是信息描述简洁，解析效率高，但可读性、结构性、操作性差，在操作系统粗粒度授权中应用较多。

（3）XML文本形式[11]。角色的权限信息和角色的继承关系以XML 文件表示，在中心服务器上维护，通过在XML中定义角色和权限的标签，利用XML的层次性的特点，来表示角色之间的关系，在简单文本的基础上大大提高了系统的可读性、可操作性和灵活性。但不同的应用都需要考虑如何定义各种元素来表达用户、角色、权限之间关系，以及如何定义规则来描述策略，因此不同的应用有不同的定义。这种非标准语义的XML文本形式，同样面临交互的问题。

（4）XACML 文本形式。随着人们对基于 XML 的 RBAC 模型研究的深入和推广，越来越需要统一、规范、权威、灵活的形式表述 RBAC 关系模型及策略，以统一的 XML 语法和规则定义标准的访问控制策略和请求/响应策略。

XACML 的主要优势有[12]：

（1）XACML 是标准。标准通过了大量专家和使用者审查，开发者不再需要考虑设计一个定义语言所涉及的所有议题，不必经常更改系统。随着 XACML 的越来越广泛的开发部署，开发者将更容易通过标准语言与其它使用同一语言的应用系统协作交互。

（2）XACML 是通用的。不仅为特殊的应用环境提供访问控制，还包括任意环境的特殊资源。一个策略可以被不同的应用所使用，当使用通用语言时，策略管理变得更容易。

（3）XACML 是分布式的。这意味着一个访问控制策略可以引用其它任意位置的其它策略。其带来的好处就是可以由不同的用户或组各自管理策略中各自业务范围内、或专业领域内的子策略并最终合并为总策略，而不是集中管理一个集成的访问控制策略。由 XACML 来处理如何正确的合并不同的子策略判断结果，

并作出统一的访问控制。

（4）XACML 是强大的。标准语言已经可以支持广泛的数据类型、功能、和合并不同策略判断结果的组合规则。另外，已经有若干标准小组致力于开发扩展和概要描述，使得 XACML 可以与其它诸如 SAML 和 LDAP 之类的标准协同工作，这使得 XACML 可以得到更广泛的应用。

XACML 对于广域分布式应用环境非常适用，支持多种授权方式和模型。对于传统访问控制系统，XACML 也提供实用而明确的权限管理和定义描述。XACML 是以 XML 形式描述的开放标准，有开源项目的支持，使得其在整合异构系统中将有很好的表现，它将成为分布式授权系统很好的选择。

1.3 研究现状

目前，国内外已有许多基于 XML 的访问控制的应用研究[13][14]，以及基于 XACML 的 RBAC 安全访问控制模型的研究，策略描述语言经历了从 XML 到 XACML 的发展。基于 XACML 的用户、策略、角色管理以及 RBAC 应用，已处在积极的应用研究阶段[15][16]。

在网络应用环境中，特别是在网格应用中，策略的描述语言有各种各样。在网格安全基础设施 GSI(Grid Security Infrastructure)中，策略语言的使用是不透明的，策略的创建者和资源提供者需要能理解共同的策略语言[17]。在公共授权服务 CAS(Community Authorization Service)[18]的代理证书中可使用任意策略语言，包括 Controlled English [19], ASL [20], Ponder [21]，不同的策略描述语言，给系统的交互和维护带来诸多不便，因此在网格 CAS、GSI 应用中需要发展标准的策略语言[22]，策略语言的发展将会使网格安全服务受益。

PRIMA[23]是网格计算环境中的分布式访问控制系统，支持多向授权。允许用户作为管理者将验证过的资源访问权限授予用户，也可将精确定义的权限赋予验证的资源。目前，PRIMA 系统已经在 Globus Toolkit 的安全机制中实现。在网格的安全访问控制系统中，还有 Akenti[24]、PERMIS [25]等，这些系统使用的访问控制策略语言都基于 XML，没有统一的策略描述语言。Akenti、 PERMIS 项目都在研究 XACML，打算将其作为核心策略描述语言代替原有的策略语言。XACML 除了定义策略语言，还定义了认证的请求和响应格式。虽然 XACML 不是标准化一套完整的授权解决方案，但却成为各种解决方案组合的基础。

XACML 的规范正在不断完善和发展，XACML1.2 规范的实现主要有 SUN XACML1.2 的 Java 实现[26]和 XACML.NET 的 C#实现[27]。通过这些开源项目的支持，可以方便的生成标准的一致的请求/响应格式，并方便处理策略/策略集，通过统一的评估算法，对访问控制策略进行评估，得出一致的决策评估结果。

目前尚无成熟的商业或开源产品，可以方便的操作、编辑 XACML 策略，因

此对于 XACML 的操作，大都限于熟悉 XACML 语法的系统开发人员。XACML 语言的灵活性和强大的表达力带来的代价是复杂和冗长，这样很难直接操作策略语言和策略文件。没有方便使用的工具，一般用户很难运用基于 XACML 的访问控制系统，这些都阻碍了 XACML 应用的推广。

XACML 对 RBAC 提出了明确的支持，对于用户、角色、权限的定义也有明确的语言规范。但用户、角色、权限描述的存放位置并没有明确规定，比如是存放在同一文件中还是分开存储，或以 LDAP 形式存放。这些都可能影响到系统检索策略和评估策略的性能。目前尚未有针对策略存储方式和策略评估算法的性能测试，对于基于 XACML 的 RBAC 访问控制策略描述，其响应性能需要进一步验证。

1.4 问题的提出

依据以上研究现状和技术分析，可以看到网络环境数据访问控制中 XACML 的重要和 RBAC 优势，因此实现基于 XACML 的 RBAC 模型，是非常有意义的工作。为了更好的维护和使用 XACML 策略，必须实现基于 XACML 的 RBAC 管理系统（简称 XRBAC），以便更方便更有效地管理和维护用户、角色和权限之间的关系，提供可移植的、一致的访问控制策略，实现网络资源的跨域、细粒度授权。

合理组织策略存储结构，保证策略检索和策略评估效率，保证策略集的可扩展性。测试策略检索和评估效率，以确认在实际应用中的可行性，为基于 XACML 的 RBAC 访问控制应用提供数据参考。

1.5 本文的工作和组织结构

1.5.1 本文的工作

本论文从分析现阶段网络环境中数据访问控制面临的问题入手，分别从理论和实际应用角度拟定解决问题的措施，设计了基于 XACML 的 RBAC 访问控制管理系统，并提出具体的可行实现技术。

本文的主要内容是设计实现基于 XACML 的 RBAC 访问控制和管理系统 XRBAC。通过分析 RBAC 理论模型，依据 XACML 实现 RBAC 的描述，结合 SUN 提供的 Sun XACML1.2 API，设计实现基于 XACML 的 RBAC 各种管理功能、系统功能等，以简化基于 XACML 访问控制策略的操作和维护，并提供性能测试功能，进行数据分析和性能评估，在不同的用户规模和权限规模下，测试系统的访问控制性能和可扩展性，验证理论模型的可行性。通过大量的实验数据，分析系统的瓶颈，为其他相关研究提供可供参考的信息。

1.5.2 论文的组织

论文的具体组织结构如下：

第一章 绪论，概述本文的技术背景、研究现状、选题的意义、本文的主要工作及组织结构。

第二章 简要介绍了 XACML 在数据访问控制中的位置，分析了访问控制技术和 RBAC 模型。

第三章 通过分析 RBAC 功能规范，获取管理 RBAC 的各功能需求，以及 XACML 为支持 RBAC 提供的相关定义和描述，进行 XRBAC 系统分析和设计，描述 XRBAC 的管理功能、系统功能、性能分析功能的实现细节。

第四章 通过对 XRBAC 的策略决策点 PDP 性能测试，验证 XRBAC 理论模型的可行性，测试系统的访问控制性能、可扩展性，分析性能瓶颈并提出解决方案。

第五章 总结和展望，对整篇论文的主要工作和取得的成果进行了总结，并指出了一些需要完善的不足和对未来研究工作的一些思考和展望。

第二章 XACML 及 RBAC 概述

本章简单介绍 XACML 的基本概念和主要行为，简单介绍如何通过 XACML 描述实现访问控制。深入分析 ANSI 标准的 RBAC 模型及相关要素，通过集合描述各要素之间的关系，为实现基于 XACML 的 RBAC 模型提供理论支持。初步研究 RBAC 管理模型，了解 RBAC 模型的管理功能的需求。

2.1 XACML 简介

目前，许多访问控制和授权系统都以各自的方式实现，所以都限定于特定的应用环境而无法在开放的网络环境中使用，因为无法在不同的区域中共享相关的访问控制信息。在没有XACML的时候，需要为网络编写和维护多个身份验证系统，因此需要一种通用语言表达不同的访问控制策略[28]。2003年2月，可扩展访问控制标记语言XACML获得了批准，成为了一个OASIS标准。XACML定义了一种通用的用于保护资源的策略语言和一种基于XML的标准的访问控制语言，用于为不同的设备和应用编写访问控制策略。

XACML 包括访问控制语言和请求/响应（Request/Response）语言两部分。访问控制语言使得开发者能以统一的语法和规则，描述谁何时何种情况下能做什么，这使得不同的节点资源能相互理解彼此的策略规则。请求/响应语言用来描述访问请求和响应结果。图 2.1 描述了 XACML 中主要行为的应用流程。

当客户端向服务端提出访问请求(Access Request)时，由策略执行点 PEP（Policy Enforcement Point）执行。为了执行授权的策略，这个实体将规范化策略描述信息，向策略决策点 PDP（Policy Decision Point）发出委托授权请求。可用的策略位于策略信息点 PIP（Policy Information Poin）中，依据资源属性由策略决策点 PDP 评估请求，然后返回授权结果。根据响应信息，策略执行点可以向客户端做出适当的反应。访问请求到达策略执行点 PEP 后，PEP 创建一个 XACML 请求并发送到策略决策点 PDP，由 PDP 评估请求并返回一个响应。该响应可以是允许访问，也可以是拒绝访问，可同时具有适当的义务(Obligations)。PDP 评估请求中的相关策略和规则后会做出决策。可以应用的策略有多种，PDP 并没有评估所有的策略，而是根据策略目标选择相关的策略进行评估。策略目标包括关于主体、动作和其他环境属性的信息。为了获得策略，PDP 要用到策略访问点 PAP，PAP 编写策略和策略集，供 PDP 使用。PDP 也可以调用策略信息点 PIP 服务检索与主体、资源或者环境有关的属性值。PDP 做出的授权决策被发送到 PEP。PEP 履行义务，并根据 PDP 发送的授权决策允许或拒绝访问。

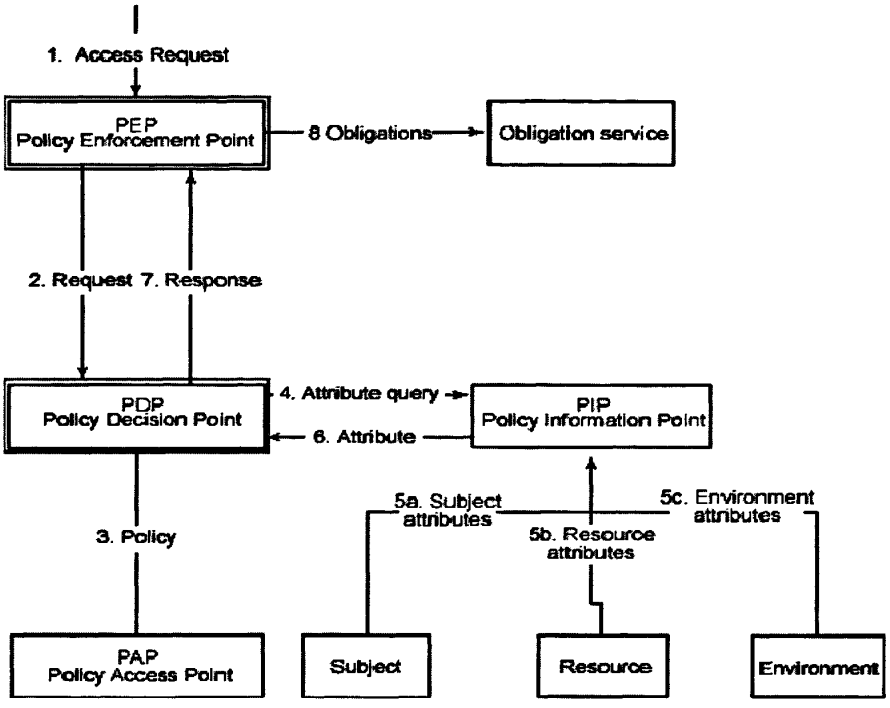


图 2.1 XACML 应用流程

在 XACML 出现之前，应用开发者必须创建自己的访问控制描述方式，很显然这种描述无法和其他的应用策略交互。XACML 的出现，是为了替代现存的面向应用的特殊策略描述方式，如 Akenti 和 PERMIS 中基于 XML 的策略。目前，XACML 已获得了初步应用[29]，XACML 的第一个 Java 实现已由 Sun Microsystem Inc.开发完成[30]。

OASIS 为了满足 RBAC 应用的需求，已经明确提出了 XACML 对 RBAC 的语言特性支持。支持核心 RBAC(Core RBAC)，继承 RBAC(Hierarchical RBAC)和职权分离模型。XACML 支持 RBAC 具体的语言特性在 3.1.4 节中说明。

2.2 RBAC 模型分析

RBAC 已不是什么新出的概念，早在 20 世纪 70 年代，基于角色的访问控制（Role-Based Access Control）的概念已随着多用户多应用在线系统的产生而出现[31]。RBAC 的核心概念就是访问权限和角色相关联，而将用户赋予特定的角色，从而使用户关联特定的权限。在用户（user）和权限（permission）之间引入角色（role）的概念，用户和特定的一个或多个角色相关联，角色可依据实际应用建立或取消，这大大简化了访问权限的管理。角色依据组织中的各种工作需要创建，用户依据不同的职责被赋予不同的角色。用户可以很方便地从一个角色转换为另一角色。角色可以被赋予系统相应的权限，权限也可在必要时被撤销。1992 年，

Ferraiolo 和 Kuhn 正式首先提出了 RBAC 的概念[31]。1996 年，由 Sandhu 等提议的由用户、角色、访问权限和 Sessions 组成的 RBAC 一系列模型被正式提出，如 RBAC96[4] 等，其中和 ANSI 模型最接近的为(美国)国家标准技术研究所（National Institute of Standard Technology）NIST 模型[32]。

2004 年，RBAC 被美国国家标准委员会（ANSI）和 IT 国际标准委员会（INCITS）接纳为 ANSI INCITS 359-2004 标准[2]，描述了 RBAC 的统一模型。RBAC 标准包括两个主要部分：RBAC 参考模型和 RBAC 功能描述。RBAC 参考模型定义了 RBAC 的基本定义和基本元素集合，并通过集合论给出了一套 RBAC 的数学模型。RBAC 功能描述定义了 RBAC 系统必须具备的功能，包括管理功能（Administrative Functions），系统功能（Supporting System Functions）和浏览功能（Review Functions）。

在图 2.2 中，RBAC0 作为最基本的模型，满足 RBAC 系统中最少的需求。RBAC1 和 RBAC2 都包含 RBAC0，RBAC1 增加了角色继承关系（角色能继承其他角色的访问权限），而 RBAC2 增加了约束条件。最顶层的模型是 RBAC3 包含 RBAC1 和 RBAC2，通过传递也包含了 RBAC0。RBAC0 由除角色继承和约束的其他元素组成。其中显示了四个实体：用户、角色、访问权限和会话。

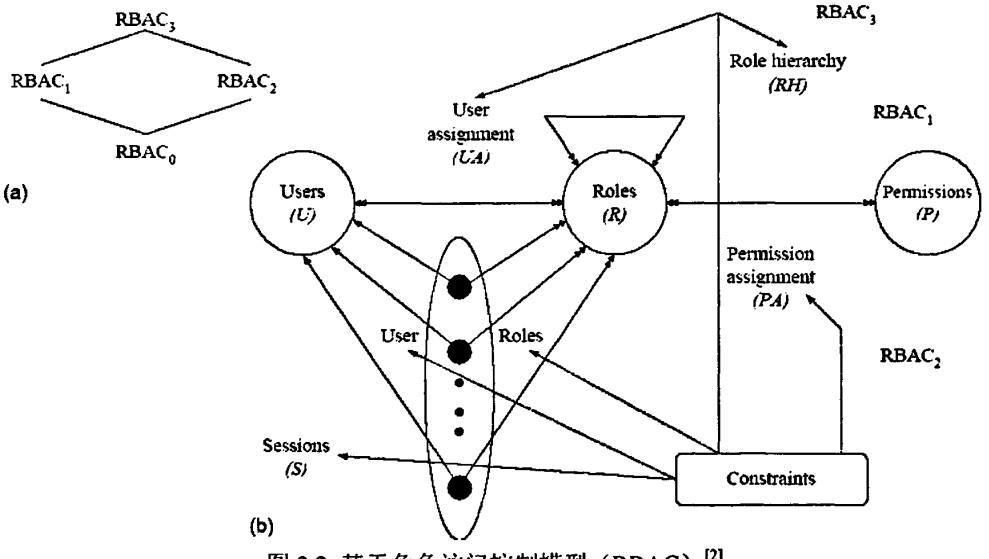


图 2.2 基于角色访问控制模型（RBAC）[2]

RBAC作为灵活高效的授权机制，在许多商业领域中已有广泛应用。RBAC模型被分为三个等级不断提升的功能层次：核心RBAC，继承RBAC和约束RBAC。后一层次是在前一层功能需求基础上的累积和扩展。如果没有标准模型作为参考将会导致功能和描述上的不确定和混乱。ANSI模型组合了各种RBAC模型、商业产品和研究模板相一致的概念，发展成为进一步的标准。ANSI模型和NIST模型比较一致，只做了很少的修正。核心RBAC对应RBAC0,继承RBAC对应RBAC1,约束RBAC则对应RBAC2。

将ANSI模型作为RBAC参考模型有两个目的：

（1）参考模型定义了标准中RBAC的特性。明确了包含在所有RBAC系统中特性元素的最小集合，有角色继承，静态约束和动态约束。

（2）参考模型使用精确和一致的语言定义了一套元素集合和功能规范。

RBAC主要组成模块包括核心RBAC、继承RBAC、以及静态职权分离SSD（Static Separation of Duty）和动态职权分离DSD（Dynamic Separation of Duty）关系，如图2.3所示。以下为参考模型及相关内容：

核心RBAC定义了RBAC元素的最小集合(包括用户-角色赋值和权限-角色赋值关系)。

继承RBAC定义了角色中的继承关系。

静态职权分离关系在角色集合上添加约束特别是在形成UA关系时。(即用户不能同时被赋予两个相互约束的角色)

动态职权分离关系允许用户被两个或两个以上的角色在行为不冲突的情况下动态授权。

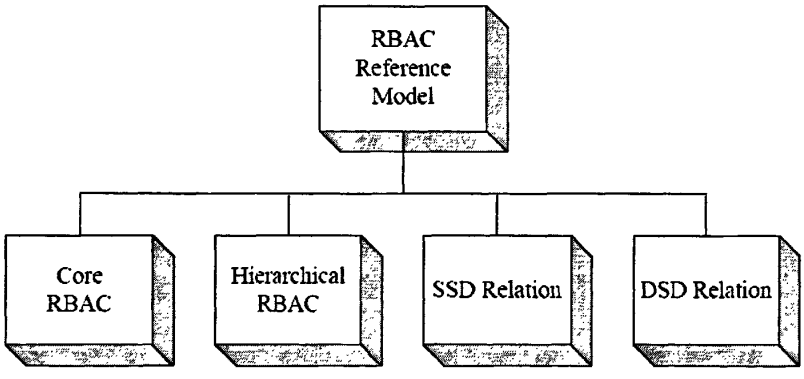


图 2.3 RBAC 组件的参考模型

2.2.1 核心 RBAC 模型

RBAC 模型作为一个整体，主要定义了被赋予角色的用户和被赋予权限的角色。这样一来，角色就在用户和权限之间，起到了多对多映射关系的作用。此外，核心 RBAC 模型还包括一套会话（SESSIONS），每个会话都是一个用户和与之对应角色的有效集合之间的映射关系。

核心 RBAC 模型主要由三个实体组成：用户(U)，角色(R)和权限(P)。它主要描述了 RBAC 的基本概念：用户被赋予相应的角色，角色被赋予相应的权限。这使得用户以角色为中介获取相应的权限。

RBAC参考模型定义了一套基本RBAC元素（用户、角色、许可、操作和对

象资源）及其功能关系。

对象资源：在此标准中，一个对象指的是任何受控的系统资源，如一个文件、打印机、终端、数据库记录、等。

操作：一个操作指一个可执行过程，它为用户提供某些可调用功能。

权限：在一个或多个受RBAC访问控制的对象资源上进行某项操作的操作许可。

角色：执行特定任务的权利或在组织中已被授予一定责任的工作头衔。它代表一种资格、权利和责任。

用户：一个用户被定义为一个人。尽管用户的概念可以被扩展，包括机器，网络，或智能自治代理，为了便于理解这里只限于人。

操作和对象资源的类型取决于其所在的系统类型。例如，在文件系统中操作就包括读、写和执行；在数据库管理系统中，操作就包括插入、删除、追加和更新。

访问控制机制的目的就是保护系统资源。和原来的访问控制模型一致，对象资源就是包含或接收信息的实体。在 RBAC 系统中，对象资源能表示信息（如操作系统中的文件、目录，数据库系统中的列、行、表和视图）或表达有限的系统资源如打印机，磁盘空间,CPU 周期等。RBAC 所涵盖的对象资源都列在赋予角色的权限中。

RBAC 的中心是角色关系的概念，角色则是描述策略的语义构造。

图 2.4 说明了用户赋值（UA）和权限赋值（PA）关系。箭头表示了一个多对多的关系（例如，一个用户能被赋予多个角色，一个角色能被指派给不同的用户）。这种安排提供了很好的灵活性和权限到角色及用户到角色的细粒度赋值。在便利之外同样有隐患，由于在用户和资源上缺少有效的访问控制手段，用户对资源的操作可能被赋予过多的权限。例如，用户需要列出目录或是更改文件而不需创建新的文件，或需要在文件中添加记录不需更改存在的记录。任何对资源访问控制灵活性提升的同时，都需要强调最小权限的应用原则。

每个会话是一个用户对可能多个角色的映射，在用户激活与之相关的角色集时产生会话。每个会话对应一个用户，而每个用户与一个或多个会话相关联。session_roles 提供会话激活的角色，session_users 提供和会话相关的用户。对当前用户有效的权限是当前用户所有会话所包含的角色被赋予的权限。

核心 RBAC 规范：

（1）USERS、ROLES、OPS 和 OBS（用户，角色，操作和对象资源）。

（2）UA $\subseteq$ USERS $\times$ ROLES，一个多对多映射用户－角色的赋值关系。

（3）assigned_users: (r:ROLES) $\rightarrow$ 2USERS，角色 r 映射的用户集合。公式：assigned_users(r) = {u$\in$USERS | (u, r) $\in$ UA}

（4）PRMS = 2(OPS × OBS)，权限集合。

（5）PA ⊆ PERMS × ROLES，多对多映射权限-角色赋值关系。

（6）assigned_permissions(r: ROLES) → 2PRMS，角色 r 映射的权限集合。公式：assigned_permissions(r) = {p∈PRMS | (p, r) ∈ PA}

（7）Op(p: PRMS) → {op⊆OPS}，权限对应操作的映射，给出与权限相关的操作集。

（8）Ob(p: PRMS) → {ob⊆OBS}，权限对应对象资源的映射，给出了与权限相关的对象资源集。

（9）SESSIONS = 会话集合。

（10）session_users (s: SESSIONS) → USERS，会话对应的用户集合。

（11）session_roles (s: SESSIONS) → 2ROLES，会话所映射的角色集合。 公式: session_roles (si) ⊆ {r∈ROLES | (session_users (si), r) ∈ UA}

（12）avail_session_perms(s:SESSIONS) → 2PRMS，当前会话中的有效权限。

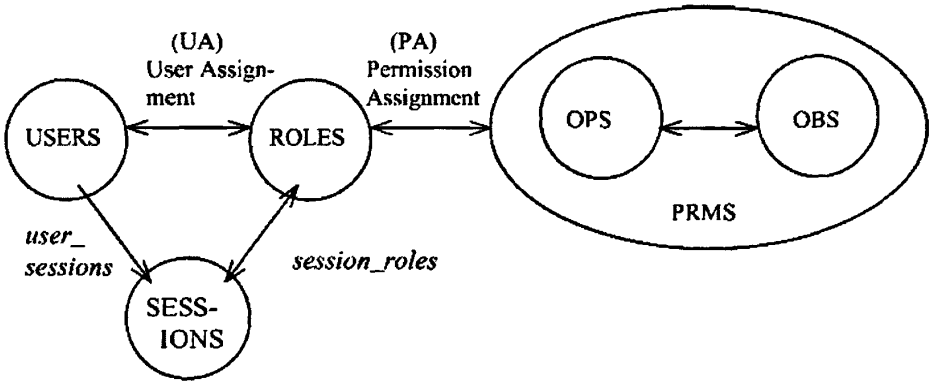


图 2.4 核心 RBAC（Core RBAC）[2]

2.2.2 继承 RBAC 模型

本模型构造介绍如图 2.5 的角色继承。角色继承是 RBAC 模型的重要概念，通常包含在 RBAC 产品中。继承是一个表示角色构成的自然方式，来反映一个组织的权限和责任。角色继承定义了角色的继承关系，继承用权限的术语表示，如 r1“继承了”r2 则表示 r1 包含了 r2 的所有权限。一般考虑两种类型的角色继承，通用角色继承和受限角色继承。通用角色继承提供一种偏序继承方式，包括权限和角色中用户成员的多继承。受限角色继承则限制一个简单的树结构（如一个角色可以有一个或多个直接后继，但只允许有一个直接前驱）。

（1）通用角色继承:

RH ⊆ ROLES × ROLES表示角色继承关系，写作≥ ，r1 ≥ r2表示当且仅当凡r2拥有的权限r1也都拥有，并且所有r1的用户也必然是r2的用户，例如：r1 ≥ r2⇒ authorized_permissions(r_2)⊆ authorized_permissions(r_1)。

authorized_users(r: ROLES) $\rightarrow 2^{USERS}$,角色继承下的角色r映射的用户集，公式：authorized_users(r) = {u∈USERS | r' ≥ r, (u, r') ∈ UA}。

authorized_permissions(r: ROLES) $\rightarrow 2^{PRMS}$,角色继承下的角色r映射的权限集，公式：authorized_permissions(r) = {p∈PRMS | r' ≥ r, (p, r') ∈ PA}。

（2）受限继承模型

受限继承模型中角色只限定在单个直接前驱。节点 r1 对节点 r2 直接继承表示为 r1 ≻≻ r2，如果 r1 ≥ 2，在 r1 和 r2 之间没有其他角色。也就是在角色继承中，不存在第三角色 r3 使得 r1 ≥ r3 ≥ r2,其中 r1 ≠ r2 且 r2 ≠ r3。

对于受限角色继承有如下定义：

∀r,r1,r2∈ROLES, r ≥ r1 ∧ r ≥ r2 ⇒ r1 = r2.

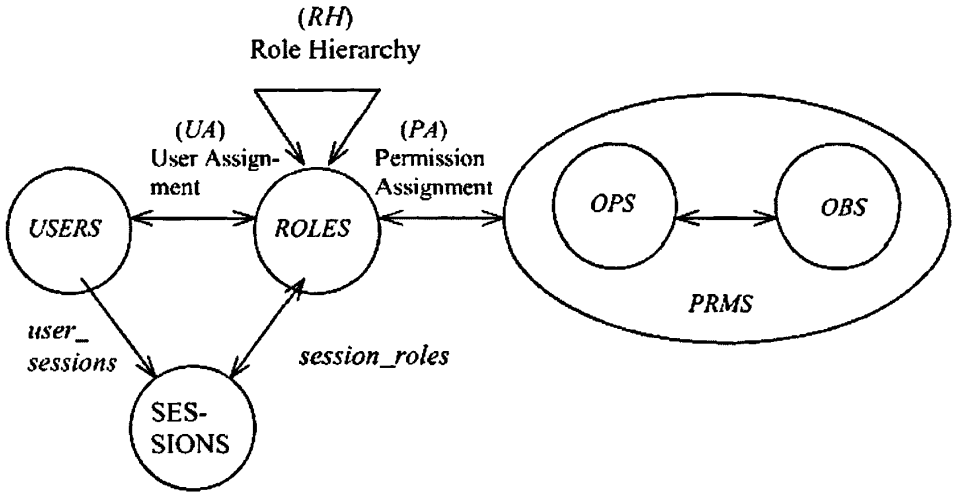


图 2.5 继承 RBAC 模型（Hierarchical RBAC）[2]

2.2.3 约束型 RBAC 模型

约束型 RBAC 在 RBAC 模型中添加了职权分离关系。职权分离关系用来加强解决策略冲突，避免组织中的用户获得过高的权限级别。做为一个安全策略，职权分离被广泛应用于商业、工业和政府部门。确保个体不同的职责或不同的利益都分配给不同的商业功能。目的是确保在没有得到多方共同参与的情况下，不会产生欺骗和主要错误。RBAC 标准定义了动态和静态的职权分离。

（1）静态职权分离

基于角色的系统中的利益冲突（Conflict of interest）往往是由于用户获得的授权和相应的角色冲突。一种防止此类利益冲突的方式是通过静态职权分离，即在将用户赋予角色时加强约束。静态约束有许多形式，处理静态职权分离(SSD)的通用方式是为用户相应地定义多个不相交角色集。静态约束被认为是实现其他重要职权分离策略的有效方式。

在此模型中定义的静态约束只限于如下关系，对角色集合特别是 UA 赋值行

为的约束。这是指如果一个用户被赋予一个角色，这个用户就被禁止赋予另外某个角色。SSD 策略可以集中分类，然后统一应用于不同角色。从策略的全局看，静态约束关系提供了一个强有力的方式解决利益冲突以及提供权限集合的分离规则。静态约束通常实施在管理操作上，这些操作有可能破坏更高层次的职权分离组织策略。

RBAC 模型已经定义了 SSD 关系约束用户-角色的多角色赋值（如，在 SSD 下没有用户能被同时赋予两个角色）。现实中也存在 SSD 策略，这些定义在两方面有过度限制。一方面是 SSD 中角色集合的大小，另一方面是用户赋值时角色集合的组合是受限制的。

如图 2.6 描述，SSD 关系可能存在于继承 RBAC 模型。当在角色继承中实现 SSD 关系时，必须注意确保继承性不破坏 SSD 策略。角色继承关系就包含 SSD 约束的继承。为应对这种潜在的矛盾，SSD 对被授权的 SSD 角色用户定义了一个约束。

静态职权分离模型公式描述

SSD $\subseteq (2^{ROLES} \times N)$ 是静态职权分离中的集合(rs, n)，每个 rs 是一个角色集合，t 是 rs 的子集，n 是≥2 的自然数，从集合 rs 中用户不能被赋予 n 或 n 个以上的角色，对于每个(rs, n) $\in$ SSD。

公式：$\forall (rs, n) \in SSD,\ \forall t \subseteq rs:\ |t| \geq n \Rightarrow \bigcap_{r \in t} assigned\_user(r) = \emptyset$。

继承关系下的静态职权分离：

角色继承关系下的静态职权分离的定义是基于用户授权而不是用户赋值。

公式：$\forall (rs, n) \in SSD,\ \forall t \subseteq rs:\ |t| \geq n \Rightarrow \bigcap_{r \in t} authorized\_user(r) = \emptyset$。

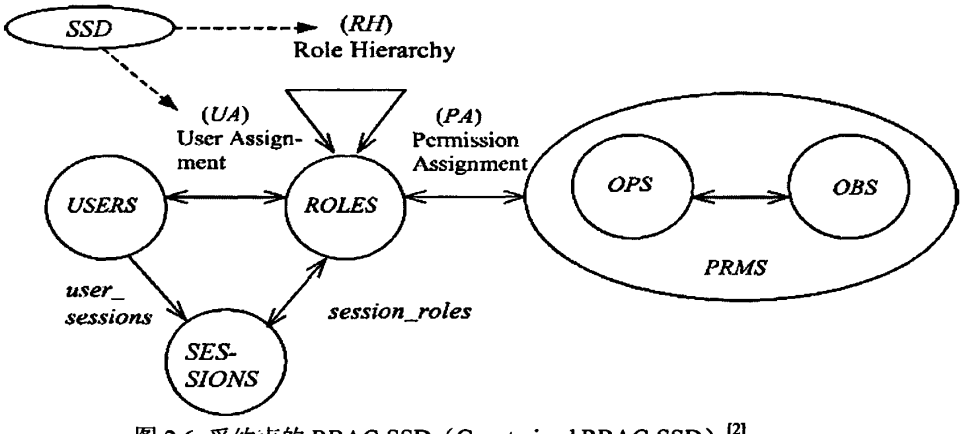


图 2.6 受约束的 RBAC-SSD（Constrained RBAC-SSD）[2]

（2）动态职权分离模型

通过在将用户赋值给角色集合时添加约束，静态职权分离减少了可能的有效权限。动态职权分离和静态职权分离一样，试图限制用户的有效权限。但是，DSD 和 SSD 在特定环境下的限制有所不同。DSD 的约束作用于用户的整个权限

空间，如图 2.7。此模块构件定义 DSD 特性，通过在用户会话或被激活的角色上设置约束，来限制用户权限空间的有效权限集。根据用户扮演的不同角色，每个用户在不同时间具有不同的权限等级，DSD 能提供额外的最小特权应用原则的支持。这些特性确保权限存在的时间不会超过执行某职责时所需要的时间。最小特权通常是指适时地撤销信任。离开动态职权分离的实现，动态权限撤销会变得相当复杂，在过去为了方便动态撤销权限都被省略。

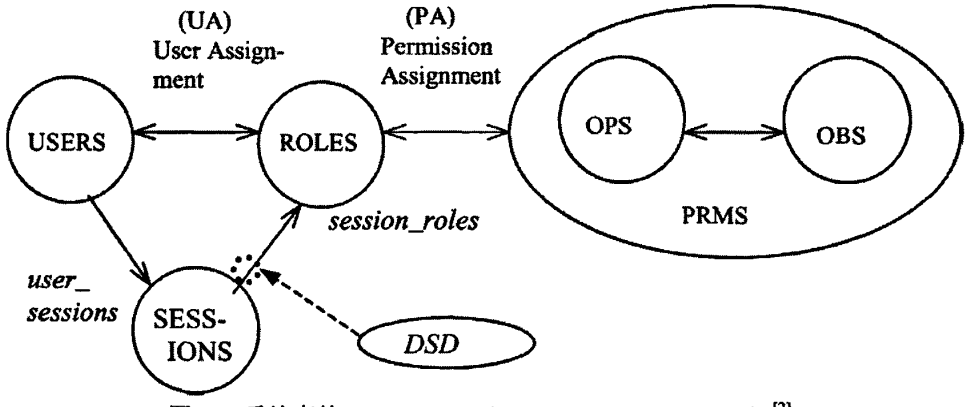


图 2.7 受约束的 RBAC-DSD（Constrained RBAC-DSD）[2]

DSD $\subseteq$ (2^{ROLES} × N)动态职权分离中的(rs,n)2 维向量集合,rs 是角色集合，n 是≥2 的自然数，对于每个 dsd∈DSD,从 rs 中不能获得 n 或 n 个以上的有效角色。

公式：$\forall rs \in 2^{ROLES}, n \in N, (rs, n) \in DSD \Rightarrow n \geq 2.$

$|rs| \geq n$ 且 $\forall s \in SESSIONS, \forall rs \in 2^{ROLES}, \forall role\_subset \in 2^{ROLES}, \forall n \in N, (rs, n) \in DSD, role\_subset \subseteq rs, role\_subset \subseteq session\_roles(s) \Rightarrow |role\_subset| < n$。

2.4 RBAC 管理模型

在大型企业级系统中，角色和用户数目可能成千上万，管理这些角色、用户及其关系，这项艰巨的任务通常集中在安全管理团队。RBAC 的主要优势就是方便管理，很自然会考虑怎样使用 RBAC 自身模型管理 RBAC。并且，使用 RBAC 管理 RBAC 将是未来 RBAC 成功的重要因素。

通过研究 Sandhu ARBAC97[33]，ARBAC97SS [34]模型,首先将管理划分为角色赋予用户，权限赋予角色，和角色赋予角色的角色继承，并称之为 ARBAC97（administrative RBAC）模型。随后，随着对 ARBAC 的进一步研究，出现了更为完善的 ARBAC99[35]和 ARBAC02 模型 [36]。

2.4.1 ARBAC97 模型

管理基于角色的访问控制模型的思想是分散管理角色和权限。ARBAC97 由三个组成部分：URA97（user-role assignment'97），PRA97（permission-role assignment'97）和 RRA97（role-role assignment'97），分别处理 RBAC 管理的不同方面。URA97 关注用户-角色管理，PRA97 关注权限-角色管理，RRA97 关注角色-角色管理，关系结构如图 2.8。

ARBAC97 模型为实现 RBAC 管理提供了一种思路。ARBAC97 中定义的关系，都可被视为 ARBAC 的系统功能。

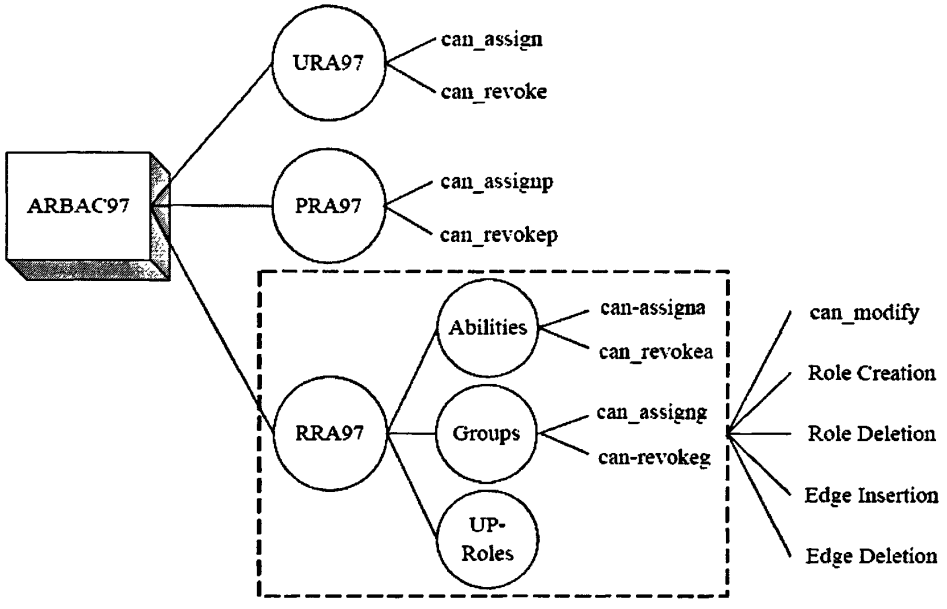


图 2.8 ARBAC97 中的功能关系[33]

2.4.2 ARBAC99 模型

ARBAC99 由三部分组成：URA99,PRA99 和 RRA99。它是 ARBAC99 模型的扩展，在用户和角色权限中添加了活动和非活动成员的概念，就是 URA99 和 PRA99。在 URA99 中，将用户赋予角色的非活动赋值方式，可以使用角色相应的权限，在任何进一步赋值时，不再确认其成员关系。而活动赋值方式则在进一步赋值或访问角色权限时，都需要确认。

PRA99 模块处理角色权限的赋值和撤销。赋予角色的用户和权限可以是活动的也可以是非活动的。就像 PRA99 是 PRA97 的扩展一样，URA99 和 RRA99 分别是 URA97 和 RRA97 的扩展，结构如图 2.9。

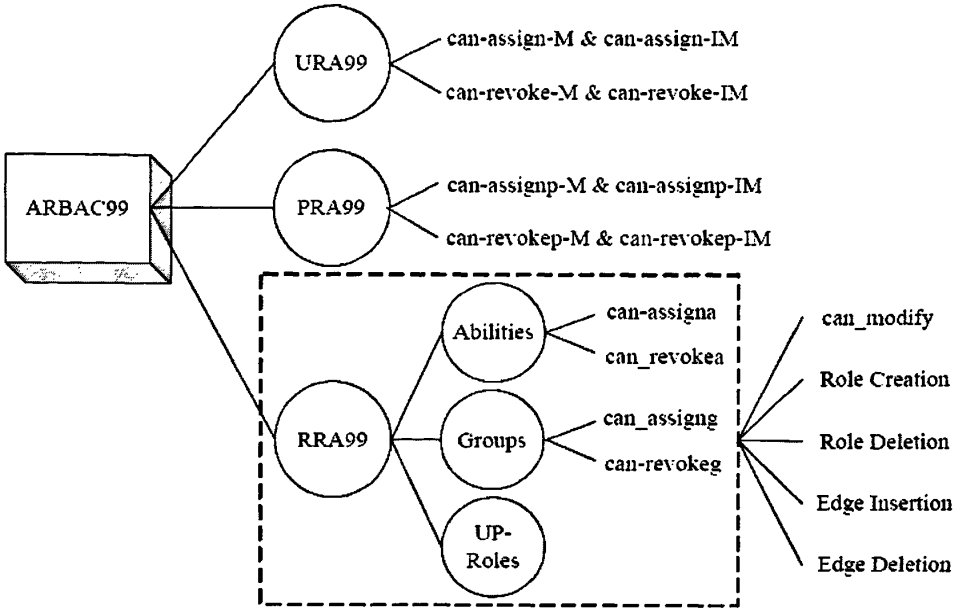


图 2.9 ARBAC99 中的功能关系[35]

2.4.3 ARBAC02 模型

ARBAC02 模型由 Sandhu 在文献 [36] 中有介绍，列出了在 ARBAC97 模型中，管理用户－角色及角色－权限关系时可能产生的问题（图 2.10）。为克服 ARBAC97 模型中的缺点，可采取两种策略。首先，在组织结构上用新的用户和权限池代替角色继承下的必要角色。在 ARBAC97 模型中，用户和角色池依赖于角色继承结构。而在 ARBAC02 中，组织单元的概念成了定义用户和权限池的基础，它独立于角色继承。将用户或权限赋予池与赋予角色无关。独立的用户和权限池增加了用户和权限池的灵活性，克服了 ARBAC97 模型中确定性的缺点。其次，ARBAC02 模型中建议以自底向上的方式管理权限-角色关系，和 ARBAC97 的自顶向下的方式相反。相对普通的权限赋予较低层次的角色，较高层次的角色从较低层次的角色继承权限，也可提供其他特别的权限。

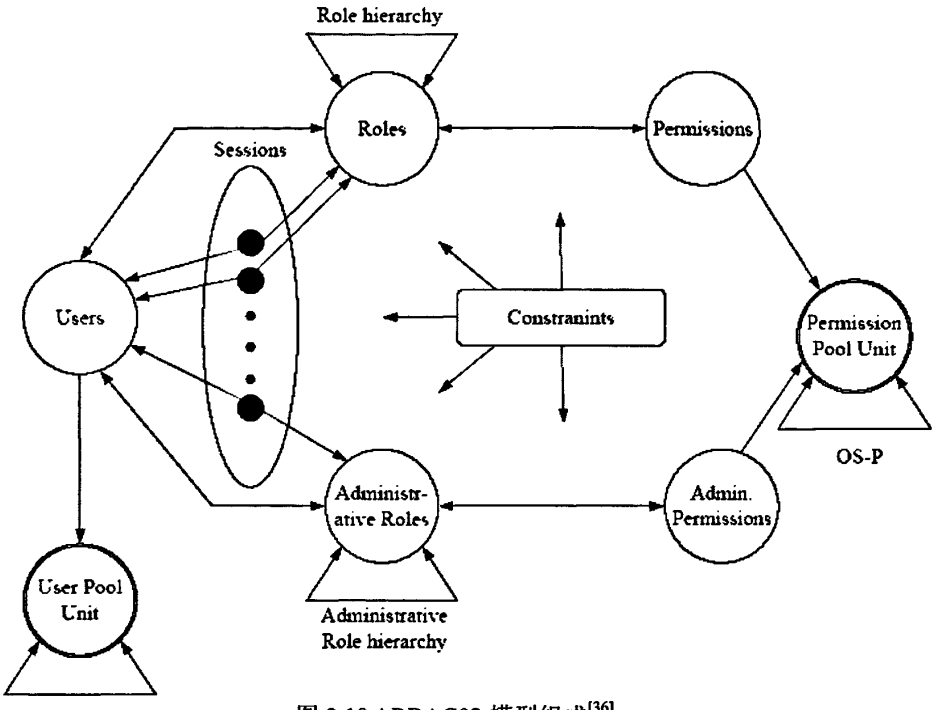


图 2.10 ARBAC02 模型组成[36]

2.5 RBAC 系统管理功能说明

RBAC 系统和管理功能规范说明了 RBAC 系统所必需的特性。主要包括如图 2.11 所示的 3 类，以下分别进行说明。

（1） 管理功能—创建和维护各种 RBAC 模块的元素集合和关系。

（2） 系统功能—在用户和系统交互时，支持 RBAC 模块的构造（如 RBAC 会话属性和访问决策逻辑）。

（3） 浏览功能—浏览管理功能所产生的结果。

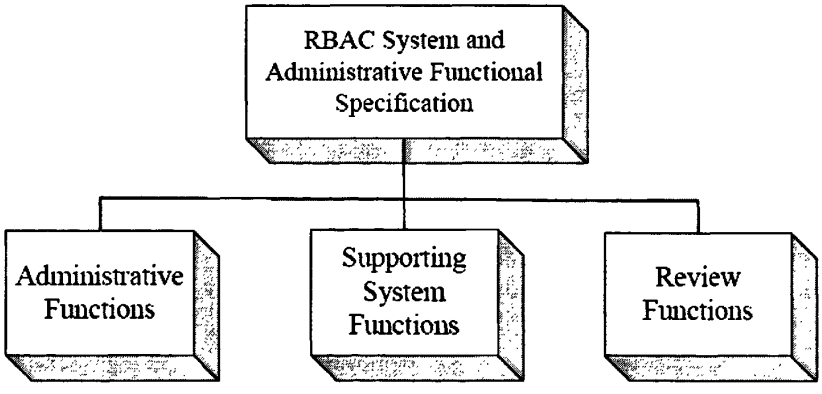


图 2.11 RBAC 组件的系统和管理功能

2.6 本章小结

本章简要介绍 XACML 的主要概念模型，分析 XACML 在网络环境中处理访问控制的优势。分析 ANSI RBAC 的理论模型和 RBAC 管理模型，总结 RBAC 管理功能的各个功能模块组成结构，为基于 XACML 的 RBAC 授权系统和管理系统的总体设计、实现奠定良好的理论基础，并获取所需的功能需求。

第三章 基于 XACML 的 RBAC 系统设计和实现

本章详细分析核心 RBAC 和继承 RBAC 各功能模块中的功能组成，抽象出各个功能模块的功能点，并分析 XACML 对 RBAC 提供的语言支持，通过 XACML 描述，实现 RBAC 各功能模块，为实现 XRBAC 的访问控制和管理系统提供语义支持。设计了基于 XACML 的 RBAC 管理系统框架结构，实现了 RBAC 核心和继承模型的主要功能，描述了包括用户管理、角色管理、权限管理和静态职权分离管理功能的具体实现。

3.1 系统分析

要实现基于 XACML 的 RBAC 访问控制系统，必须对各 RBAC 模型描述的功能进行分析，抽象出各系统功能。

3.1.1 核心 RBAC 功能规范

3.1.1.1 核心 RBAC 管理功能

核心 RBAC 中的管理功能主要是创建和维护元素集。核心 RBAC 中的基本元素集是 USERS、ROLES、OPS 和 OBS。在这些元素集中，OPS 和 OBS 是在 RBAC 配置时底层信息系统预定义好的。管理员创建、删除用户及角色，并在角色和已存在的操作和对象资源间建立关联。对用户的管理功能是 AddUser 和 DeleteUser，对角色的管理功能是 AddRole 和 DeleteRole。

创建和维护关系：核心 RBAC 两个主要关系是 User-to-Role 赋值关系（UR）和 Permission-to-Role 赋值关系（PR）。创建和删除 User-to-Role（UR）关系的功能是 AssignUser 和 DeassignUser。对于 Permission-to-Role 赋值(PR)需要 GrantPermission 和 RevokePermission 功能。

3.1.1.2 核心 RBAC 系统功能

系统功能需要会话管理和执行访问控制决策。在用户会话中，有效角色需要调节访问控制。在用户会话开始，要为用户产生默认的有效角色集。通过添加删除角色，用户在会话过程中可以改变默认值。以下为添加、取消有效角色和其他辅助功能：

（1）CreateSession，创建用户会话并为用户提供默认的有效角色集。

（2）AddActiveRole，为当前会话添加一角色作为有效角色。

（3）DropActiveRole，在当前会话中从有效角色集中删除一角色。

（4）CheckAccess，决策会话主体是否有权限对某对象资源执行请求的操作。

3.1.1.3 核心 RBAC 浏览功能

当创建 User-to-Role 赋值（UR）和 Permission-to-Role 赋值（PR）实例时，应该可以从用户和角色的角度浏览相关内容。例如，对于 UA 关系，管理员应该可以方便地浏览指定角色的所有相关用户和指定用户的所有相关角色。此外，应该还可以浏览系统功能产生的结果来决定某些会话特性，如特定会话的有效角色，以及所在的权限范围。当然，并不是所有的 RBAC 实现提供浏览角色、用户和会话权限或会话有效角色的功能便利，这些功能在规范中是可选的。强制的（M）和可选的（O）的浏览功能有：

（1）AssignedUsers（M），返回赋予指定角色的用户集。

（2）AssignedRoles（M），返回赋予指定用户的角色集。

（3）RolePermissions（O），返回授予指定角色的权限集。

（4）UserPermissions（O），返回指定用户的相关权限，此权限来源于用户被赋予的角色。

（5）SessionRoles（O），返回与会话相关的有效角色集。

（6）SessionPermissions（O），返回会话的有效权限集(会话中所有效角色相关权限的并集。

（7）RoleOperationsOnObject（O），返回指定角色在指定对象资源上的操作集。

（8）UserOperationsOnObject（O），返回指定用户在指定对象资源上的操作集 (直接获取或通过相应的角色获取)。

3.1.2 继承 RBAC 功能规范

3.1.2.1 继承 RBAC 管理功能

继承 RBAC 包括所有核心 RBAC 中的管理功能。但是，DeassignUser 的语义必须重新定义，因为角色的继承性使用户的授权角色的概念变化。即用户的角色是通过继承得来而非直接赋予。继承性使用户可从指定角色的后继角色中获取权限。一个重要的议题是用户只能从直接赋予的角色剥离还是能从（间接）授权的

角色剥离。适当的行为过程留给具体实现，规范中并未描述。

继承 RBAC 模型的附加管理功能就是关于创建和维护角色的偏序关系。其操作包括：创建（删除）角色集合中两个已存在角色的继承关系和在恰当的层次位置添加一新建的角色使其成为层次上某角色的前驱或后继。命名及相应功能如下：

（1）AddInheritance（O），在两个已有角色间添加直接继承关系。

（2）DeleteInheritance（O），删除两角色间的直接继承关系。

（3）AddAscendant（O），创建一新角色并添加为已有角色的直接前驱。

（4）AddDescendant（O），创建一新角色并添加为已有角色的直接后继。

模型提供了普通和受限继承两种。普通继承允许多继承而受限继承只允许单继承。对于受限继承，AddInheritance 被约束为单个前驱。DeleteInheritance 的功能在不同情形下会产生不同结果。当 DeleteInheritance 被调用时，关系到两个角色 A 和 B，系统实现时需要考虑以下两情况之一。系统需要维持角色 A 和 B 同其他角色的层次关系。如角色 A 通过 B 继承于 C,在 A 和 B 的关系被删除后，A 需要保持 C 的权限。另一情形是割断这些关系，因为 A 和 B 之间的关系已不存在。DeleteInheritance 的语义留给具体的系统实现。

3.1.2.2 继承 RBAC 系统功能

继承 RBAC 和核心 RBAC 一样提供相同的系统功能。但由于角色继承的出现，CreateSession 和 AddActiveRole 的语义需要重新定义。在角色继承中，指定角色继承于一个或多个角色。当一个角色为用户的有效角色时，问题是被继承的角色是自动被设置为有效角色还是需要明确说明，这个议题要视具体实现而定，规范中未有说明。但是如果是明确说明就要前后一致。例如，CreateSession 的功能中创建的新会话的有效角色集，不仅包括用户的直接相关角色，还包括所有的间接继承角色。同样在 AddActiveRole 中,用户能激活直接被赋值的角色及一个或多个继承角色。

3.1.2.3 继承 RBAC 浏览功能

在核心 RBAC 中的浏览功能在继承 RBAC 模型中仍有效。指定角色的用户成员集不仅包括直接赋予指定角色的用户，还包括继承角色的相关用户。同样，指定用户的角色成员集不仅包括直接赋予指定用户的角色，还包括继承于直接赋值角色的继承角色。为获取扩展的“角色的用户成员集”和“用户的角色成员集”，定义了以下功能：

（1）AuthorizedUsers（M），返回指定角色的直接赋值用户集，以及“指定角

色的被继承角色"的用户成员集。

（2）AuthorizedRoles（O），返回指定用户的直接赋值的角色集以及"直接赋值角色的被继承角色集"。

由于继承关系的存在，指定角色的权限集不仅包括指定角色的直接相关权限集，还包括指定角色的被继承角色的权限集。相应地，用户的相关角色权限集也会扩展。并非所有的 RBAC 实现提供此便利，这些功能是可选的。"权限浏览"功能如下：

（3）RolePermissions（O），返回直接被授予角色的权限集以及被继承角色的权限集。

（4）UserPermissions（O），返回指定用户被授权的角色的权限集 (直接被赋值的角色集以及其被继承的角色集之和。

（5）RoleOperationsOnObject（O），返回指定角色对某指定对象资源的（直接或继承的）操作集。

（6）UserOperationsOnObject（O），返回指定用户对某指定对象资源的操作集(从用户赋值的角色集及被继承的角色集中获取)。

3.1.3 RBAC 功能模块组成

实现复杂的 RBAC 模型，需要整合许多 RBAC 相关的研究。在理论上 RBAC 的思想容易实现，要具体实施则会相对复杂，基于角色的 RBAC 管理则成为关键。首先，通过整合相关的 RBAC 研究介绍系统构架，然后分析 XACML 对 RBAC 的语法及语义支持，设计基于 XACML 的 RBAC 管理、系统及浏览功能。

RBAC 的主要优势就是方便管理，可以分散管理 URA,RPA 和 RRA 的操作。在前面相关 RBAC 研究的基础上，可以给出 RBAC 管理的整体框架，如图 3.1。引用了 ARBAC97 的 3 个主要部分，分别是 URA,PRA 和 RRA 以及 ANSI 规范的建议功能。相应的 RBAC 管理由 3 大主要功能，包括管理功能，系统功能和浏览功能，每块功能有自身的功能规范。而管理功能被进一步细分成 3 部分：用户－角色管理(URM)，角色－角色管理(RRM)，角色-权限管理(RPM 或 PRM)。

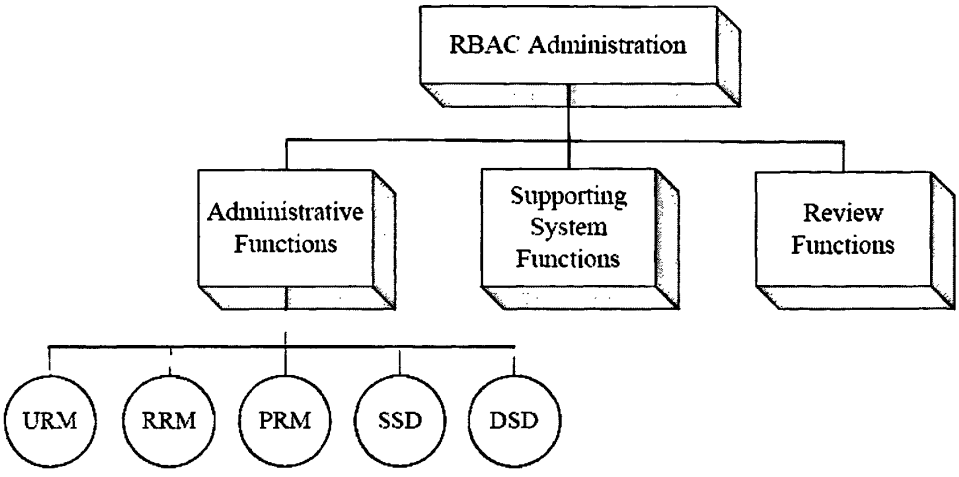


图 3.1 RBAC 管理模块

3.1.3.1 管理功能

管理功能处理创建，删除，和维护 RBAC 元素及关系的操作。图 3.2 给出了功能规范，表 3.1 有进一步解释。URM 模块管理用户帐号和用户—角色关系。RRM 模块管理角色，如添加删除系统角色及维护角色继承关系。RPM 模块管理角色-权限关系，如授予权限、撤销权限等。表 3.1 为核心 RBAC（C）和继承 RBAC（H）的功能描述。

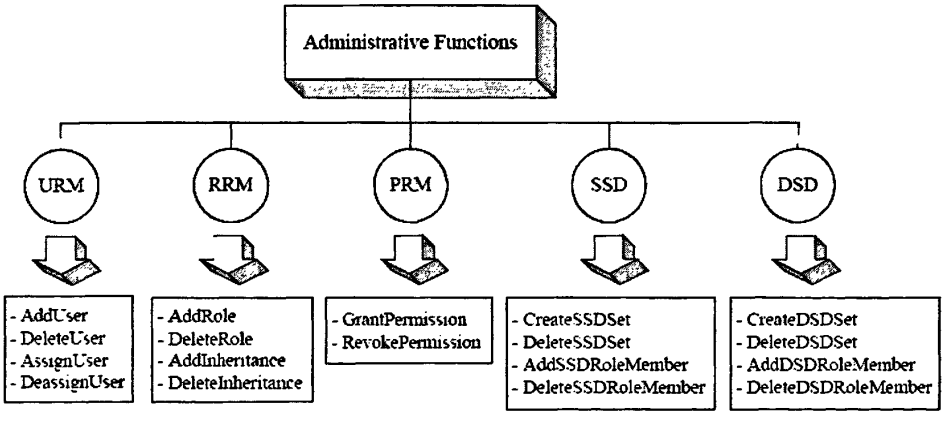


图 3.2 管理功能

表 3.1

URM	
AddUser（C，H）	创建 RBAC 新用户

DeleteUser（C，H）	删除一个已存在用户
AssignUser（C，H）	将用户赋予一角色
DeassignUser（C，H）	解除次用户和特定角色的复制
RRM	
AddRole（C，H）	创建 RBAC 新角色
DeleteRole（C，H）	删除一个已存在角色
AddInheritance（H）	在两个已有角色间添加直接继承关系
DeleteInheritance（H）	删除两角色间的直接继承关系
AddAscendant（H）	创建一新角色并添加为已有角色的直接前驱
AddDescendant（H）	创建一新角色并添加为已有角色的直接后继
PRM	
GrantPermission（C，H）	授予角色权限
RevokePermission（C，H）	收回角色权限

3.1.3.2 系统功能

系统功能主要处理会话管理，如创建和删除会话，表 3.2 进行详细描述。

表 3.2

Supporting System Functions	
CreateSession	创建用户会话并为用户提供默认的有效角色集
DeleteSession	删除用户已存在会话
AddActiveRole	为当前会话添加一角色作为有效角色
DropActiveRole	在当前会话中从有效角色集中删除一角色
CheckAccess	决策会话主体是否有权限对某对象资源执行请求的操作

3.1.3.3 浏览功能

浏览功能为 RBAC 的元素及其关系提供查询功能。图 3.3 描述了浏览功能规范，表 3.3 给出了详细解释。

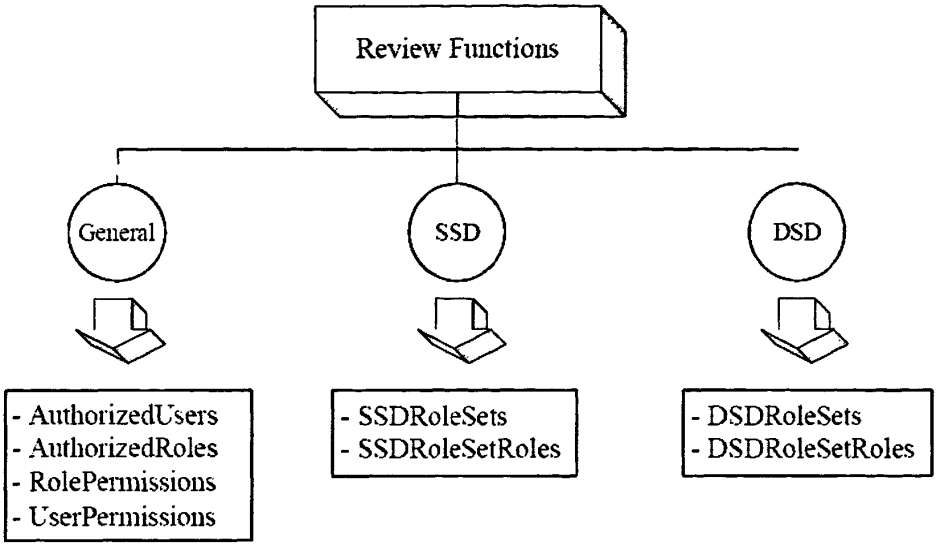


图 3.3 浏览功能

表 3.3 浏览功能规范说明

通用功能	
AuthorizedUsers	指定角色的直接赋值用户集，以及“指定角色的被继承角色”的用户成员集
AuthorizedRoles	指定用户的直接赋值的角色集以及“直接赋值角色的被继承角色集”。
RolePermissions	直接被授予角色的权限集以及被继承角色的权限集
UserPermissions	指定用户被授权的角色的权限集 (直接被赋值的角色集以及其被继承的角色集之和)
RoleOperationsOnObject	指定角色对某指定对象资源的（直接或继承的）操作集
UserOperationsOnObject	用户对某指定对象资源的操作集 (从用户赋值的角色集及被继承的角色集中获取)

3.1.4 XACML 对 RBAC 的支持

为了实现上述 RBAC 功能，需要利用 XACML 规范中定义的语言特性，定义 3 个主要描述模版，分别为角色、权限和角色赋值。由于目前 Sun XACML 开发组还未正式发布对 XACML2.0 支持的开发包，所以所有的描述都基于 1.2，开发使用 Sun XACML1.2 API 包，相关的 XML Schema 和可扩展的定义由 SunXACML 提供。一般来说，用户信息有专门的人力资源部门维护，许多的用户个人信息包括姓名、地址、电话号码等，都由专门的存储设备存储，因此不需要将这些信息

记录在 XACML 中。

XACML 策略语言包括 3 个模块：<PolicySet>,<Policy>和<Rule>，如图 3.4。所有 XACML 策略的根节点都是一个策略（<Policy>）或策略集（<PolicySet>）。一个策略是包含其他策略或策略集的容器，引用的策略可以是远程的。每个 XACML 策略文档都包含一个<Policy>或<PolicySet>的 XML 根节点。在<Policy>或<PolicySet>中，还需要定义<Target>为 PDP 提供可用的策略条件。Rule 是 XACML <Policy>的最基本组成，<Rule>又包含<Target>，<Target>由<Subjects>，<Resources>，<Actions>和<Environments>组成，用来监测规则<Rule>是否适用于特定的请求。规则生效则产生 Permitted 或 Denied 的响应值。<Rule>中的<Condition>布尔表达式用来限制<Rule>的适用范围。<Rule>需要包含在 Policy 中，不能单独存在。<Policy>通过一套 Rules 来表示访问控制策略，还包括<Target>，Obligations 和规则组合算法（Rules combining algorithm）。义务（Obligation）描述的是在获得许可后，PEP 必须执行的行为，如向资源的拥有者发送 Email。<Target>通过定义一组 Resource，Subject 和 Action，比较<PolicySet>,<Policy>和<Rule>中的 Target 使 PDP 能查找适用的<Policy>。

由于每个<Policy>或<PolicySet>都包含多个策略或规则，而每个策略和规则都会有不同的访问控制决策结果，XACML 需要某些方式来做决策。这就要通过组合算法集(a collection of Combining Algorithms)实现。每个算法代表将多个判断结果组合成单个决策的不同方式。这里有策略组合算法(Policy Combining Algorithm)(在策略集中使用)和规则组合算法(Rule Combining Algorithms)（在规则中使用）。例如否定覆盖算法，只要出现否定或未有许可的结果，则最终结果就是否定。这些组合算法用来建立复杂策略，并使 XACML 策略分散。有几个标准算法可以组合以适应应用需求。

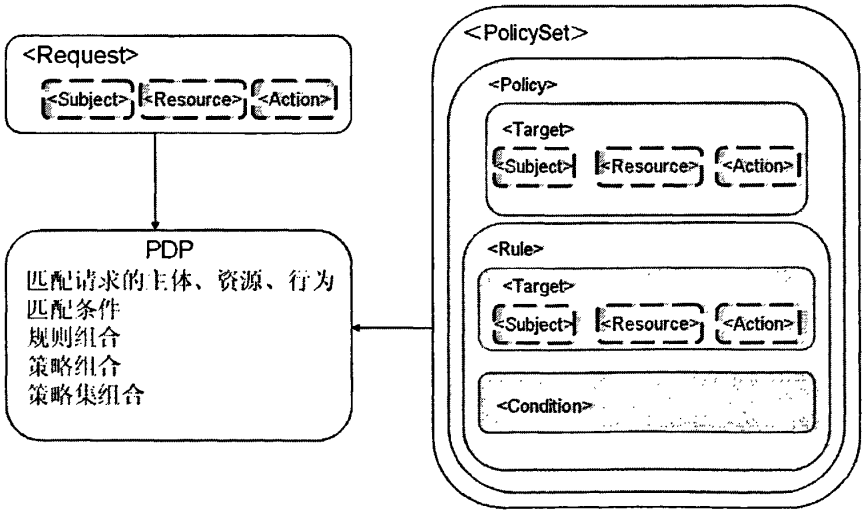


图 3.4 XACML 元素模块结构

3.1.4.1 XACML 的角色描述

角色都表示为XACML主体属性。XACML支持多主体请求，表示在请求中包含多个实体。通常一个用户可能有多个应用或代码产生较低等级的请求。在这些应用中有些计算设备或代码段，这些设备有唯一的标识如IP地址。XACML使用SubjectCategory属性描述每个实体区分主体的类型。主体在访问请求时，角色属性可以和任意主体类型相关联。以下所有

角色<PolicySet>或 RPS 表示，一个角色<PolicySet>使用权限集<PolicySet>关联指定角色的属性，权限集<PolicySet>包含指定角色相关的实际权限。角色<PolicySet>的<Target>元素限制<PolicySet>与拥有相关属性和值的主体的适用性。每个角色<PolicySet>只使用单个相应的权限集<PolicySet>进行角色描述。

可以为角色定义以下模版，将所有角色存储在 RPS:AllRole:Role 中。上级<PolicySet>用来存储所有角色，子<PolicySet>为上级<PolicySet>的子集，用来存储每个单独的角色。以下的例子中存放了角色 Manager, <PolicySetIdReference>通过引用 ID 为 PPS:Manager:Role 的<PolicySet>而获得相应权限。

```
<PolicySet
        xmlns="urn:oasis:names:tc:xacml:1.0:policy"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        PolicySetId="RPS:AllRole:Role"
        PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:per
        mit-overrides">
    <Target>
        <Subjects>
            <AnySubject/>
        </Subjects>
        <Resources>
            <AnyResource/>
        </Resources>
        <Actions>
            <AnyAction/>
        </Actions>
    </Target>
    <PolicySet

    PolicySetId="RPS:Manager:Role"
    PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-o
    verrides">
    <Description>
            角色 Manager 的描述
    </Description>
    <Target>
        <Subjects>
            <Subject>
                <SubjectMatch
                MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
                <AttributeValue
                DataType="http://www.w3.org/2001/XMLSchema#string">Manager</Attribu
                teValue>
                <SubjectAttributeDesignator
                DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"
                AttributeId="urn:oasis:names:tc:xacml:1.0:subject:Role"/>
                </SubjectMatch>
            </Subject>
        </Subjects>
        <Resources>
            <AnyResource/>
        </Resources>
        <Actions>
            <AnyAction/>
```

```
        </Actions>
    </Target>
    <!-- Use permissions associated with the manager role -->
    <PolicySetIdReference>PPS:Manager:Role</PolicySetIdReference>
    </PolicySet>
    <PolicySet PolicySetId="RPS:Employee:Role"
    PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-o
    verrides">
    <Description>
            角色 Employee 的描述
    </Description>
    <Target>
        <Subjects>
            <Subject>
                <SubjectMatch
                MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
                <AttributeValue
                DataType="http://www.w3.org/2001/XMLSchema#string">Employee</Attrib
                uteValue>
                <SubjectAttributeDesignator
                DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"
                AttributeId="urn:oasis:names:tc:xacml:1.0:subject:Role"/>
                </SubjectMatch>
            </Subject>
        </Subjects>
        <Resources>
            <AnyResource/>
        </Resources>
        <Actions>
            <AnyAction/>
        </Actions>
    </Target>
    <!-- Use permissions associated with the manager role -->
    <PolicySetIdReference>PPS:Employee:Role</PolicySetIdReference>
    </PolicySet>
</PolicySet>
```

3.1.4.2 XACML 的权限描述

权限集<PolicySet>或PPS表示，一个包含给定角色的实际权限的<PolicySet>。它包含<Policy>元素和主体有权访问的资源及能在其上实施的行为<Rules>，以及能访问相应的条件，如在某一天时间内。指定的权限集<PolicySet>可能包含指定角色的下级角色相关的权限<PolicySet>的引用，因此允许指定的权限集<PoilcySet>继承下级角色引用权限集<PolicySet>中的所有权限。权限集<PolicySet>中的<Target>元素必需不限制<PolicySet>所适用的主体。

XACML 通过定义<PolicySet>存储角色的权限集和权限-角色赋值(PRA)。权限和角色描述一样，使用<PolicySet>存储权限。以下的例子中，用一个<PolicySet>存储所有的权限，另一个<PolicySet>存储特定的角色权限集。赋予角色 Manager 的权限，通过指定 PolicySetId 的属性值 PPS:Manager:Role，标示可以被引用的权限集。以下的 XACML 描述表示，角色 Mangaer 通过引用<PolicySetIdReference>PPS:Manager:Role</PolicySetIdReference>而获得了角色 Manager 相应的权限。<Policy>用来描述单个权限，<Rule>用来描述权限细节，可以通过添加<Policy>元素增加新的权限。图 3.5 为通过<PolicySetIdReference>继承角色权限的示意图。

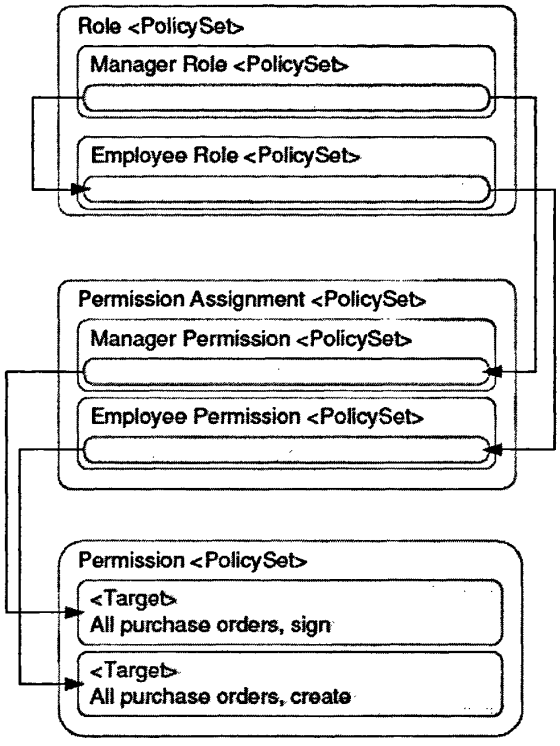


图 3.5 角色权限继承示意图

```
<PolicySet
        PolicySetId="PPS:AllRole:Role"
        PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:per
        mit-overrides">
    <Target>
        <Subjects>
            <AnySubject/>
        </Subjects>
        <Resources>
            <AnyResource/>
        </Resources>
        <Actions>
            <AnyAction/>
        </Actions>
    </Target>
    <!-----  角色 Manager 的权限集 ----->
    <PolicySet PolicySetId="PPS:Manager:Role"
        PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:per
        mit-overrides">
        <Description>
            角色 Manager 的权限集
        </Description>
        <Target>
            <Subjects>
                <AnySubject/>
            </Subjects>
```

```
        <Resources>
            <AnyResource/>
        </Resources>
        <Actions>
            <AnyAction/>
        </Actions>
</Target>
<Policy PolicyId="Permission:for:Manager:Role"
    RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:or
    dered-permit-overrides">
<Description>
    角色 Manager 的权限描述。
</Description>
<Target>
    <Subjects>
        <AnySubject/>
    </Subjects>
    <Resources>
        <AnyResource/>
    </Resources>
    <Actions>
        <AnyAction/>
    </Actions>
</Target>
<Rule RuleId="Permission:to:sign" Effect="Permit">
    <Target>
        <Subjects>
            <AnySubject/>
        </Subjects>
        <Resources>
            <Resource>
            <ResourceMatchMatchId="urn:oasis:names:tc:xacml:1.0:function:anyU
            RI-equal">
            <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#anyURI">Order</Att
            ributeValue>
            <ResourceAttributeDesignator
            DataType=http://www.w3.org/2001/XMLSchema#anyURI
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
            </ResourceMatch>
                </Resource>
            </Resources>
            <Actions>
                <Action>
                    <ActionMatc
                        MatchId="urn:oasis:names:tc:xacml:1.0:function:string-e
                        qual">
                        <AttributeValue
                        DataType="http://www.w3.org/2001/XMLSchema#string
                        ">Sign</AttributeValue>
                    <ActionAttributeDesignator
                    DataType="http://www.w3.org/2001/XMLSchema#string"Attri
                    buteId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
                  </ActionMatch>
                </Action>
              </Actions>
            </Target>
```

```
            </Rule>
        </Policy>
        <PolicySetIdReference>PPS:Employee:Role</PolicySetIdReference>
    </PolicySet>
    ..... .....
</PolicySet>
```

3.1.4.3 XACML 的用户-角色赋值描述

角色赋值包括两种情况，一种是在 AssignUser 操作时产生的纯赋值操作。另一种是在建立 Sessions 时产生。在进行 RBAC 的管理操作时，只考虑第一种情况。角色赋值<Policy>或<PolicySet>：一个<Policy>或<PolicySet>定义了哪些角色能被激活或赋予哪些主体。可能会在角色组合上、特定主体能被赋值或被激活的角色数目上，说明约束条件。此类策略应用在角色授权认证（Role Enablement Authority）上。角色赋值中可以选择<Policy>或<PolicySet>。在 XRBAC 系统中，选择<PolicySet>作为角色赋值的载体，<Subject>描述用户信息，< PolicySetIdReference> 描述所属的角色。

```
<PolicySet                                                    PolicySetId="UserRole:*"
  PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:ordered-pe
  rmit-overrides">
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <AnyResource/>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>
  <PolicySet                                                  PolicySetId="UserRole:User418"
PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:ordered-perm
it-overrides">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">User418</AttributeValue>
            <SubjectAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <AnyAction/>
```

```
      </Actions>
    </Target>
    <PolicySetIdReference>RolePermission:Role14</PolicySetIdReference>
    <PolicySetIdReference>RolePermission:Role39</PolicySetIdReference>
  </PolicySet>
.....
</PolicySet >
```

以上的策略语言描述表示用户 User418 拥有角色 RolePermission:Role14、RolePermission:Role39 定义的所有权限。

3.1.4.4 XACML 的职权分离和权限约束描述

一个用户可能必须同时拥有几个角色才能访问某权限或一个用户不能同时拥有某几个角色的权限，都可被认为权限约束。这些策略可以通过使用角色<PolicySet>表示，其<Target>元素需要主体有全部必要的角色属性。可以通过使用一个<Subject>元素包含多个<SubjectMatch>子元素实现。相关的权限集<PoliceSet>应该说明同时拥有所有角色主体相关的权限。

关于多角色策略的权限集<PolicySet>可能引用其它角色的权限集<PolicySet>,这样可能从其它角色继承权限。如果其它角色在其自己的权限集<PolicySet>中包含和多角色策略相关的权限集<PolicySet>的引用,指定多角色<PolicySet>的相关权限可能被其它角色继承。或者某些角色的权限不能同时被赋予给某个角色。

RBAC 中的另一种情况，比如创建购物单和签单的权限不同同时赋予同一个用户，也不能赋予同一个角色。同样一个角色也不能同时为创建购物单和签单的上级角色。另一种实现基于用户的职权分离方案在文献[38]中提到，可以通过动态维护用户的"黑名单"实现。这种形式也可用在其它职权分离中。比如，同一个用户不能同时拥有某些角色权限，则角色被添加到一个约束集中，如 UserSSD（role0，role1）表示角色 role0、role1 不能同时被赋予某个用户。

3.1.4.5 XACML 的规则描述

<Rule>是 XACML <Policy>的最基本组成，<Rule>又包含<Target>和<Condition>，<Target>由<Subjects>，<Resources>，<Actions>来监测规则<Rule>是否适用于特定的请求。规则生效则产生 Permitted 或 Denied 的响应值。<Rule>中的<Condition>布尔表达式用来限制<Rule>的适用范围。

为了使规则具有直观性，分别对 3 种规则进行说明：

（1）时间控制规则

时间控制可以包括时间点和时间段，比如某段日期之间或某个时刻后，可以对资源进行访问。以下例子描述了满足从 9 点到 17 点的请求，可获得许可。

```
<Condition>
  <Apply FunctionId=" urn:oasis:names:tc:xacml:1.0:function:and">
      <Apply FunctionId="
      urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal">
        <Apply FunctionId=" urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
          <EnvironmentAttributeDesignator
          AttributeId=" urn:oasis:names:tc:xacml:1.0:environment;current-time"
          DataType=" urn:oasis:names:tc:xacml:1.0:xml;time"/>
          </Apply>
        <AttributeValue
        DataType=" urn:oasis:names:tc:xacml:1.0:xml;time">9h</AttributeValue>
      </Apply>
      <Apply FunctionId="
      urn:oasis:names:tc:xacml:1.0:function;time-less-than-or-equal">
        <Apply FunctionId=" urn:oasis:names:tc:xacml:1.0:function;time-one-and-only">
          <EnvironmentAttributeDesignator
          AttributeId=" urn:oasis:names:tc:xacml:1.0:environment;current-time"
          DataType=" urn:oasis:names:tc:xacml:1.0:xml;time"/>
        </Apply>
        <AttributeValue
        DataType=" urn:oasis:names:tc:xacml:1.0:xml;time">17h</AttributeValue>
      </Apply>
  </Apply>
</Condition>
```

（2）URI 控制规则

针对不同用户对不同网页资源的控制，被授予该权限的用户，拥有访问该资源的权限。以下的规则描述表示获得该权限的用户可以对资源 http://server.example.com/rule.html 进行 Read操作，即能访问该页面。

```
<Rule RuleId="ReadRule" Effect="Permit">
    <Target>
        <Subjects>
            <AnySubject />
        </Subjects>
        <Resources>
            <Resource>
                <ResourceMatch
                        MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
                    <AttributeValue
                DataType="http://www.w3.org/2001/XMLSchema#anyURI">
                      http://server.example.com/rule.html
                    </AttributeValue>
                    <ResourceAttributeDesignator
                DataType="http://www.w3.org/2001/XMLSchema#anyURI"
                    AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" />
                </ResourceMatch>
            </Resource>
        </Resources>
        <Actions>
            <Action>
                <ActionMatch
     MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                    <AttributeValue
                    DataType="http://www.w3.org/2001/XMLSchema#string">
                        read
                    </AttributeValue>
```

```
                    <ActionAttributeDesignator
                    DataType="http://www.w3.org/2001/XMLSchema#string"
        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" />
                </ActionMatch>
            </Action>
        </Actions>
    </Target>
    <Condition/>
</Rule>
```

（3）数据库表控制规则

由于所有的资源在 XML 中都可以以字符表示，如果需要使用户对数据库中的 userinfo 表有 update 的权限，则规则表示如下：

```
<Rule RuleId="ReadRule" Effect="Permit">
    <Target>
        <Subjects>
            <AnySubject />
        </Subjects>
        <Resources>
            <Resource>
                <ResourceMatch
                        MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
                                <AttributeValue
                        DataType="http://www.w3.org/2001/XMLSchema#string">
                         userinfo
                    </AttributeValue>
                    <ResourceAttributeDesignator
                    DataType="http://www.w3.org/2001/XMLSchema#table">
                        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" />
                </ResourceMatch>
            </Resource>
        </Resources>
        <Actions>
            <Action>
                <ActionMatch
     MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                    <AttributeValue
                    DataType="http://www.w3.org/2001/XMLSchema#string">
                        update
                    </AttributeValue>
                    <ActionAttributeDesignator
                    DataType="http://www.w3.org/2001/XMLSchema#string"
        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" />
                </ActionMatch>
            </Action>
        </Actions>
    </Target>
    <Condition/>
</Rule>
```

3.2 系统设计

通过系统分析以后，可以进行基于 XACML 的 RBAC 管理系统(XRBAC)的设计和实现，主要管理 XACML 的策略信息点 PIP 逻辑模块，有效组织用户、角色、权限信息。出于通用性和可移植性的考虑，选择使用 Eclipse 的 JAVA RCP 应用程序实现 XRBAC 功能，大范围引用 Sun XACML1.2 提供的开发 API[30]，以生成各种<Policy>和<PolicySet>，进行策略评估,以及在此基础上的策略决策点 PDP 性能测试。XACML 作为一个强有效的媒介，良好的可扩展性、可读性以及访问控制标准定义，大大降低了不同组织系统的整合难度。

分析 XACML 应用流程后，可以看到在 XACML 的应用模块中，进行管理和维护的模块为 PAP 策略访问点，或者叫 PMP 策略管理点，通过 PAP 来维护用户、角色、权限、策略之间的关系。在基于 XACML 的 RBAC 管理系统中，就是要满足 ANSI 中 RBAC 模型描述的功能需求，通过实现其管理功能达到有效管理和维护 XACML 访问控制策略的目的。PAP 操作的对象就是策略信息点 PIP，所有的用户、角色、权限和策略信息都包含在逻辑 PIP 中。通过分析文献[39]描述的通过 XML 工具实现 RBAC 的模式，可以将 PIP 作为逻辑模块抽象出来，无论是存放在数据库中还是文件目录中，XRBAC 的操作直接操作对象都是以 XACML 语言描述的 XML 文件，因此不必要关心其具体的存放位置，将 PIP 与物理存储位置分离。

通过对核心 RBAC 管理模型的分析，获得相关的功能需求，将 PAP 划分为以下功能模块：

（1） URM：用户-角色管理。

实现用户信息的呈现和相关角色维护，由于 RBAC 中的用户不直接和策略关联，降低了用户策略信息的耦合，使得用户策略方便维护。直接和用户关联的是角色，因此只需要维护用户和角色之间的关系，包括 AddUser、DeleteUser、AssignUser、DeassignUser，具体功能描述参考表 3.1。改变用户-角色关系，只会影响到与用户相关的信息，而不会对角色、权限和策略信息产生影响，因此将用户-角色信息单独存放在 UserRole.xml 中，减小因数据变化的范围，提高数据存取效率。在后面的实验中可以看到，将所有信息存放在一个文件中是不可行的，特别是当用户节点数增大，改变任何信息都需要操作叫大的数据文件。

（2） RRM、RPM：角色-角色、角色权限管理。

实现角色信息的呈现和角色权限维护，角色-权限关系的改变会影响到用户被赋予的策略。RPM 具体功能包括：AddRole、DeleteRole、Grant Permission 、Revoke Permission，在继承 RBAC 中还可包括 RRM 功能：AddInheritance、

DeleteInheritance 等。改变角色-权限关系，只需要修改角色信息，不需要对其他信息作改动，因此将信息单独存放在 RolePermission.xml 中。

（3） PPM:权限-角色管理。

实现权限信息的呈现和维护，以及策略的编辑。在 XACML 中请求通过策略决策，因此 XRBAC 中策略为最小的访问控制单位，需要通过工具编辑详细的策略信息。PPM 具体功能包括：AddPerm、DeletePerm、AssignPerm、DeassignPerm 和 EditPolicy，分别表示添加权限、删除权限、给权限添加策略、将策略从权限中删除以及编辑策略。所有的权限策略信息存放在 PermissionSet.xml 中。

（4） SDM:职权分离管理。

实现职权分离的管理和维护。对于 XRBAC 的管理功能，目前只考虑静态职权分离，包括角色的静态约束和权限的静态约束。SDM 具体功能包括 AddSSD、DeleteSSD，分别表示添加静态约束集合、删除静态约束集合。静态约束集合存放在 SSDSet.xml 中，URM、RPM 和 PPM 模块中的赋值操作将受 SSD 的约束。

图 3.6 描述了 XRBAC 的主要功能模块和逻辑层次结构，系统管理功能都包含在 PAP 模块中，PAP 模块由 URM、RPM、RRM、PPM 和 SDM 组成，所有的信息以 XACML 描述，用户、角色、权限和策略信息都以<PolicySet>或<Policy>的形式表示，所以<PolicySet>和<Policy>的生成、删除、编辑、查找以及策略评估，必须通过 API 层操作，以实现用户-角色、角色-权限、策略集的管理。

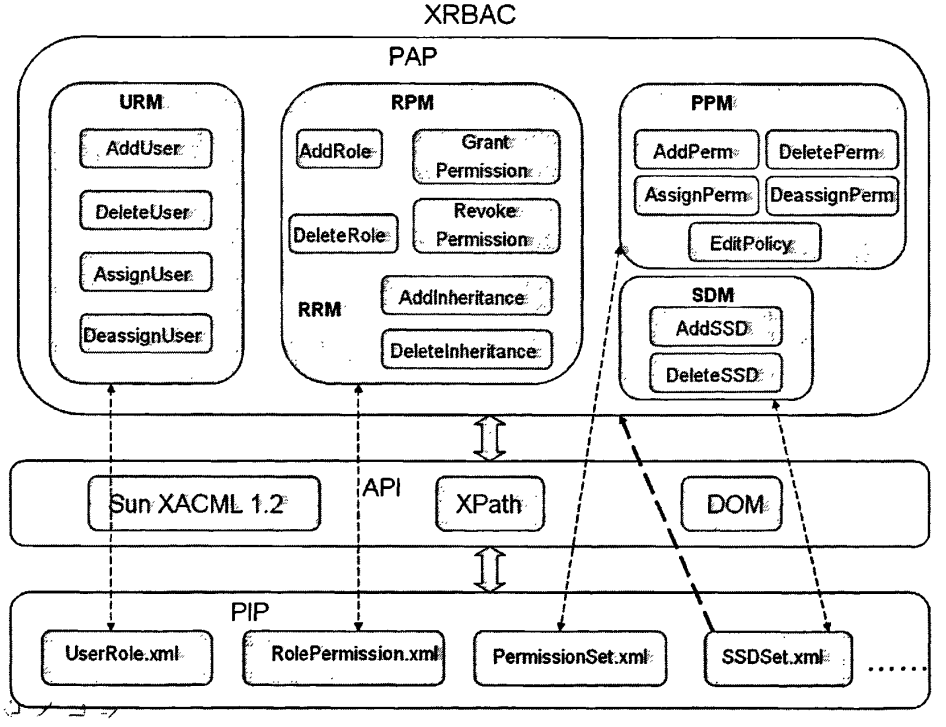


图 3.6 XRBAC 系统层次结构

PAP 操作的所有信息都存放在策略信息点 PIP 中，包括用户、角色、策略信

息，以及资源或者环境有关的属性值。文献[37]所描述的系统机构将用户、角存放在 APD 数据库中，而权限信息存放在 RMD 数据库中，其信息组织的逻辑结构和 PIP 相似。PIP 信息在进行策略评估时供 PDP 使用，因此 PIP 的组织结构直接影响到 PDP 的性能。PDP 需要根据请求中的 3 元素，在 PIP 中定位用户、角色、权限信息，并有 Sun XACML 1.2 提供的评估功能，进行策略评估并产生响应，整个过程如图 3.7。RBAC 访问控制模式的一个大的优点就是通过角色授权，降低用户信息和权限、策略信息间的耦合性。因此 PIP 的设计也需要继承这种优点，将用户-角色、角色-权限、策略集、静态约束信息分别存放在 UserRole.xml、RolePermission.xml、PermissionSet.xml、SSDSet.xml 中，使得信息的改变所影响的范围减小，URM、RPM、PPM 和 SDM 只改变相对应的信息模块，但所有相关的赋值操作都受 SSDSet 的静态约束。由于用户信息的改变不会影响到角色和策略信息，也使得用户规模易于扩充。

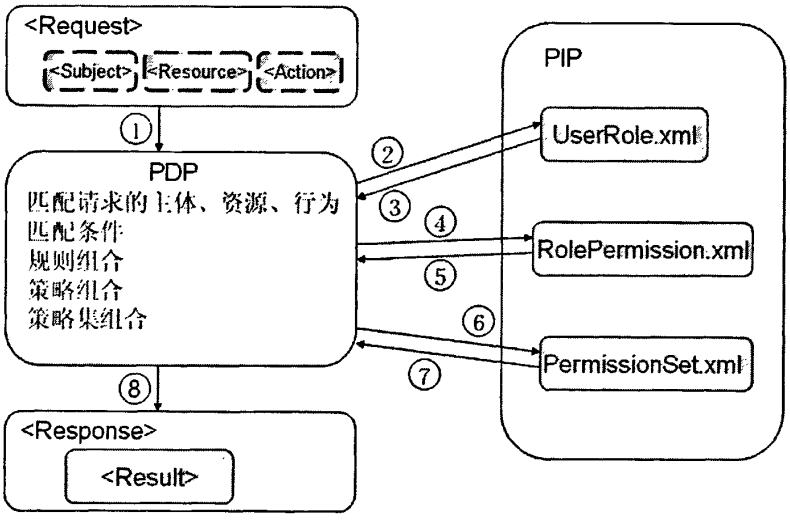


图 3.7　PDP 进行访问控制的过程

在 XRBAC 系统中，大部分的操作都涉及对数据的修改及数据修改后的呈现，因此采用 MVC 框架模式很适合这种需求[40]。MVC 不仅是一种设计模式也是一种框架，对于应用程序的也同样适用，它可清晰划分系统层次。MVC(Model-View-Controller)把一个应用的输入、处理、输出流程按照 Model、View、Controller 的方式进行分离，这样一个应用被分成三个层——模型层、视图层、控制层。

视图(View)代表用户交互界面，MVC 设计模式对视图的处理仅限于视图上数据的采集和处理，以及用户的请求，而不包括在视图上的业务流程的处理。

模型(Model)最主要是指数据模型，包括业务数据组成和持久化等。

控制(Controller)可以理解为从用户接收请求，将模型与视图匹配在一起，共同完成用户的请求。划分控制层的作用也很明显，它清楚地说明这就是一个分发

器，选择什么样的模型，选择什么样的视图，可以完成什么样的用户请求。控制层并不做任何的数据处理。

模型、视图与控制器的分离，使得一个模型可以具有多个显示视图。如果用户通过某个视图的控制器改变了模型的数据，所有其它依赖于这些数据的视图都应反映到这些变化。因此，无论何时发生了何种数据变化，控制器都会将变化通知所有的视图，导致显示的更新。这实际上是一种模型的变化-传播机制。模型、视图、控制器三者之间的关系和各自的主要功能，如图 3.8 所示。

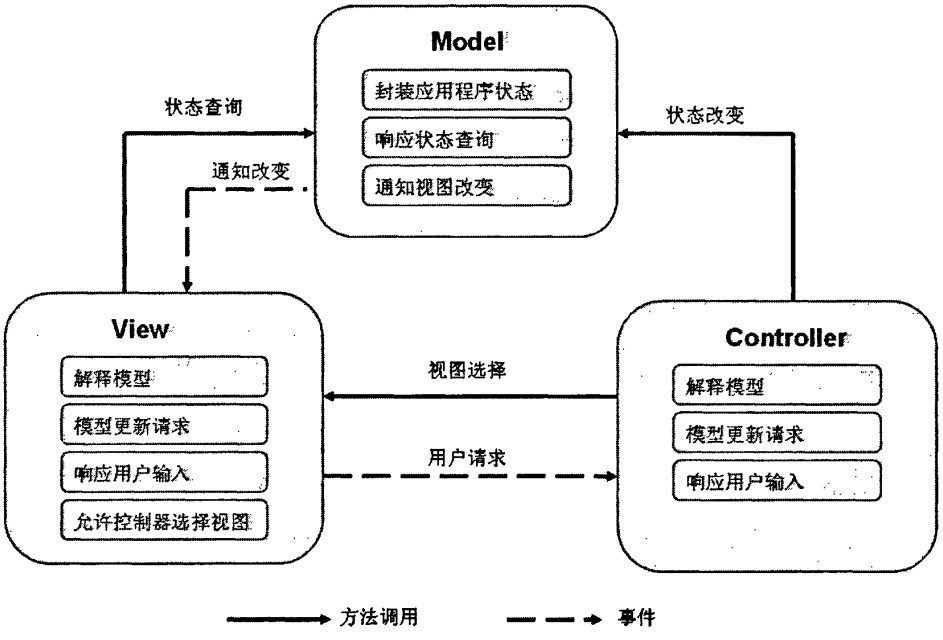


图 3.8 MVC 模块关系和功能

在 XRBAC 系统中，也采用 MVC 的框架模式，使数据模型和表现形式分离，给操作和表示带来更大的灵活性。虽然 View 模块和 Controll 模块很难做到彻底分离，但 MVC 框架模式使功能逻辑更清晰。

3.3 XRBAC 系统实现

图 3.9 描述了 XRBAC 中使用的主要类，能很好地反映了 MVC 框架模式的层次结构。所有的信息显示以 View 类表示，组成 View 模块；数据模块存放在 Entity 类中，组成 Model 模块；Factory 类联系 View 类和 Model 模块，根据用户请求改变数据 Entity 类状态，以及实现数据状态查询和向 View 类发出改变通知；

View 模块和 Controller 模块紧密联系，很难从结构上分离，部分的 Controller 由监听器组成，和 View 类组织在一起。每一个 View 类和图 3.6 中的 PAP 功能模块对应，实现其中的主要功能。

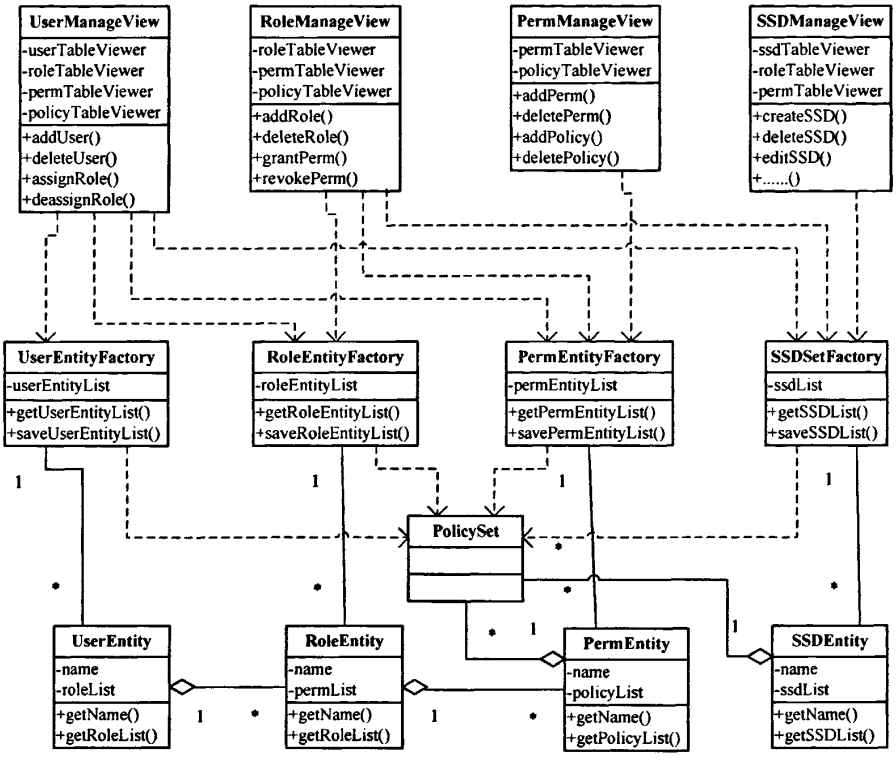


图 3.9 XRBAC 主要类静态图

从图 3.9 中可以看到，所有的 View 类都依赖 Factory 类，由 Factory 类提供所需要的数据，而 Factory 类都依赖 PolicySet，通过 PolicySet 实现用户、角色、权限和策略信息的存储和提取。

XRBAC 关注的是如何有效的管理和组织用户信息，实现基于角色的细粒度授权，以及如何使请求获得及时响应。XACML 所有关于策略评估算法的细节在这里不作考虑，设计只关注和用户、角色、权限管理相关的语言特性。主要相关的特性有<PolicySetIdReference>、<Target>、<Subject>、<Resource>和<Action>元素等，其他<Condition>、<Obligataion>等元素侧重于策略编辑相关的范围。图 3.10 表示了 PolicySet 中各元素的结构关系。

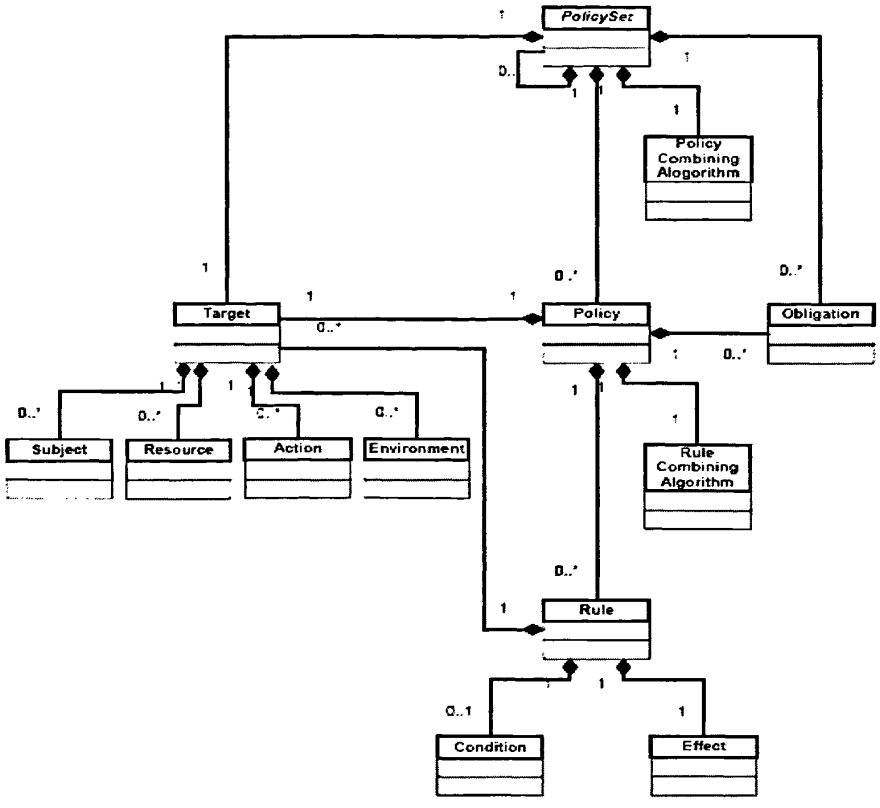


图 3.10 XACML 元素关系

管理功能中主要有用户-角色管理，角色-权限管理、权限集-权限管理和职权分离管理。用户角色管理主要管理用户-角色的赋值关系，用户的信息管理只作为辅助功能。用户-角色的赋值关系通过<PolicySetIdReference>的引用关系实现，用户的权限通过角色授予，不直接和权限或权限集联系。用户-角色关系信息存放在 UserRole.xml 文件中。角色-权限管理功能主要管理角色和权限集以及权限的赋值关系，通过<PolicySetIdReference>和<PolicyIdReference>实现角色权限的赋值。角色可以通过<PolicySetIdReference>的引用获得角色权限集中的所有权限，也可通过<PolicyIdReference>获得角色权限集中指定的权限。角色-权限对应关系存放在 RolePermission.xml 中。权限的所有描述信息都存放在 PermissionSet.xml 中，角色的所有权限放在<PolicySet>元素中，<Policy>描述每个权限中访问策略的具体信息，主要包括对资源(<Resources>)的合法操作行为(<Actions>)。角色的继承可被表示为角色权限的继承，通过<PolicySetIdReference>和<PolicyIdReference>实现角色权限的继承关系。由于在 RolePermission.xml 可以通过<PolicySetIdReference>指定被继承的角色权限，因此要避免角色权限集被重复引用。

3.3.1 管理功能

3.3.1.1 用户管理

在 URM 模块中，主要有四大功能，AddUser,DeleteUser,AssignUser,和 DeassignUser。通常，用户可能有大量相关存储信息，此系统的主要功能是针对 RBAC 的管理，因此只需要使用用户 ID 或用户名构造用户数据。在添加用户(AddUser)时只需要用户 ID 和用户名即可，而与用户相关的细节信息可以存放在其他信息存储位置中。

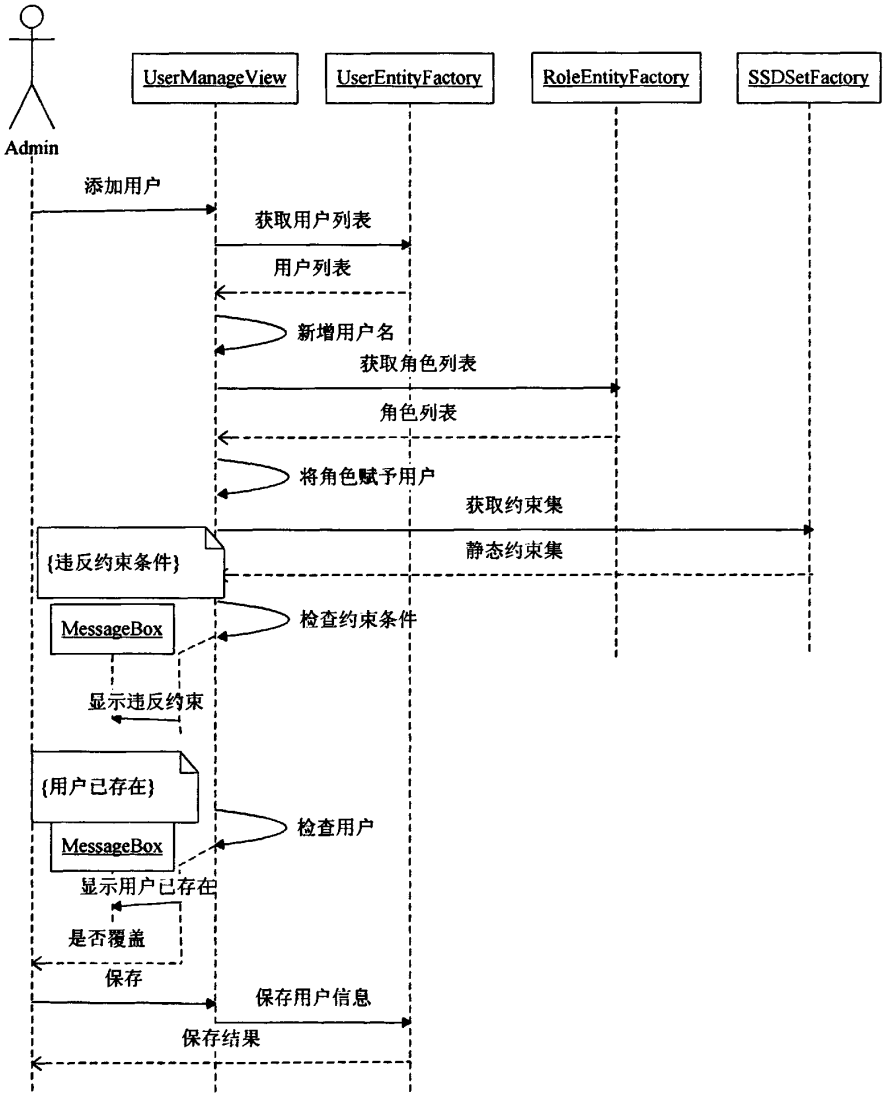


图 3.11 AddUser 时序图

图 3.11 为 AddUser 时序图。URM 中所有的功能由 UserManageView 实现，依赖 UserEntityFactoy 提供用户 UserEntity 信息，依赖 RoleEntityFactory 提供角色 RoleEntity 信息，依赖 SSDSetFactory 提供静态约束集，所有的用户和角色信息都依赖 PolicySet，由它产生 XACML 语言描述形式保存到文件中。

图 3.12 为用户信息界面，当点击用户名，用户所有的角色、权限、策略信息将显示在列表中。选择需要删除的用户，可以删除用户，实现 DeleteUser。

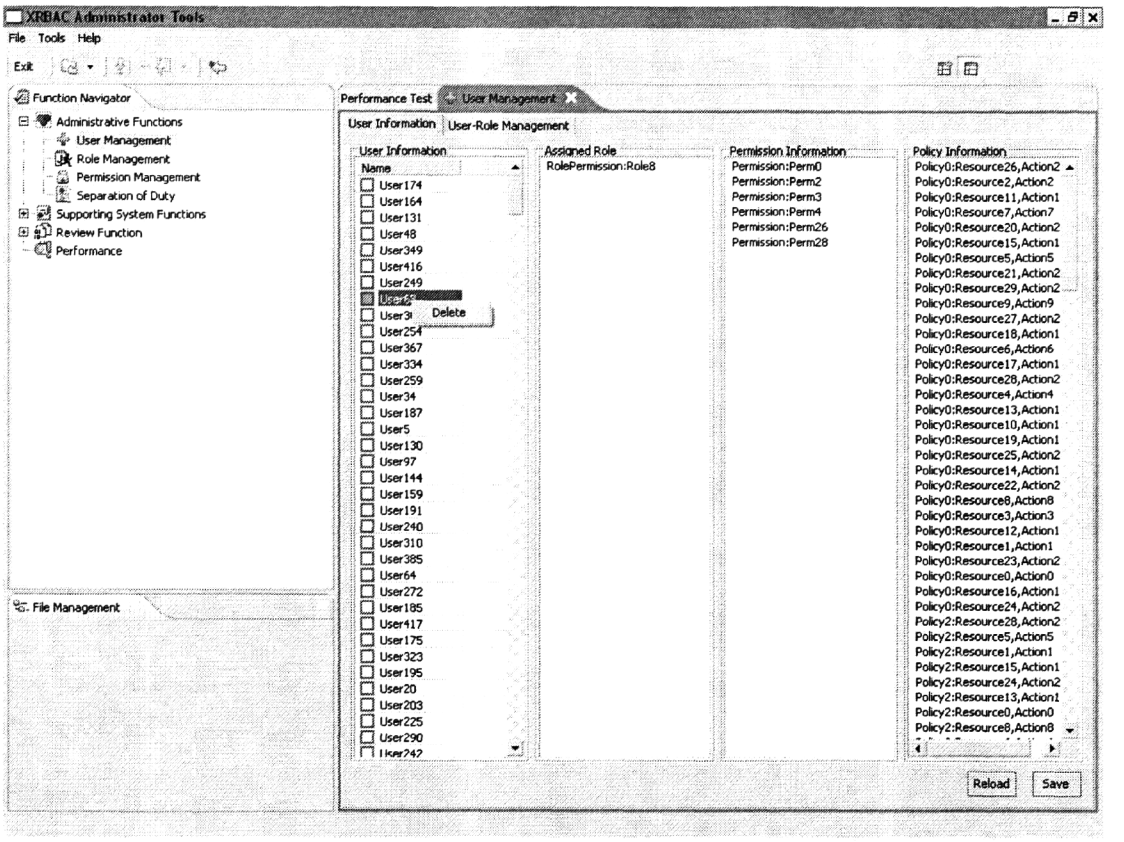


图 3.12 用户信息界面

图 3.13 为用户-角色编辑界面，输入不同的用户名，添加新用户。通过 RoleEntityFactory 获取角色列表，UserManageView 呈现所有角色信息，管理员可以通过选择角色或取消选择角色，实现 AssignUser 或 DeassignUser，将指定角色的权限赋予相应的用户，或将权限取消。在进行角色赋值时，需要检测静态约束集，看是否违反角色约束。被赋予用户的角色权限和策略也在表中相应地列出。

图 3.13 中表示的是将 Role8 的权限赋予用户名为 User673 的用户。右边列表中显示出用户拥有的所有权限和策略。所有的操作在 Save 后使操作结果保存。

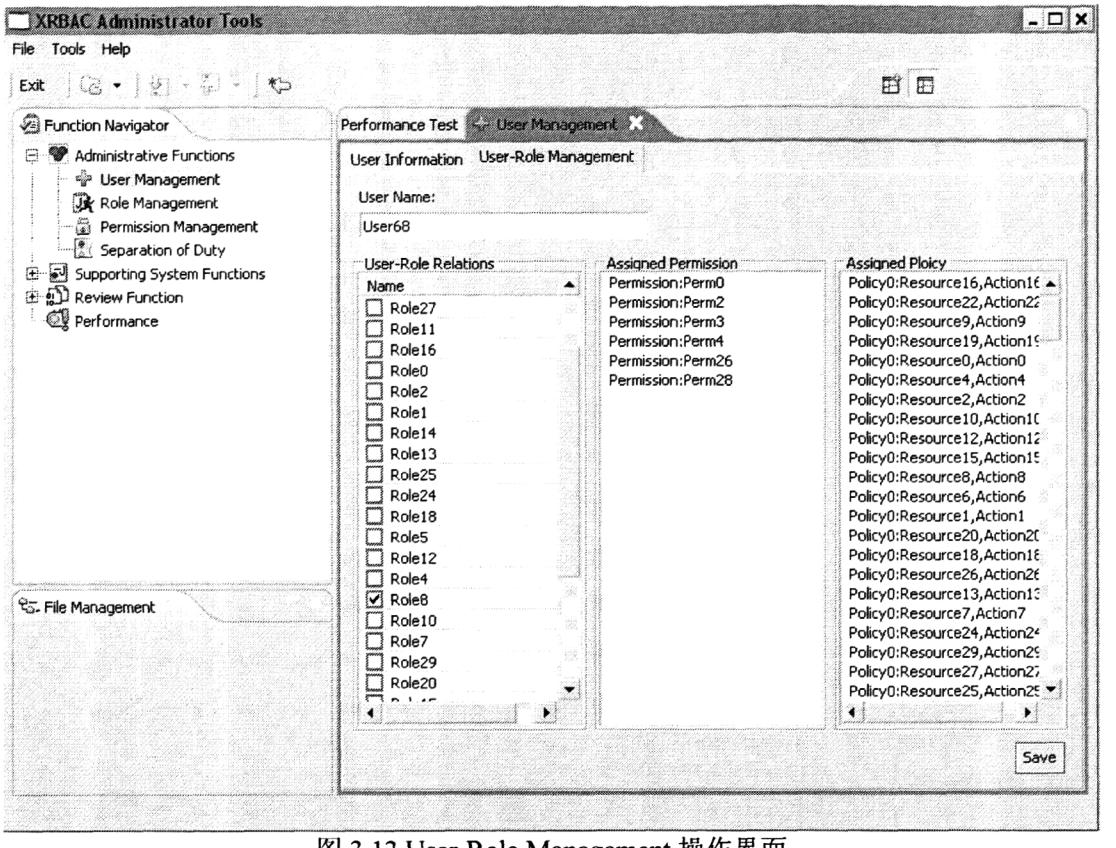


图 3.13 User-Role Management 操作界面

3.3.1.2 角色管理

RRM 模块中包括 AddRole，DeleteRole，AddInheritance 和 DeleteInheritance。AddAscendant 和 AddDescendant 并不在 XACML1.2 规范中有明确的支持，所以不在此系统中实现。角色的继承关系通过 AddInheritance 实现。在 RBAC 中有两种类型的角色继承，一般继承和限制继承。为了便于说明，此系统实现限制继承，即只继承单个角色的权限。AddRole 操作产生新角色 NewRole 时，会生成新的 <PolicySet> RolePermission:Role 的角色模板，和 <PolicySetIdReference> Permissin:Role 的权限引用模板，< PolicySetIdReference >描述引用的角色权限，将角色-权限信息存放在 RolePermission.xml 文件中。DeleteRole 则删除相应的 <PolicySet> 元素。角色的继承则在对应的 <PolicySet> 中添加或删除 <PolicySetIdReference>的角色引用描述。例如，NewRole 具有 Manager 的权限，由于 Manager 的<PolicySetIdReference>引用了 Employee 的权限，因此将自动继承 Employee 的权限，见图 3.5。

图 3.14 为 AddRole 时序图。RPM 中所有的功能由 RoleManageView 实现，依赖 RoleEntityFactoy 提供角色 RoleEntity 信息，依赖 PermEntityFactory 提供角色 PermEntity 信息，依赖 SSDSetFactory 提供静态约束集，所有的角色和权限信息都依赖 PolicySet，由它产生 XACML 语言描述形式保存到文件中。

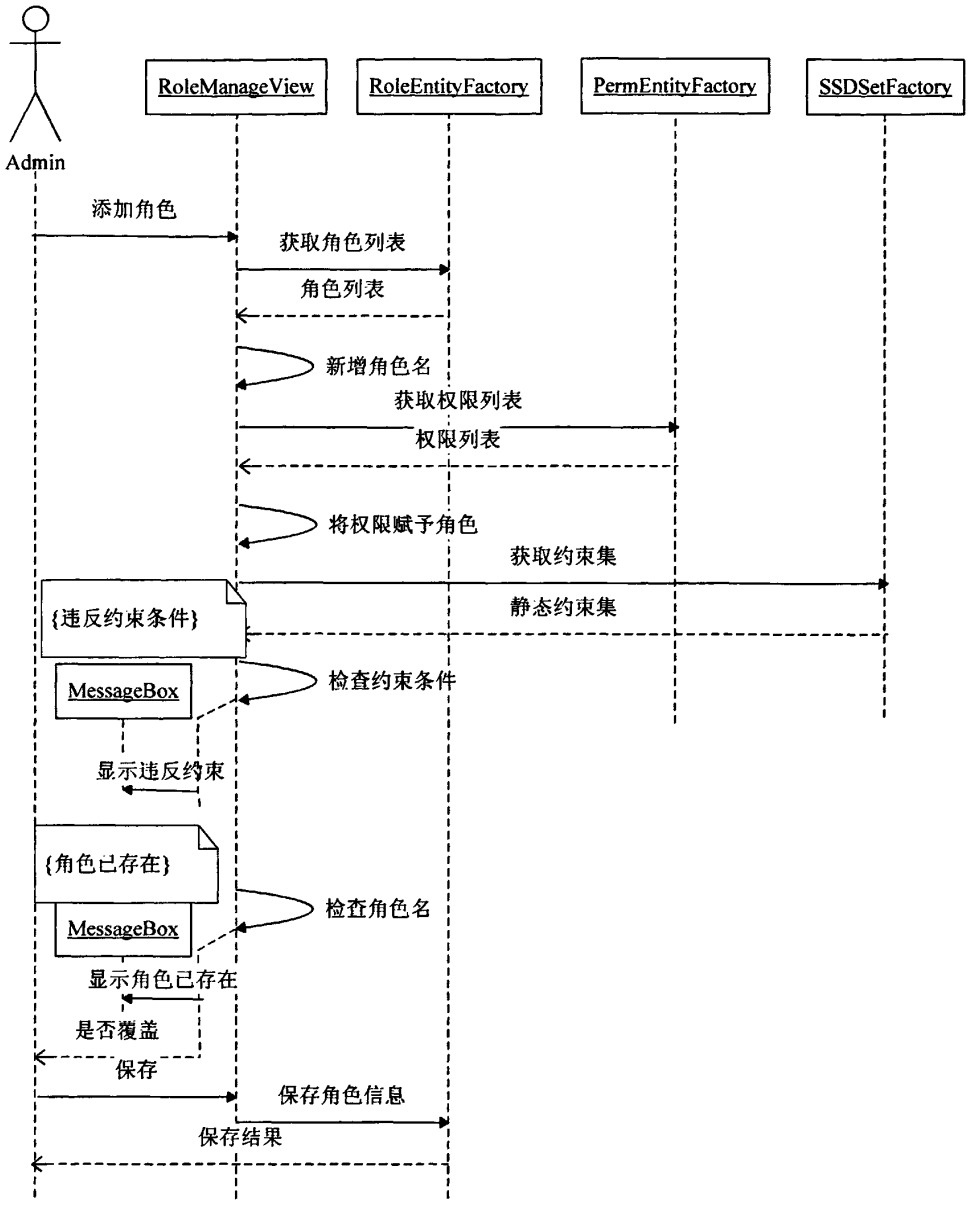


图 3.14 AddRole 时序图

图 3.15 为角色信息界面，当点击角色名，角色所有的权限、策略信息将显示在列表中。选择需要删除的角色，可以从角色列表中删除角色，实现 DeleteRole。

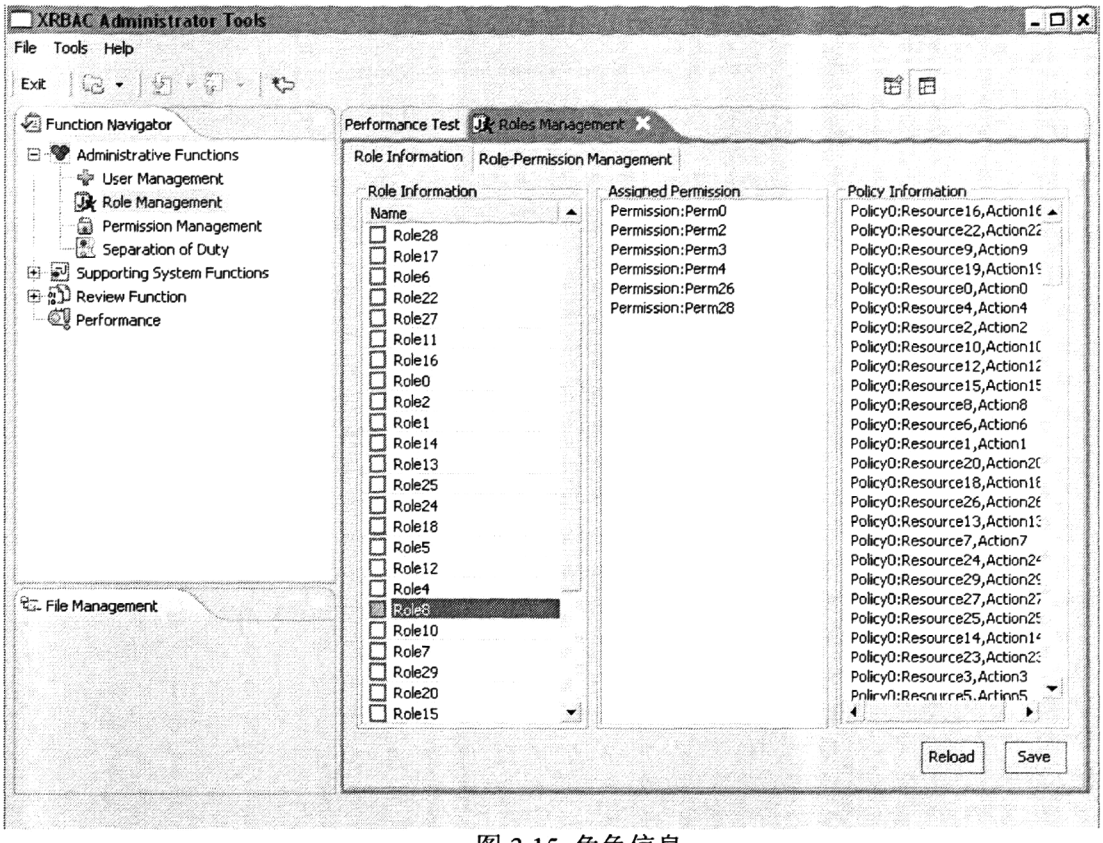


图 3.15 角色信息

图 3.16 为角色-权限编辑界面，输入不同的角色名，添加新角色。通过 PermEntityFactory 获取权限列表，Role-Permission Management 中呈现所有权限信息，管理员可以通过选择权限或取消选择权限，实现 GrantPerm 或 RevokePerm，将指定的权限赋予相应的角色，或将权限收回。在进行权限赋值时，需要检测静态约束集，看是否违反权限约束。被赋予权限的策略也在表中相应地列出。

图 3.16 中表示的是将 Perm28 和 Perm10 的权限赋予角色名为 Role8 的角色。右边列表中显示出角色被赋予的所有策略。所有的操作在 Save 后保存。

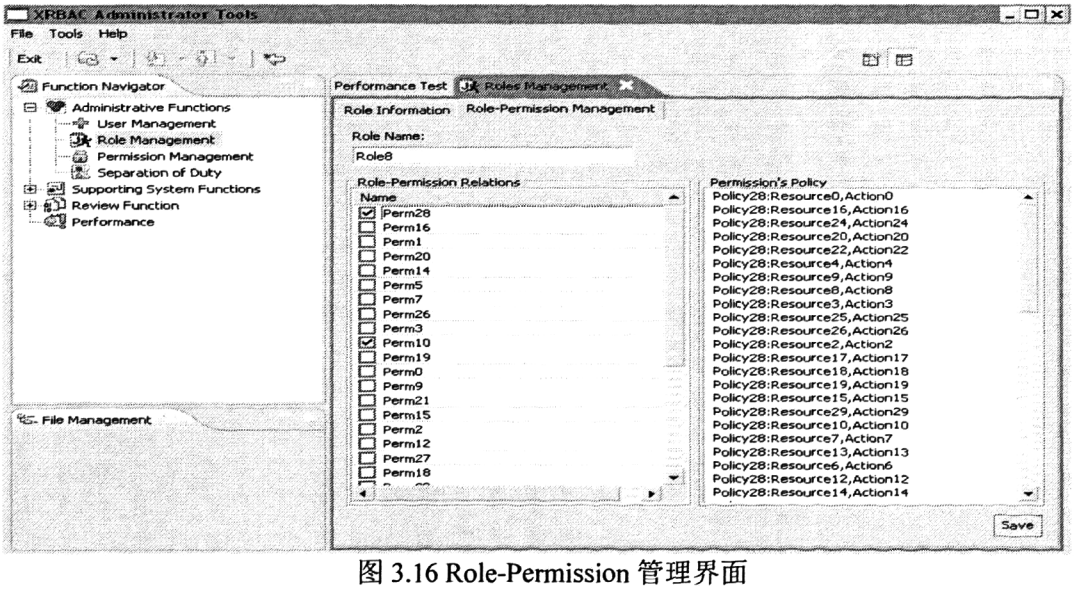


图 3.16 Role-Permission 管理界面

3.3.1.3 权限管理

图 3.17 为 AddPerm 添加权限时序图。PPM 中所有的功能由 PermManageView 实现，依赖 PermEntityFactoy 提供用户 UserEntity 信息，依赖 PermEntityFactory 提供角色 PermEntity 信息，所有的权限和策略信息都依赖 PolicySet，由它产生 XACML 语言描述形式保存到文件中。

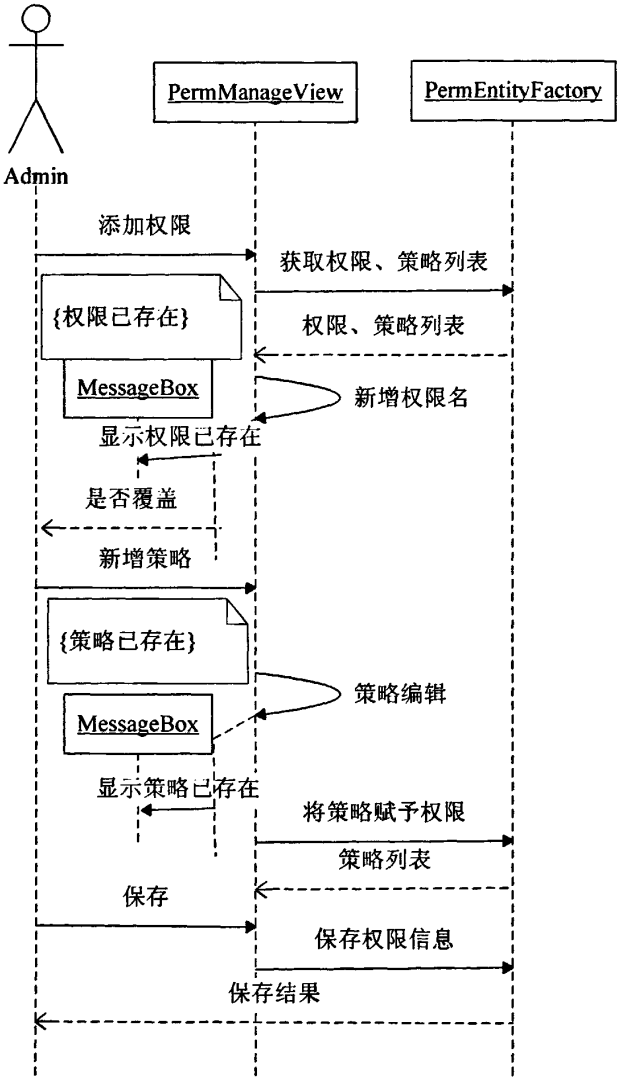


图 3.17 AddPerm 时序图

图 3.18 为权限管理界面，当点击权限名，所选权限的所有策略信息将显示在列表中。选择需要删除的权限，可以从权限列表中删除权限，选择权限的不同策略可以从策略列表中删除策略，也可以添加和编辑策略。所有操作在 Save 后将操作结果保存。

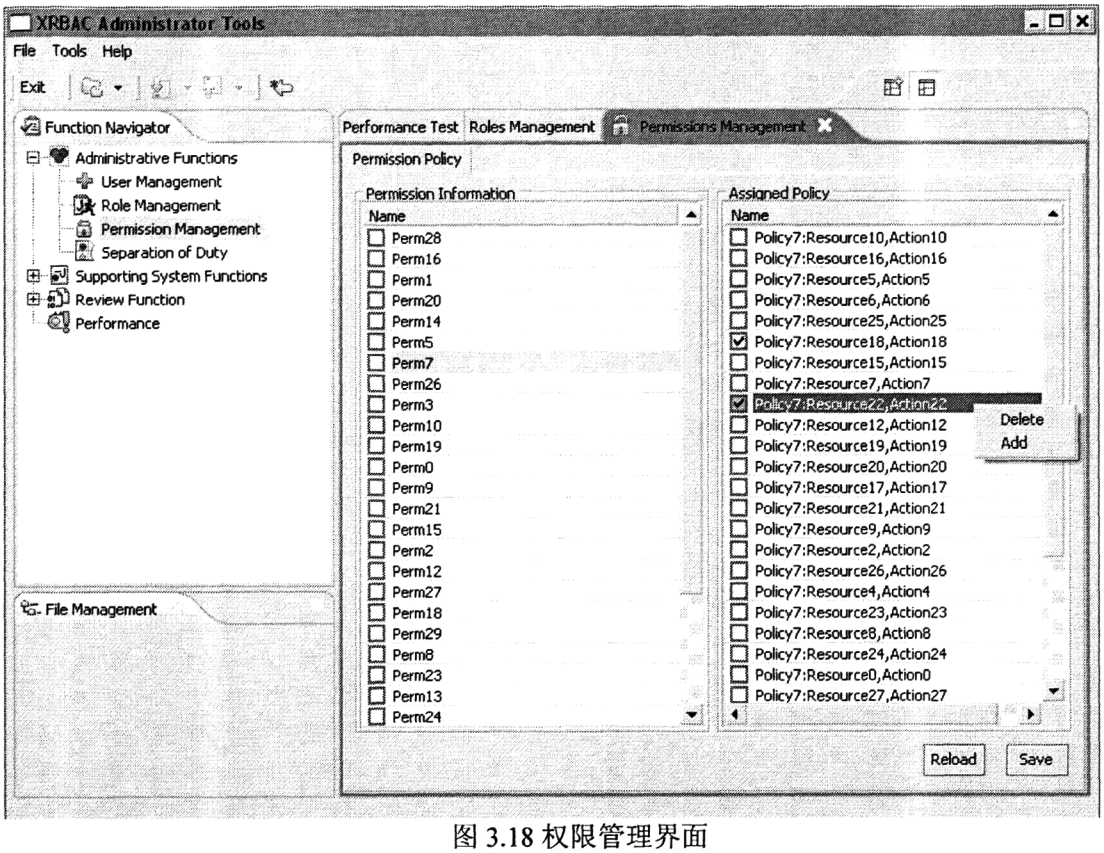


图 3.18 权限管理界面

3.3.1.4 职权分离

角色的策略冲突，可以通过<PolicySet>SD 职权分离管理。职权分离包括动态和静态职权分离，在 XRBAC 管理中，只关注静态职权分离。详细说明见 2.3.3 节。根据权限约束的粒度大小不同，也可以添加权限静态约束和角色静态约束。通过建立<PolicySet>SD:SD01:PolicySet 将所有冲突的权限存储在这个<PolicySet>中，同一个角色不能同时拥有 SD01 中的权限。<PolicySetIdReference>元素中定义了相关的冲突权限的引用。同样，也可以对角色进行 SSD 静态约束集维护，使互相约束的角色不能同时赋予某个用户。

图 3.19 为添加权限 SSD 时序图。SDM 中所有的功能由 SSDManageView 实现，依赖 RoleEntityFactoy 提供角色 RoleEntity 信息，依赖 PermEntityFactory 提供角色 PermEntity 信息，依赖 SSDSetFactory 提供静态约束集，所有的 SSD、角色和权限信息都依赖 PolicySet，由它产生 XACML 语言描述形式保存到文件中。

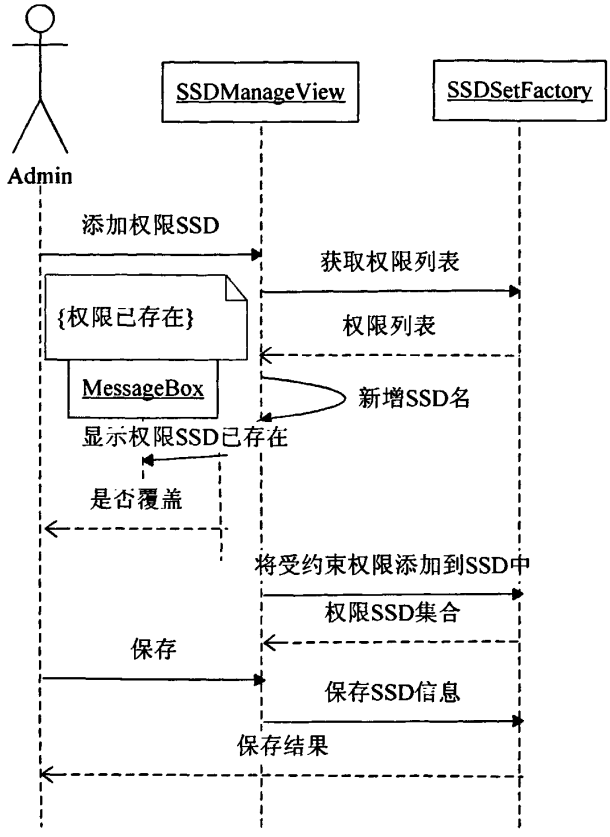


图 3.19AddPermSSD 时序图

图 3.20 为静态权限 SSD 管理界面。选择权限约束集，右边的权限列表中，互相约束的权限将被钩选。通过选择全县或取消选择权限，可以将受约束的权限添加到被选择的 SSD 中，或将其从被选择的 SSD 中删除。通过选择 Add 或 Delete 菜单，可以添加新的权限 SSD 或删除已存在的 SSD。图 3.19 表显示被选中的权限约束 SSD5 中，Perm28 和 Perm5 为受约束权限，不能同时赋予同一个角色。在进行角色编辑时，Role8 违反了 SSD5 的权限约束条件，将给出提示信息。对于角色的静态约束集在 Role SSD 中以相同的原理实现。

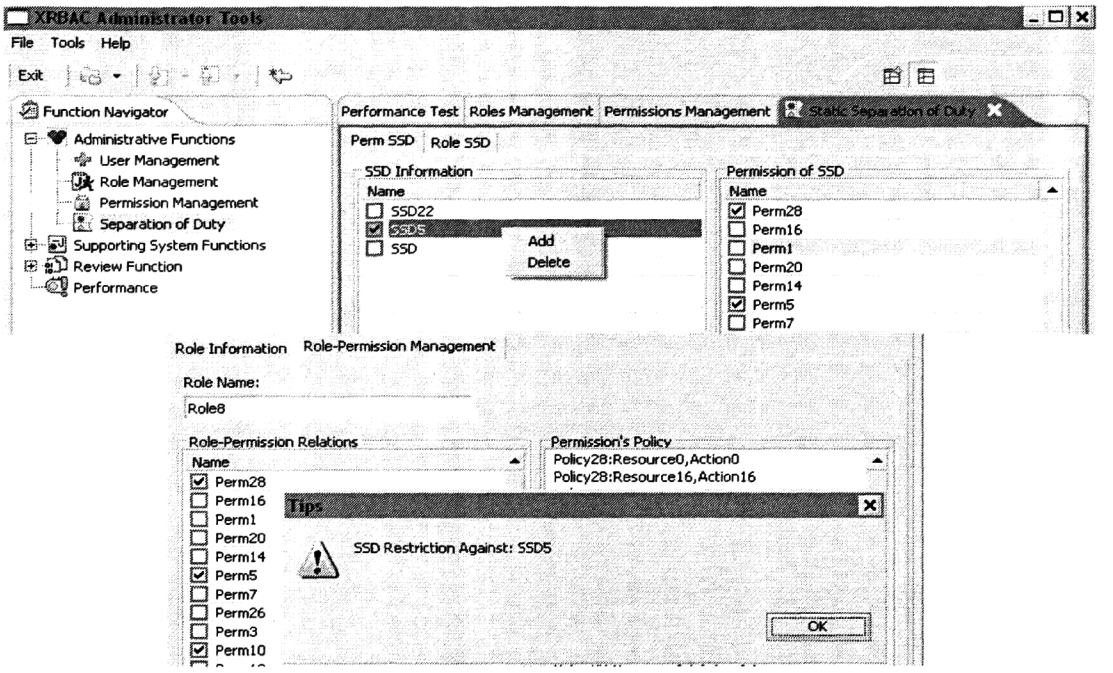


图 3.20 SSD 管理界面和违反约束条件提示

3.3.2 系统功能

系统功能维护 session 信息。此系统中定义了 CreateSession 和 DeleteSession。当用户创建一个新 Session 时，需要将角色、用户和 Session 联系起来。在 Deletesession 时只需要知道角色和用户 ID 就可以终止他们的关系。对 Session 的描述模板和用户-角色(URA)赋值相似。不同的是 Session 有<Condition>元素而 URA 没有。<Condition>保存时间范围使得用户可以在指定的条件下激活角色。

事实上，在进行用户和角色赋值的时候，都需要检测职权分离<PolicySet>，以保证被赋予的权限没有冲突。在这里可以显示不同职权分离策略中的所有相关角色。

3.3.3 浏览功能

通用浏览功能大部分已经包含在前面的个功能模块中，这里只是做为一种集中显示和有特别需求的浏览功能，目的只是为了方便管理。

3.3.4 性能分析功能

提供性能分析功能的目的是在大用户数目情况下，对系统性能进行评估，以便能及时维护用户信息文件，改善系统性能。图 3.21 显示的是系统响应时间分布图。性能分析还提供了平均响应时间及响应时间组成分析，响应时间随规模变化趋势图。

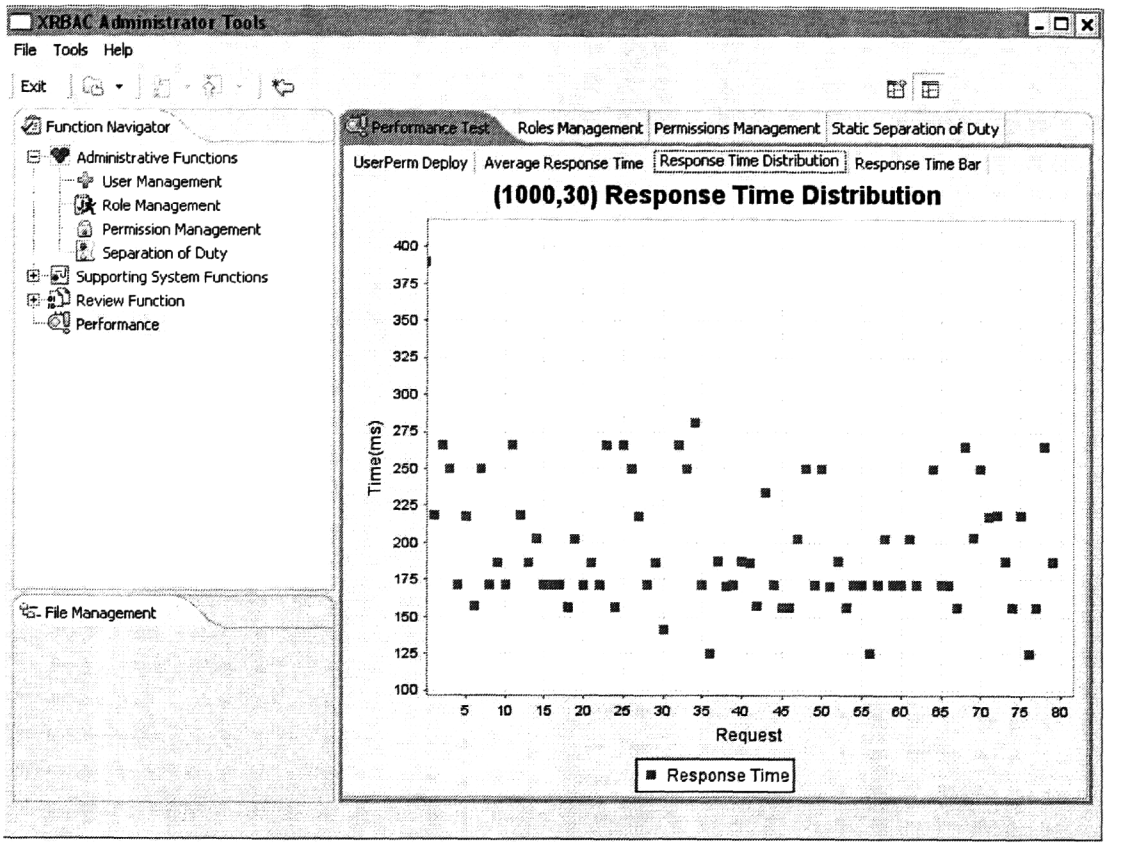


图 3.21 性能分析界面

3.4 本章小结

本章依据第 2 章 RBAC 理论模型，详细分析核心 RBAC 和继承 RBAC 各功能模块中的功能组成，为 XRBAC 的实现提供功能参考。分析 XACML 对 RBAC 提供的语言支持，通过 XACML 描述，实现 RBAC 各功能模块，为实现 XRBAC 的访问控制和管理系统提供语义支持。根据核心 RBAC 的功能需求，设计了基于 XACML 的 RBAC 管理系统框架结构，采用 MVC 框架模式，实现了 RBAC 核心和继承模型的主要功能。描述了 RBAC 的用户管理、角色管理、权限管理和 SSD 管理功能的具体实现，并介绍 XRBAC 的性能测试模块，验证了 XACML 对 RBAC 支持的可行性，为其他基于 XACML 的 RBAC 实现提供参考。通过 XRBAC 的 PIP 和 PAP 模块，能有效组织和管理用户、角色、策略信息，简化 XACML 的维护和编辑过程，大大提升工作效率，使得一般的系统安全管理员通过简单的培训，就能维护基于 XACML 的权限控制系统，降低了维护成本。

第四章 XRBAC 的 PDP 性能测试

XRBAC 实现的基于角色的访问控制系统所面临的一个重要问题就是请求的响应性能。目前，Sun 的 XACML1.2 还没有关于决策评估性能测试方面的报告，文献[30]只有关于依据 XACML1.2 规范提出的访问控制功能实现模型。以下内容将评测 XRBAC 访问控制模型下的 PDP 性能和扩展性。说明系统的性能指标和可能出现的瓶颈。这些方法和结果对于 XACML 的相关研究有参考价值。

4.1 性能测试

4.1.1 系统配置

硬件环境：1.86 Pentium IV 处理器，512M 内存。

操作系统::：Windows XP SP2。

软件包: JDK1.4.2，Sun XACML 1.2。

开发平台及辅助包： Eclipse 3.2，log4J[41]，JFreeChart 1.2[42]。

4.1.2 PDP 模块实验分析

每个请求<Request>，由<Subject>，<Resource>和<Action>3 部分组成。PDP 需要根据 3 元素，在策略信息点 PIP 中定位策略信息，在 UserRole.xml 中定位用户，根据用户的角色信息在 RolePermission.xml 中定位角色权限集，根据权限信息在 PermissionSet.xml 中定位策略，根据策略<Policy>中的定义，评估请求。因此，策略决策点 PDP 的性能只和策略的定位和评估或决策有关，不论是访问网页还是访问数据库表的请求，都通过 PDP 解析完成，因此 PDP 的性能主要受信息检索定位和策略评估性能影响。

PermissionSet.xml 的规模定义：30 个角色* 30 个策略，这样大小的规模已经能很好反应实际应用规模。

角色权限 RolePermission.xml 规模: 30 个角色，每个角色对应 5-10 个左右的角色权限集，这个值相对固定。

用户角色 UserRole.xml 的规模在 1000 个用户,每个用户对应随机个角色，一般不超过 10 个。这种规模的样本空间，能比较好地反映应用实际情况。

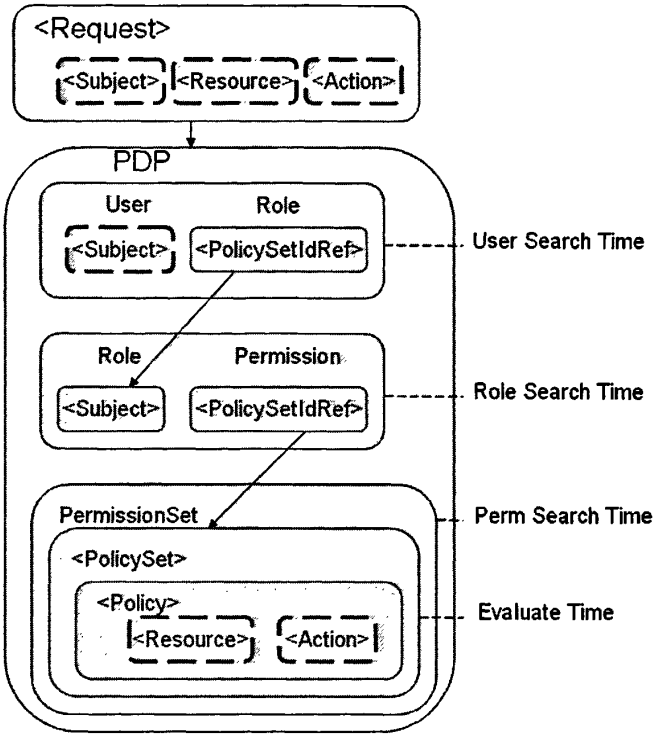


图 4.1 请求响应流程和响应时间构成

请求的响应时间组成如图 4.1。Request 中的<Subject>、<Resource>、<Action>元素分别和 UserRole.xml 中的<Subject>、PermissionSet.xml 中的<Resource><Action>相对应。

首先产生所有的策略和策略集，存放在策略配置文件中，30*30 规模的策略文件根据实际情况大小有所不同，实验大小在 1371KB，至少有 900 个节点。用户描述文件 1028KB，至少 1000 个节点。由于角色的数目固定数量级，节点数量少。所有的策略和用户在文件中的位置随机存放。在对请求 Request 处理之前，将所有的策略文件读入内存，分配好相关的处理资源，等待接收请求。

本系统在 Sun XACML1.2 的基础上做了部分扩展，引用其提供的功能，以生成各种<Policy>和<PolicySet>，以及进行策略评估。由于目前的研究，并没有对策略评估做性能测试，因此在对 XRBAC 的性能测试中，采用固定请求和评估模型，降低评估性能对测试的影响。各部分响应时间由 log4J 输出到日志文件中，其中响应时间组成与分布状态，借助 JFreeChart 提供的强大图表功能以直观图表形式表示。

请求描述包括三个要素:主体<Subject>，资源<Resource>和行为<Action>。先根据三要素产生请求，依据请求<Subject>在 UserRole.xml 中找到与用户匹配的角色引用（User Search Time）。如果没有匹配的角色，则对请求返回 NoAppliable。依据角色引用在 RolePermission.xml 中查找角色相关的角色权限或权限集引用

(Role Search Time)。如果没有相应的权限或权限集引用则返回 NotAppliable。依据权限或权限集引用，在 PermissionSet.xml 中查找相应的权限和权限集（Permission Search Time），使用 Sun XACML1.2 提供的功能对请求 Request 进行评估，返回评估结果 Response（Evaluation Time）。整个响应时间主要由 4 部分组成：Total Time = User Search Time + Role Search Time + Permission Search Time + Evaluate Time。数据的获取通过每次都随机生成策略集和用户集，评估 80 个获得许可的请求，所得的加权平均值。图 4.2 显示了响应时间的分布状况。

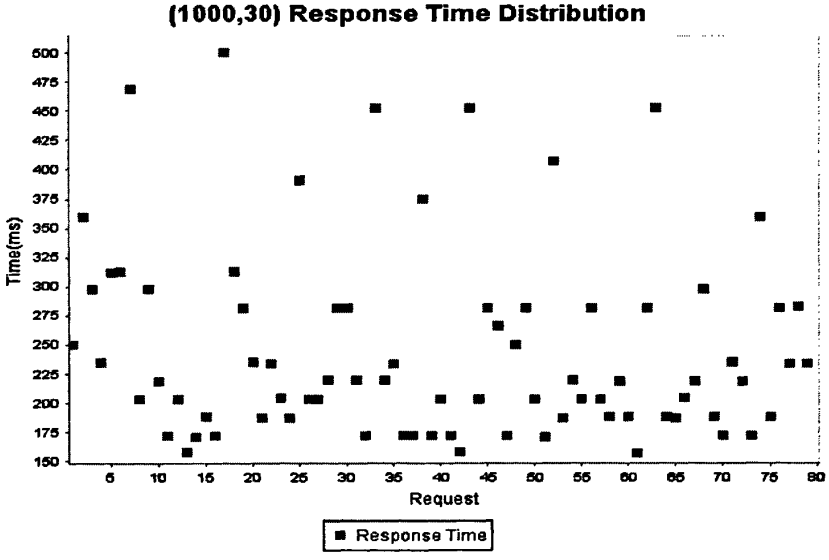


图 4.2　(1000,30)规模下的响应时间分布状况

图 4.3 显示的响应时间构成图表明，在（1000，30）的样本规模下，主要时间花费在用户和权限集的搜索上，而角色查找的时间可以忽略，策略评估的时间也相对固定，这和直观的事实相符。节点查询方法，成为影响性能的主要因素。（1000, 30）表示用户节点的规模为 1000，角色和权限的规模为 30*30。

在当前的规模下，响应时间的平均值为 245ms，对于当前的用户和策略规模有很好的性能表现。图 4.3 显示了平均响应时间在 245 ms 的各个时间花费的组成情况。平均响应时间直接受系统运行环境影响，在较好的硬件环境下，特别是内存充分状态下，性能表现更好。当用户和策略集的规模扩大，对性能会有怎样的影响，因此必须做可扩展性测试。

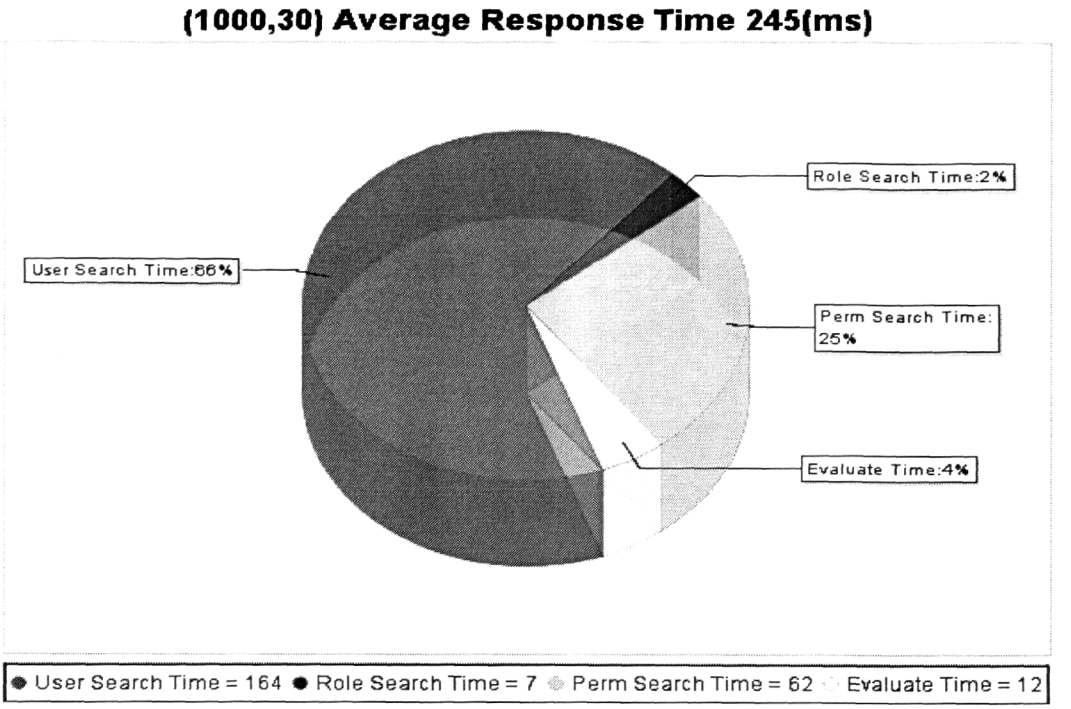
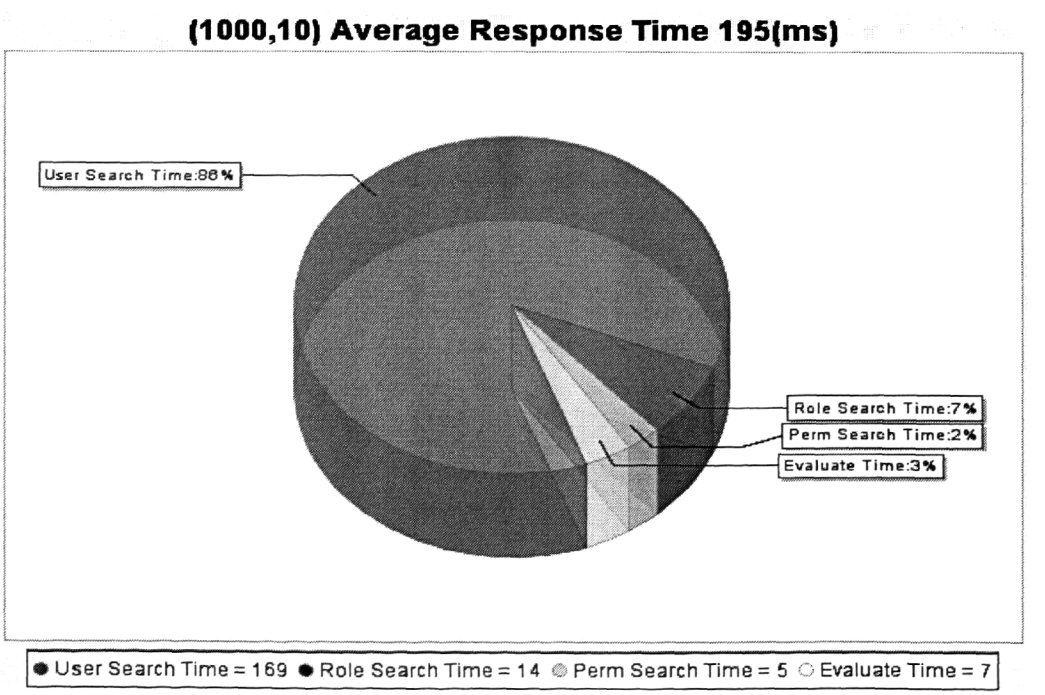


图 4.3 (1000，30)规模的平均响应时间

4.1.3 可扩展性测试

可扩展性测试，通过改变用户规模和权限集规模实现。首先在用户规模 1000 不变的前提下改变策略集的规模，测试点有 20*20，30*30，40*40，50*50，太大的策略集没有实际意义，在实际应用中不会用到，而且会导致 JDOM，XPath 无法处理。但这并不会成为此系统的瓶颈，通过在后面的实验数据分析，可以看到其良好的可扩展性。



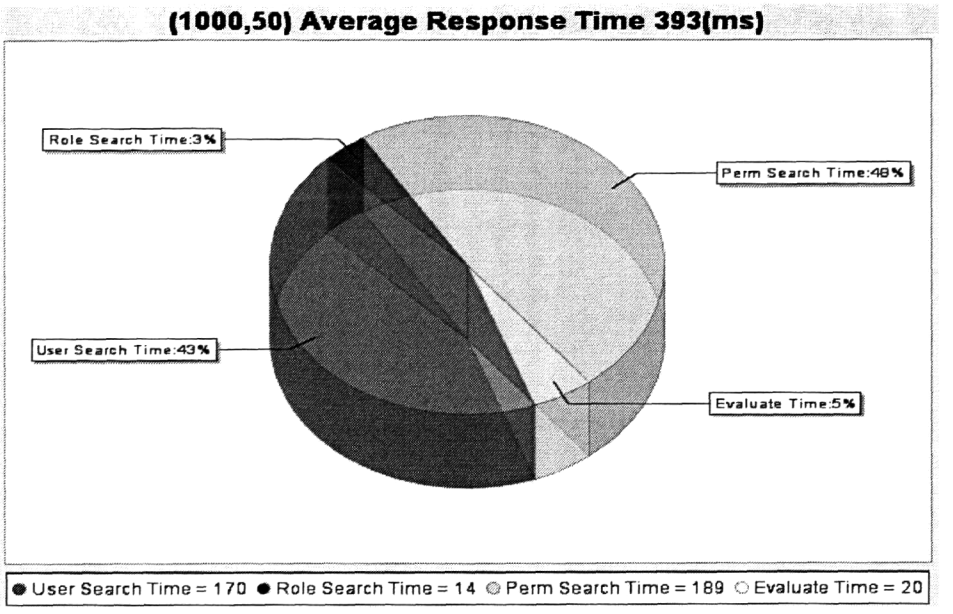


图 4.4 10*10 和 50*50 规模下的相应时间比较

图 4.4 的数据表明，随着权限集规模的扩大，权限集的节点查找时间有明显增加，增长的幅度决定于权限集节点的数量级。用户和角色的节点查找时间几乎没有变化，因为规模没有改变。而策略评估时间有小幅增加，因为策略集增大，对策略集的评估时间也有所增大。XPath 对 1000 个节点范围的平均查找时间为 170ms，在用户规模 1000 不变的条件下，时间的增长由策略集节点规模决定，这些都在可以预计和接受的范围，对性能的影响有限。图 4.5 反映了响应时间随权限集规模增长的幅度。

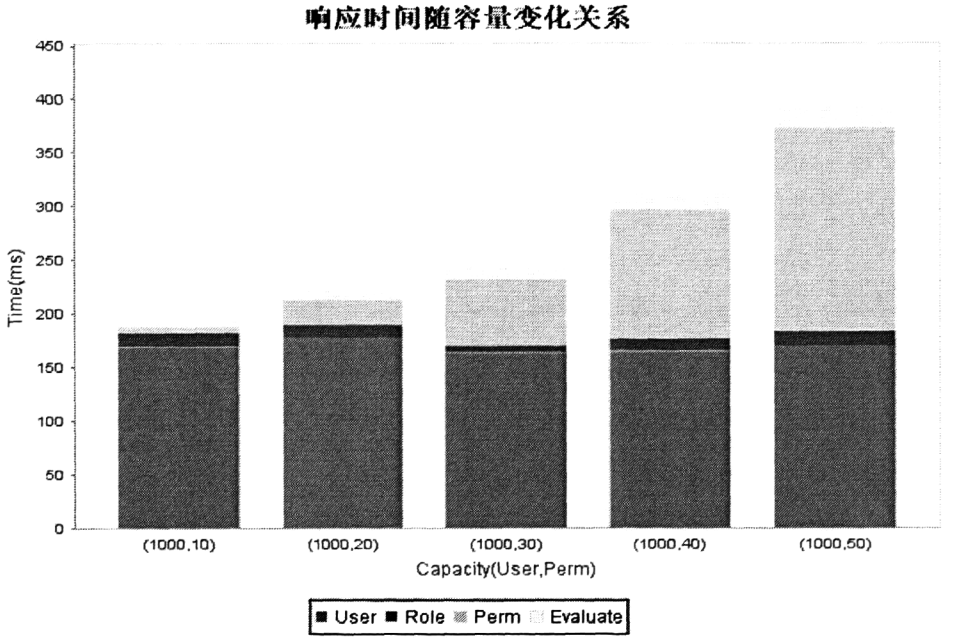
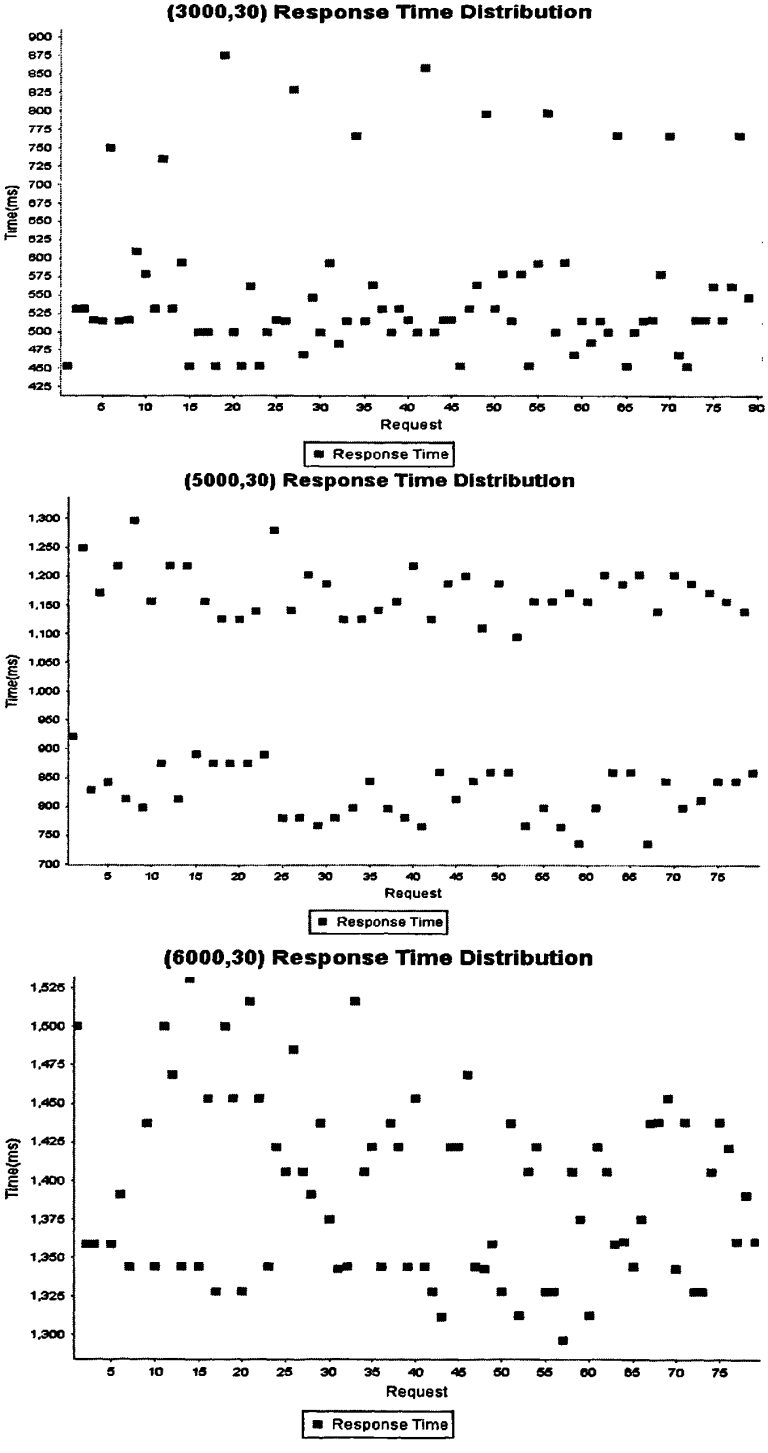


图 4.5 权限集规模改变对响应时间的影响

保持策略规模在 30*30 不变，改变用户规模，测试点有 500,1000,2000,3000, 5000，6000，7000。通过分析各个规模下响应时间的分布状况，很清楚可以看到，

5000 个用户节点以内，相应时间相对集中且集中分布在平均响应时间附近。7000 点的分布分散，几乎在一定范围内均匀分布，平均响应时间会明显增大。由此看出，本系统对 5000 个节点内的查找有较好的性能表现，增加到 7000 点后，性能明显降低，有时会因为内存分配不足产生溢出异常。见图 4.6。



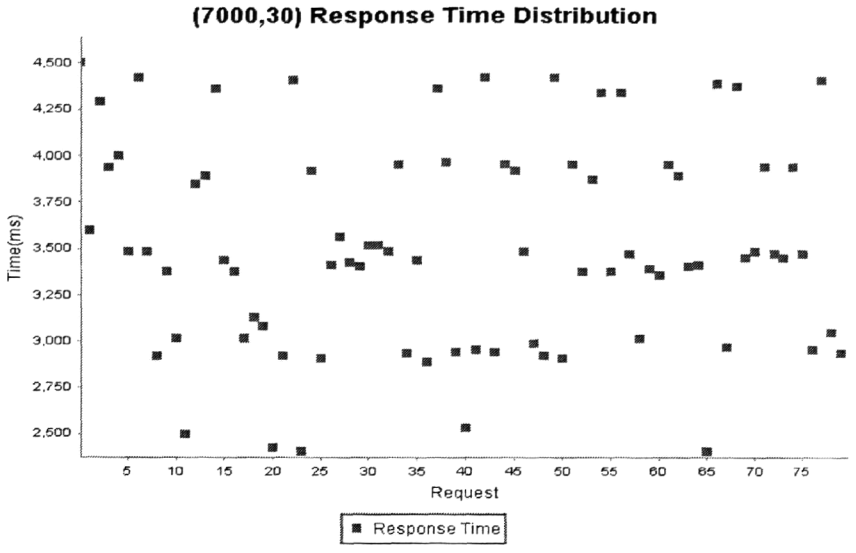


图 4.6 各测试点响应时间分布状况

图 4.7 反映了在规模为 30*30 不变的情况下，响应时间随用户节点增长的幅度。6000 点以内的增长幅度比较平缓，到达 7000 点时产生了性能上的瓶颈，所有时段的处理时间都显著增加。

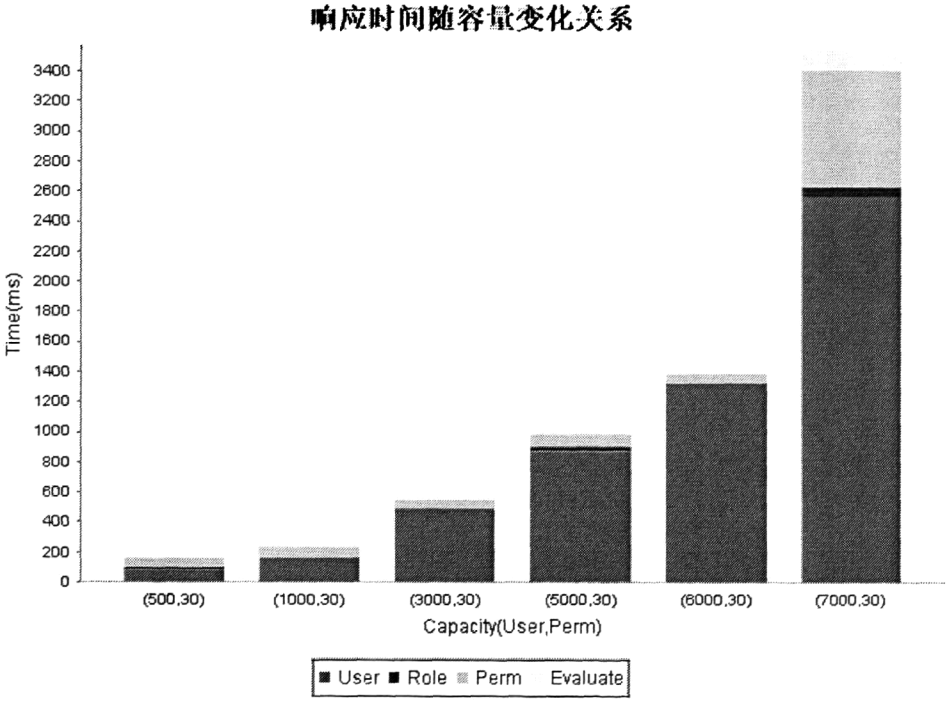


图 4.7 用户规模改变对响应时间的影响

4.2 系统性能分析

从各个查询阶段的耗时可以看出影响性能的主要因素是节点查找，耗费的时间主要集中在 XPath 的节点查找上，而策略的评估时间稳定，原因是请求评估的复杂度没有变化，策略集的节点也可控制，唯一不可确定的因素是用户节点数。

在当前的硬件环境下，本系统在（5000，30）的规模以内，有很好的性能表现，超过 7000 的用户信息节点后，性能出现瓶颈，XPath 会无法处理。可以通过将用户分组的方式，将大量用户分散在不同组的文体中，比如 50000 个用户分布在 10 个组中，每个组的节点控制在 5000 以内，系统性能将有极大改善。

4.3 本章小结

本章说明了实验方法和实验数据的获取方式，通过分析响应时间的组成份额、响应时间分布状态以及响应时间随用户权限规模变化趋势，通过实验分析 XRBAC 的请求响应性能。此系统对于 5000 以内的用户节点有很好的响应性能，对于大量用户节点的需求，可以通过用户分组的方式快速实现扩展，具有很好的可扩展性。

第五章 总结与展望

本章对全文的工作进行了总结，介绍了本论文的研究意义和价值，并对未来的工作进行展望。

5.1 工作总结

5.1.1 本文工作

本文在了解目前各种访问控制策略的基础上，分析各种访问控制策略的优缺点，重点说明 RBAC 的优点，同时对比不同策略的描述方式，分析 XACML 在分布式环境中的特殊优势。从 ANSI RBAC 理论模型和 ARBAC 97/99 功能模型入手，获得实现 RBAC 管理功能的功能需求。详细说明 XACML 支持 RBAC 的语言特性，为实现 XRBAC 提供理论支持和准备。通过分析 RBAC(Role Based Access Control)理论模型，依据 XACML 实现 RBAC 的描述，结合 SUN 提供的 Sun XACML1.2 实现，设计实现基于 XACML 的 RBAC 各种管理功能，系统功能等，从实现的角度成功构造了 XRBAC 管理系统，并提供性能测功能，进行数据分析和性能评估。在不同的用户规模和权限规模下，测试系统性能和可扩展性，验证理论模型的可行性。最后通过大量的实验数据，详细分析 XRBAC 在各种角色和权限规模下的性能表现和瓶颈，并提出可扩展方案，为其他相关研究提供可供参考的信息。

本文的主要内容是提出并实现了基于 XACML 的 RBAC 管理系统（简称 XRBAC），对其中的关键模块的设计和实现做了详细讨论，特别详细介绍了 XRBAC 管理功能的设计和实现、PIP 的设计和 PDP 的性能测试。通过 XRBAC 能更有效地管理和维护用户、角色和策略之间的关系，为提供可移植的、一致的访问控制策略，实现网络资源的跨域、细粒度授权，提供了一种可行的解决方案。

5.1.2 研究意义及价值

在 ANSI RBAC 规范的基础上，分析 ARBAC97/99 核心和继承 RBAC 的功能模型，研究 XACML 所提供的相关语言特性支持，以一种可行的方式，设计并成功实现基于 XACML 的 RBAC 管理系统，验证基于 XRBAC 模型的可行性。通过 XRBAC 可以有效地管理和组织用户、角色、权限信息，大大降低使用和维护 XACML 的难度，使得不同自治区域的权限控制，可以分别定义和管理，增强系

统的交互性和协作性。

最后通过大量实验测试数据，分析系统性能和瓶颈，为基于XACML的RBAC研究与应用提供参考。

XRBAC 访问控制系统可应用于目前电力市场集团辅助报价系统中，逐步替换基于 XML 描述的 RBAC 访问控制系统，提高系统的可维护性、可扩展性、健壮性及交互性，使发电集团和各电厂间真正实现资源管理自治，降低维护成本。

5.2 进一步的工作展望

在系统设计和实现的过程中，由于主要从功能实现的角度构造系统，系统的界面设计可进一步完善，可依据 ARBAC02 功能模型，增加更多的功能模块以满足更多的现实功能需求。

XACML 包括访问控制策略语言和请求/响应语言。本文主要涉及的是和 RBAC 特性相关的访问控制策略语言，构造 PDP 模块，在网络的请求/响应中必然包括 PEP 模块。可结合 PEP 和 PDP 进行负荷测试，验证在并发条件下的响应性能。可和他系统比如开放式网格数据安全访问 OGSA-DAI 整合，验证整合后的性能。

在大范围的区域内，必然存在大量的策略描述，以及动态职权分离 DSD，策略设计不仔细必然会有策略冲突。有必要设计定义 XACML 模板添加更详细的权限约束，更有效地加强访问控制。

对于大量的用户信息，可以添加用户分组信息，控制用户节点数，提升系统性能。

致 谢

本论文是在万定生导师的悉心指导和帮助下完成的。万老师渊博的学识、严谨的治学态度和务实的工作作风使我受益匪浅。在三年的学习和生活中给我提供了良好的学习和研究环境，认真耐心地解答我在生活和学习中的问题，在论文选题上和细节上都给予了富有远见的指导，没有万老师的指导，不可能完成这篇论文。至此论文完成之际，首先向万老师表示衷心的感谢！

同时感谢岳金桂老师、胡吉明老师在学习过程中，对我的鼓励和支持。

感谢陈强、任翔、王春风、陈静以及在一起生活的同学们，没有你们的友善，不可能有这么愉快的研究生生活。

感谢所有关心、帮助我的老师、同学和朋友们！

感谢我的父母和其他所有的亲人，是他们多年来无私的奉献和关爱，才能使我顺利地完成学业。

参考文献

[1] Sandhu, R.S., Samarati. P. Acess control: principle and practice. Communications Magazine, IEEE, 1994, 32(9):40~48

[2] ANSI INCITS 359-2004-Role Based Access Control: http://csrc.nist.gov/rbac/.

[3] 刘宏月,范九伦,马建峰.访问控制技术研究进展.小型微型计算机系统,2004,25(1):56~59

[4] Sandhu, R.S., Coyne.E.j, Feinstein.H.L, et al. Role-based access control models. Computer, 1996,29 (2):38~47

[5] David F Ferraiolo, Ravi Sandhu, Serban Gavrila et al. Proposed NIST standard for role based access control. ACM Transaction Information and System Security, 2001,4(3):224~274

[6] 张纲,李晓林,游赣梅等.基于角色的信息网格访问控制的研究.计算机研究与发展,2002,39(8):952~956

[7] Osborn, S., Sandhu, R., & Munawer, Q. Configuring Role-based Access Control to Enforce Mandatory and Discretionary Access Control Policies. ACM Transaction on Information and System Security, 2000, 3(2):85~106

[8] eXtensible Access Control Markup Language, (XACML) Version 2.0 OASIS Standard, 1 Feb 2005.http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-corespec-os.pdf 2006-06-15.

[9] 薛伟,怀进鹏.基于角色的访问控制模型的扩充和实现机制研究.计算机研究与发展,2003,40(11):1635~1641

[10] 陈林,阳富民,胡贯荣.基于角色的多级访问控制模型.华中科技大学学报(自然科学版),2002,30(2):102~104

[11] 耿晖，王海波．基于 XML 的角色访问控制 (RBAC). 计算机应用研究,2002,12(1):12~14

[12] A Brief Introduction to XACML. http://www.oasisopen.org.

[13] Vuong, N. N., Smith, G. S., & Deng, Y. Managing Security Policies in a Distributed Environment Using eXtensible Markup Language(XML). Symposium on Applied Computing, Las Vegas, NV, USA. 2001, 3:405~411.

[14] 孟珂, 阮永良. Web 服务中基于 XML 的 RBAC 策略模型. 计算机工程与设计.2005, 2, 26(2):397~399.

[15] Schaad,A., Mofett, J., & Jacob J. The Role-Based Access Control System of a Europeran bank: A Case Study and Discussion. ACM Press, 2001:3~9.

[16] 杨宏伟, 李晶, 虞淑瑶, 一种基于 XACML 访问控制策略决策服务的安全模型. 微电子学与计算机，2005,22,(8):151~154.

[17] Lorch, M., Kafura, D., Shah, S. An XACML-based Policy Management and Authorization Service for Globus Resources. Grid Conputering, 2003. In Proceedings of Fourth International Workshop, 2003, 11(17):208~210.

[18] L. Pearlman, V. Welch, I. Foster, C. etc. A Community Authorization Service for Group Collaboration. Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2002:50~60.

[19] J. Bacon, M.L., K. Moody. Translating Role-Based Access Control Policy within Context. Policies for Distributed Systems and Networks, 2001:1-7~119.

[20] S. Jajodia, P.S., V.S. Subrahmanian. A Logical Language for Expressing Authorizations. IEEE Symposium on Security and Privacy. 1997(5):31~42.

[21] N. Darnianou, N.D., E. Lupu, M. Sloman. The Ponder Policy Specification Language.Policies for Distributed Systems and Networkds. 2001:18~38.

[22] Pearlman, L., Welch, V., Foster, I., Kesselman, C., Tuecke, S., The Community Authorization Service: Status and Future, CHEP 2003:24~28.

[23] Markus Lorch, David Adams, Dennis Kafura, etc., The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments. Workshop on Grid Computing. 2003(4):109.

[24] M. Thompson, A. Essiari, S. Mudumbai, Certificate-based Authorization Policy in a PKI Environment. ACM Transactions on Information and System Security, 2003(8):566~588.

[25] D.W.Chadwick, A. Otenko. The PERMIS X.509 Role Based Privilege Management Infrastructure. In: Trent. Jaeger. Proc 7th ACM Symposium On Access Control Models And Technologies. California. 2002 .USA: Monterey, 2002:135~140.

[26] http://sunxacml.sourceforge.net
[27] http://mvpos.sourceforge.net/
[28] XML Security: Control information access with XACML. The objectives, architecture, and basic concepts of eXtensible Access Control Markup Language. http://www-128.ibm.com/developerworks/xml/library/x-xacml/ 2006-05-30
[29] M Lorch, S Proctor, R Lepro, D Kafura, S Shah, First experiences using XACML for access control in distributed systems, Proceedings of the 2003 ACM workshop on XML security, 2003.
[30] Sun's XACML Implementation, Programmer's Guide for Version 1.2. http://sunxacml.sourceforge.net/guide.html 2006-09-01.
[31] Ferraiolo, D., & Kuhn, R. Role-Based Access Control. In Proceedings of 15th National Computer Security Conference, 1992(10):554~563.
[32] Sandhu, R., Ferraiolo, D., & Kuhn, R. The NIST Model for Role-Based Access Control: Towards A Unified Standard. In Proceedings of the 5th ACM Workshop on Role-based Access Control, 2000, 7:47~63.
[33] Sandhu, R., Bhamidipati, V., Coyne, E., Ganta, S., & Youman C. The ARBAC97 Model for Role-Based Administration of Roles: Preliminary Decriptioin and Outline. In Proceedings of the second ACM Workshop on Role-Based Access Control, 1997, 11:41~50.
[34] Sandhu, R, & Munawer, Q, The ARBAC97 Model for Role-Based Administrating of Roles. ACM Transtraction on Information and System Security, 1999, 2(1):105~135.
[35] Sandhu, R., & Munawer, The ARBAC99 Model for Administrating of Roles. In Proceedings of 15th Annual Computer Security Application Conference, Phoenix, Arizona, USA, 1999, 12:229~238.
[36] Oh, S. & Sandhu R., A Model for Role Administration Using Organization Sturcture. In Procesdings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002), Monterey, California, USA, 2002, 6:155~168.
[37] Chen Y., Yang S.B., Guo L.T., etc, Design and Implementation of Dynamic-Role Based Access Control Framework in Grid Environment. Proceedings of the International Conference on Information Technology: Coding and Computing, 2005, 2:758~759.
[38] Crampton J. Specifying and enforcing constraints in role-based access control[A]. In: Proceedings of ACM Symposium on Access Control Models and Technologies[C]. New York: ACM Press, 2003.43~50.
[39] Chandramouli, R. (2000, July). Application of XML Tools for Entrtprise-Wide RBAC Implementation Tasks. In Proceedings of the 5th ACM Workshop on Role-based Access Control, 11~18.
[40] 阎宏. 专题：MVC 模式与用户输入数据检查. Java 与模式，电子工业出版社，2005.1:729~735.
[41] Log4j Project: http://logging.apache.org/log4j
[42] JFreeChart Project: http://www.jfree.org/jfreechart/