Um Modelo Discricionário de Delegação e Revogação

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Fábio Negrello e aprovada pela Banca Examinadora.

Campinas, 24 de junho de 2007.

Prof. Dr. Jacques Wainer (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Júlia Milani Rodrigues – CRB8a / 2116

Negrello, Fábio
N222m Um modelo discricionário de delegação e revogação / Fábio Negrello -- Campinas, [S.P. :s.n.], 2007.

Orientador : Jacques Wainer
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Computadores – Controle de acesso. 2. Computadores – Medidas de segurança. 3. Banco de dados – Medidas de segurança. I. Wainer, Jacques. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: A discretionary model of delegation and revocation.

Palavras-chave em inglês (Keywords): 1. Computers – Access control. 2. Computer – System security. 3. Data bases – Security measures.

Área de concentração: Sistemas de Informação

Titulação: Mestre em Ciência da Computação

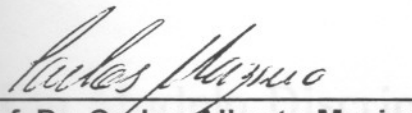
Banca examinadora: Prof. Dr. Jacques Wainer (IC-UNICAMP)
Prof. Dr. Carlos Alberto Maziero (PUC-PR)
Prof. Dr. Ricardo Dahab (IC-UNICAMP)
Prof. Dr. Paulo Lício de Geus (IC-UNICAMP)

Data da defesa: 14/05/2007

Programa de Pós-Graduação: Mestrado em Ciência da Computação

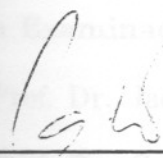
TERMO DE APROVAÇÃO

Tese defendida e aprovada em 14 de maio de 2007, pela Banca examinadora composta pelos Professores Doutores:



Prof. Dr. Carlos Alberto Maziero
PUC-PR.

Prof. Dr. Ricardo Dahab
IC – UNICAMP.



Prof. Dr. Jacques Wainer
IC – UNICAMP.

Instituto de Computação
Universidade Estadual de Campinas

Um Modelo Discricionário de Delegação e Revogação

Fábio Negrello

Junho de 2007

Banca Examinadora:

- Prof. Dr. Jacques Wainer (Orientador)
- Prof. Dr. Carlos Alberto Maziero
- Prof. Dr. Ricardo Dahab
- Prof. Dr. Paulo Lício de Geus (Suplente)

Resumo

Esta dissertação apresenta um modelo discricionário de delegação que permite controlar a formação de cadeias de delegações, tanto através da limitação no comprimento de tais cadeias, como através da definição de condições para utilização e aceitação de novas delegações. Juntamente com o mecanismo de delegação proposto, é apresentado um mecanismo de revogação que considera o máximo comprimento de cada cadeia de delegações, e a relação de força entre delegações, permitindo assim que os sujeitos existentes permaneçam com o maior conjunto de direitos após uma revogação.

Uma das principais vantagens em relação à definição de condições associadas à cada delegação é possibilidade de reforçar restrições por conteúdo e contexto. Enquanto o controle de acesso por conteúdo permite que o acesso a determinado objeto, ou recurso, seja controlado com base em atributos e características do próprio objeto em questão, o controle de acesso por contexto considera características de contexto relativas ao sistema como um todo, ou referentes ao contexto em que o usuário solicitou determinado acesso. Será apresentado um mecanismo que permite a utilização deste tipo de informação na definição de condições em delegações.

Será apresentado um mecanismo para definição de proibições, que torna possível proibir que usuários utilizem determinadas direitos, mesmo que estes usuários tenham recebido tais direitos através de delegações de outros usuários do sistema.

Através da utilização de condições também é possível a definição de delegações temporais, que são delegações que devem ser consideradas válidas somente durante determinados períodos de tempo, ou enquanto condições de dependência em relação a outras delegações forem atendidas, como será discutido.

Finalmente, será apresentado um arcabouço de um servidor de autorizações, que permitiu avaliar o modelo proposto. Neste arcabouço foram implementados os principais algoritmos apresentados, e foi formulada uma arquitetura unificada para criação e revogação de delegações, bem como para verificação de autorizações.

Palavras-chave: Delegação, Revogação, Autorização, Discricionário

Abstract

This thesis presents a model of delegation that makes it possible to control the creation of delegation chains, both by limiting the lenght of such chains, and by defining restrictions for the use and acceptance of new delegations. Together with the proposed delegation mechanism, it is presented a revocation mechanism that considers the maximum length of each delegation chain, and the strength relation between delegations, allowing the existing subjects to retain the maximum set of rights after a revocation.

One of the biggest advantages regarding the definition of conditions associated with each delegation is the possibility of enforcing context and content based restrictions. While the content based access control allows the access to a specific object to be controlled based on its attributes and characteristics, the context based access control considers context information related to the system as a whole, or regarding the context in which a user made an access request. It will be presented a mechanism that allows the use of this type of information in the definition of conditions in delegations.

A prohibition mechanism will be presented, which prevents users from using certain rights, even though these users have received such rights through other users delegations.

As it will be discussed, it is also possible, through the use of conditions, to define temporal delegations, which are delegations that must be considered valid only during specific periods of time, or while dependency condition regarding other delegations are met.

Finally, it will be presented a prototype of an authorization server, that was used to validate the proposed model. In this prototype, the main algorithms were implemented, and a unified architecture was formulated both for the creation and recation of delegations, as well as for the verification of authorizations.

Keywords: Delegation, Revocation, Authorization, Discretionary

Agradecimentos

Agradeço principalmente aos meus pais por seu apoio incondicional, que possibilitou que eu me dedicasse aos estudos.

Agradeço sinceramente ao Prof. Dr. Jacques Wainer, pela sua valiosa orientação e tempo, sempre com ótimas sugestões e idéias.

Agradeço aos colegas do CPqD, que permitiram sempre que necessário a minha dedicação ao curso de mestrado e ao desenvolvimento desta dissertação.

Agradeço a todos os professores e colegas que contribuíram direta ou indiretamente para que eu alcançasse meu objetivo.

Sumário

Resumo **v**

Abstract **vi**

Agradecimentos **vii**

1 Introdução e Motivação **1**

2 Trabalhos Relacionados **3**

3 Modelo de Delegação **6**
3.1 Representação de Autorizações 6
3.2 Cadeias de Suporte 7
3.3 Algoritmo para Revogação com Rebaixamento 9
3.4 Conclusão 13

4 Delegações Condicionais **14**
4.1 Funções Avaliadoras 15
4.2 Representação de Condições 16
4.3 Propriedades de Delegações Condicionais 18
4.4 Aceitação de Delegações Condicionais 22
4.5 Verificação de Autorizações de Domínio 24
4.6 Revogação de Delegações Condicionais 25
4.7 Algoritmo de Rebaixamento Condicional 27
4.8 Revogação com Delegações Contextuais 28
4.9 Rebaixamento de Delegações com Forças Diferentes 30
4.10 Autorizações para Revogação 32
4.11 Conclusão 34

5 Delegações Contextuais **35**
5.1 Controle de Acesso Baseado em Conteúdo 35
5.2 Controle de Acesso Baseado em Contexto 37
5.3 Aplicando Restrições MAC ao Modelo 38
5.4 Representação da Política *Chinese Wall* 40
5.5 Reforçando Restrições com Funções Avaliadoras 42
5.6 Conclusão 44

6 Autorizações Negativas **45**
6.1 Utilização de Autorizações Negativas 46
6.2 Algoritmo para Resolução de Conflitos 49
6.3 Bloqueio de Delegações 53
6.4 Conclusão 59

7 Delegações Temporais **60**
7.1 Principais Conceitos 60
7.2 Utilização de Delegações Condicionais Temporais 62
7.3 Algoritmo para Aplicação de Dependências 64
7.4 Conclusão 68

8 Implementação **69**
8.1 Futuro Refinamento da Implementação 73
8.2 Conclusão 73

9 Conclusão **75**
9.1 Trabalho Futuro 76

Bibliografia **78**

Lista de Figuras

3.1 Revogação da Delegação AB 10
3.2 Caminhos com Maiores Pesos Acumulados (λ) 12

4.1 Propagação de Condições de Controle 18
4.2 Propagação de Condições de Domínio 18
4.3 Cadeias de Suporte Parciais 20
4.4 Exemplo de Cálculo da Condição Efetiva 21
4.5 Autorizações de Controle e Domínio 25
4.6 Cadeia de Suporte da Delegação DF 27
4.7 Rebaixamento da Delegação AC 31
4.8 Passos para Rebaixamento de AB 32

6.1 Conflito entre S_0 e S_1 sobre S_2 50
6.2 Conflito entre S_0 e S_1 Independentes 52
6.3 Bloqueio de Delegações Dependentes 54
6.4 Delegações Ativas da Figura 6.3 57
6.5 Conflitos Dependentes 58

7.1 Grafo de Dependências 65

8.1 Arcabouço de um Servidor de Autorização 70

Um Modelo Discricionário de Delegação e Revogação

Fábio Negrello

24 de junho de 2007

Capítulo 1

Introdução e Motivação

Entre as principais políticas de controle de acesso atualmente utilizadas destacam-se a política de controle de acesso discricionária (DAC) e a política baseada em papéis (RBAC) [Samarati and di Vimercati 2001]. Entre os tipos de sistemas que normalmente utilizam a política DAC, embora não de forma isolada, estão os sistemas de gerenciamento de bases de dados (DBMS).

O modelo de autorização proposto por Griffths e Wade [Griffiths and Wade 1976], relacionado ao framework de bancos de dados relacionais System R [Astrahan et al. 1976], foi utilizado como base para a maioria dos sistemas DBMS comerciais atuais. Este modelo, como proposto inicialmente, possuía caráter essencialmente discricionário e descentralizado. O criador, ou dono, de um objeto podia conceder autorizações sobre este objeto a outros sujeitos do sistema. Opcionalmente, era possível conceder o direito de administração sobre este objeto a outros sujeitos. Desta forma, os sujeitos com direito de administração, assim como o criador, podiam conceder autorizações, ou ainda direito de administração, a outros sujeitos do sistema, formando assim uma cadeia de delegações. Neste modelo, ao se revogar uma autorização, todas as autorizações concedidas após a autorização revogada, e que não possuam outras autorizações de suporte, também são recursivamente revogadas. Este mecanismo de revogação considera que uma autorização pode suportar somente autorizações que tenham sido criadas posteriormente a ela mesma. Ou seja, durante uma revogação considera-se o conjunto de direitos existentes no momento da criação de cada delegação, e não o conjunto atual de direitos, o que seria mais adequado na maioria das aplicações.

Com base neste histórico, a maioria dos mecanismos de autorização atuais, seja voltados aos sistemas de bases de dados, ou voltados a aplicações de uso geral, permitem utilizar o conceito de delegação, que é a concessão de autorizações de um sujeito a outro. Embora seja necessário em sistemas descentralizados que outros sujeitos, além do criador de um objeto, tenham o privilégio de conceder autorizações, esta possibilidade pode levar

a dois problemas principais.

Primeiramente, as cadeias de autorizações podem crescer sem controle, tornando difícil para o dono de um objeto controlar quais sujeitos possuem acesso a este objeto, e em quais condições este acesso deve ser autorizado. Isto é especialmente significativo em sistemas onde sujeitos, objetos e autorizações representam, respectivamente, pessoas, recursos, e processos reais dentro de uma empresa ou organização. Como exemplo, observe que um gerente deveria ser capaz de garantir que uma tarefa especialmente importante não seja delegada a funcionários em um nível hierárquico distante, digamos, dois níveis dele próprio, o que poderia tornar difícil o acompanhamento sobre a execução da tarefa.

Além disso, sujeitos com interesses conflitantes em relação ao dono do objeto podem receber autorizações sobre este objeto via uma cadeia de delegações. Uma solução comum para este problema, empregada pela maioria dos modelos de controle de acesso, é a utilização de autorizações negativas. Entretanto, empregando-se esta solução em um sistema discricionário, o dono de um objeto deve atribuir autorizações negativas a todos os sujeitos que não deseja que o acessem. Isto nem sempre é factível, especialmente se o número de sujeitos é grande, ou se novos usuários são acrescentados com uma freqüência alta. Outra abordagem utilizada com o objetivo de controlar quais sujeitos têm acesso a determinadas autorizações é a definição de regras, ou restrições, genéricas. Esta abordagem é utilizada principalmente em modelos avançados de controle de acesso que permitem a especificação de autorizações através de linguagens lógicas [Woo and Lam 1993].

Resumindo, além de ser possível transmitir direitos de administração a sujeitos em um sistema, são necessárias também formas de se controlar a profundidade das cadeias de delegações formadas a partir do dono de um objeto e, além disso, prover mecanismos de controle que permitam especificar as condições que os sujeitos devem satisfazer para que possam adquirir determinadas autorizações, evitando assim conflitos em relação ao dono do objeto.

Neste documento, primeiramente, serão resumidamente apresentados os trabalhos principais na área de modelos de autorização e revogação. Em seguida será apresentada a base do modelo conceitual proposto. Serão tratadas questões relacionadas à revogação de delegações simples, e será apresentado um algoritmo eficiente para revogação de tais delegações. Em seguida, este algoritmo será estendido para considerar delegações condicionais, que serão utilizadas como base para um mecanismo de controle de acesso baseado em contexto e conteúdo nas seções seguintes. No restante do documento será apresentado um mecanismo de autorizações negativas, permitindo um controle ainda mais fino sobre delegações e, finalmente, será apresentado um mecanismo para definição de delegações temporais, e um algoritmo para aplicação de dependências entre delegações.

Capítulo 2

Trabalhos Relacionados

Dentre os principais modelos de autorização, os mais conhecidos são os modelos *Take-Grant* [A.K. Jones and Snyder. 1976], e *Action-Entity* [U. Bussolati and Martella. 1993], sendo o segundo basicamente um aprimoramento do primeiro com um conjunto de privilégios de administração, e suporte a predicados em autorizações. O modelo *Take-Grant* utiliza uma estrutura em forma de grafo para representar sujeitos e objetos, bem como autorizações. Um determinado arco rotulado com um ou mais direitos indica que o sujeito, representado pelo vértice fonte, é autorizado a exercer os direitos sobre o objeto, representado pelo vértice destino.

Na área de sistemas de gerenciamento de bases de dados, o modelo proposto por Griffths e Wade dentro do framework do System R [Griffiths and Wade 1976], foi utilizado como base na maioria dos sistemas comerciais utilizados atualmente [Bertino et al. 1997]. O modelo baseado em grafos proposto em [Ruan and Varadharajan 2004], suporta delegação de autorizações e, ao contrário do modelo *Take-Grant* possui a vantagem de suportar autorizações negativas, e possuir mecanismos para resolução de conflitos quando autorizações divergentes são atribuídas ao mesmo sujeito.

Na literatura é ampla a gama de trabalhos relacionados ao emprego de delegações em sistemas que utilizam a política RBAC. Em especial, o modelo RBDM0 [Barka and Sandhu. 2000] é proposto como uma extensão ao modelo RBAC para permitir a inclusão de delegações. Neste modelo, todas as autorizações associadas a um papel são delegadas, como um conjunto. O modelo de delegação proposto em [Zhang et al. 2003a] permite delegações na forma DLGT(User1, Role1, User2, Role2), onde User1, associado ao papel Role1, delega este papel ao usuário User2, o qual já se encontra associado ao papel Role2. Assim como o modelo RBDM0, as autorizações são delegadas em conjunto. A necessidade de delegações parciais, isto é, de somente um subconjunto das autorizações associadas a um papel, é considerada pelo modelo PBDM [Zhang et al. 2003b]. Este modelo traz a vantagem de permitir delegações parciais, além de permitir delegações de um

papel a outro, bem como de um usuário a outro. O modelo proposto em [Yao et al. 2001] utiliza o conceito de nomeação, ou designação. Um usuário nomeia outro usuário, através da passagem de uma credencial, permitindo que este usuário ative um ou mais papéis. A ativação de papéis é baseada em regras. Assim, é possível que usuário nomeie outro usuário, mesmo que o nomeador não possua as autorizações que o nomeado venha a receber. Este conceito é similar ao conceito de delegações fortes, que será apresentado neste artigo, na medida em que usuário delega a outro usuário autorizações que ele mesmo não possui.

O modelo $UCON_{ABC}$ [Park and Sandhu 2004] introduz o conceito de *controle de uso*, em uma tentativa de unificar as várias formas de controle de acesso modernas e DRM (*Digital Rights Management*). A base deste modelo é uma família de submodelos que consistem de oito componentes básicos: sujeitos, atributos de sujeitos, objetos, atributos de objetos, direitos, autorizações, obrigações e condições. Autorizações são predicados que devem ser avaliados com base em atributos do sujeito ou objeto, e em relação ao direito requisitado, e que retornam se o sujeito tem permissão para executar o direito requisitado sobre o objeto. Obrigações são predicados que verificam pré-requisitos obrigatórios que um sujeito tem que satisfazer para o uso do direito. Em relação a obrigações, os atributos de sujeitos e objetos são utilizados somente para selecionar o conjunto de obrigações que devem ser satisfeitas, e não para decidir o acesso. Condições têm a mesma função de autorizações, mas podem ser baseadas somente em informações de ambiente, ou orientadas ao sistema, que não têm relação direta com atributos do sujeito ou objeto. Autorizações, obrigações e condições podem ser avaliadas antes, durante ou após a utilização do direito, ao contrário de modelos convencionais de controle de acesso. Os atributos de sujeitos e objetos podem ser considerados mutáveis ou imutáveis. No primeiro caso, os atributos podem ser utilizados para refletir o uso do direito, podendo ser utilizados como fatores de decisão no acesso corrente ou em requisições futuras. O modelo $UCON_{ABC}$ permite um controle preciso sobre o uso de direitos, porém não permite controlar como tais direitos são delegados ou revogados.

Em relação a mecanismos de revogação, o modelo *Griffths-Wade* utiliza *timestamps* para identificar as dependências entre as delegações existentes no sistema. Quando uma delegação é revogada, todas as delegações dependentes, ou seja, conferidas depois desta sendo removida, e que não possuam outra delegação de suporte, são também removidas. O mecanismo de revogação proposto em [Bertino et al. 1997], denominado *noncascading-revocation*, é uma extensão ao modelo *Griffths-Wade*. Este mecanismo também utiliza o *timestamp* das autorizações como um modo de se determinar dependências entre as delegações existentes no sistema. Porém, esta extensão permite que as autorizações revogadas sejam automaticamente reatribuídas pelo sujeito que as revogou, evitando assim a revogação em cascata.

Parte do modelo proposto nesta dissertação foi baseada no modelo DW-RBAC [Wainer et al. 2007]. Este modelo, como proposto originalmente, é voltado à área de sistemas de *workflow*, permitindo a delegação de tarefas entre usuários, sem a necessidade que o executor da tarefa pertença a um papel específico, como exigido por modelos baseados puramente no modelo RBAC. Este modelo definiu os conceitos de comprimento de cadeias de delegações, e delegações condicionais, que permitem restringir os usuários que podem executar uma tarefa segundo propriedades que devem ser obedecidas. O modelo também permite a definição de regras de alto nível que controlam a criação de delegações segundo restrições organizacionais. Tais regras, que podem ser tanto gerais quanto específicas para um determinado caso de execução, podem ser utilizadas para reforçar a separação (*separation of duties*) ou a ligação de deveres (*binding of duties*).

O modelo proposto nesta dissertação difere do modelo DW-RBAC, visto que o modelo aqui proposto é puramente discricionário, embora ainda possa ser utilizado em conjunto com o modelo RBAC. Outra diferença principal se refere ao fato de que o modelo DW-RBAC não define uma separação entre direitos de administração e utilização de autorizações, portanto não permite que sejam impostas condições distintas para utilização destes direitos. Neste documento, é dada continuidade ao trabalho iniciado em [Negrello and Wainer 2006], no qual são definidos os algoritmos principais de aceitação e revogação de delegações, e são propostas extensões ao modelo, tais como delegações temporais e autorizações negativas, além do mecanismo de controle de acesso por conteúdo e contexto.

Capítulo 3

Modelo de Delegação

3.1 Representação de Autorizações

Uma autorização representa o direito, ou privilégio, de execução de determinada operação sobre um objeto, ou recurso, do sistema.

À cada autorização, é associada uma condição Q, que limita a utilização desta autorização. Esta condição pode ser baseada em informações relativas ao contexto atual, bem como relativas ao objeto, receptor ou delegador de determinada autorização. Seja O o conjunto de objetos a serem autorizados no sistema, e F o conjunto de operações que podem ser executadas sobre estes objetos. Uma Autorização é então definida por uma tripla (o, f, Q) tal que $o \in O$, $f \in F$ e Q é a condição restritiva da autorização.

Uma autorização pode também ser considerada um objeto de outra autorização. Neste caso, as operações possíveis são delegar ou revogar a autorização. Vamos representar a autorização de delegar uma autorização $a_1 = (o, f, Q_1)$ como $a_2 = (a_1, \textit{delegate}, Q_2)$. Neste caso, a condição restritiva Q_2 se refere à ação de delegar em si, e não à execução da operação f sobre o objeto o. Enquanto a condição Q_1 deve ser respeitada pelos sujeitos que vierem a usufruir da autorização a_1, a condição Q_2 deve ser respeitada pelos sujeitos que a delegarem. Chamamos a autorização a_1 de *autorização de domínio*, por se referir a objetos e operações reais do domínio considerado, enquanto que chamamos a autorização a_2 de *autorização de controle*, por controlar aspectos internos relativos ao modelo de autorização.

Neste modelo consideramos que uma delegação é a ação através da qual um sujeito concede a outro sujeito determinada autorização, e que tal delegação não implica na perda da autorização por parte do delegador, mas sim que o receptor da delegação passa a possuir esta autorização enquanto houver uma cadeia de suporte válida. Tal tipo de delegação é chamada de *monotônica*.

Observe que é possível que um sujeito possua somente o direito de delegar determinada

autorização de domínio, sem entretanto possuir a autorização delegada em si. Chamamos de delegação forte esta situação, em que o delegador não possui a autorização delegada, mas somente a autorização de delegar.

Permitir que um sujeito receba somente o direito de delegar determinada autorização é desejável para assegurar separação de deveres. Por exemplo, em um hospital, um médico chefe pode ser responsável por encaminhar pacientes a outros membros da equipe, sem necessariamente possuir ele mesmo o direito, ou a função, de medicar o paciente. Da mesma forma, um médico que possui autorização para medicar um paciente pode não possuir autorização para delegar esta função a outro médico, visto que a delegação de tarefas é responsabilidade do médico chefe. Entretanto, em outras situações, pode ser necessário requerer que o delegador de determinada autorização possua a autorização delegada em si. Neste modelo, isto poderia ser obtido associando-se uma condição restritiva que obrigue o delegador a possuir a autorização de domínio delegada.

3.2 Cadeias de Suporte

Além de definir condições para que determinada autorização seja delegada, é necessário em alguns casos definir se determinada autorização pode ser delegada adiante a terceiros, ou não. Caso seja possível delegá-la a terceiros, é interessante definir em até que profundidade isto é possível. Se um sujeito possui a autorização (a, *delegate*, Q), este sujeito possui o direito de delegar a autorização a adiante, mas os sujeitos que a receberem não possuem tal direito. Para que isto seja possível, é necessário delegar o direito de delegar.

Se um sujeito possui a autorização aninhada $((a, delegate, Q), delegate, Q)$, então este sujeito possui autorização de delegar o direito de delegar. Podemos generalizar este tipo de autorização aninhada como $(a, delegate, Q)^n$. Desta forma, a autorização $(a, delegate, Q)^n$ permite que seu possuidor crie uma cadeia de delegações de no máximo n passos. Neste documento iremos nos referir a n como o *peso* de uma delegação. No restante do documento, a autorização de delegar a autorização a, ou seja $(a, delegate, Q)^n$, será representada de forma abreviada por $del(a)^n$, ou por $del(a, Q)^n$ quando for necessário especificar explicitamente a condição Q.

Seja S o conjunto de sujeitos definidos no sistema, e A, o conjunto de autorizações possíveis. Então uma delegação é representada por del(g, r, a), onde g e $r \in S$, $a \in A$. Denotamos por g e r, o delegador e o receptor da delegação, respectivamente. A autorização sendo delegada é representada por a.

A função *decrement*, utilizada para representar a perda natural do poder de delegação ao longo de uma cadeia de delegações, possui a seguinte propriedade:

- $decrement(del(a)^n)) = del(a)^{n-1}$ para $n > 1$.

Uma delegação com peso n é uma delegação que pode criar uma cadeia de delegações de no máximo n passos, tal que uma delegação no passo i suporta a delegação no passo *i+1*. Dizemos que uma delegação $d_1 = \text{del}(g_1, r_1, a_1)$ suporta outra delegação $d_2 = \text{del}(g_2, r_2, a_2)$ se as seguintes condições são satisfeitas:

- $r_1 = g_2$: o receptor de d_1 é o delegador de d_2;
- $decrement(a_1) \sqsupseteq a_2$: o direito de delegação em d_2 é no máximo tão forte quanto $decrement(a_1)$.

A *relação de força* entre duas autorizações T=(o, f, Q) e T'=(o', f', Q'), utilizada acima, é definida como T $\sqsupseteq$ T' se e somente se:

$$\begin{cases} \text{T} = \text{T'}, \text{ ou} \\ o \sqsupseteq o' \text{ e } f \sqsupseteq f', \text{definidos segundo critérios específicos do domínio em questão.} \end{cases}$$

A relação de força entre operações ou entre objetos pode ser baseada em critérios específicos de uma aplicação. Como exemplo, em uma aplicação da área de saúde, podemos definir que a operação *tratar paciente* é composta pelas operações *medicar paciente* e *dar alta a paciente*. Portanto, *tratar paciente* $\sqsupseteq$ *medicar paciente* e *tratar paciente* $\sqsupseteq$ *dar alta a paciente*. Similarmente, podemos definir que o objeto *prontuário* compreende os objetos *dados cadastrais* e *exames realizados*, portanto um médico com acesso ao prontuário de um paciente teria acesso tanto aos dados cadastrais quanto aos exames realizados por este paciente.

Observe que as condições restritivas não são utilizadas na relação de força entre autorizações. Portanto, uma delegação pode suportar outra delegação independentemente da condição restritiva imposta pela autorização desta outra delegação. A relação de força entre duas delegações é definida como:

- $\text{del}(g, r, del(a)^n) \sqsupseteq \text{del}(g, r, del(a)^k)$ se $n > k$
- $\text{del}(g, r, del(a)^n) \sqsupseteq \text{del}(g, r, del(a')^n)$ se $a \sqsupseteq a'$

Se para duas delegações D e D_0, temos que $D \sqsupseteq D_0$, dizemos que D_0 é mais fraca que D. Podemos também definir $D \sqsupset D_0$ como $D \sqsupseteq D_0$ e $D \neq D_0$, e chamamos $\sqsupset$ de relação estritamente forte.

Finalmente, uma cadeia de delegações é representada por uma lista ordenada de delegações do tipo $d_i = \text{del}(G_i, R_i, A_i)$, tal que:

- d_0 é a raiz da cadeia de delegações;
- d_i suporta d_{i+1};

- não há nenhum par d_i e d_j na cadeia, tal que $d_i = d_j$ (ciclos são permitidos mas não são considerados na formação de cadeias de suporte).

Uma vez que $del(a)^i \sqsupseteq del(a)^k$ para todo $i > k$, não necessariamente uma delegação deve ocorrer a partir do último sujeito de uma cadeia de delegações. Intuitivamente, isto significa que se um delegador possui o direito de delegar o direito de delegar uma autorização, este sujeito possui também o direito de delegar diretamente a autorização em questão. Observe também que um delegador pode escolher qual o poder de delegação que será passado adiante, podendo até mesmo escolher não delegar o poder de delegar.

O conjunto de delegações é representado através de um grafo simples cíclico, dirigido, com pesos, em que cada aresta representa uma delegação, e o peso da aresta representa o peso da delegação correspondente. Para cada objeto o, o grafo G[o] é utilizado para representar todas as delegações em relações ao objeto o. Seja G[o] = (V, E) um grafo dirigido com pesos, onde V é um conjunto finito de vértices representando os sujeitos que possuem alguma autorização em relação a o, E é um conjunto finito de arcos tal que se existe uma delegação del(g, r, a), então (g, r) $\in$ E. Desta forma, o conjunto total de delegações G pode ser representado como G = $\{$G[o] | o $\in O\}$, ou seja, a união dos subgrafos relativos a cada objeto o. As definições apresentadas acima serão utilizadas a seguir na discussão do problema de revogação de delegações com rebaixamento (diminuição de peso).

3.3 Algoritmo para Revogação com Rebaixamento

Uma delegação inicial pode permitir a formação de uma cadeia independente de delegações de tamanho arbitrariamente grande, possuindo possivelmente ciclos. Ao revogarmos uma delegação que pertença a uma cadeia de suporte, para cada delegação d_i restante poderá ser necessário:

1. remover a delegação d_i pois a mesma não é mais alcançável a partir de outras cadeias de delegações, ou

2. diminuir o peso de d_i, para que esta delegação passe a possuir uma cadeia de suporte alternativa válida.

Estes dois casos são ilustrados na figura 3.1. Ao revogarmos a delegação de A a B, a delegação de B a C passa a não possuir uma cadeia de suporte, pois B não é alcançável a partir de qualquer outro sujeito. Portanto, a delegação de B a C deve ser removida. Isto ilustra o caso (1). As delegações CD, DE e EC devem ter seus pesos atualizados para que possam utilizar uma cadeia de suporte alternativa válida. No caso do sujeito D, que antes possuía a cadeia de suporte (AB, BC, CD), passa a possuir a cadeia de

suporte alternativa (AE, EC, CD), ilustrando o caso (2). Entretanto, é necessário que os pesos ao longo da seqüência de delegações (AE, EC, CD) sejam atualizados, para que a mesma possa ser considerada uma cadeia de suporte válida, com início no sujeito dono da autorização, representado por A.

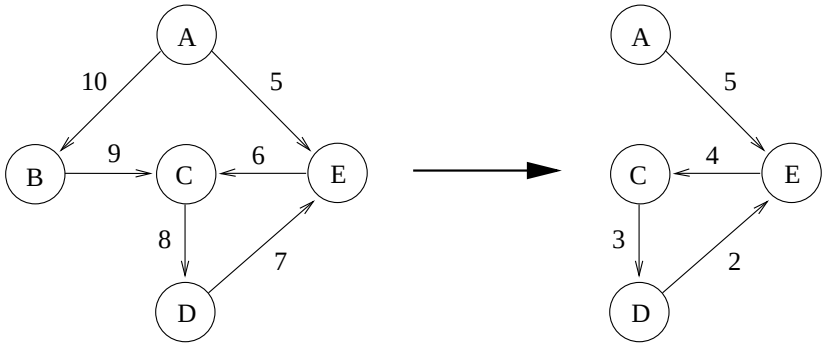


Figura 3.1: Revogação da Delegação AB

Uma determinada delegação d possui uma cadeia de suporte válida se e somente se existe ao menos uma cadeia de delegações $d_0...d_k$ tal que d_k suporta d, o que nos leva à seguinte definição:

Definição 1: Um conjunto de delegações é válido se e somente se toda delegação possui ao menos uma cadeia de suporte válida.

Claramente, se um conjunto de delegações em relação a um único objeto e operação, representado pelo grafo G = (V, E), é válido, então toda aresta $uv \in$ G apresenta a seguinte propriedade:

Propriedade 1: $w(uv) < max\{w(t_iu)\}$, para todo t_iu $\in$ E.

Isto é, o peso da aresta entre os vértices u e v, representado por $w(uv)$, é inferior ao maior peso das arestas incidentes em u.

Definição 2: Seja P = $v_0...v$ um caminho em G. Dizemos que P é válido se para todo par de arestas subsequentes e_i e e_{i+1} em P, temos que $w(e_{i+1}) < w(e_i)$.

Definição 3: o máximo peso acumulado em um vértice v_i, denotado por $\lambda(v_i)$, é o máximo peso que toda aresta v_iv_{i+1} pode adquirir, sem ferir a propriedade 1, e tal que o caminho $v_0...v_{i+1}$ seja um caminho válido.

Definição 4: o peso efetivo de uma aresta uv, denotado por $\psi(uv)$, é o mínimo valor entre $w(uv)$ e $\lambda(u)$. Ou seja: $\psi(uv) = min(w(uv), \lambda(u))$

Claramente, se G é válido, temos que $w(uv) < \lambda(u)$ para todo $uv \in$ E. Portanto a seguinte propriedade também é válida:

Propriedade 2: $\psi(uv) = w(uv)$, para todo $uv \in$ E.

Suponha que uma aresta tenha seu peso diminuído, tornando o grafo G inválido, ou seja, não obedecendo as propriedades 1 e 2. Temos que recalcular os pesos de todas as arestas de G, de forma que estas propriedades sejam novamente satisfeitas, e que as delegações mantenham o máximo peso possível. Para tanto, é necessário para todo $v \in$ G, calcular $\lambda(v)$. Em seguida, é necessário recalcular os novos pesos de todas as arestas uv em G, denotados por $w'(uv)$, utilizando-se a seguinte fórmula:

$$w'(uv) = \psi(uv) = min(w(uv), \lambda(u))$$

Assim garantimos que todas as arestas possuem o peso máximo permitido pelas cadeias alternativas, assim obedecendo as propriedades 1 e 2. O algoritmo de Dijkstra [Bondy and Murty 1976], ou uma variação deste, normalmente é utilizado para o cálculo do caminho mais curto de um vértice inicial a todos os outros vértices de um grafo. Entretanto, este algoritmo pode ser facilmente modificado para calcular o caminho mais longo, em lugar do caminho mais curto, de um vértice inicial u_0 aos outros vértices do grafo. Neste caso, u_0 representa o sujeito dono da autorização, e portando a raiz da cadeia de delegações. Suponha que S é um subconjunto de V tal que $u_0 \in$ S, e seja $\overline{S}$ o conjunto complementar V\S. Se P = $u_0 ... \overline{uv}$ é o caminho mais longo de u_0 a $\overline{S}$ então claramente $\overline{u} \in$ S e a secção (u_0, $\overline{u}$) de P deve ser o caminho mais longo entre u_0 e $\overline{u}$. Portanto:

$$dist(u_0,\overline{v}) = dist(u_0, \overline{u}) + w(\overline{uv})$$

A distância mais longa de u_0 a $\overline{S}$ é dada pela fórmula: $dist(u_0, \overline{S}) = max_{u \in S, v \in \overline{S}}\{dist(u_0, u) + w(uv)\}$. Utilizando esta idéia, utilizaremos $\psi(uv)$ em lugar de $dist(u_0, u) + w(uv)$, para refletir o peso efetivo de cada aresta. Desta forma, obtemos: $dist(u_0, \overline{S}) = max_{u \in S, v \in \overline{S}}\{\psi(uv)\}$. Para o cálculo de $\psi(uv)$, é necessário o valor de $\lambda(u)$. À medida que o algoritmo apresentado a seguir prossegue, os vértices são rotulados de tal que forma que no estágio i, temos:

$$l(u) = dist(u_0, u) \text{ para } u \in S_i$$

$l(v) = max_{u \in S_{i-1}}\{\psi(uv)\}$ para $v \in \overline{S}_i$

Podemos notar que $l(u)$ é exatamente igual ao valor $\lambda(u)$, necessário para o cálculo de $\psi(uv)$ para toda aresta $uv \in E$.

Algoritmo de Rebaixamento:

1. Seja $l(u_0) = \infty$, $l(v)$ = -∞ para $v \neq u_0$, $S_0 = \{u_0\}$ e $i = 0$.
2. Para cada $v \in \overline{S_i}$, substitua $l(v)$ por max$\{l(v)$, $\psi(u_i$v) - 1$\}$. Calcule $max_{v \in \overline{S}_i}\{l(v)\}$ e seja u_{i+1} o vértice em que esse máximo é obtido. Faça $S_{i+1} = S_i \cup \{u_{i+1}\}$.
3. Se $i = v - 1$, pare. Se $i < v - 1$, substitua i por $i + 1$ e volte para o passo 2.
4. Para cada $e \in$ E, substitua $w(e)$ por $\psi(e)$, e se $\psi(e) < 0$, remova a aresta e.

Através da figura 3.2 são representados os caminhos, a partir do nó raiz até os outros nós, com maiores pesos acumulados (λ) antes (figura 3.2.a) e após (figura 3.2.c) a revogação da delegação AB. Na figura 3.2.b são apresentados os pesos acumulados em cada nó, que são obtidos, neste exemplo, após quatro iterações do passo 2. Ao término deste algoritmo temos os pesos de todas as arestas substituídos pelos seus respectivos pesos efetivos $\psi(e)$. Portanto, todas as arestas remanescentes pertencem a ao menos um caminho válido. A ordem de complexidade deste algoritmo é similar à do algoritmo de Dijkstra, citado acima. Para o algoritmo de Dijkstra são necessárias aproximadamente $\nu(\nu\text{-}1)/2$ adições e $\nu(\nu\text{-}1)$ comparações. O algoritmo de rebaixamento, entretanto, necessita de Δ(G) $\times$ $\nu(\nu\text{-}1)$ comparações, devido ao cálculo de ψ, onde Δ(G) é máximo grau dos vértices em G. Desta forma, a complexidade do algoritmo proposto é da ordem de $O(V^2)$.

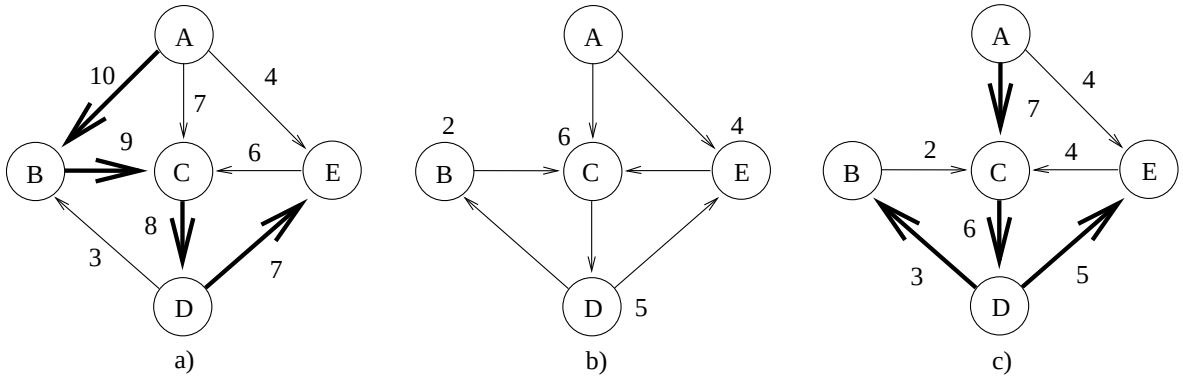


Figura 3.2: Caminhos com Maiores Pesos Acumulados (λ)

3.4 Conclusão

Neste capítulo foram introduzidos os principais conceitos necessários para o desenvolvimento do modelo de delegação e revogação, tais como os conceitos de cadeia de suporte e rebaixamento de delegações.

O algoritmo de revogação proposto caracteriza-se por manter as delegações com o maior peso possível, considerando as cadeias de suporte válidas para cada delegação. As delegações que não podem ser rebaixadas, por não possuírem cadeias de suporte alternativas, são revogadas. Este algoritmo será utilizado como base no restante deste documento para outros algoritmos necessários para a definição do modelo.

No próximo capítulo, o modelo será estendido para considerar delegações condicionais, e serão apresentados os algoritmos para aceitação de delegações e verificação de autorizações.

Capítulo 4

Delegações Condicionais

Uma autorização é considerada válida, isto é, pode ser utilizada, somente se a condição associada a esta autorização for satisfeita. Uma vez que o próprio direito de delegar pode ser representado como uma autorização, a criação de novas delegações também pode ser restringida utilizando-se o mesmo mecanismo para autorizações comuns.

Enquanto a condição imposta pela autorização de domínio é mais adequada para controlar a utilização de recursos do sistema, a condição imposta pela autorização de controle é mais adequada para restringir os sujeitos que possivelmente podem adquirir determinada autorização, ou restringir os sujeitos que podem delegar autorizações.

A condição de uma autorização pode ser baseada em atributos da autorização em si, tais como o delegador, o receptor, a operação, ou o objeto referenciado pela autorização, permitindo com isso definir uma ampla gama de critérios tanto para aceitação quanto para utilização de delegações.

A condição de uma autorização pode ser utilizada para representar restrições temporais. Como exemplo, digamos que que um sujeito possui autorização para acessar determinado módulo de um sistema. Entretanto, visto que a execução do módulo consome quantidade significativa de recursos das máquinas de determinada rede, queremos restringir o horário em que tal módulo pode ser executado. Neste caso, podemos associar à esta autorização uma condição temporal restringindo o horário de utilização do módulo.

Muitas aplicações requerem que o acesso a recursos seja restringido tanto com base no contexto atual do sistema, quanto com base em determinados atributos dos recursos acessados, ou seja, com base em informações de conteúdo de objetos, portanto também deve ser permitido que estas informações sejam utilizadas na formulação de condições.

4.1 Funções Avaliadoras

Uma função avaliadora tem como objetivo avaliar o valor de uma informação utilizada na definição da condição de uma autorização.

Definição 5: Uma Função Avaliadora é definida como uma função $Av(d, t, S)$, onde d representa uma delegação, t representa o instante atual do sistema, e S representa o conjunto de informações de contexto do sistema.

O resultado da execução de uma função avaliadora pode ser qualquer valor, ou tipo, necessário para a definição de condições. As informações obtidas por uma função avaliadora podem ser:

1. Informações relativas a uma delegação $d = \text{del}(g, r, a)$:
   - Sujeito concessor da delegação (g).
   - Sujeito receptor da delegação (r).
2. Informações relativas a autorização $a = (o, f)$:
   - Modo de acesso (f).
   - Identificador do objeto (o).
3. Informações de conteúdo do objeto o.
4. Informações relativas ao mecanismo de autorização:
   - Relacionamento de sujeitos com grupos.[1]
   - Relacionamento de sujeitos com papéis.
5. Informações relativas a uma sessão específica:
   - Sujeito executor, ou seja, o sujeito cujo acesso a determinada autorização está sendo verificado.
   - Forma de autenticação utilizada para iniciar a sessão. Esta informação pode ser utilizada para restringir o acesso a determinados objetos, dependendo do grau de confiança correspondente ao mecanismo de autenticação utilizado.
   - Localização do acesso: endereço de rede, localização física, ou simplesmente uma categorização da localização, tal como acesso remoto ou local.

[1]Embora o modelo proposto seja essencialmente discricionário, os conceitos de papel e grupo são ortogonais a este modelo, e podem ser suportados por mecanismos que o implementem.

6. Informações relativas a uma determinada regra de negócio do sistema que utiliza o mecanismo de autorização:
   - Relacionamento entre o objeto e os sujeitos receptor e delegador. Como exemplo, pode-se determinar se o prontuário que um médico quer acessar pertença a um de seus pacientes.
   - Disponibilidade de determinado pré-requisito para possuir acesso a um objeto.
7. Informações dependentes do instante t da verificação de acesso.

4.2 Representação de Condições

As funções avaliadoras utilizadas em uma condição são representadas como um literal, ou como uma função $f(p_0,...,p_n)$, onde p_i são funções avaliadoras das quais o resultado de f depende. Funções são mais adequadas para representar o relacionamento entre dois ou mais objetos. Por exemplo, podemos definir a função *tratou(médico, paciente)* para representar se um médico tratou de determinado paciente. Funções avaliadoras também podem ser definidas pela notação *Objeto.Atributo.Sub-Atributo*, para representar atributos de um objeto ou sujeito.

As implementações de funções avaliadoras podem utilizar bibliotecas, ou invocação a serviços, permitindo que condições sejam definidas externamente ao sistema de autorização. As funções avaliadoras predefinidas, isto é, disponibilizadas pelo próprio mecanismo de autorização, são listadas a seguir:

- Sujeito concessor de uma delegação: *Grantor*
- Sujeito receptor de uma delegação: *Delegate*
- Sujeito executor de uma autorização: *Executor*
- Modo de acesso de uma autorização: *AccessMode*
- Objeto de uma autorização: *Object*
- Instante da verificação de autorização: t

Definida a representação de informações, podemos definir a representação de uma condição:

Definição 6: Uma condição Q é definida como uma função $Q(d, t, S)$, onde d representa uma delegação, t representa o instante atual do sistema, e S representa o

conjunto de informações de contexto do sistema.

Os parâmetros d, t e S não são obrigatoriamente utilizados pela condição, permitindo a existência de condições dependentes de somente um tipo de informação. Como exemplo, em uma empresa, considere a seguinte condição para a delegação de uma autorização:

$Q = Delegate \in Coordenadores$

Esta condição depende de informações sobre a delegação (d) e informações de contexto (S). Ela restringe que uma delegação somente pode ser criada se o receptor da delegação pertencer ao grupo de coordenadores.

No restante deste documento, para uma delegação $d = (g, r, a)$, onde $a = (o, f, Q)$, quando verificarmos se uma outra delegação d' satisfaz a condição Q, iremos representar esta verificação por $Q(d')$.

Definição 7: Uma condição Q é representada por uma expressão lógica que recebe os valores de n funções avaliadoras, e retorna *Verdadeiro* ou *Falso*.

Q: $\{Av_i, 0 \leqslant i \leqslant n\} \rightarrow \{Verdadeiro, Falso\}$

Os operadores permitidos em uma condição são:

- Aritméticos: $+$, - (unário e binário), *, /, *div*, %, mod.
- Lógicos: $\wedge$, $\vee$, $\neg$.
- Relacionais: $=$, $\neq$, $<$, $>$, $\leq$, $\geq$.
- Sobre conjuntos: $\supset$, $\supseteq$, $\subset$, $\subseteq$, $\in$, $\cup$, $\cap$.
- Condicionais: $(E\ ?\ Y\ :\ N)$. Retorna Y se a expressão E é verdadeira, N caso contrário, onde Y e N são duas expressões.
- De dependência entre delegações (será explicado no capítulo 7): $\rightarrow$.

Neste documento, chamaremos as delegações com condições dependentes de t de *delegações temporais*. As delegações condicionais dependentes de S serão referidas como *delegações contextuais*. No restante deste capítulo, serão discutidos os conceitos e algoritmos relacionados a delegações condicionais em geral. As questões específicas de delegações temporais serão discutidas separadamente no capítulo 7 e as contextuais serão discutidas no capítulo 5.

4.3 Propriedades de Delegações Condicionais

Podem ser definidas condições tanto para a criação de novas delegações, quanto para a utilização de autorizações já delegadas.

Uma delegação d_0 = del(g, r, a_d) somente pode ser criada se for obedecida a condição Q_c imposta pela autorização de controle $a_c = (a_d, delegate, Q_c)^n$, ou abreviadamente $del(a_d, Q_c)^n$, que está associada ao sujeito g. Isto é, a utilização da autorização de delegar somente é permitida se a sua condição for satisfeita.

A associação de condições a autorizações de controle e domínio permite que os sujeitos limitem ainda mais os poderes de utilização e delegação de uma autorização. Por exemplo, um sujeito s_1 pode acrescentar a restrição de que somente sujeitos pertencentes ao papel de coordenador devem delegar determinada autorização. Um outro sujeito s_2, que recebeu a autorização de delegação de s_1, pode impor uma outra restrição, tal como que somente coordenadores seniores devem delegá-la. Um terceiro sujeito s_3, que recebeu a autorização de delegação de s_2, pode não querer impor restrições novas. Embora o sujeito s_3 não tenha imposto novas condições, as delegações criadas a partir de s_3 devem ainda obedecer as condições impostas por s_1 e s_2. Isto é, as condições são acumulativas, ou seja, são propagadas ao longo das cadeias de delegações.

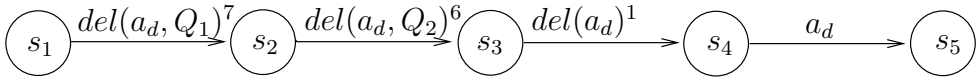


Figura 4.1: Propagação de Condições de Controle

No exemplo da figura 4.1, a delegação de s_2 a s_3 deve satisfazer a condição Q_1, enquanto as delegações de s_3 a s_4, e de s_4 a s_5, devem satisfazer as condições Q_1 e Q_2.

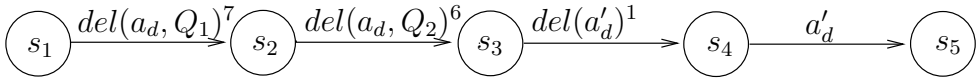


Figura 4.2: Propagação de Condições de Domínio

No exemplo da figura 4.2, suponha que o sujeito s_3 altere a autorização a_d = (o, f, Q_d) substituindo a sua condição Q_d pela condição Q'_d. Denotemos por a'_d = (o, f, Q'_d) esta nova autorização. Para que s_4 utilize a autorização a'_d será então necessário satisfazer não somente a condição Q'_d, mas também Q_d. Ou seja, as condições de autorizações de domínio também são propagadas.

A propagação de condições permite que cada sujeito preocupe-se somente com as restrições que ele mesmo deve impor. Não é necessário repetir as restrições impostas anteriormente, visto que isto seria inviável em sistemas com múltiplas cadeias de delegações

chegando a cada sujeito. Além disso, caso as condições não fossem automaticamente propagadas, algum sujeito poderia remover restrições impostas anteriormente, ou substituí-las por restrições mais fracas, criando assim possíveis falhas de segurança. A propagação de restrições também permite que novas restrições adicionadas a uma delegação sejam automaticamente propagadas a todas as delegações dependentes desta delegação.

Portanto, para que uma delegação seja aceita é necessário satisfazer não somente a condição imposta pela autorização de controle associada ao sujeito que deseja criar a delegação, mas é também necessário satisfazer as condições impostas pelos sujeitos pertencentes a pelo menos uma cadeia de suporte desta delegação.

Anteriormente, com relação a delegações não condicionais, foi definida a seguinte propriedade para que um conjunto de delegações seja válido:

Propriedade 1: $w(uv) < \max\{w(s_i u)\}$, $\forall\ s_i u$ em E.

Esta propriedade reflete o fato de que cada delegação uv deve pertencer a pelo menos uma cadeia de delegações tal que a delegação anterior possua um peso maior que o próprio peso de uv. Para delegações não condicionais, esta propriedade é intuitivamente suficiente para garantir a validade de todas as delegações referentes a uma autorização. Entretanto, com a introdução de delegações condicionais ao modelo, torna-se necessário redefinir o conceito de validade de uma cadeia de suporte.

Definição 8: Uma cadeia de suporte parcial de uma delegação d é uma cadeia de n delegações condicionais $d_0 ... d_{n-1}$, com condições denotadas respectivamente por $Q_0 ... Q_{n-1}$, tal que d_i suporta d_{i+1}, d_{n-1} suporta d, e $Q_i(d)$ é verdadeiro para $0 \leq i \leq n$.

Em uma cadeia de suporte parcial de uma delegação d, é necessário que esta delegação satisfaça todas as condições impostas ao longo desta cadeia. Entretanto, não é necessário que cada delegação satisfaça as condições impostas pelas delegações anteriores pertencentes à esta mesma cadeia de delegações.

Definição 9: Uma delegação condicional d possui uma cadeia de suporte válida se e somente se existe ao menos uma cadeia de n delegações $d_0 ... d_{n-1}$ tal que d_i suporta d_{i+1}, d_{n-1} suporta d, e toda delegação d_i possui ao menos uma cadeia de suporte parcial.

A definição acima significa que ao longo de uma cadeia válida de delegações condicionais toda delegação deve satisfazer as condições impostas pelas delegações pertencentes a pelo menos uma cadeia de delegações. Ou seja, toda delegação deve ter uma cadeia de delegações que justifique sua existência. No exemplo da figura 4.3, a delegação EF

pode possuir as cadeias de suporte parciais <AB,BC,CE> ou <AB,BD,DE>, enquanto a delegação FG pode possuir as cadeias <AB,BC,CE,EF> ou <AB,BD,DE,EF>. Mesmo que EF possua a cadeia de suporte parcial <AB,BC,CE>, a delegação FG pode possuir a cadeia de suporte parcial <AB,BD,DE,EF>. Ou seja, embora a delegação EF pertença necessariamente à cadeia de suporte de FG, suas cadeias de suporte parcial podem ser distintas.

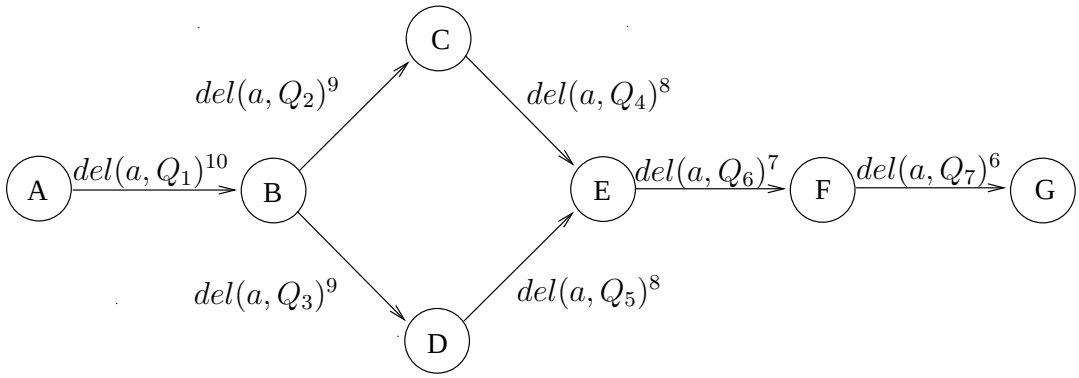


Figura 4.3: Cadeias de Suporte Parciais

Tomando-se como base a nova definição de validade apresentada, é necessário estabelecer uma propriedade suficiente para garantir que um conjunto de delegações seja válido, assim como foi estabelecido para condições não condicionais utilizando-se a propriedade 1. Considere a delegação condicional entre dois sujeitos, denotada por $d = \text{del}(g, r, a)$, onde a $= (o, f, Q)$. Sejam $d_k = \text{del}(s_k, g, a)$ as k delegações da autorização a ao sujeito g. Observe que a seguinte propriedade é válida:

Propriedade 3: $\bigvee_{i=0}^{k} (Q_i(d) \wedge (w(d) < w(d_i)))$

Entretanto, esta propriedade não é suficiente para garantir a validade de um conjunto de delegações, visto que a concordância com a condição imposta por uma delegação imediatamente anterior não implica que todas as condições ao longo de determinada cadeia sejam obedecidas, o que é necessário segundo a definição de validade apresentada acima. De fato, estamos interessados na condição efetiva imposta por determinada delegação, isto é, a condição que ela própria impõe, somada às condições ao longo da cadeia de suporte desta delegação. Apesar de ser necessária somente uma cadeia de suporte válida para cada delegação, é possível a existência de mais de uma cadeia de suporte. Portanto, a condição efetiva imposta por determinada delegação deve refletir as condições impostas por todas as cadeias as quais esta delegação pertença. Não é possível, portanto, definir uma propriedade local, tal como no caso de delegações não condicionais,

que seja suficiente para garantir a validade de um conjunto de delegações. Entretanto, a condição efetiva $\dot{Q}$ de uma autorização $a = (o, f, Q)$ pode ser recursivamente definida como:

Definição 10: $\dot{Q}= Q \wedge (\bigvee_{i=0}^{k} (\dot{Q}_i \wedge (w(d) < w(d_i))))$

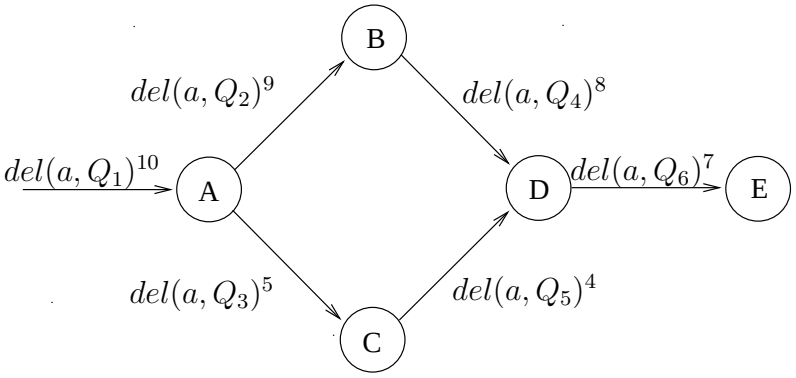


Figura 4.4: Exemplo de Cálculo da Condição Efetiva

Como exemplo, utilizando a definição da condição efetiva, podemos calcular a condição efetiva $\dot{Q}_6$ da delegação DE da figura 4.4. Segundo a definição de $\dot{Q}$, temos:

$\dot{Q}_6 = Q_6 \wedge ((\dot{Q}_4 \wedge (7 < 8)) \vee (\dot{Q}_5 \wedge (7 < 4)))$
$\Rightarrow \dot{Q}_6 = Q_6 \wedge (\dot{Q}_4)$
$\Rightarrow \dot{Q}_6 = Q_6 \wedge (Q_4 \wedge (\dot{Q}_2 \wedge (8 < 9)))$
$\Rightarrow \dot{Q}_6 = Q_6 \wedge (Q_4 \wedge (\dot{Q}_2)$
$\Rightarrow \dot{Q}_6 = Q_6 \wedge (Q_4 \wedge (Q_2 \wedge (\dot{Q}_1 \wedge (9 < 10))))$
$\Rightarrow \dot{Q}_6 = Q_6 \wedge (Q_4 \wedge (Q_2 \wedge (\dot{Q}_1)))$
$\therefore \dot{Q}_6 = Q_6 \wedge Q_4 \wedge Q_2 \wedge Q_1$

Com base na expressão para o cálculo da condição efetiva de delegações, podemos redefinir a propriedade necessária para a validade de uma delegação:

Propriedade 4: $\bigvee_{i=0}^{k} (\dot{Q}_i(d) \wedge (w(d) < w(d_i)))$

Esta propriedade, se atendida por todas as delegações, é suficiente para que um conjunto de delegações seja válido. Prova: Por definição, um conjunto C de delegações em relação a uma autorização é válido se toda delegação d em C possui uma cadeia de suporte parcial válida. Seja C um conjunto válido de delegações condicionais. Desta forma, segundo a definição de condição efetiva, a propriedade 4 é válida para toda delegação

em C. Suponha, por contradição, que ao acrescentarmos ao conjunto uma nova delegação $d = \text{del}(g, r, a)$, este conjunto passe a ser inválido, entretanto ainda obedecendo à propriedade 4. Sejam $t_0...t_k$ as delegações em C cujo receptor seja g. Pela proposição, as delegações $t_0...t_k$ possuem cada uma ao menos uma cadeia de suporte válida. Seja t_i a delegação tal que $\dot{Q}_i(d) \wedge (w(d) < w(d_i))$ é verdadeiro, e seja $d_0...d_k$ a cadeia de suporte desta delegação. Então a cadeia $d_0, ..., d_k, t_i$ é uma cadeia de suporte válida para d, o que contraria a suposição de que C é inválido.

4.4 Aceitação de Delegações Condicionais

A introdução de delegações condicionais ao modelo, como explicado acima, impõe não somente uma condição à aceitação de cada nova delegação, mas um número de condições igual ao da cadeia de suporte desta delegação. Intuitivamente, quando mais distante uma delegação se encontra do dono da autorização, mais condições o sujeito receptor de determinada delegação deve satisfazer. Para determinar se determinada delegação pode ser concedida, devemos verificar se a mesma possui uma cadeia de suporte válida, tanto em relação ao seu peso, quanto em relação às condições impostas ao longo da cadeia de suporte.

Para uma mesma delegação, pode ser possível a existência de uma ou mais cadeias de suporte válidas, entretanto estamos preocupados em estabelecer um algoritmo para determinar se nova delegação possui ao menos uma destas cadeias. Ou seja, devemos garantir que a propriedade 4, anteriormente apresentada, é válida para a nova delegação, já que esta propriedade é suficiente para garantir a validade do conjunto de delegações.

Algoritmo para Verificação de Cadeia de Suporte Parcial:
Entrada: Grafo G e delegação $d_0 = u_0v_0$.
Saída: Verdadeiro se d_0 possui uma cadeia de suporte *parcial* em G.

1. Seja r_0 a raiz de G.
2. Seja G' = (V', E') o subgrafo induzido pelos caminhos de r_0 a u_0.
3. Para todo $uv \in$ E', faça:
4. Se $Q(d_0)$ é *Falso*, faça:
5. E' $\leftarrow$ E' - uv
6. Aplique Rebaixamento(G').
7. Se $d_0 \in$ E' Retorne *Verdadeiro*, Senão Retorne *Falso*.

Este algoritmo primeiramente remove as delegações que possuem condições que não são atendidas pela delegação d_0. A condição de cada delegação, Q, é sempre avaliada em relação a d_0, isto é, verifica-se se a condição Q definida pela delegação uv é satisfeita pela delegação d_0. Uma vez removidas as delegações que não podem pertencer a cadeias de suporte de d_0, aplica-se o algoritmo de rebaixamento para verificar se os pesos das cadeias restantes podem suportar a delegação d_0. A complexidade deste algoritmo é a mesma do algoritmo de rebaixamento. Uma vez definida esta rotina auxiliar, podemos apresentar o algoritmo que verifica se existe uma cadeia de suporte válida para d_0 tal que a propriedade 4 seja satisfeita.

Algoritmo para Verificação de Cadeia de Suporte:
Entrada: Grafo G e delegação $d_0 = u_0v_0$.
Saída: Verdadeiro se d_0 possui uma cadeia de suporte em G.

1. Seja r_0 a raiz de G.
2. Seja G'=(V',E') o subgrafo induzido em G pelos caminhos de r_0 a u_0.
3. Faça $C \leftarrow \{r_0v_i \mid r_0v_i \in \text{E'}\}$, $W \leftarrow \text{E'}\backslash C$
4. Enquanto $W \neq \emptyset$, faça:
5. Para todo $\{uv \mid uv \in W, u \text{ in } C\}$, faça:
6. Se *PossuiSuporteParcial*(G'[C], uv), faça:
7. Se $uv = d_0$, retorne *Verdadeiro*.
8. $C \leftarrow C + \{uv\}$.
9. $W \leftarrow W$ - $\{uv\}$.
10. Retorne *Falso*.

Em cada iteração, todas as delegações do subgrafo induzido pelo conjunto C, ou seja, G[C], possuem uma cadeia de suporte válida. Uma delegação uv somente é acrescentada ao conjunto C se possui ao menos uma cadeia de suporte parcial em G[C]. Portanto, a delegação uv também possui uma cadeia de suporte válida. A complexidade deste algoritmo é $O(n^3)$, já que o algoritmo de rebaixamento é aplicado para cada aresta de G.

4.5 Verificação de Autorizações de Domínio

A verificação de uma requisição de acesso, ou seja, a verificação se um sujeito s possui uma autorização de domínio $a_d = (o, f, Q)$ deve considerar:

1. Se existe ao menos uma delegação da autorização de domínio $d = \text{del}(g_i, s, a_d)$.
2. Se existe uma delegação de uma autorização de controle $a_c = (a_d, \textit{delegate})$ à g_i, com uma cadeia de suporte válida.
3. A condição acumulada de utilização da autorização a_d é satisfeita.

Além de verificarmos se existe uma delegação da autorização com uma cadeia de suporte válida, devemos verificar se as condições de domínio existentes nesta cadeia também são satisfeitas. A propagação de condições de controle é realizada da mesma forma que para condições de domínio. Portanto, na verificação de existência de cadeia de suporte parcial, devemos verificar também se as condições de domínio são satisfeitas. Esta alteração é apresentada a seguir:

Algoritmo Modificado para Verificação de Cadeia de Suporte Parcial:
Entrada: Grafo G e delegação $d_0 = u_0v_0$.
Saída: Verdadeiro se d_0 possui uma cadeia de suporte *parcial* em G.

1. Seja r_0 a raiz de G.
2. Seja G' = (V', E') o subgrafo induzido pelos caminhos de r_0 a u_0.
3. Para todo $uv \in$ E', faça:
4. Se (Q_d é *Falso*) $\vee$ ($Q_c(d_0)$ é *Falso*), faça:
5. E' $\leftarrow$ E' - uv
6. Aplique Rebaixamento(G').
7. Se $d_0 \in$ E' Retorne *Verdadeiro*, Senão Retorne *Falso*.

Em cada delegação uv, a condição da autorização de controle é representada por Q_c, enquanto a condição de domínio é representada por Q_d. Se a delegação d_0 não satisfaz a condição Q_c, ou a condição Q_d não é atendida, uv não é utilizada como na cadeia de suporte parcial de d_0. Desta forma, garante-se a propagação tanto de condições de controle quanto de condições de domínio.

Algoritmo para Verificação de Autorizações de Domínio
Entrada: Grafo G, sujeito s, e autorização de domínio a_d .
Saída: Verdadeiro se s possui autorização válida a_d.

1. Seja $a_c = (a_d, \textit{delegate})$ a autorização de delegação de a_d.
2. Para cada delegação $d_i = \text{del}(g_i, s, a_d)$ faça:
3. Para cada delegação $d_j = \text{del}(g_j, g_i, a_c)$ faça:
4. Se $(Q_d \wedge \textit{PossuiCadeiaSuporte}(\text{G}, d_j))$ retorne *Verdadeiro*.
5. Retorne *Falso*.

O algoritmo de verificação de autorizações de domínio verifica se existe uma cadeia de suporte válida para as delegações de autorizações de domínio cuja condição seja atendida, ou seja, Q_d é verdadeiro.

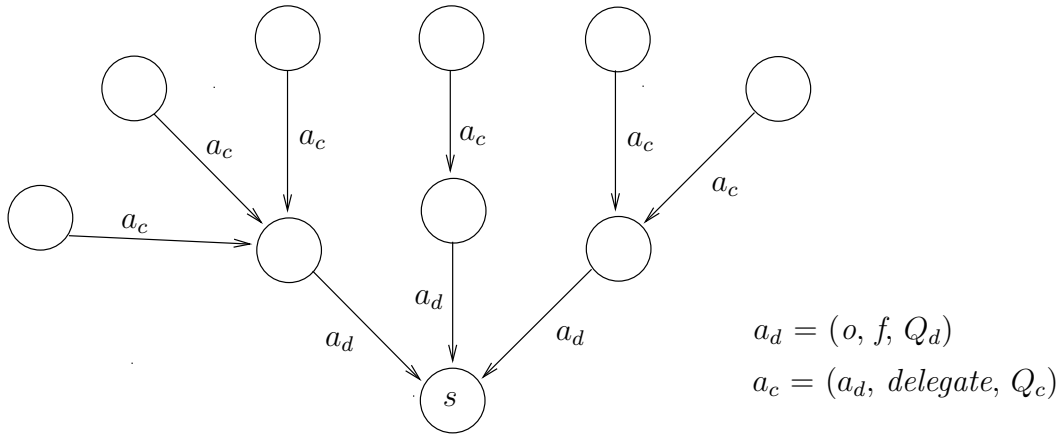


Figura 4.5: Autorizações de Controle e Domínio

Na figura 4.5 é apresentado um exemplo de grafo em que o sujeito s recebe uma autorização a_d via diferentes cadeias de delegações. As arestas representadas com rótulo a_c são delegações de autorizações de controle, ou seja, $a_c = del(a_d)^n$, enquanto a_d são as delegações de autorizações de domínio.

4.6 Revogação de Delegações Condicionais

Contrariamente ao que ocorre em um conjunto formado somente por delegações não condicionais, o efeito da revogação de uma delegação em um conjunto de delegações condicionais

depende não somente do peso desta delegação, mas também da condição por ela imposta. Se determinada delegação não pertencer a uma cadeia de suporte de outras delegações, sua revogação não terá efeito algum, a não ser sobre o próprio sujeito que a recebeu, que possivelmente perderá o direito sobre a autorização anteriormente delegada. Por outro, caso esta delegação pertença a uma cadeia de suporte, possivelmente outras delegações deverão ter seus pesos reduzidos para poderem utilizar cadeias de suporte alternativas.

Com base no algoritmo para revogação com rebaixamento anteriormente proposto, obtemos uma forma eficiente de se manter consistente o conjunto de autorizações referentes a uma determinada autorização. Neste caso, entretanto, considerou-se a validade de cadeias de delegações somente com base nos pesos máximos permitidos para cada delegação. Agora, ao introduzirmos o conceito de delegações condicionais ao modelo, para determinar a validade de uma cadeia de delegações, torna-se também necessário satisfazer as condições impostas ao longo desta cadeia, como discutido acima. A introdução do conceito de delegações condicionais tem conseqüências, portanto, no algoritmo proposto para revogação com rebaixamento, na medida em que uma cadeia de suporte válida formada por delegações não condicionais pode passar a ser inválida se considerarmos a concordância ou não das condições ao longo desta cadeia em relação à delegação em questão. O algoritmo para revogação com rebaixamento para delegações não condicionais utiliza o conceito do peso efetivo de cada aresta: $\psi(uv) = min(w(uv), \lambda(u))$

O peso efetivo foi utilizado para recalcular o peso de cada delegação levando-se em consideração o máximo peso permitido por sua cadeia de suporte. Observe que ψ não pode ser diretamente utilizado no caso de delegações condicionais, visto que $\lambda(u)$, o máximo peso acumulado em um vértice, depende da aresta de destino. A principal conseqüência desta observação é que temos que considerar que o peso a ser transferido de uma delegação à outra imediatamente adjacente pode ser menor que o peso da primeira delegação subtraído de uma unidade. Isto se deve ao fato de que duas delegações adjacentes podem possuir cadeias de suporte diferentes, mesmo que a primeira delegação pertença à cadeia de suporte da segunda. Ou seja, a cadeia que permitiu determinado peso à primeira delegação, pode não ser válida como suporte para a segunda delegação. Portanto, neste caso, é possível que a segunda delegação tenha que utilizar uma cadeia de suporte que permita um peso inferior ao que seria possível se utilizasse a mesma cadeia de suporte da primeira delegação. Ou seja, o peso da primeira delegação, digamos w, impõe restrições ao peso da segunda delegação, neste caso no máximo $w - 1$, mas não é suficiente para garantir que o peso da segunda delegação seja realmente $w - 1$.

Um exemplo desta situação é apresentado na figura 4.6. Para tornar o entendimento mais simples, foi representado somente um subgrafo do conjunto total de delegações em relação a uma determinada autorização. Considere que delegações AC, BC e CD possuem as condições Q_{ac}, Q_{bc} e Q_{cd}, respectivamente. Suponha que estas três condições são

atendidas pela delegação DE, mas somente as condições Q_{ac} e Q_{cd} são atendidas por DF. Observa-se que o máximo peso acumulado em D em relação à DE é 6, enquanto que o máximo peso acumulado neste mesmo vértice em relação à DF é 3. Isto ocorre porque a delegação DE pode utilizar a cadeia de suporte (BC, CD), enquanto que delegação DF deve utilizar a cadeia de suporte alternativa (AC, CD), visto que a condição imposta por Q_{bc} não é atendida por DF. Observe que se este mesmo subgrafo fosse formado por delegações não condicionais, o máximo peso acumulado em D seria o mesmo para DE e DF, ou seja, 6. Portanto, ao rebaixarmos a delegação AC de 5 para 3, como apresentado na figura, devemos também rebaixar a delegação DF de 3 para 1, uma vez que o máximo peso acumulado em D em relação à DF passa a ser somente 1.

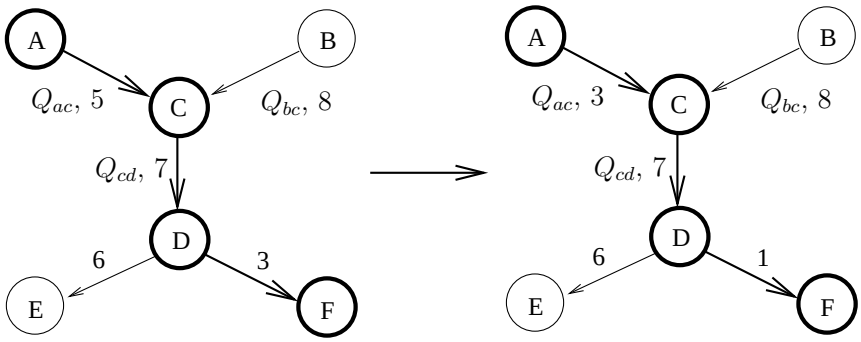


Figura 4.6: Cadeia de Suporte da Delegação DF

4.7 Algoritmo de Rebaixamento Condicional

A seguir será apresentado um algoritmo de rebaixamento para delegações condicionais, considerando as questões levantadas acima. A solução mais simples para um algoritmo de rebaixamento condicional, é utilizar a mesma idéia do algoritmo para delegações não condicionais. Entretanto, para tanto, é preciso recalcular $\lambda(u)$, necessário para o cálculo de $\psi(uv)$ no passo 2, dependendo da aresta uv em questão. Denotemos este novo valor de $\lambda(u)$, por $\lambda_{uv}(u)$, representado o máximo peso acumulado em u em relação à aresta uv. A definição de ψ pode ser reformulada como:

$$\psi'(uv) = min(w(uv),\lambda_{uv}(u))$$

Observe que λ_{uv} reflete o fato de que as arestas cuja condição não seja atendida por uv possuem um peso efetivo nulo em relação a uv. Isto é, tais arestas não podem pertencer a uma cadeia de suporte de uv. Para obtermos λ_{uv}, podemos, primeiramente, ajustar os pesos das arestas para refletir esta propriedade. Denotemos por $w_{uv}(e)$ o peso da aresta e em

relação à aresta uv. Este peso é obtido aplicando-se a seguinte regra a toda aresta e em E:

- $w_{uv}(e) = w(e)$ se $Q_e(uv)$ é verdadeiro
- $w_{uv}(e) = 0$ caso contrário.

Denotemos por G_{uv} o grafo obtido após a aplicação desta regra. Observe que, se obtermos o valor de $\lambda(u)$ neste novo grafo, este valor será justamente o valor que queremos obter, ou seja, $\lambda_{uv}(u)$. Podemos obtê-lo aplicando o algoritmo para delegações não condicionais em G_{uv}. Com este modificação, a complexidade do algoritmo também é altamente alterada, uma vez que em cada iteração do passo 2, devemos aplicar uma instância do algoritmo de rebaixamento não condicional ao subgrafo induzido pelo conjunto S_i. Assim, a complexidade deste algoritmo inicialmente proposto é da ordem de O(V^4).

Algoritmo de Rebaixamento Condicional:

1. Seja $l(u_0) = \infty$, $l(v)$ = -∞ para $v \neq u_0$, $S_0 = \{u_0\}$ e $i = 0$.
2. Para cada $v \in \overline{S_i}$, aplique *Rebaixamento* $(G_{u_iv}[S_i])$, e substitua $l(v)$ por max$\{l(v), \psi'(u_i$v) - 1$\}$. Calcule $max_{v \in \overline{S}_i}\{l(v)\}$ e seja u_{i+1} o vértice em que esse máximo é obtido. Faça $S_{i+1} = S_i \cup \{u_{i+1}\}$.
3. Se $i = v$ - 1, pare. Se $i < v$ -1, substitua i por $i + 1$ e volte para o passo 2.
4. Para cada $e \in$ E, substitua w(e) por $\psi(e)$, e se $\psi(e) < 0$, remova a aresta e.

4.8 Revogação com Delegações Contextuais

O algoritmo de revogação anteriormente apresentado considera que condições são avaliadas somente no momento em que as delegações são criadas. Isto é, se uma delegação d atende a condição Q, então isto é sempre verdadeiro a menos que Q seja alterada, adicionando-se um novo predicado. Entretanto, com a utilização de referências de contexto, esta mesma delegação d pode não mais satisfazer a condição Q, caso o contexto que levou a isto seja alterado. Portanto, ao aplicarmos o algoritmo de rebaixamento condicional, o conjunto de delegações que são revogadas, ou sofrem rebaixamento, pode variar significativamente. Podemos lidar com esta situação segundo as seguintes opções:

Adicionar a cada autorização uma condição de reavaliação Q_r: se esta condição é verdadeira em um dado momento, sabemos que a condição deve ser reavaliada,

não importando se a última avaliação resultou verdadeiro ou falso. Esta abordagem possui grande flexibilidade, porém requer que o delegador preveja quando será reavaliada a condição que impôs para que uma delegação seja aceita. Isto não é intuitivo, e pode levar a resultados não desejados, tais como negar determinado acesso mesmo quando pudesse ser concedido. Outra dificuldade desta abordagem é o aumento da complexidade dos algoritmos propostos, e conseqüente perda de desempenho.

Nunca reavaliar condições: impossibilita a vantagem principal de funções de avaliação, que é permitir o controle de acesso a recursos dependendo do contexto atual do sistema.

Sempre reavaliar condições: esta solução pode levar a uma revogação em cascata no caso de uma revogação, visto que as delegações válidas podem variar expressivamente dependendo de cada contexto. Isto pode levar a um sistema de difícil manutenção, que necessita que delegações sejam reatribuídas com uma frequência demasiadamente grande.

Reavaliar quando mudanças importantes de contexto ocorrem: no caso de um banco de dados, tal evento poderia ser por exemplo a alteração de dados em tabelas. Entretanto, tais eventos podem ocorrer com uma frequência grande, levando em último caso à mesma semântica da reavaliação contínua, discutida acima, levando aos mesmos problemas apresentados naquele caso.

Reavaliar somente quando explicitamente requisitado: o administrador do sistema poderia requisitar a reavaliação das condições existentes no sistema. Se o sistema for altamente complexo, é difícil determinar com que frequência, ou em quais situações a reavaliação deve ser solicitada.

A principal dificuldade em relação à utilização de condições dependentes de contexto, é que tais condições podem levar à revogação em cascata, qualquer que seja a abordagem adotada. Portanto, a melhor solução seria não considerar tais condições durante a execução de uma revogação, ou rebaixamento. Isto é, as cadeias de suporte são consideradas somente em relação ao peso das delegações, como definido anteriormente para o algoritmo de rebaixamento simples.

Por outro lado, durante a criação de uma nova delegação devemos considerar tais condições na formação das cadeias de suporte. Se uma delegação d possui uma cadeia válida de suporte sem considerarmos as condições desta cadeia, mas deixa de possuí-la se passarmos a considerar tais condições, esta delegação permanece temporariamente inativa. Isto é, ela não é removida do sistema, e pode ser reativada caso volte a atender as condições impostas ao longo de ao menos uma cadeia de suporte.

Esta abordagem, além de trazer grande flexibilidade, permite simplificar o processo de revogação, já que somente o algoritmo simples de rebaixamento deve ser utilizado, e não mais o algoritmo condicional. Isto é, durante uma revogação, devemos aplicar o algoritmo de rebaixamento sem considerar as condições impostas por cada delegação. Estas condições devem consideradas somente durante a aceitação de novas delegações, ou verificação da validade de autorizações, no caso de requisições de acesso. Outra vantagem desta abordagem refere-se ao fato de que o sistema pode ser mais facilmente gerenciado, já que não há o risco de revogações indesejáveis em cascata, como discutido.

4.9 Rebaixamento de Delegações com Forças Diferentes

Quando permitimos a existência de autorizações de diferentes forças em um mesmo conjunto, pode ser necessário dividir uma única delegação em uma ou mais novas delegações, de forma a manter cada sujeito com o maior conjunto possível de direitos após uma revogação. Isto é, cada sujeito deve ter revogados apenas os direitos que não possuem mais cadeias de suporte válidas. Desta forma, evita-se revogações em cascata de direitos que ainda são suportados por delegações alternativas.

No exemplo a seguir, considere que x $\sqsupset$ y. Como apresentado na figura 4.7.a, ao diminuir o peso da delegação AC, de $del(x)^7$ para $del(x)^5$, a delegação CD também deve ser alterada, isto é, sofrer rebaixamento, de forma a manter o grafo consistente. Se nós simplesmente diminuirmos o peso de CD de $del(x)^7$ para $del(x)^4$, a delegação DF $del(y)^4$ não será mais suportada, apesar de poder ainda ser suportada pela cadeia de suporte iniciada em BC. Analogamente, se diminuirmos CD de $del(x)^6$ para $del(y)^5$, a delegação DE $del(x)^3$ não será mais suportada, mesmo se diminuirmos seu peso também. Portanto a delegação CD deve ser dividida em duas delegações, $del(x)^4$ e $del(y)^5$, mantendo assim as delegações DE e DF ainda suportadas.

O algoritmo de rebaixamento simples, anteriormente apresentado neste documento, pode ser utilizado somente para aplicar o rebaixamento em um grafo composto por delegações de mesma força. Suponha que o grafo seja composto por um conjunto parcialmente ordenado de delegações $del(R_i)$, tal que $R_i \sqsupseteq R_{i+1}$, $\forall$ i e j. As delegações são agrupadas em uma lista de conjuntos S_i, tal que cada conjunto é composto somente por delegações com a mesma força. Por exemplo, delegações $del(x)^4$ e $del(x)^5$ pertencem ao mesmo conjunto, enquanto que as delegações $del(x)^5$ e $del(y)^5$ pertencem a diferentes conjuntos. Note que, de acordo com este procedimento, para quaisquer duas delegações r e s, r $\sqsupset$ s se r $\subseteq S_i$, e s $\subseteq S_k$, e $i < k$. Denotemos por $Right(S_i)$ o direito sendo delegado pelas delegações no conjunto S_i.

Como uma delegação pode ser suportada somente por delegações que são mais fortes que ela mesma, podemos afirmar que cada delegação é suportada pelo subgrafo induzido

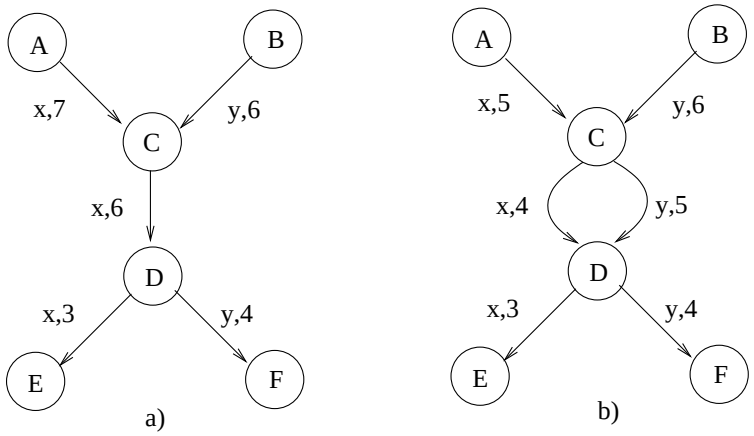


Figura 4.7: Rebaixamento da Delegação AC

por estas delegações. Denotemos por G_n o subgrafo induzido pela união das delegações nos conjuntos S_i, $\forall$ i $\leq$ n. Por exemplo, G_2 = G[S_1] $\cup$ G[S_2]. Similarmente, podemos afirmar que as delegações em um determinado conjunto S_i são suportadas pelo subgrafo G_{i-1}. Como citado acima, o algoritmo de rebaixamento apresentado anteriormente, pode ser utilizado para aplicar o rebaixamento em um grafo composto de delegações de um mesmo direito, portanto este pode ser utilizado para aplicar o rebaixamento nos subgrafos induzidos por cada conjunto S_i. Em adição, este também pode ser utilizado para aplicar o rebaixamento em cada subgrafo G_i, se considerarmos cada subgrafo G_i como composto por delegações do mesmo direito.

A principal idéia do algoritmo a seguir é progressivamente aplicar o rebaixamento em um grafo, começando com o grupo de delegações detendo o direito mais forte (S_0), e terminando com as delegações mais fracas (S_n). Este procedimento pode ser realizado devido ao fato que cada conjunto S_i não pode afetar, isto é suportar, nenhuma delegação mais forte no subgrafo G_{i-1}.

Em cada passo, as delegações no subgrafo G_{i-1} são copiadas para o subgrafo G_i, substituindo o direito em cada delegação no grafo G_{i-1} pelo direito sendo delegado no conjunto S_i, ou $Right(S_i)$, como definido. Por definição, note que del(u) $\cup$ del(v) $\equiv$ del(u), se $u \sqsupseteq$ v. Portanto, este procedimento não confere mais direitos aos sujeitos, do que eles já possuíam, uma vez que $Right(S_i) \sqsupset Right(S_{i-1})$, por construção.

O grafo final é a união das delegações obtidas em cada subgrafo G_i. Entretanto, para evitar delegações desnecessárias, todas delegações comparáveis entre os mesmos usuários são substituídas pela mais forte dentre elas. Assim, delegações são divididas somente se estritamente necessário.

Algoritmo de Rebaixamento Composto

1. Faça $G_0 = \emptyset$.
2. Para $i = 1$ a n, copie as delegações em G_{i-1} para G'_{i-1}, substituindo todos direitos por $Right(S_i)$. Faça $G_i = G'_{i-1} \cup G[S_i]$, e aplique o algoritmo de rebaixamento em G_i.
3. Faça $G = \bigcup_{i=1}^{n} G_i$. Substitua todas delegações comparáveis entre os mesmos usuários pela mais forte dentre elas.

As iterações necessárias para aplicar o algoritmo ao grafo 4.7.a são apresentadas na figura 4.8. Como existem somente dois direitos distintos, somente duas iterações são necessárias. No grafo final, as delegações $del(x)^5$ e $del(y)^5$ são substituídas pela mais forte, $del(x)^5$. O mesmo acontece para $del(x)^3$ e $del(y)^3$. Como as delegações $del(x)^4$ e $del(y)^5$ não podem ser comparadas, ambas são mantidas no resultado (figura 4.8.c).

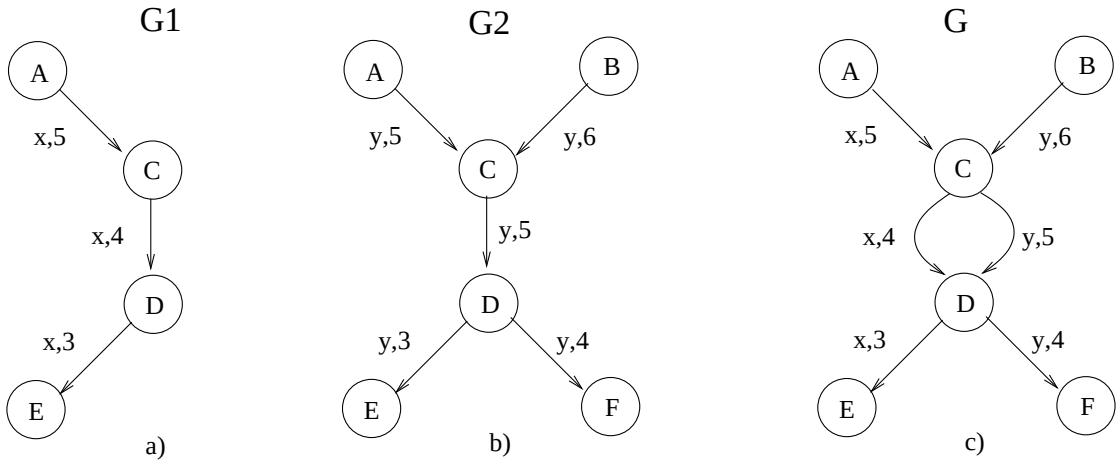


Figura 4.8: Passos para Rebaixamento de AB

4.10 Autorizações para Revogação

Na maioria dos modelos convencionais de autorização, normalmente derivados do modelo *Griffths-Wade* [Griffiths and Wade 1976], um sujeito pode revogar diretamente somente suas próprias delegações. Entretanto, este sujeito somente pode revogar delegações realizadas por outros sujeitos se estas revogações forem realizadas de forma indireta, através do mecanismo de revogação em cascata. Isto é, ao revogar uma delegação, todas as delegações dependentes também são revogadas.

Entretanto, em algumas situações é necessário que um sujeito revogue diretamente determinada delegação mesmo que esta delegação seja concedida por outro sujeito. Porém, a revogação de autorizações deve ser realizada somente por sujeitos autorizados, de forma similar à delegação de autorizações.

Em nosso modelo, assim como a autorização para criação de delegações, a autorização para revogação é uma autorização de controle, pois diz respeito ao funcionamento interno do mecanismo de autorização. A autorização para revogação de uma autorização a pode ser representada como $a_{rv} = (a, \mathit{revoke}, Q)$.

Definição 11: Um sujeito R é autorizado a revogar a delegação del(A, B, a) sse:

- R = A, ou seja, R é o delegador da delegação AB, ou
- R possui uma autorização válida $a_{rv} = (a, \mathit{revoke}, Q)$.

O sujeito R deve receber a autorização a_{rv} através de uma cadeia de delegações, assim como é feito para autorizações de domínio. A verificação se um sujeito possui a autorização válida a_{rv} deve ser realizada através do mesmo algoritmo proposto anteriormente para autorizações de domínio (ver seção *Verificação de Autorizações de Domínio*). Ou seja, devemos verificar tanto se existe uma cadeia de suporte válida para a autorização, quanto se são obedecidas as condições para utilização desta autorização.

Este mecanismo permite grande controle e flexibilidade sobre o mecanismo de revogação. Por exemplo, assim como definimos a autorização de revogação de uma autorização de domínio a, podemos definir a autorização de revogação de uma autorização de delegação:

$a_{rv} = (a, \mathit{revoke}, Q)$: revogar a delegação da autorização a.
$a_{rd} = ((a, \mathit{delegate}, Q), \mathit{revoke}, Q)$: revogar a delegação da autorização de delegação de a.

A autorização de delegação de uma autorização de revogação é representada por $((a, \mathit{revoke}, Q), \mathit{delegate}, Q)$. O recebimento de uma autorização a não está condicionada ao recebimento da autorização de revogação de a. Portanto, podemos controlar quais sujeitos poderão realizar revogações no sistema, independentemente dos direitos de delegação possuídos por estes sujeitos.

Finalmente, por se tratar de uma autorização como qualquer outra, a autorização de revogação também pode sofrer restrições através da sua condição. Desta forma, por exemplo, podemos conceder a determinado administrador o direito de revogar delegações que atendam determinados critérios específicos da aplicação que utiliza o mecanismo de autorização.

4.11 Conclusão

Neste capítulo, primeiramente, foi apresentado um mecanismo que permite definir condições com base em informações de contexto, conteúdo, ou tempo, através da utilização de delegações condicionais. Este mecanismo permite grande flexibilidade, uma vez que as condições podem ser definidas com base em informações externas ao próprio mecanismo de autorização, de acordo com as necessidades de uma aplicação específica.

A avaliação de cadeias de suporte considera tanto os pesos das delegações, quanto as condições impostas ao longo das cadeias de delegações. As condições impostas por uma delegação são automaticamente propagadas às delegações dependentes, evitando possíveis falhas de segurança, e ao mesmo tempo facilitando o gerenciamento do sistema. Conforme discutido, cada delegação que pertence a uma cadeia de suporte deve possui uma cadeia de suporte parcial que permita que esta delegação seja utilizada.

O algoritmo para aceitação de novas delegações verifica se a delegação a ser criada possui ao menos uma cadeia de suporte válida, com base nas condições das autorizações de controle e nos pesos das delegações das cadeias de suporte. O algoritmo para verificação de autorizações de domínio verifica tanto se as condições impostas pelas autorizações de controle quanto pelas autorizações de domínio são válidas. Isto é, as delegações das cadeias de suporte devem ser válidas, e as condições para a utilização da autorização de domínio devem ser satisfeitas.

O algoritmo de rebaixamento inicialmente proposto foi estendido para considerar delegações com diferentes forças na composição de cadeias de suporte. Este algoritmo possui a propriedade de manter os sujeitos com o maior conjunto de direitos suportados após a execução de revogações. Foi apresentado um algoritmo de rebaixamento que considera as condições impostas pelas delegações além dos seus pesos. Foram apresentadas as alternativas e implicações da utilização deste mecanismo de revogação em especial.

Finalmente, foi discutido como o modelo proposto permite controlar também a revogação de delegações, uma vez que a revogação de delegações também é controlada através de autorizações condicionais de controle.

A partir desta base construída, no próprio capítulo iremos tratar em particular a questão da definição de delegações contextuais, que são delegações cujas condições são baseadas em informações de contexto do sistema.

Capítulo 5

Delegações Contextuais

Uma condição Q foi definida como uma função $Q(d, t, S)$, onde d representa uma delegação, t representa o instante atual do sistema, e S representa o conjunto de informações de contexto do sistema. Uma delegação contextual é definida como uma delegação del(g, r, a), com $a = (o, f, Q)$, onde Q é uma função de S. Ou seja, uma delegação contextual é uma subclasse de delegações condicionais.

Neste capítulo serão apresentados os principais trabalhos relacionados ao controle de acesso baseado em contexto e ao controle de acesso baseado em conteúdo. Em seguida, serão discutidos exemplos de utilização de delegações contextuais na implementação destes mecanismos.

5.1 Controle de Acesso Baseado em Conteúdo

O Controle de acesso baseado em conteúdo visa limitar o acesso a recursos através de seus conteúdos, ou atributos. Por exemplo, utilizando este tipo de controle de acesso, um paciente poderia restringir o acesso a seus registros baseando-se no tipo de doença, ou tratamento, que estes descrevem. Desta forma, uma enfermeira poderia ter acesso a testes rotineiros de sangue, enquanto não poderia ter este acesso a testes de sangue para detecção de *HIV*. Sob o ponto de vista conceitual, sistemas de anti-vírus podem ser considerados um tipo de controle de acesso baseado em conteúdo, na medida em que permitem o acesso somente se o arquivo, ou recurso, a ser acessado for livre de código potencialmente danoso ao usuário.

Freqüentemente, a granularidade relativa ao controle de acesso dos sistemas atuais não é suficiente para satisfazer os requisitos de uma aplicação específica. Como exemplo, os sistemas DBMS atuais de uso comercial não fornecem funcionalidades para se restringir a acesso de usuários a somente um subconjunto das linhas de uma tabela específica. Este necessidade é normalmente implementada através de visões, o que não garante que o

acesso à tabela original seja devidamente controlado.

Igualmente, em sistemas específicos, normalmente é necessário definir critérios de acesso dependentes do conteúdo do recurso a ser acessado. Como exemplo, em um hospital um médico normalmente deve ser autorizado a acessar somente os dados relativos a seus próprios pacientes. Este caso em específico representa um caso de restrição positiva, ou *binding of duties*.

Os exemplos referem-se a instâncias específicas de determinados tipos de objetos. É comum, especialmente em sistemas que suportam o mecanismo RBAC, a atribuição de autorizações em relação aos objetos pertencentes a determinadas classes de objetos. Como exemplo, um médico possui autorização para ler prontuários. Neste caso, prontuário representa uma classe específica de objetos. Entretanto, note que neste mesmo exemplo, o médico não deve ter autorização para ler todos os prontuários, mas sim um subconjunto específico de prontuários que digam respeito aos pacientes atendidos por este médico específico, ou que sejam, como exemplo, referentes aos atendimentos realizados em determinado setor do hospital.

Este tipo de restrição também é caracterizado como um tipo específico de *binding of duties*, como no exemplo anterior. Normalmente estas restrições são conseguidas através de regras, agrupadas na forma de uma política de acesso. Um determinado conjunto de regras aplica-se a uma classe de objetos, permitindo-se que restrições sejam especificadas. Embora a utilização de regras seja uma forma normalmente empregada para impor restrições de acesso, este mecanismo pode ser de difícil gerenciamento, especialmente se o conjunto de regras, e igualmente exceções a estas regras, é muito grande.

Outra abordagem para se limitar o acesso a um subconjunto de objetos de uma determinada classe é a enumeração manual destes objetos específicos. Este tipo de abordagem é comumente utilizado em conjunto com o mecanismo RBAC. Neste caso, enquanto o papel é responsável por associar autorizações genéricas de acesso a determinadas classes de objeto do sistema, as listas de objetos obtidas através de enumeração são utilizadas para indicar quais instâncias destas classes de objetos podem ser acessadas pelo usuário específico associado ao papel. Esta solução não é suportada pelo modelo RBAC, portanto deve ser reforçada pela aplicação em que os acessos devem checados. Ou seja, além de trabalhosa, esta solução diminui a vantagem intrínseca do modelo RBAC, que é a sua capacidade de facilitar o gerenciamento de autorizações e usuários do sistema.

O agrupamento de instâncias é uma abordagem também utilizada, sendo mais eficiente que a anterior em relação ao trabalho de gerenciamento. Neste caso, as instâncias não são associadas diretamente aos usuários do sistema, mas sim indiretamente através de grupos de objetos. O critério para se incluir uma instância em um grupo pode ser arbitrário, sendo normalmente realizado manualmente, ou de forma dependente da aplicação, fora do controle do sistema de autorização. Os grupos podem ainda ser computados em tempo

de execução, através de caracteres coringa. Nesta direção, o modelo conhecido como Generalized RBAC [Covington et al. 2000] inclui o conceito de papéis de objetos (*object roles*), que são basicamente grupos de instâncias computados com base em determinadas propriedades dos objetos.

Também no domínio do modelo RBAC, o mecanismo proposto em [Kumar et al. 2002] utiliza o conceito de contexto de papel (*role context*), que é uma construção que permite a aplicabilidade das autorizações de um papel a um subconjunto de instâncias. À cada contexto de papel, é associada uma restrição booleana, chamada de *context filter*, que é avaliada em tempo de execução com base nos atributos do usuário que requisitou o acesso, e nos atributos do objeto específico requisitado. Assim pode-se restringir o conjunto de instâncias a que as autorizações do papel são aplicáveis.

O modelo proposto em [Giuri and Iglio 1997] utiliza a idéia de *templates* de papéis, bem com permissões parametrizadas. Neste modelo, é possível especificar políticas em que o acesso depende do conteúdo do objeto acessado. Isto é conseguido através de restrições, que podem referenciar atributos do objeto acessado. No momento da verificação, os valores reais dos atributos do objeto acessado são substituídos nas variáveis pertencentes às autorizações parametrizadas. Através deste mecanismo, é possível, por exemplo, restringir o acesso de um médico aos registros dos seus próprios pacientes.

5.2 Controle de Acesso Baseado em Contexto

O controle de acesso baseado em contexto é muito similar ao baseado em conteúdo, na medida em que ambos permitem definir condições para o acesso a determinados recursos. Entretanto, o controle baseado em contexto pode refletir condições dinâmicas que independem dos atributos dos objetos acessados. Com a crescente preocupação em relação à segurança e controle de acesso é comum a necessidade de suporte à decisões complexas de acesso, que dependem de parâmetros dinâmicos, baseados no contexto atual do acesso, ou do sistema como um todo.

Algumas aplicações, ou até mesmo sistemas de auto-atendimento, podem disponibilizar determinadas funcionalidades dependendo do tipo de autenticação, e correspondente grau de confiança, utilizado por determinado usuário. Entre os tipos de autenticação normalmente utilizados estão a autenticação por retina, íris, impressão digital, e senha, correspondendo, respectivamente, a graus decrescentes de confiança. Em um sistema, o administrador pode definir uma regra tal como: O acesso a dados laboratoriais deve ser permitido somente se o nível de confiança da autenticação é maior ou igual ao da autenticação por retina. Este tipo de restrição refere-se ao contexto de autenticação do usuário, e faz sentido quando o sistema pode ser acessado através de diferentes mecanismos de autenticação.

Também é comum a utilização de autorizações que são válidas em determinadas situações que dependem do estado geral do sistema, e não das características de um acesso específico. Por exemplo, um médico pode ter autorização para acessar o prontuário de pacientes que se encontram em uma situação de emergência , ou analogamente, caso ocorra um caso de calamidade pública. Neste último caso, o contexto se refere ao sistema como um todo, portanto independe das características do paciente, ou médico, em questão.

Mecanismos convencionais de controle de acesso não são adequados para aplicações móveis, já que tais mecanismos não consideram informações de contexto. Em um ambiente distribuído, os usuários podem acessar recursos, tais como serviços ou sensores, através de dispositivos móveis. Assim, estes usuários têm a liberdade de mudar sua localização freqüentemente. Como resultado, o contexto do usuário (localização, recursos do sistema, estado ou configuração de segurança da rede, etc) pode ser alterado na mesma freqüência. Permitir o acesso ao sistema sem considerar informações de contexto pode comprometer a segurança, já que os direitos do usuário dependem não somente de sua identidade, mas também do estado deste usuário, bem como do ambiente computacional ao seu redor. Mesmo um usuário com credenciais válidas de autenticação pode comprometer o sistema, já que diferentes contextos podem requerer diferentes contextos de segurança e autorização.

Em computação móvel é cada vez mais comum que o acesso a determinados recursos seja baseado em informações sobre a localização do usuário em potencial. Um possível cenário é o de usuários de linhas ferroviárias, que acessam anonimamente serviços online através de redes *Wireless* dispostas ao longo destas linhas. Neste caso, é necessário garantir que somente os usuários de trens tenham acesso ao serviço. Note ainda que o acesso é completamente realizado com base em informações contextuais, não necessitando da identificação destes usuários. Este cenário, e as possíveis dificuldades em relação à necessidade de se determinar a veracidade de informações de localização, são discutidos em [Hulsebosch et al. 2005].

5.3 Aplicando Restrições MAC ao Modelo

A política de segurança MAC (*Mandatory Access Control*) impõe restrições de acesso com base na classificação de sujeitos e objetos no sistema segundo classes de acesso, visando impedir fluxos indevidos de informações, tanto de forma direta quanto indireta [Samarati and di Vimercati 2001].

As classes de acesso existentes são ordenadas segundo uma relação de dominância. Uma classe de acesso pode ser simplesmente um rótulo, ou pode corresponder a um nível de segurança associado a um conjunto de categorias. Os níveis de segurança são normalmente representados por TS (ultra secreto), S (secreto), C (confidencial) e U (sem

classificação), possuindo a relação de dominância TS > S > C > U. As categorias não possuem uma relação de força, e representam divisões lógicas ou funcionais dentro de uma organização. Quando uma classe de acesso é composta por um nível de segurança associado a categorias de acesso, uma classe de acesso c_1 domina outra classe de acesso c_2 se e somente se o nível de segurança de c_1 é maior ou igual ao nível de segurança de c_2, e as categorias de c_1 incluem as de c_2.

O nível de segurança de uma classe de acesso associada a um objeto reflete o grau de confidencialidade da informação contida neste objeto, e a importância relativa de se manter esta informação sigilosa. O nível de segurança associado a um usuário, também denominado de *Clearance*, por sua vez, reflete o grau de confiança de que este sujeito não irá revelar informações sigilosas a usuários não autorizados.

Segundo o modelo *Bell LaPadula* [Bell and LaPadula 1976] um sistema é composto de um conjunto de sujeitos S, objetos O, e ações A, que incluem *read* e *write*. O modelo também assume que exista um conjunto L de classes de acesso e uma função λ: $S \cup O \rightarrow L$, que quando aplicada a um sujeito ou objeto retorna a sua classificação neste estado. Visando manter a confidencialidade de informações, este modelo propõe os seguintes princípios:

1 - Proibida leitura acima (No-read-up): Um sujeito s possui acesso de leitura (*read*) a determinado objeto o somente se $\lambda(s) \geq \lambda(o)$.

2 - Proibida escrita abaixo (No-write-down): Um sujeito s possui acesso de escrita (*write*) somente se $\lambda(o) \geq \lambda(s)$.

A concordância com estes dois princípios previne fluxos de informação de sujeitos e objetos em direção a outros sujeitos e objetos pertencentes a classes de acesso com níveis de segurança inferiores. Com isso, assegura-se que nenhum sujeito terá acesso indevido a informações que não sejam permitidas para seu nível de segurança, mesmo que de forma indireta.

De uma maneira geral, pode-se dizer que as políticas DAC e MAC não são mutuamente exclusivas, pois podem ser utilizadas em conjunto. Neste caso, para que o acesso a determinado objeto seja permitido, é necessário tanto que as condições de confidencialidade impostas pelo mecanismo MAC sejam satisfeitas, quanto que o sujeito possua autorização para o acesso, segundo o modelo discricionário. Ou seja, em tal situação a política DAC atua dentro dos limites impostos pela política MAC, uma vez que a política DAC pode restringir somente os acessos que são permitidos pela política MAC.

Em nosso modelo, os dois princípios de confidencialidade, acima citados, podem ser garantidos através da utilização de delegações condicionais. Para tanto, é necessário:

- Definir uma função avaliadora, como discutido anteriormente no capítulo referente a delegações condicionais, que indique o nível de segurança de um sujeito. Vamos representá-la por *Executor.Clearance*.

- De forma similar, definir uma função avaliadora que indique o nível de segurança de um objeto. Vamos representá-la por *Object.AccessClass*.

O princípio 1 do modelo *Bell LaPadula* é reforçado pela condição:

$Q_r = (Executor.Clearance \geq Object.AccessClass)$

O princípio 2, por sua vez, é reforçado pela condição:

$Q_w = (Object.AccessClass \geq Executor.Clearance)$

Enquanto a condição Q_r deve ser associada a autorizações de leitura (o, $read$, Q_r), a condição Q_w deve ser associada a autorizações de escrita, ou seja, (o, $write$, Q_w). Assim como a função λ do modelo *Bell LaPadula* reflete mudanças de estado, as funções avaliadoras também dependem do estado do sistema, refletindo possíveis alterações em relação às classes de acesso de sujeitos e objetos do sistema.

Como discutido anteriormente, a inclusão de uma condição nova em uma delegação é automaticamente propagada às delegações dependentes desta delegação. Portanto, não é necessário repetir a condição acima para cada delegação no sistema, basta que a condição seja imposta pela raiz da cadeia de delegações de cada objeto que possua restrições de confidencialidade.

5.4 Representação da Política *Chinese Wall*

A política *Chinese Wall* [Brewer and Nash 1989] foi criada com a finalidade de introduzir controles similares ao da política MAC em sistemas que utilizam a política DAC. Procurou-se definir um mecanismo para prevenir fluxos de informação que poderiam causar conflitos de interesse quando uma única pessoa possui informações sobre corporações concorrentes, ou conflitantes. A idéia desta política é similar ao conceito de separação de deveres (*separation of duties*), que é o impedimento de que um mesmo indivíduo adquira direitos conflitantes sobre um objeto, tal como criar e aprovar uma mesma ordem de compra. Entretanto, a política *Chinese Wall* procura restringir o acesso a informações conflitantes, em lugar de direitos. Esta política também difere da política MAC na medida em que o acesso aos dados não é restringido com base em classificações, tais como

confidencial ou secreto, mas com base nas informações que o indivíduo já acessou. Este modelo é baseado em uma organização hierárquica dos objetos:

- *Objetos básicos* são itens individuais de informação relacionados à uma única corporação.
- *Conjuntos de dados de empresas* definem grupos de *objetos básicos* relacionados à uma mesma corporação.
- *Classes de conflito de interesse* definem *conjuntos de dados de empresas* que são relacionados a corporações competidoras.

A política *Chinese Wall* restringe o acesso aos dados segundo as seguintes propriedades:

Propriedade Simples: Um sujeito s pode ter acesso a um objeto o somente se este objeto:

- a) pertence a um dos conjuntos de dados referentes aos objetos já acessados por s;
- b) pertence a um conjunto de dados que está em uma classe de conflito cujos conjuntos de dados nunca foram acessados por s.

Propriedade Geral: um acesso de escrita é permitido somente se:

- a) o acesso é permitido pelo regra simples;
- b) não pode ser lido nenhum objeto que pertença a um conjunto de dados diferente daquele do objeto para o qual está sendo solicitada permissão de escrita.

Enquanto a regra simples impede fluxos indesejáveis que podem ocorrem com relação à um único usuário, a propriedade geral impede fluxos que podem ocorrer com a interação entre dois ou mais usuários.

Um aspecto importante a ser notado é o fato de que a aplicação da política *Chinese Wall* requer um mecanismo de histórico de acessos, visto que é necessário verificar a quais conjuntos de dados pertencem os objetos já acessados por cada sujeito do sistema.

Por simplicidade, podemos considerar em nosso modelo que a verificação se um sujeito s possui uma autorização $a = (o, f, Q)$ como a prova de que s de fato acessou o objeto o. Neste caso, um registro (usuário, autorização) = (s, a) é automaticamente armazenado no histórico de acessos do mecanismo de autorização a partir de cada verificação de acesso. Com base no mecanismo de histórico, devemos definir as seguintes funções avaliadoras:

1. Função que verifica no histórico de acessos todos os conjuntos de dados já acessados por um sujeito: *DataSets(s)*

2. Função representando o conjunto de dados a que um objeto pertence: *Object.DataSet(o)*

3. Função que retorna as classes de conflito a que um ou mais conjuntos de dados pertencem: *conflictClasses(DataSets)*

4. Função que retorna todos os conjuntos de dados a que um sujeito s possui acesso, isto é, os conjuntos de dados dos objetos para os quais s possui ao menos uma autorização na forma (o, *read*, Q), para qualquer Q: *readableDataSets(s)*

A propriedade simples é assegurada pelas condições:

$Q_{1a} = DataSet(Object) \in DataSets(Executor)$

$Q_{1b} = conflictClasses(DataSet(Object)) \cap conflictClasses(DataSets(Executor)) \neq \varnothing$

A propriedade geral é assegurada pela condição:

$Q_{2b} = (Q_{1a} \wedge Q_{1b}) \wedge (readableDataSets(Executor) \subseteq DataSet(Object))$

5.5 Reforçando Restrições com Funções Avaliadoras

Existem dois tipos principais de restrições implementadas por mecanismos de controle de acesso. O primeiro tipo, denominado ligação de deveres (*binding of duties*), representa uma restrição positiva. Restrições negativas, por sua vez, são denominadas de separação de deveres (*separation of duties*). Enquanto o primeiro tipo visa garantir que um sujeito desempenhando determinada função possua as autorizações necessárias relativas às atividades que desempenha, o segundo tipo procura impedir acessos que levem a conflitos de interesse sob o ponto de vista global de uma organização.

Por exemplo, observe que um médico deve ter permissão para dar alta aos pacientes que ele mesmo tratou. Não estamos interessados em proibir que este médico dê alta aos pacientes que não tratou, mas estamos interessados em garantir que ele possua as autorizações necessárias para realizar as tarefas relativas aos pacientes que tratou. Portanto, este é um exemplo de ligação de deveres.

Em uma empresa é comum a necessidade de se impedir que uma mesma pessoa realize determinada atividade, e ao mesmo tempo, realize a função de regulação sobre esta atividade. Por exemplo, uma pessoa que faz ordens de compra deveria ser impedida de aprovar ordens de compra. Este caso representa um tipo estático de separação de deveres,

já que estamos tratando das atividades fazer e aprovar, sem considerar ordens de compra específicas.

Outra empresa pode ser mais permissiva, impedindo somente que uma pessoa aprove as ordens que ela mesma criou. Neste caso, trata-se de uma separação dinâmica de deveres, já que se considera uma ordem de compra específica, e portanto não se pode, à priori, determinar se o acesso deve ser autorizado ou não. Através de referências de contexto que avaliem regras específicas de negócio, é possível reforçar restrições. Considere a seguinte restrição:

- *Um médico m pode dar alta a um paciente p se m admitiu p.*

Neste caso, é necessário definir através de uma função avaliadora o relacionamento *admitiu*, existente entre médico e paciente: *admitiu(médico, paciente)*. A autorização de conceder alta é representada por $a = (*, concederAlta, Q)$, onde:

$Q =$ *admitiu(Executor, Object)*

Note que ao verificarmos a autorização a, *Executor* é o médico, e *Object* é o paciente a ser concedida alta. Considere outro exemplo de restrição:

- *Um funcionário s pode aprovar uma ordem de compra o somente se s não criou o.*

É necessário definir o relacionamento *criou*, existente entre funcionário e ordem de compra: *criou(funcionário, ordem de compra)*. A autorização é representada por $a = (*, aprovar, Q)$, onde:

$Q = \neg$ *criou(Executor, Object)*

Note que ao verificarmos a autorização a, *Executor* é o funcionário, e *Object* é a ordem de compra. Nestes dois exemplos, estamos concedendo autorizações sobre todos os objetos. Ou seja, em lugar de restringirmos as autorizações a um objeto específico, concedemos autorizações sobre todos os objetos, com a restrição que a condição Q seja atendida. Porém, no momento de uma verificação de acesso, somente um único objeto é referenciado.

Também, observe que no segundo caso não estamos proibindo que o funcionário aprove uma ordem de compra caso ele não a tenha criado, mas estamos concedendo a autorização para aprovar qualquer ordem de compra que ele não tenha criado. Isto é, se outro sujeito diferente conceder uma autorização para que A aprove mesmo as ordens de compra que

ele tenha criado, A poderá aprová-las apesar de nossa restrição. Podemos evitar que isto ocorra através da criação de uma proibição explícita de que A aprove ordens de compra que ele mesmo tenha criado. Assim, mesmo que outro sujeito conceda a A o direito de aprovar qualquer ordem de compra, A não poderá fazê-lo devido à proibição. A utilização de proibições será discutida a seguir no capítulo 6.

5.6 Conclusão

Delegações cujo controle é baseado em informações de contexto do sistema têm uma ampla gama de aplicações, tais como sistemas na área de saúde, aplicações móveis, e outras aplicações de uso específico. Como citado, existem diversas abordagens para a definição do controle de acesso baseado em contexto, principalmente em relação ao modelo RBAC, com base em definição de regras que visam controlar a aplicações de autorizações de um papel com base em informações de contexto. O controle de acesso baseado em conteúdo possui igual importância, tendo também abordagens similares à do controle de acesso baseado em contexto.

Como discutido, o modelo proposto neste documento permite a implementação de restrições de contexto e conteúdo, com base em um mesmo mecanismo. A vantagem principalmente em relação a outros modelos de delegação ou autorização é a possibilidade de controlar individualmente determinadas delegações, com base no mecanismo já definido para delegações condicionais.

Neste capítulo foi apresentado como o mecanismo anteriormente proposto de delegações condicionais pode ser utilizado para reforçar restrições de separação ou ligação de deveres. Foi discutido como é possível implementar controles mandatários, que podem ser utilizados para fornecer um grau de controle adicional sobre o sistema. Finalmente, através da definição de um histórico de acessos, o modelo permite também a implementação da política *Chinesse Wall*, que caracteriza-se por controlar o acesso a recursos de forma a evitar fluxos indesejados de informações.

Capítulo 6

Autorizações Negativas

Proibições, ou autorizações negativas, como são normalmente denominadas na literatura, provêem um mecanismo para impedir que sujeitos exerçam direitos sobre objetos do sistema. Este mecanismo é especialmente importante em sistemas descentralizados, onde outros sujeitos, além do próprio criador de determinado objeto, podem delegar autorizações sobre este objeto. Nestes casos, sem o emprego de autorizações negativas, um sujeito poderia adquirir autorizações sobre este objeto mesmo que o seu criador não o queira.

Em nosso modelo, autorizações negativas não são estritamente necessárias, visto que podemos impedir que um sujeito tenha acesso a determinado objeto, através somente da utilização de delegações condicionais. Por exemplo, se o criador do objeto o quer impedir que o usuário s tenha acesso a este objeto, o criador do objeto o pode incluir em cada delegação realizada a outros sujeitos a condição que os receptores das delegações realizadas por estes sujeitos sejam diferentes de s. Uma vez que as condições são propagadas em futuras delegações, s nunca receberá uma delegação que permita o acesso ao objeto o. Entretanto, se o número de sujeitos que devem ser impedidos de acessar um objeto for demasiadamente grande, esta solução pode levar a um número grande de condições a serem checadas durante a criação de cada nova delegação, além de tornar a administração do sistema demasiadamente complexa. Portanto, o conceito de autorizações negativas deve também ser considerado em nosso modelo, como uma forma auxiliar de limitar o acesso a objetos.

A introdução de autorizações negativas leva a alterações nos algoritmos de aceitação e revogação de delegações, visto que devemos considerar a existência de autorizações negativas direcionadas a sujeitos pertencentes às cadeias de suporte. Além disso, deve ser estabelecido um modelo formal que permita definir políticas para resolução de conflitos, quando determinado sujeito possuir autorizações positivas e negativas em relação ao mesmo objeto.

6.1 Utilização de Autorizações Negativas

Embora o modelo de delegações proposto neste documento já permita controlar de forma precisa a atribuição de autorizações, o uso de autorizações negativas pode ser utilizado como um mecanismo estático que impeça definitivamente que um sujeito adquira autorizações sobre determinado objeto. O uso de delegações condicionais, ao contrário, é mais adequado para restringir o acesso a determinado objeto com base em características de contexto ou conteúdo que podem sofrer alterações com o decorrer do tempo, sendo portanto um mecanismo inerentemente dinâmico.

Normalmente, o emprego de autorizações negativas é especialmente útil para a definição de exceções, quando autorizações são concedidas a grupos de sujeitos, tornando o gerenciamento do sistema mais simples. Como exemplo, imagine que queiramos delegar uma autorização a todos os membros de grupo específico, com exceção de um membro s. Sem o emprego de autorizações negativas, teríamos que especificar autorizações a cada membro deste grupo, com exceção de s. Entretanto, o mesmo resultado pode ser obtido se atribuirmos uma única autorização positiva ao grupo, e uma autorização negativa ao membro s, impedindo assim que este adquira a autorização destinada ao seu grupo.

De uma maneira geral, as diferentes abordagens, ou políticas, em relação ao controle de acesso podem ser classificadas em:

- Políticas fechadas: permitem o acesso a um determinado objeto ou direito somente se existir uma autorização explícita que permita isto;
- Políticas abertas: negam o acesso se existir uma proibição explícita, e o permitem caso contrário;

Políticas abertas são normalmente utilizadas em aplicações que não requerem um controle de acesso muito restrito, seja porque os dados ou ações considerados não são especialmente críticos, ou em razão de comodidade na administração de usuário e suas respectivas permissões. As políticas fechadas, por outro lado, consideram que o acesso deve ser normalmente negado. Este tipo de política oferece maior proteção ao sistema, na medida em que um número menor de usuários tende a possuir direitos, ao mesmo tempo em que tais direitos são geralmente delegados segundo critérios ou papéis específicos.

É comum a existência de sistemas que utilizem não somente uma única política aberta ou fechada, mas sim uma composição de tais políticas. Isto é, dependendo do contexto considerado, tal como um conjunto específico de objetos ou direitos, utiliza-se uma política fechada ou, se for mais conveniente, uma política aberta. Em razão de eficiência na administração, além da composição de políticas, é possível a utilização de uma política flexível, que não torne o acesso tão direto como no caso de políticas abertas, mas que também não torne a administração demasiadamente trabalhosa, o que normalmente

ocorre em políticas fechadas. Ou seja, é necessário que políticas flexíveis permitam que se especifique permissões para um conjunto grande de usuários e objetos, mas que ao mesmo tempo suportem exceções considerando determinados conjuntos especialmente críticos de objetos ou usuários, que devem ter acesso finamente controlado. Este tipo de idéia é normalmente utilizando em conjunto com a idéia de agrupamento de usuários, de acordo com um critério específico, tal como departamento, projeto, função, etc. Neste caso, atribui-se uma permissão para o conjunto desejado, e em seguida proibe-se o acesso para determinados usuários que não devem, por alguma razão, ter acesso ao objeto sendo considerado, e vice-versa.

A idéia de proibir, ou negar, o acesso a determinados objetos é normalmente relacionada ao conceito de autorizações negativas. Estas são comumente utilizadas tanto para representar uma exceção, como no caso do agrupamento de usuários citado acima, mas também podem ser utilizadas para proibir o acesso de um usuário a determinado recurso crítico, mesmo que este usuário não esteja inserido em grupo maior de usuários que possua tal acesso.

Embora o conceito de autorizações negativas facilite a administração com o suporte a exceções, sua utilização leva à questão de como tratar os casos em que tanto autorizações negativas quanto positivas são atribuídas a um mesmo sujeito e objeto. Ou seja, é necessária a definição de políticas de resolução de conflitos, que considerem os vários casos em que tal situação pode ocorrer. Em sistemas que permitem a atribuição de autorizações tanto a grupo, papéis e usuários, é normalmente empregada a política que diz que a autorização mais específica é a que deve prevalecer. Como exemplo, em uma empresa, imagine que um usuário trabalhe em um projeto, tal que todos os empregados possuem acesso a determinados documentos. Entretanto, por razões específicas, tal usuário específico foi explicitamente proibido de acessar estes documentos. Neste caso, a autorização mais específica, ou seja, a proibição, é a que deve prevalecer.

Este é o exemplo mas simples de conflito que normalmente se encontra. Entretanto, são comuns os casos em que um usuário específico pertence a grupos disjuntos de sujeitos, com autorizações muitas vezes conflitantes. Além disso, tornando a definição de políticas de resolução de conflitos ainda mais difícil, há casos em a autorização mais específica não deve prevalecer. Baseando-se no mesmo exemplo, considere que, por se tratar de uma decisão estratégica, definiu-se que o acesso a determinados documentos deve ser proibido a funcionários que trabalhem em determinado projeto, sem exceções. Se utilizarmos a política do mais específico prevalece, é claro que a proibição não será obedecida, visto que uma autorização atribuída a um usuário específico irá prevalecer sobre tal proibição.

Como solução a este tipo de conflito, uma possível abordagem utiliza a idéia de atribuir prioridades às autorizações presentes no sistema, enquanto outra abordagem se caracteriza simplesmente definir um conjunto de autorizações fortes que nunca deve ser sobrescrito

[Rabitti et al. 1991]. Enquanto a primeira abordagem, caracterizada por definir prioridades, pode resultar em um sistema de difícil gerenciamento, a segunda solução exige que as todas autorizações sejam consistentes, ou seja, que não haja conflitos entre autorizações fortes. Entretanto isto nem sempre é possível, dependendo da complexidade inerente ao sistema.

Existem ainda outras abordagens para resolução de conflitos, ainda com a intenção de definir prioridades entre autorizações, tal como se basear em uma ordem de listagem arbitrária, estabelecida pelo administrador do sistema [Shen and Dewan 1992], ou ainda definir as prioridades de acordo com a ordem temporal de criação das autorizações.

A abordagem análoga, que consiste em atribuir prioridades aos sujeitos do sistema também é utilizada. Neste tipo de abordagem, para a resolução de conflitos considera-se a prioridade relativa dos sujeitos que concederam autorizações conflitantes. Por exemplo, segundo esta política, as autorizações delegadas por um coordenador de projetos em uma empresa poderiam ser sobrescritas por um sujeito com prioridade maior, tal como um gerente, mas não por funcionário comum. Entretanto esta política não é suficiente nos seguintes casos:

- Um dos sujeitos, ou possivelmente os dois sujeitos, não pertence a grupos ou papéis, e portanto os sujeitos não podem ser comparados segundo este critério.
- Os sujeitos pertencem a grupos e papéis que não podem ser comparados por pertencerem a hierarquias disjuntas.
- Os sujeitos pertencem exatamente aos mesmos papéis e grupos, e portanto possuem a mesma prioridade.

Visto que esta política nem sempre resulta em um vencedor e, além disso, o modelo de delegação proposto neste documento apresenta caráter principalmente discricionário, não iremos optar por estabelecer prioridades com base em papéis ou grupos.

É importante notar que não existe uma solução perfeita, que seja adequada para todos os casos. Sempre existirão situações que não haviam sido previstas no momento em que o modelo foi formulado, ou que simplesmente não são suportadas por se tratarem de casos muito específicos. Entretanto, é possível utilizar uma composição de políticas de resolução de conflitos nos casos em que uma única política não é suficiente, visto que políticas de resolução de conflitos não são necessariamente exclusivas. Isto é, pode-se utilizar uma primeira política, e caso a resolução do conflito não seja possível com esta política, aplica-se uma segunda política ao caso considerado.

Recentes trabalhos nesta direção [Al-Kahtani and Sandhu 2004], [Jajodia et al. 2001], [Siewe et al. 2003] procuram tornar flexível a escolha da política adotada, tanto com relação à decisão de acesso em si, quanto em relação à resolução de conflitos. Geralmente,

isto é conseguido através da utilização de regras que se aplicam a determinados objetos ou sujeitos no sistema. Este abordagem, embora torne o sistema extremamente flexível, também torna a sua administração complexa. Da mesma forma que podem existem conflitos entre autorizações negativas ou positivas, podem haver conflitos entre tais regras. Além disso, pelo fato destas regras se aplicarem a cada tomada de decisão, esta solução pode não ser aplicável, dependendo da complexidade do sistema em termos de número de regras, objetos e usuários. Uma vez que não está no escopo deste trabalho permitir tal flexibilidade, iremos adotar uma política única de resolução de conflitos que seja adequada ao contexto de delegações entre sujeitos, sem considerar informações hierárquicas de grupos e papéis, ou regras flexíveis como nas abordagens citadas.

6.2 Algoritmo para Resolução de Conflitos

Antes de apresentar a política adotada, devemos definir quando ocorre uma condição de conflito. Primeiramente devemos estender a definição de autorizações para incluir o sinal da autorização: negativo ou positivo. Uma autorização passa a ser representada por uma quádrupla $(o,\ f,\ Q,\ t)$, onde $t \in \{+,-\}$ indica uma autorização positiva ou negativa, respectivamente.

A delegação de uma autorização negativa, representada da mesma forma que para autorizações positivas por $\mathrm{del}(g,\ r,\ -a)$, significa que o sujeito g está impedindo que o sujeito r usufrua da autorização a, caso r venha a receber esta autorização de um terceiro sujeito. Por definição $a \sqsupseteq -a$. Portanto, um sujeito que possui direito de delegar a autorização a, possui direito de atribuir uma proibição, através da delegação de uma autorização negativa. Portanto, a relação de força entre delegações pode ser estendida como a seguir:

- $\mathrm{del}(g,\ r,\ del(a)^n) \sqsupseteq \mathrm{del}(g,\ r,\ -del(a)),\ \forall\ n.$
- $\mathrm{del}(g,\ r,\ -del(a)) \sqsupseteq \mathrm{del}(g,\ r,\ -del(a'))$ se $a \sqsupseteq a'$

Dizemos que uma autorização r $= (o_r,\ f_r,\ Q_r,\ t_r)$ é comparável a uma autorização s $= (o_s,\ f_s,\ Q_s,\ t_s)$ sse $o_r \sqsupseteq o_s$ e $f_r \sqsupseteq f_s$ ou $o_s \sqsupseteq o_r$ e $f_s \sqsupseteq f_r$.

Definição 12: Duas delegações são conflitantes se as duas autorizações delegadas são comparáveis, e se seus sinais são divergentes.

Finalmente, dizemos que dois sujeitos S_0 e S_1 estão em conflito sobre o sujeito S_2, se S_2 recebeu delegações conflitantes de S_0 e S_1.

Como exposto acima, quando é possível a atribuição de autorizações negativas a grupos e, simultaneamente a sujeitos individuais, são possíveis várias políticas para resolução de conflitos. Entretanto, considerando o modelo até agora proposto neste documento, que visa prover mecanismos de delegação somente entre sujeitos individuais do sistema, torna-se necessário selecionar uma política de resolução de conflitos adequada a este contexto. Isto é, estamos mais preocupados em definir políticas para resolução de conflitos entre sujeitos, e não entre sujeitos e grupos. Com o fim de adotar uma política adequada para resolução de conflitos, devemos estabelecer uma ordem de prioridade entre os sujeitos existentes no sistema, tal como citado anteriormente. Primeiramente, deve ser observado que uma autorização delegada por um sujeito inicial S_0 não pode ser sobrescrita por um sujeito S_1, se a autorização conflitante delegada pelo sujeito S_1 tiver sido recebida indiretamente através do sujeito S_0. Esta situação é representada na figura 6.1.

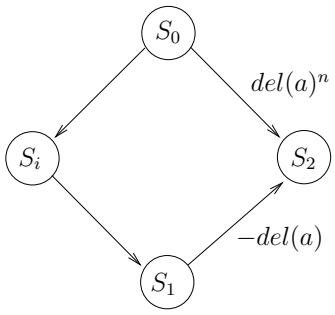


Figura 6.1: Conflito entre S_0 e S_1 sobre S_2

Observe que a proibição imposta por S_1 a S_2, não pode sobrescrever a autorização delegada por S_0 a S_2, visto que S_1 recebeu a autorização também de S_0 em primeiro lugar. Assim, chegamos à primeira regra da política que iremos adotar:

Regra 1: Seja G[o,f] o subgrafo induzido pelas delegações da autorização de execução da operação f sobre o objeto o. Dados dois sujeitos quaisquer S_0 e S_1, se S_0 pertencer a todas as cadeias de suporte do sujeito S_1, então S_0 tem prioridade sobre o sujeito S_1.

Se S_0 pertence a todas as cadeias de suporte de S_1, dizemos que S_1 é subordinado à S_0. Observe que é necessário que S_0 pertença a todas as cadeias de suporte de S_1 para podermos afirmar que S_1 recebeu sua autorização, seja direta ou indiretamente, através de S_0. Não é suficiente exigir que exista ao menos um caminho de S_0 a S_1, ou seja, que S_0 pertença a ao menos uma cadeia de suporte de S_1. Basear a relação de prioridade nesta única condição, o que é feito em modelos similares ao apresentado em [Ruan and Varadharajan 2004], seria uma falha de segurança. Como exemplo, suponha que dois sujeitos S_j e S_k deleguem autorizações conflitantes a um mesmo sujeito. Para

que, por exemplo, S_j adquira artificialmente maior prioridade sobre S_k, e portanto ganhe a disputa, bastaria que S_j delegasse a autorização conflitante a S_k. Assim, S_j obteria maior prioridade sobre S_k, embora S_k já possuísse a autorização antes desta delegação realizada por S_j. Solucionado o caso mais simples através da *Regra 1*, em que a autorização de um sujeito depende do outro sujeito, é necessário agora considerar o caso que os sujeitos conflitantes possuem cadeias de suporte independentes.

Observe que intuitivamente um sujeito com maior "poder de delegação"deve possuir maior prioridade em casos de conflitos. Lembrando, uma autorização de controle pode ser representada, de forma abreviada, por $del(a)^n$, onde n é o peso da delegações e a é a autorização a ser delegada. Como definido anteriormente, a relação de força entre duas delegações é definida como:

- del(g, r, $del(a)^n$) $\sqsupseteq$ del(g, r, $del(a)^k$) se n $> k$
- del(g, r, $del(a)^n$) $\sqsupseteq$ del(g, r, $del(a')^n$) se $a \sqsupseteq a'$

Seja c a autorização conflitante recebida via delegação pelo sujeito S. Seja D_{S_i} o conjunto de delegações recebidas pelo sujeito S_i, e seja $max_c\{D_{S_i}\}$ a delegação mais forte neste conjunto cuja autorização é comparável à autorização c.

Definição 13: Um sujeito S_0 possui maior poder de delegação que um sujeito S_1, com relação a uma autorização c, sse $max_c\{D_{S_0}\} \sqsupset max_c\{D_{S_1}\}$.

Observe que se S_1 é subordinado à S_0 então obrigatoriamente S_0 possui maior poder de delegação que S_1.

Prova: Suponha que S_1 seja subordinado à S_0, e mesmo assim S_1 possua maior poder de delegação que S_0. Por definição, se S_1 é subordinado à S_0, então todas as cadeias de suporte de S_1 passam por S_0. Seja G_0S_0 a delegação de maior peso recebida por S_0. Por outro lado, se S_1 possui maior poder delegação que S_0, então existe uma cadeia de suporte válida $<d_0, d_1, ..., G_1S_1>$, tal que $w(G_1S_1) > max_c\{D_{S_0}\} = w(G_0S_0)$. Mas como S_0 pertence à todas as cadeias de suporte válidas de S_1, então deve existir uma cadeia de suporte válida $<d_0, d_1, ..., G_0S_0, ..., G_1S_1>$. Entretanto, isto contraria a definição de cadeia de suporte válida, uma vez que $w(G_1S_1) > w(G_0S_0)$, por hipótese.

A figura 6.2 apresenta outro exemplo de conflito. Neste caso, os dois sujeitos conflitantes, S_0 e S_1, não são dependentes um em relação ao outro. Visto que $max_c\{D_{S_0}\} = del(a)^9 \sqsupset max_c\{D_{S_1}\} = del(a)^8$, S_0 possui maior prioridade na resolução do conflito sobre S_2.

Desta forma, podemos reformular a *Regra 1*, tornando-a mais geral:

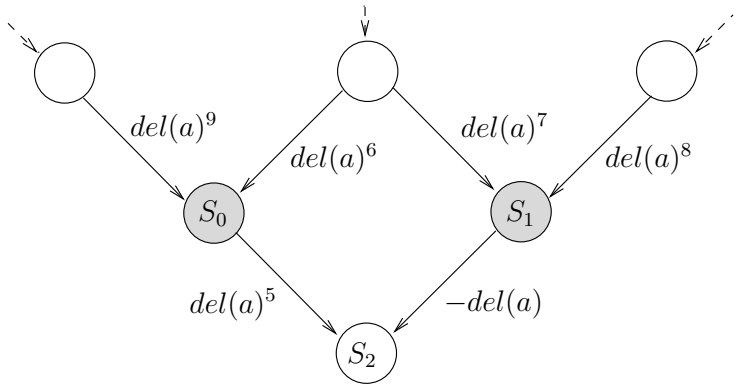


Figura 6.2: Conflito entre S_0 e S_1 Independentes

Regra 1 (Geral): Dizemos um sujeito S_1 possui maior prioridade que outro sujeito S_2 sobre um sujeito S, com relação à autorização c, se S_1 possui maior poder de delegação que S_2.

A definição de poder de delegação é útil na resolução de conflitos porque reflete a idéia intuitiva de que um sujeito que recebeu determinado direito diretamente, ou via uma cadeia de delegações curta, possui maior prioridade, ou confiança, que outro sujeito que recebeu esta mesma autorização via uma cadeia longa de delegações. Este tipo de comparação seria de difícil obtenção em modelos convencionais de autorização, mas é facilmente obtido no nosso modelo através do peso das delegações.

Agora devemos prover uma regra que resolva os conflitos que ocorrem quando os dois sujeitos conflitantes pertencem a cadeias disjuntas de delegações, e possuem o mesmo poder de delegação, pois neste caso as duas regras já definidas não são suficientes para resolver o conflito.

Uma das primeiras abordagens para resolução de conflitos [Bertino et al. 1993] é a abordagem de atribuir prioridade às autorizações negativas (conhecida como *denials take precedence*). Esta abordagem sempre leva à resolução do conflito, e é adequada ao modelo de delegações essencialmente discricionário proposto, pois independe de informações de papéis ou grupos. Além disto, esta abordagem reforça o princípio do menor privilégio, especialmente adequado à sistemas que utilizam políticas fechadas de decisão de acesso.

Regra 2: Uma autorização negativa possui maior prioridade que uma autorização positiva comparável à esta.

Se não for possível definir prioridade entre os sujeitos conflitantes, a regra 2 deve ser utilizada para resolver o conflito. Assim chegamos à política final para resolução de conflitos, composta por duas regras que devem ser aplicadas na ordem definida a seguir:

Regra 1: S_0 possui maior prioridade que S_1 se S_0 possui maior poder de delegação que S_1.

Regra 2: Uma autorização negativa possui prioridade sobre uma delegação positiva comparável.

6.3 Bloqueio de Delegações

Quando uma autorização negativa possui maior prioridade sobre a autorização positiva, a autorização negativa permanece *ativa*, e a autorização positiva permanece *bloqueada*. Isto é, a positiva não é revogada; simplesmente não pode ser utilizada pelo sujeito que a recebeu através de delegações. O inverso ocorre se a autorização positiva possui maior prioridade: neste caso a autorização positiva permanece ativa, enquanto a autorização negativa permanece bloqueada.

Como exposto anteriormente neste documento, existem dois tipos de autorizações: autorizações de domínio e de controle. Enquanto as autorizações de domínio conferem direitos sobre os objetos de um sistema específico, as autorizações de controle conferem direitos de delegação e revogação aos sujeitos deste sistema.

Um sujeito somente pode delegar uma determinada autorização se possuir o respectivo direito para delegar esta autorização. Representamos este direito de delegação como $del(a)^n$. Neste cenário, existem duas situações principais que podem ocorrem devido à utilização de autorizações negativas:

- o sujeito possui através de delegação uma autorização de domínio a, e recebe a autorização negativa correspondente.
- o sujeito possui uma autorização de controle $del(a)^n$, e recebe a autorização negativa correspondente $-del(a)$.

Vamos assumir que após a aplicação da política para resolução de conflitos a autorização negativa foi ativada em ambos casos. Em relação ao primeiro caso, simplesmente a autorização de domínio a permanece bloqueada. Esta autorização não tem efeito sobre outras autorizações existentes no estado de autorização atual, ou seja, não possui efeitos em relação ao conjunto de autorizações correntemente ativas no sistema. Outra alternativa seria revogar a autorização a. Entretanto, esta solução limita o poder de utilização de autorizações negativas, na medida em que impossibilita que estas sejam utilizadas como uma forma temporária de se retirar direitos de sujeitos no sistema. É desejável, além de tornar o sistema mais facilmente administrável, que a atribuição de uma autorização negativa, e posterior revogação desta mesma autorização, leve novamente o sistema ao estado de autorização inicial. Caso contrário, poderia ser necessário atribuir novamente autorizações com uma frequência demasiadamente grande. Outro problema associado à solução de revogar a autorização a é que isto significaria que um sujeito, através da

atribuição de uma autorização negativa, poderia revogar autorização positivas de outros sujeitos. Isto deve ser evitado, pois teríamos que formular uma política que definisse quando um sujeito possui direito de revogar autorizações de outro sujeito, o que tornaria o modelo demasiadamente complexo.

O segundo caso exige uma análise mais detalhada por se tratar de uma autorização de controle, pois esta pode ter efeito sobre outras autorizações, de domínio ou de controle, no sistema. Suponha que a autorização bloqueada $del(a)^n$ tenha sido utilizada para criação de uma delegação $d = \text{del}(g, r, a)$, que pertença a cadeias de suporte de outras delegações. Precisamos decidir o efeito do bloqueio desta autorização com relação a:

- a autorização de domínio a, delegada a r
- outras delegações cujas cadeias de suporte passem por d (delegações dependentes).
- outras delegações cujas cadeias de suporte passem por d, mas tenham cadeias de suporte alternativas.
- as autorizações de domínio delegadas pelas delegações cujas cadeias de suporte passem por d.

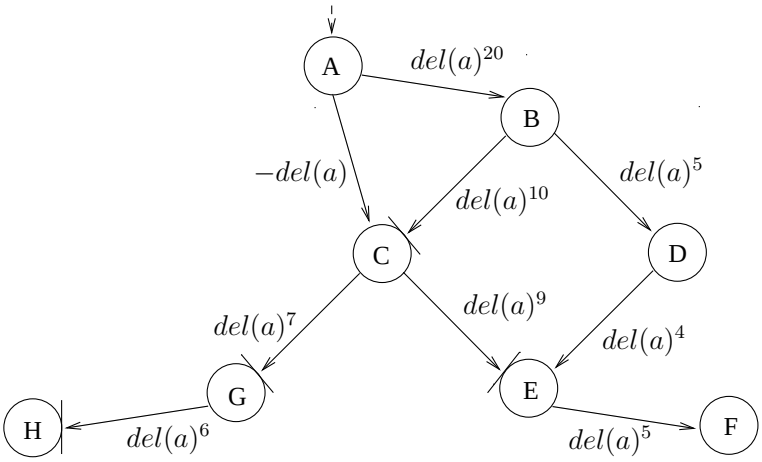


Figura 6.3: Bloqueio de Delegações Dependentes

Primeiramente, se um sujeito tem seu direito de delegação bloqueado, é desejável que as autorizações por ele delegadas tornem-se igualmente bloqueadas. Uma outra alternativa seria revogar estas autorizações, uma vez que estas não podem mais ser utilizadas. Entretanto, decidimos deixá-las no sistema, na forma de autorizações bloqueadas, pois desta forma estas autorizações podem ser reativadas novamente se a delegação da autorização negativa que causou o bloqueio for revogada. Este tipo de situação representa a perda temporária de direitos por parte do sujeito que recebeu a autorização negativa.

Imaginando um cenário em que isto seja factível, imagine um funcionário com acesso a dados confidenciais que deva ser impedido de delegar direitos de acesso a estes dados caso esteja de férias. Uma forma de desativar os direitos dos funcionários que tenham recebido autorizações deste funcionário seria atribuindo autorizações de controle negativas, que desativariam temporariamente os direitos delegados por este funcionário.

Com relação às delegações que são dependentes de d, ou seja, que possuem todas as cadeias de suporte passando por d, temos as seguintes opções:

1. Revogar as delegações dependentes.
2. Não tomar ação alguma.
3. Bloquear as delegações dependentes.

Este caso é apresentado na figura 6.3, através da delegação GH, que é dependente da delegação CG, bloqueada devido à autorização negativa atribuída em AC.

A primeira opção, revogá-las, apresenta o mesmo problema discutido para delegações de domínio. Não seria possível desativar temporariamente certas delegações, além de eventualmente levar a efeitos de revogação em cascata de difícil previsão, ou controle, pelo administrador do sistema.

A segunda opção, não tomar ação alguma, apresenta o mesmo problema discutido para delegações de domínio, ou seja, não permite que delegações sejam desativadas, diminuindo assim o potencial de utilização de autorizações negativas, mais especificamente, autorizações de controle negativas que controlam o direito de delegação de sujeitos.

Portanto, a terceira opção, pelos mesmos motivos apresentados para autorizações de domínio, é a mais adequada à situação apresentada. Bloquear as delegações dependentes impede que direitos bloqueados sejam utilizados na construção de novas cadeias de suporte, ao mesmo tempo em que permite um mecanismo de revogação temporária de direitos.

Continuando no próximo cenário, se as delegações apresentam cadeias de suporte alternativas às cadeias que passam pela delegação bloqueada, temos as mesmas opções apresentadas para delegações dependentes. Este caso é também apresentado na figura 6.3, através da delegação EF, que possui a delegação bloqueada CE em uma de suas duas cadeia de suporte: $<AB, BC, CE>$ e $<AB, BD, DE>$.

A mesma opção adotada para delegações dependentes parece, à princípio, ser a melhor opção para o caso de delegações não dependentes: simplesmente bloqueá-las. Porém, note que não é desejável bloquear certa delegação se esta pode ser ainda suportada, sem sofrer rebaixamento, por outra delegação ativa no sistema. Considerando isto, poderíamos adotar a solução de bloquear somente as delegações que não possuam cadeias de suporte alternativas que as possibilitem manter o peso atual. Entretanto, observe que mesmo as

delegações que não podem ser suportadas por cadeias alternativas podem ser utilizadas para criação de novas delegações, supondo que exista uma cadeia de suporte válida para as novas delegações.

Assim, há a opção de aplicar rebaixamento nestas delegações, de forma que os pesos de novas delegações não sejam influenciados pela delegação bloqueada. Esta opção, entretanto, pode levar ao mesmo problema da revogação direta, pois caso não existam cadeias de suporte alternativas para as delegações, estas serão revogadas. Portanto, a melhor solução, neste caso, é aplicar o algoritmo de rebaixamento somente para determinar as delegações desbloqueadas do sistema. Caso somente parte do peso de uma delegação seja suportado por cadeias de suporte ativas, esta delegação deve ser dividida em duas delegações: uma ativa, refletindo os direitos correntemente ativos, e uma bloqueada, refletindo os direitos totais no sistema, estejam estes bloqueados ou não.

Finalmente, com relação às autorizações de domínio originadas de delegações bloqueadas, também é desejável que estas sejam bloqueadas, já que autorizações originadas de delegações bloqueadas diretamente devem ser bloqueadas, como discutido anteriormente. Não há motivos para se diferenciar os dois casos: bloqueio direto de uma delegação via autorização negativa de controle, ou bloqueio de forma indireta, via bloqueio de sua cadeia de suporte.

Definidas condições para ativação ou bloqueio de autorizações, devemos agora considerar o efeito de delegações bloqueadas no algoritmos de aceitação e revogação com rebaixamento, considerando as questões levantadas.

Com relação à aceitação de delegações, devemos considerar considerar somente as autorizações ativas como pertencentes à cadeias de suporte válidas. Isto é, se uma autorização de controle positiva está bloqueada, devido à uma autorização negativa, esta autorização positiva não deve ser considerada na criação de novas delegações. Observe que, pelo fato de possíveis cadeias de suporte estarem bloqueadas, é necessário verificar o máximo peso disponível para uma nova delegação com base nas cadeias alternativas de suporte que se encontram ativas.

Definição 14: O estado de autorização é o conjunto de autorizações positivas e negativas existentes no sistema. Denotaremos este conjunto por AS.

Definição 15: O estado ativo é o conjunto de autorizações positivas ativas no sistema, denotado por *Ativo(AS)*. Este conjunto é obtido a partir de AS, após a aplicação da política de resolução de conflitos composta pelas regras definidas anteriormente.

Definição 16: O estado válido, denotado por *Válido(AS)* é o conjunto de autorizações positivas ativas no sistema com cadeias de suporte válidas. Este conjunto é obtido

a partir de *Ativo(AS)*, após a aplicação do algoritmo de rebaixamento.

Como exemplo, o estado válido do grafo da figura 6.3 é apresentado na figura 6.4.

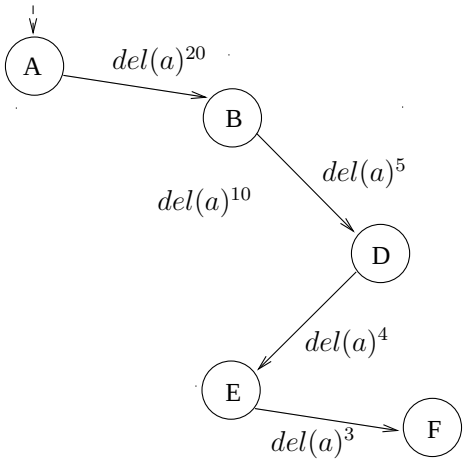


Figura 6.4: Delegações Ativas da Figura 6.3

A aplicação do algoritmo de rebaixamento no estado de autorizações ativas é necessário para impedir que delegações bloqueadas, e seus respectivos pesos, sejam utilizadas como cadeias de suporte para a nova delegação a ser criada. Portanto, o algoritmo de aceitação de delegações deve ser aplicado no conjunto *Válido(AS)*.

Algoritmo para cálculo de **Válido(*AS*)**:

1. Seja $l(u_0) = \infty$, $l(v)$ = -∞ para $v \neq u_0$, $S_0 = \{u_0\}$ e $i = 0$.
2. Para cada $v \in \overline{S_i}$:
3. Substitua $l(v)$ por $max\{l(v), \psi(u_i\text{v})$ - 1$\}$, onde u_iv é positiva e não bloqueada.
4. Calcule $max_{v \in \overline{S}_i}\{l(v)\}$ e seja $u_m u_{i+1}$ a aresta em que esse máximo é obtido.
5. Se existe u_k tal que $max_{u_{i+1}}\{D_{u_k}\} > max_{u_{i+1}}\{D_{u_m}\}$ e $u_k u_{i+1}$ é negativa:
6. Bloqueie toda aresta $u_{i+1}v$.
7. Faça $S_{i+1} = S_i \cup \{u_{i+1}\}$.
8. Se $i = v$ - 1, pare. Se $i < v$ -1, substitua i por $i + 1$ e volte para o passo 2.
9. Para cada $e \in$ E, substitua w(e) por $\psi(e)$, e se $\psi(e) < 0$, remova a aresta e.

Este algoritmo é uma adaptação ao algoritmo de rebaixamento simples. À medida que um novo vértice é acrescentado ao conjunto S_i, contendo os vértices com maiores pesos acumulados até o passo i, é aplicada a política de resolução de conflitos neste vértice. Assim, garante-se que somente delegações ativas sejam utilizadas na construção de cadeias de suporte. No exemplo da figura 6.5, observe que o conflito entre C e D deve ser solucionado somente após a conflito entre A e B ser solucionado, já que a validade da delegação CE depende do resultado deste conflito.

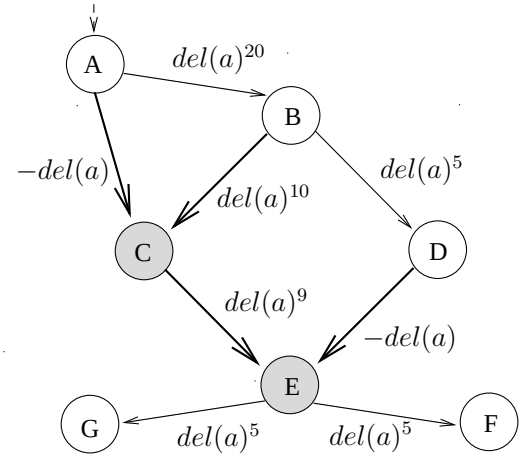


Figura 6.5: Conflitos Dependentes

Este algoritmo garante que as seguintes propriedades sejam respeitadas:

1. Cada conflito é solucionado, antes que as delegações resultantes sejam utilizadas em cadeias de suporte, ou na resolução de outros conflitos: Cada conjunto S_i possui os vértices com maiores pesos acumulados até o passo i. Segundo a política de resolução de conflitos proposta, o vértice com maior peso acumulado possui prioridade no caso de um conflito. Portanto, o vértice u que possui maior prioridade em um conflito sobre u_{i+1} pertence necessariamente ao conjunto S_i.

2. Delegações bloqueadas não são utilizadas em cadeias de suporte: para o cálculo do maior peso acumulado somente arestas positivas e não bloqueadas são utilizadas (condição no passo 3).

3. Todas delegações possuem uma cadeia de suporte válida, isto é, sem autorizações bloqueadas, e com pesos suportados: por construção, o peso acumulado de um vértice é o maior peso suportado das arestas que chegam a este vértice.

Com relação ao algoritmo de rebaixamento, note que mesmo as delegações bloqueadas devem ser atualizadas com relação aos seus pesos, ou condições, visto que estas delegações podem vir a se tornarem ativas novamente, caso a autorização negativa de controle

causando cada bloqueio seja removida. Portanto, durante a execução de uma revogação, devemos aplicar o algoritmo de rebaixamento no conjunto *AS*, ou seja, tanto nas delegações ativas quanto bloqueadas. Por outro lado, tanto a aceitação de novas delegações, quanto a verificação de requisições de acesso, devem ser aplicadas no conjunto *Válido(AS)*, refletindo assim somente as delegações ativas e com cadeias de suporte válidas.

6.4 Conclusão

Neste capítulo foi apresentado um mecanismo para definição de proibições, o que torna possível impedir que determinado sujeito faça uso de autorizações já recebidas, ou que poderão ser recebidas no futuro. A utilização de proibições, ou seja, autorizações negativas, é também importante para bloquear temporariamente os direitos recebidos por um ou mais sujeitos que tenham recebido uma autorização via uma cadeia de delegações. Como discutido, o bloqueio de uma delegação é automaticamente propagado, visto que delegações bloqueadas não podem ser utilizadas na composição de cadeias de suporte de novas delegações ou delegações já existentes.

Foi apresentado um mecanismo para resolução de conflitos quando um mesmo sujeito recebe autorizações negativas e positivas concomitantemente. Este mecanismo considera os poderes de delegação de cada um dos sujeitos ativos envolvidos no conflito. Caso não seja possível solucionar o conflito, a política *denials take precedence* é utilizada.

A definição de autorizações negativas será utilizada no próximo capítulo, onde será possível definir relações de dependência entre delegações de autorizações positivas e negativas, trazendo ao modelo proposto neste documento um grau adicional de controle sobre delegações.

Capítulo 7

Delegações Temporais

Uma extensão especialmente útil de delegações condicionais é possibilidade de definir delegações temporais. Uma delegação temporal é uma delegação cuja condição seja dependente de t, ou seja, do instante atual do sistema. Uma delegação que depende da existência de outras delegações também é considerada uma delegação temporal, na medida em que é válida enquanto as outras delegações das quais depende permanecerem válidas. Neste capítulo, iremos definir um mecanismo para utilização de delegações temporais, através da utilização de delegações condicionais.

7.1 Principais Conceitos

Um dos primeiros trabalhos relacionados à area de autorizações temporais foi proposto em [Bertino et al. 1996]. Neste modelo, uma expressão temporal é associada à cada autorização, indicando os instantes em que esta autorização é válida. Uma expressão temporal é formada por uma expressão periódica, tal como por exemplo *9am to 1pm on WorkingDays*, e um intervalo temporal limitando o escopo da expressão periódica definida.

Expressões periódicas são uma representação de conjuntos infinitos de intervalos periódicos. A definição destas expressões é baseada na noção de calendários [Niezette and Stevenne 1992]. Um calendário é definido como um conjunto contável de intervalos consecutivos, onde cada intervalo é numerado por um inteiro, denominado de índice do calendário. *Horas*, *Dias*, *Semanas*, *Meses* e *Anos* são exemplos de calendários básicos.

Calendários podem ser combinados para representar conjuntos mais complexos de intervalos periódicos, não necessariamente contíguos. Estes conjuntos são representados através de uma expressão periódica definida como $\mathrm{P} = \sum_{i=1}^{n} O_i.C_i \rhd r.C_d$, onde $O_1 = all$, $O_i \in 2^{\mathbb{N}} \cup \{all\}$ e $C_i \sqsubseteq C_{i-1}$ para $i = 2, ..., n$, $C_d < C_n$ e $r \in \mathbb{N}$.

O símbolo $\rhd$ separa a primeira parte da expressão, identificando o conjunto de pontos

iniciais, da especificação da duração de cada intervalo em termos do calendário C_d. Por exemplo, *all.Years* + 3,7.*Months* $\triangleright$ 2.*Months* representa o conjunto de intervalos começando no mesmo instante em que o terceiro e o sétimo mês de cada ano, e tendo uma duração de dois meses. O_i é omitido quando seu valor é *all*, e é representado por seu único valor se possui somente um elemento. O termo $r.C_d$ é omitido quando é igual a $1.C_n$.

O conjunto de intervalos de tempo correspondente a expressões periódicas é representado formalmente pela função Π(P). Por exemplo, para a expressão periódica P = *Weeks + {2,...,6}.Days + 10.Hours* $\triangleright$ *4.Hours*, Π(P) é o conjunto de intervalos de tempo com uma duração de 4 horas, começando com a décima hora (9am à 10am) do segundo, terceiro, quarto, quinto e sexto dia de cada semana.

A autorização periódica ([*begin*, *end*], P, (s, o, m, pn, g)) representa que a autorização especificada é válida em cada instante em Π(P) que seja maior ou igual ao instante denotado por *begin*, e menor ou igual ao instante denotado por *end*, se *end* for diferente de ∞. Por exemplo, ([1/1/94, ∞], *Mondays*, 6 (s_1, o_1, *read*, 1, s_2)), especificada pelo sujeito s_2, estabelece que o sujeito s_1 possui a autorização para ler o_1 toda segunda-feira a partir de 1/1/94.

Observe que a possibilidade de se expressar tanto autorizações negativas, isto é, proibições, quanto autorizações positivas pode levar ao surgimento de conflitos. Isto ocorre quando duas autorizações temporais relativas ao mesmo objeto, operação e modo de acesso são atribuídas ao mesmo sujeito. Neste caso, a política de resolução de conflitos estabelece que a negação toma precedência sobre a autorização negativa. Segundo este critério, uma autorização é válida em um determinado instante t somente se uma autorização temporal ([begin, end], P, A), com $t \in \Pi(P) \cap \{[t_b, t+e]\}$ é especificada, ou pode ser derivada através de uma regra de derivação, e se nenhuma autorização negativa ([begin', end'], P', A'), com o mesmo sujeito, objeto e modo de acesso tal que $t \in \Pi(P') \cap \{[t_b', t_e']\}$ é especificada ou pode ser derivada, onde begin' e end' denotam, respectivamente, os instantes t_b' e t_e'.

O segundo componente básico deste modelo é um mecanismo de inferência baseado em regras de derivação, expressando dependências temporais entre autorizações. Estas regras permitem a criação de novas autorizações periódicas com base na validade, ou não validade, de autorizações periódicas definidas. Assim como autorizações, cada regra de derivação possui um intervalo de tempo e uma indicação de periodicidade, representando os instantes em que esta regra é válida.

Uma regra de derivação é definida como uma tripla ([begin, end], P, *A* $\langle$OP$\rangle$ *expression*), onde *A* é a autorização a ser derivada, *expression* é uma expressão booleana de autorizações, e $\langle$OP$\rangle$ é um dentre os três possíveis operadores *WHENEVER*, *ASLONGAS* e *UPON*. Autorizações derivadas através desta regras são criadas como se tivessem sido atribuídas pelo criador da respectiva regra.

A expressão booleana pode ser baseada em um número qualquer de autorizações,

conectadas por operadores lógicos. Por exemplo, a expressão $(\neg\ A_1) \vee A_2$ é verdadeira em cada instante t tal que A_1 não seja válida neste instante, e ao mesmo tempo A_2 seja válida.

Os operadores mencionados permitem que se especifique critérios de derivação baseados na existência, ou ausência, de outras autorizações, permitindo assim grande expressividade. Estes operadores funcionam indicando restrições adicionais sobre a validade da autorização definida, enquanto agindo dentro dos limites especificados pela expressão temporal P associada ao intervalo de validade.

O operador *WHENEVER* indica que a autorização é válida em todo instante em $\Pi(\mathrm{P}) \cap [\{t_b, t_e\}]$ tal que a expressão especificada seja válida. O operador *ASLONGAS*, sendo mais restritivo que o primeiro, indica que a autorização referenciada é válida em todo instante $t \in \Pi(\mathrm{P})$ que seja maior ou igual a t_b e menor que t. Por exemplo, A_1 *ASLONGAS* $\neg\ A_2$ indica que A_1 é válida até o primeiro instante em que A_2 é válida. Funcionando como um complemento ao operador *ASLONGAS*, o operador *UPON* indica que a autorização é válida a partir do primeiro instante em que a expressão booleana é verdadeira.

Um problema importante em relação à definição de dependências entre autorizações é a ordem em que estas dependências devem ser aplicadas, visto que não é desejável que esta ordem altere o conjunto final de autorizações válidas. Por exemplo, considere as duas dependências à seguir:

1. A *WHENEVER* $\neg$ B
2. B *WHENEVER* $\neg$ A

Nesse exemplo, dependendo da ordem de aplicação das autorizações, o resultado obtido é diferente. O modelo citado acima evita este problema através de um algoritmo de detecção de dependências cíclicas. Isto é, à cada autorização é associada uma prioridade, e caso o algoritmo detecte que uma autorização possui maior prioridade sobre si mesma, isto indica que há uma dependência cíclica.

7.2 Utilização de Delegações Condicionais Temporais

Considerando o nosso modelo, delegações temporais podem ser modeladas como uma extensão a delegações condicionais, tal que a condição é uma expressão temporal. De fato, uma autorização temporal, como definido no trabalho citado, pode ser traduzida como uma autorização com a conjunção de dois predicados: um predicado indica o intervalo de validade da autorização, enquanto o outro predicado indica relações de

dependência:

$$\text{Q} = (WeekDay \in WorkingDays) \wedge (\text{del}(g,\, r,\, a))$$

No exemplo acima, $WeekDay$ é uma função de t, representando o dia da semana. $WorkingDays$ é uma expressão periódica definida como P $= Semanas + \{2, ..., 6\}Dias$, representando os dias normais de trabalho durante a semana. Tanto $WeekDay$ quanto $WorkingDays$ podem ser definidas através de funções avaliadoras, como discutido no capítulo 4, sobre delegações condicionais. Devido ao segundo tipo de predicado, neste exemplo "del(g, r, a)", representando dependências, o mesmo problema de dependências cíclicas, explicado no modelo citado acima, pode ocorrer em nosso modelo. Portanto, é necessário um mecanismo de dependência que previna que esta situação ocorra.

O operador *WHENEVER* é essencial à modelagem de delegações temporais. Por outro lado, os operadores *UPON* e ASLONGAS se assemelham mais à um mecanismo de gatilho que pode ser modelado como um mecanismo independente do conceito de delegações temporais. Portanto, iremos tratar aqui somente de um mecanismo similar ao operador *WHENEVER*.

De uma forma geral, note que da mesma forma que uma delegação del(g_1, r_1, a_1) pode depender de outra delegação del(g_2, r_2, a_2), uma proibição -del(g_1, r_1, a_1) poderia depender de uma delegação del(g_2, r_2, a_2).

Assim, dada uma delegação $d =$ del(g, r, a), com $a = (o,\, f,\, Q)$, e $Q =$ "del$(g', r', -b)$", indica que d deve estar ativa sempre que a proibição del$(g', r', -b)$ existir. Podemos ainda generalizar este tipo de condição para expressão a ausência de uma delegação. Como exemplo, $Q =$ "$\neg$ del(g', r', b)", ou ainda $Q =$ "$\neg$ del$(g', r', -b)$". Esta última define que a delegação d deve estar ativa sempre não existir a proibição del$(g', r', -b)$. É permitida a definição de uma expressão contendo várias delegações dependentes, como por exemplo a condição "del(g_1, r_1, b) $\wedge \neg$ del(g_2, r_2, $-c$)". As delegações utilizadas na definição de dependências podem ser tanto contextuais quanto temporais. No caso de delegações que são somente contextuais, isto é, sem relações de dependência em relação a outras delegações, simplesmente é necessário verificar seu estado atual: bloqueada ou ativa.

Portanto, este mecanismo possibilita que tanto delegações, quanto proibições, tenham condições temporais dependentes de um conjunto de outras delegações ou proibições, permitindo assim grande flexibilidade.

Se a condição de uma delegação del(g_1, r_1, a) possui as delegações del(a_i), para i de 0 a n, em sua condição, dizemos que del(g_1, r_1, a) depende de cada del(a_i). Iremos expressar a relação de dependência com o operador $\rightarrow$. Isto é, se del(g_1, r_1, a) possui del(g_2, r_2, b) em sua condição, então del(g_1, r_1, a) $\rightarrow$ del(g_2, r_2, b). Se del(g_1, r_1, a) possui $\neg$ del(g_2, r_2, b) em sua condição, chamaremos esta dependência de dependência exclusiva, no sentido de

que existe uma relação de exclusão entre as duas delegações. Caso contrário, chamaremos a delegação de inclusiva.

Note que nem sempre uma dependência cíclica leva a um estado imprevisível. Como exemplo, a sequência de dependências del(g_1, r_1, a) $\rightarrow$ del(g_2, r_2, b) $\rightarrow$ del(g_3, r_3, c) $\rightarrow$ del(g_1, r_1, a), não leva à um estado diferente dependendo da ordem de aplicações das dependências. Observe outro exemplo, em que o mesmo não pode ser dito:

1. del(g_1, r_1, a) $\rightarrow$ del(g_2, r_2, b)
2. del(g_2, r_2, b) $\rightarrow$ del(g_3, r_3, c)
3. del(g_3, r_3, c) $\rightarrow \neg$ del(g_1, r_1, a)

Quando a condição temporal de uma delegação não é satisfeita, diremos que ela está bloqueada, caso contrário, está ativa. No caso acima, supondo por simplicidade que todas as delegações estejam ativas em um dado instante t, ao aplicarmos as dependências (1, 2, 3), nesta mesma ordem, obtemos o conjunto de delegações ativas {del(g_1, r_1, a), del(g_2, r_2, b)} e bloqueadas {del(g_3, r_3, c)}. Por outro, se alterarmos a ordem para (3, 2 e 1), obtemos o conjunto de delegações ativas {$\varnothing$} e bloqueadas {del(g_1, r_1, a), del(g_2, r_2, b), del(g_3, r_3, c)}. Observe que isto ocorre devido ao fato de que del(g_1, r_1, a) depende ciclicamente de si mesma.

7.3 Algoritmo para Aplicação de Dependências

Podemos modelar o problema de detecção de dependências cíclicas conflitantes através de um grafo de dependências, como apresentado na figura 7.1. Dependências simples são representadas por linhas contínuas, enquanto dependências exclusivas são representadas por linhas tracejadas. Se del(g_1, r_1, a) $\rightarrow$ del(g_2, r_2, b), então $\overrightarrow{AB} \in G$. Se uma delegação se encontra ativa, o vértice correspondente aparece destacado, caso contrário, caso a delegação esteja bloqueada, ou não exista, esta é representada por um vértice não destacado.

Um exemplo de dependência representada no grafo da figura 7.1 é a dependência H $\rightarrow \neg$ G $\wedge$ E. Note que é possível representar expressões conjuntivas de dependências segundo este modelo.

A detecção de dependências cíclicas conflitantes se resume à detecção de ciclos que contenham ao menos uma dependência opositiva exclusiva. Isto pode ser facilmente conseguido com algoritmos simples com esta finalidade. No exemplo da figura 7.1, é possível verificar o ciclo <BC,CF,FI,IH,HG,GB>. Ao criarmos um nova delegação temporal, tal como exemplo a delegação FI, o algoritmo de aceitação deve rejeitá-la, pois levaria à criação deste ciclo.

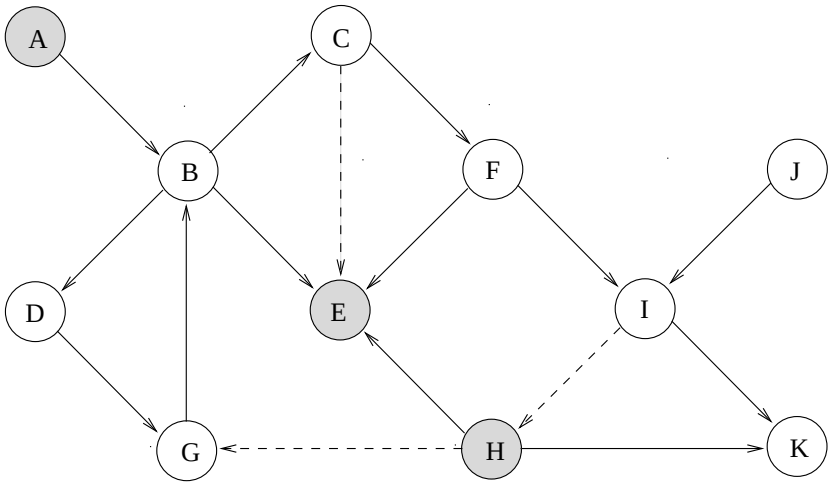


Figura 7.1: Grafo de Dependências

O conjunto de delegações em que dependências devem ser aplicadas é formado por três subconjuntos. São eles:

1. Delegações ativas que não dependem de outras delegações, e que portanto continuarão ativas após a aplicação das dependências. Estas delegações estão ativas porque possuem ao menos uma cadeia de suporte válida, e não estão explicitamente bloqueadas via proibições.

2. Delegações bloqueadas que não possuem uma cadeia de suporte válida, ou estão bloqueadas explicitamente. Estas delegações continuarão bloqueadas, não importando eventuais dependências a serem aplicadas. Observe que é possível definir dependências em relação a delegações que ainda não existem no sistema. Estas delegações, embora não existentes, também são consideradas neste conjunto para permitir a aplicação de tais dependências.

3. Delegações potencialmente bloqueadas que possuem dependências entre si, em relação às delegações ativas, ou em relação às delegações bloqueadas. Estas delegações estão inicialmente ativas, mas podem ser afetadas pela aplicação de dependências, migrando eventualmente para o primeiro ou segundo conjuntos.

Seja G um grafo de dependências, A o conjunto de delegações ativas em G, B o conjunto de delegações bloqueadas em G, e A_p o conjunto de delegações potencialmente bloqueadas. As dependências podem ser classificadas em seis diferentes situações, ou conjuntos:

1. $A_I = \{\overrightarrow{a_p a} \mid a_p \in A_p,\ a \in A,\ \overrightarrow{a_p a}$ é inclusiva$\}$

2. $B_I = \{\overrightarrow{a_p b} \mid a_p \in A_p,\ b \in B,\ \overrightarrow{a_p b}$ é inclusiva$\}$
3. $P_I = \{\overrightarrow{a_i a_j} \mid a_p \in A_p,\ a_j \in A_p,\ \overrightarrow{a_i a_j}$ é inclusiva$\}$
4. $A_E = \{\overrightarrow{a_p a} \mid a_p \in A_p,\ a \in A,\ \overrightarrow{a_p a}$ é exclusiva$\}$
5. $B_E = \{\overrightarrow{a_p b} \mid a_p \in A_p,\ b \in B,\ \overrightarrow{a_p b}$ é exclusiva$\}$
6. $P_E = \{\overrightarrow{a_i a_j} \mid a_p \in A_p,\ a_j \in A_p,\ \overrightarrow{a_i a_j}$ é exclusiva$\}$

Visto que o algoritmo de aceitação evita a formação de ciclos com dependências exclusivas, o grafo final obtido após a aplicação das dependências é único e consistente. A unicidade ocorre porque não existem dependências cíclicas exclusivas, como discutido anteriormente, enquanto que a consistência se deve ao fato de que se $\overrightarrow{a_p a} \in A_E$, e $a \in A$, então $a_p \in B$, e se $\overrightarrow{a_p b} \in B_I$, e $b \in B$, então $a_p \in B$. Isto é, se uma delegação depende de uma delegação bloqueada, esta primeira também é bloqueada. Além disso, se uma delegação possui uma relação de exclusão com uma delegação ativa, então esta primeira delegação é bloqueada. Finalmente, se não existe uma dependência que evite a validade de uma delegação, esta delegação é ativa.

Algoritmo para Aplicação de Dependências

1. Enquanto $A_p \neq \varnothing$, faça:
2. Se existe $\overrightarrow{a_p b} \in B_I$, faça $B \leftarrow B + \{a_p\}$, $A_p \leftarrow A_p$ - $\{a_p\}$.
3. Se existe $\overrightarrow{a_p a} \in A_E$, faça $B \leftarrow B + \{a_p\}$, $A_p \leftarrow A_p$ - $\{a_p\}$.
4. Se existe um bloco conexo de vértices C, tal que toda aresta $\overrightarrow{ij}$ é inclusiva, e $j \in A_p$ ou $j \in A$, faça $A \leftarrow A + C$, $A_p \leftarrow A_p$ - C.

O passo 4 é necessário para incluir no bloco de delegações ativas as delegações que são dependentes entre si, e que não possuem restrições de exclusão com outras delegações no sistema. Este conjunto pode ser determinada em tempo $O(V^2)$. Assim a complexidade total do algoritmo é de $O(V^3)$.

O algoritmo de dependências funciona como um filtro, bloqueando delegações que não atendam dependências em relação a outras delegações. A aplicação deste algoritmo deve ser realizada antes da aplicação do algoritmo de resolução de conflitos, uma vez que autorizações bloqueadas devido à existência de dependências não devem ser utilizadas em cadeias de suporte durante a resolução de conflitos.

Algoritmo para Aceitação de Delegações

1. Seja C o conjunto de delegações comparáveis à nova delegação $d = \text{del}(g, r, a$.
2. Aplique dependências em C
3. Aplique resolução de conflitos em C
4. Aplique algoritmo de aceitação condicional em C.

O algoritmo para aceitação não modifica o estado das delegações existentes no sistema, mas verifica se a nova delegação possui uma cadeia de suporte válida no momento de sua criação, considerando-se todos os eventos que podem levar ao bloqueio de delegações que poderiam pertencer à cadeia de suporte desta delegação.

A verificação de uma requisição de acesso, ou seja, a verificação se um sujeito S possui uma autorização de domínio $a_d = (o, f, Q)$ deve considerar:

1. Se existe ao menos uma delegação da autorização de domínio $d = \text{del}(g_i, S, a_d)$.
2. Se existe uma delegação de uma autorização de controle $a_c = (a_d, \textit{delegate})$ à g_i, com uma cadeia de suporte válida. A verificação da validade da cadeia de suporte deve ser realizada somente após os algoritmos de dependências, e resolução de conflitos, nesta ordem.

Algoritmo para Verificação de Autorizações de Domínio

1. Seja C o conjunto de delegações comparáveis à autorização de domínio $a_d = (o, f, Q_d)$.
2. Seja $a_c = (a_d, \textit{delegate})$ a autorização de delegação de a_d.
3. Aplique dependências em C
4. Aplique resolução de conflitos em C
5. Para cada delegação $d_i = \text{del}(g_i, \text{S}, a_d)$, tal que Q_d é *Verdadeiro*, faça:
6. Para cada delegação $d_j = \text{del}(g_j, g_i, a_c)$ faça:
7. Se $(Q_d \wedge \textit{PossuiCadeiaSuporte}(\text{G}, d_j))$ retorne *Verdadeiro*.
8. Retorne *Falso*.

Com a utilização deste algoritmo, são reforçadas tanto restrições estáticas, como autorizações negativas, quanto restrições dinâmicas, baseadas no contexto atual do sistema, e na dependência entre delegações.

7.4 Conclusão

Uma delegação temporal pode possuir tanto um predicado que represente restrições temporais, definidas com base na noção de calendários, quanto um predicado que represente relações de dependência entre delegações.

O mecanismo proposto permite que uma mesma delegação possua dependências em relação a delegações positivas ou negativas. Também é possível definir dependências em relação a não existência de determinadas delegações, através do operador $\neg$. As delegações que são referenciadas em uma dependência podem ou não ser temporais, permitindo grande flexibilidade na definição de dependências entre quaisquer delegações.

O algoritmo para aplicação de dependências recebe um conjunto inicial de delegações bloqueadas ou ativas, que possuem dependências ou são referenciadas por dependências de outras delegações. Progressivamente as delegações ativas que possuem dependências não atendidas são também bloqueadas, resultando em um estado final único e consistente.

Capítulo 8

Implementação

O modelo proposto neste documento foi avaliado através da implementação em Java dos principais algoritmos propostos em um servidor de autorização. Através das interfaces deste servidor, é possível:

- Verificar se um sujeito possui uma autorização: para tanto, verifica-se se o sujeito possui a autorização, via uma ou mais delegações, e se esta autorização é válida no contexto atual.
- Criar delegações: a criação de uma delegação é permitida a partir da verificação de uma cadeia de suporte válida para a nova delegação.
- Revogar delegações: verificação se o sujeito que requisitou a revogação possui a autorização válida necessária para realizar a revogação.

O sistema de autorização define interfaces para a implementação de avaliadores de funções, e provê o mecanismo para avaliação de expressões, conforme a sintaxe de condições definida neste documento. Assim, é possível conectar novas implementações que recuperam informações de serviços externos ao mecanismo de autorização. O arcabouço do modelo de delegação proposto foi implementado conforme ilustrado na figura 8.1. Uma requisição de acesso envolve os seguintes passos:

1. O *verificador de acesso* recebe uma requisição para verificação de uma autorização.
2. O *verificador de acesso* recupera os avaliadores de contexto da requisição, que serão responsáveis por avaliar funções de avaliação ligadas à aplicação que requisitou o acesso.
3. O *verificador de acesso* solicita ao *ponto de decisão* para que verifique se a autorização solicitada é válida.

4. O *ponto de decisão* solicita ao *gerenciador de políticas* o estado atual de autorização, e posteriormente decide se o acesso é permitido.

O gerenciador de políticas é responsável por centralizar o gerenciamento do estado de autorizações, mantendo o conjunto de autorizações consistente. O gerenciador de delegações, por sua vez, é responsável por centralizar a criação ou revogação de delegações, com o auxílio do ponto de decisão, que contém a lógica para decidir se as cadeias de suporte necessárias são válidas.

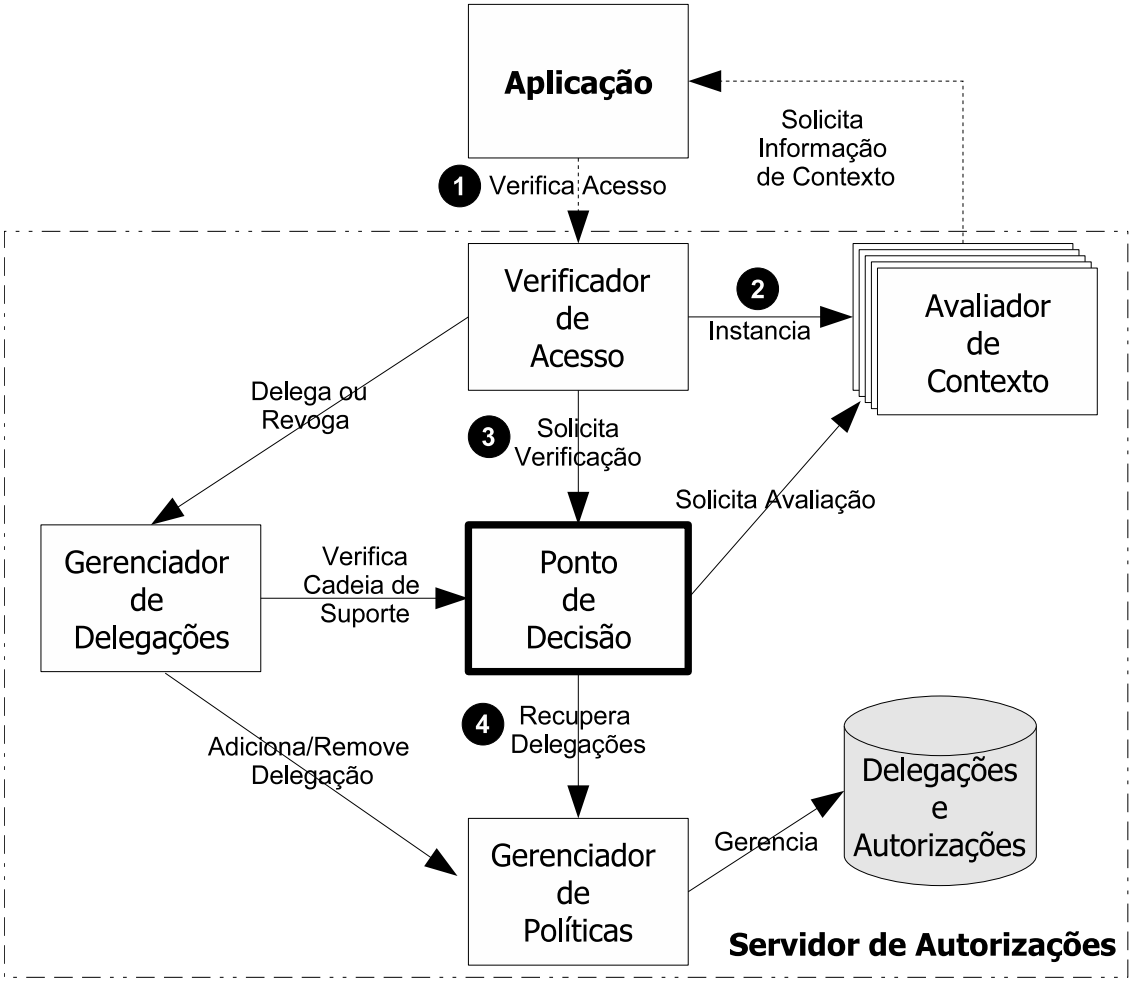


Figura 8.1: Arcabouço de um Servidor de Autorização

Tanto a verificação de autorizações de domínio quanto a verificação de autorizações de controle são realizadas pelo ponto de decisão, uma vez que ambos os casos é necessário verificar a validade das cadeias de suporte das delegações.

Ilustrando a chamada aos serviços implementados, considere a seguinte autorização, que representa o direito de aprovar uma ordem de compra com a condição de que o valor da compra seja menor que R$1000:

$a = (o, f, Q) =$ ("OrdemDeCompra", "aprovar", "ValorCompra $<$ 1000")

Considere a seguinte seqüência de delegações:

1. del(A, B, $del(a)^2$)
2. del(B, C, a)

A requisição *SOAP* para a criação da delegação BC é apresentada a seguir:

```
1. <?xml version="1.0"encoding="UTF-8"?>
2.     <soapenv:Envelope
3.         xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
4.         xmlns:q0="http://ws.auth.ic.unicamp.br"
5.         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
6.         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance»
7.         <soapenv:Body>
8.             <q0:delegate>
9.                 <q0:grantor>B</q0:grantor>
10.                <q0:delegate>C</q0:delegate>
11.                <q0:action>aprovar</q0:action>
12.                <q0:object>OrdemDeCompra</q0:object>
13.                <q0:condition>ValorCompra < 1000</q0:condition>
14.                <q0:weight>1</q0:weight>
15.            </q0:delegate>
16.        </soapenv:Body>
17.    </soapenv:Envelope>
```

Na requisição acima, foram definidos o delegador (*grantor*), o receptor (*delegate*), a operação da autorização (*action*), o objeto da autorização (*object*), e finalmente a condição para utilização da autorização (*condition*). Uma requisição que verifica se o sujeito C tem acesso à autorização a = ("OrdemDeCompra#475563", "aprovar") é apresentada a seguir:

```
1. <?xml version="1.0"encoding="UTF-8"?>
2.     <soapenv:Envelope
3.         xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
4.         xmlns:q0="http://ws.auth.ic.unicamp.br"
5.         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
6.         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance»
7.         <soapenv:Body>
8.             <q0:isAllowed>
9.                 <q0:subject>C</q0:subject>
10.                <q0:action>aprovar</q0:action>
11.                <q0:object>OrdemDeCompra#475563</q0:object>
12.            </q0:isAllowed>
13.        </soapenv:Body>
14.    </soapenv:Envelope>
```

Para uma verificação de acesso, é necessário especificar o sujeito que verifica o acesso (*subject*), a operação requisitada (*action*), e o objeto sobre o qual será realizada a operação (*object*). Note que neste exemplo a requisição depende da avaliação da condição "ValorCompra $<$ 1000", definida durante a criação da delegação da autorização a. Neste exemplo, é necessário recuperar o valor referenciado pela ordem de compra número 475563, e avaliar se a condição é atendida. Em um sistema real, o valor da ordem de compra poderia ser obtido consultando-se um serviço externo ao servidor de autorização. Neste exemplo, somente para testes, as condições foram avaliadas utilizando-se chamadas a classes implementadas localmente, permitindo simular um sistema real.

8.1 Futuro Refinamento da Implementação

Neste momento inicial procurou-se avaliar os algoritmos e modelos propostos. Entretanto, sugere-se que as seguintes questões sejam consideradas em um futuro refinamento da arquitetura proposta:

1. Deve ser garantida a autenticidade e a integridade na comunicação entre a aplicação e o servidor de autorização. Isto pode ser reforçado com a assinatura de mensagens com o certificado digital do servidor.

2. O protocolo SAML (*Security Assertion Markup Language*) [Cover 2007b] é um protocolo padrão, aprovado em 2002 pela OASIS (*Organization for the Advancement of Structured Information Standards*), para troca de informações de autenticação e autorização entre sistemas. Este protocolo pode ser utilizado na definição de uma linguagem padrão para comunicação com o servidor de autorização.

3. Deve haver um mecanismo que garanta a veracidade das informações de contexto fornecidas pelo avaliador de contexto, como discutido anteriormente no capítulo 5.

4. As autorizações e delegações devem ser gerenciadas de uma forma eficiente, permitindo que o conjunto de autorizações referentes a um determinado recurso sejam recuperadas ou alteradas de forma íntegra. O padrão XACML (*Extensible Access Control Markup Language*) [Cover 2007a], também aprovado pela OASIS, que provê uma linguagem que permite a definição dos requisitos de controle de acesso para recursos, ou seja, as políticas de acesso. Este padrão poderia ser utilizado para tanto para a definição de autorizações no sistema, quanto a sua aplicabilidade, dependendo do contexto e do tipo de recurso acessado.

5. Nesta implementação inicial não foram consideradas funcionalidades tais como criação e remoção de usuários, ou a própria associação entre usuários e autorizações, papéis ou grupos. Estas funcionalidades não se referem diretamente ao mecanismo de delegação portanto não foram necessárias em uma validação inicial. Entretanto, tais funcionalidades podem ser consideradas em um futuro refinamento, ou integração com mecanismos de autorização já existentes.

8.2 Conclusão

O mecanismo de delegação e revogação foi implementado em Java, visando avaliar o funcionamento dos algoritmos propostos neste documento. O arcabouço implementado utiliza o mesmo mecanismo para verificar autorizações de domínio e de controle, através

de um ponto central de decisão que é responsável por verificar as cadeias de suporte necessárias.

Foram realizados testes através de chamadas aos serviços implementados para verificação de autorizações, bem como para criação, revogação e rebaixamento de delegações, simulando a utilização em um sistema real.

Através de funções avaliadoras, o arcabouço proposto permite que o servidor de autorização seja estendido utilizando implementações que invoquem serviços internos ou externos ao servidor de autorização, dependendo de uma aplicação específica.

Capítulo 9

Conclusão

Neste documento foi apresentado um modelo que tem como principal vantagem, em relação a outros trabalhos relacionados à área de modelos de autorização, a capacidade de controlar de forma precisa a formação de cadeias de delegações. Como apresentado, os dois conceitos principais que permitem tal controle são o conceito de comprimento de cadeias de delegações, e o conceito de delegações condicionais. Entre as principais contribuições deste documento estão:

- definição de um mecanismo que permite controlar o acesso a recursos com base tanto em informações de contexto quanto em informações de conteúdo destes recursos. Entre as aplicações de tal mecanismo, como discutido, estão a aplicação de restrições de ligação ou separação de deveres, computação móvel e controle de acesso com alta granularidade.

- definição de um mecanismo que permite controlar separadamente autorizações de administração (autorizações de controle) e utilização de objetos (autorizações de domínio).

- definição de algoritmos eficientes, isto é, polinomiais, de revogação, bem como de aceitação, para o modelo apresentado, caracterizando-se por manter os sujeitos remanescentes do sistema com o maior conjunto possível de privilégios após a revogação de delegações [Negrello and Wainer 2006].

- definição de um algoritmo para resolução de conflitos quando proibições e autorizações são concedidas a um mesmo sujeito.

- definição de um algoritmo para aplicação de dependências entre delegações que garante um conjunto único e consistente de delegações após sua aplicação.

Através do modelo proposto procurou-se lidar com algumas dificuldades enfrentadas por mecanismos de autorização e delegação atuais. A abordagem apresentada difere de outras propostas em trabalhos relacionados, na medida em que permite controlar individualmente cada delegação do sistema, seja com informações de conteúdo, contexto, ou dependências, permitindo assim grande flexibilidade e potencial de utilização em diversas aplicações.

O conjunto de mecanismos apresentados oferece a possibilidade de controlar delegações tanto dinamicamente, através de condições em delegações, quanto estaticamente, através de autorizações negativas. O controle dinâmico é aplicável tanto à criação de novas delegações, quanto à utilização de delegações já existentes, visto que é necessário, para a utilização de uma delegação, que as condições impostas ao longo de pelo uma cadeia de suporte sejam atendidas considerando o contexto atual da requisição de acesso. Um ponto adicional relacionado a cadeias de delegações, e que torna o modelo mais flexível, se refere ao fato de que a relação de força entre delegações é considerada na composição de cadeias de suporte. Isto é, para a verificação se um sujeito possui uma determinada autorização, são verificadas as delegações de todas autorizações que possuem ao menos a mesma força da autorização verificada.

Embora o foco desta dissertação tenha sido na formulação de um modelo teórico de delegação e revogação, procurou-se avaliar o modelo proposto através da construção de um arcabouço simples de um servidor para criação e revogação de delegações, e verificação de autorizações. Tal arcabouço permitiu verificar que é possível utilizar os algoritmos e mecanismos em uma única arquitetura integrada. Isto é, o mesmo mecanismo utilizado para verificação autorizações específicas de uma aplicação, é utilizado para verificar a autorização necessária para criação e revogação de delegações.

9.1 Trabalho Futuro

O modelo proposto nesta dissertação é essencialmente discricionário, isto é, visa o controle de delegações somente entre sujeitos do sistema. Entretanto, o modelo pode ser estendido para considerar tanto o agrupamento de usuários, quanto a obtenção de autorizações via papéis, já que a princípio é indiferente a forma com que cada delegador recebeu a autorização que deseja delegar. Entretanto, juntamente com a adição de tais conceitos ao modelo, é necessário considerar a existência de hierarquias, e a política para resolução de conflitos associada, como discutido no capítulo referente a autorizações negativas. É importante notar que trabalho desenvolvido nesta dissertação é ortogonal a estes conceitos, e continua válido mesmo que tais conceitos sejam adicionados ao modelo.

Como mencionado anteriormente, vários modelos de delegação existentes atualmente permitem controlar delegações através da especificação de regras gerais para criação e

utilização de delegações. Estas regram não são associadas a uma delegação específica entre dois sujeitos determinados, mas se aplicam a todas as delegações de um determinado tipo, permitindo assim a definição de regras organizacionais, tais como nos casos de separação e ligação de deveres. Portanto, o conceito de regras pode ser incluído no modelo para permitir um controle adicional mais centralizado, ideal para sistemas baseados em papéis, mas também aplicável a sistemas discricionários, oferecendo assim um grau adicional de controle.

Um mecanismo de registro de histórico de operações é especialmente útil tanto na implementação do controle de acesso baseado em histórico, quanto na definição de restrições relativas a conflitos de interesse, como procurado pela política *Chinese Wall*. A existência de um mecanismo de registro é especialmente útil se associado ao mecanismo de regras, acima citado, pois permite a definição de regras organizacionais que especificam restrições em relação aos objetos que podem ser acessados por um sujeito, com base nos objetos já acessados anteriormente por este mesmo sujeito.

Finalmente, com relação à validação do modelo, são necessários testes adicionais para verificar a sua aplicabilidade em sistemas de grande porte, tal como um sistema de gerenciamento de bases de dados. Isto permitiria verificar até que ponto os mecanismos propostos neste documento podem contribuir para o aprimoramento dos mecanismos de autorização destes sistemas.

Referências Bibliográficas

[A.K. Jones and Snyder. 1976] A.K. Jones, R. L. and Snyder., L. (1976). A linear time algorithm for deciding security. In *Proc. 17th Anual Symposium on Foundations of Computer Science*, pages 33–41.

[Al-Kahtani and Sandhu 2004] Al-Kahtani, M. A. and Sandhu, R. (2004). Rule-based RBAC with negative authorization. In *ACSAC '04: Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, pages 405–415, Washington, DC, USA. IEEE Computer Society.

[Astrahan et al. 1976] Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D., Eswaran, K. P., Gray, J. N., Griffiths, P. P., King, W. F., Lorie, R. A., McJones, P. R., Mehl, J. W., Putzolu, G. R., Traiger, I. L., Wade, B. W., and Watson, V. (1976). System R: A relational approach to database management. *ACM Trans. Database Syst.*, 1(2):97–137.

[Barka and Sandhu. 2000] Barka, E. and Sandhu., R. (2000). A role-based delegation model and some extensions. In *23rd National Information Systems Security Conference, Baltimore, MD, October 2000.*

[Bell and LaPadula 1976] Bell, D. E. and LaPadula, L. J. (1976). Secure computer system: Unified exposition and multics interpretation. Technical Report MTR-2997 Rev. 1, The MITRE Corporation, Bedford, MA.

[Bertino et al. 1996] Bertino, E., Bettini, C., Ferrari, E., and Samarati, P. (1996). Supporting periodic authorizations and temporal reasoning in database access control. In *Proc. International Conference on Very Large DataBases (VLDB)*, pages 472–483.

[Bertino et al. 1993] Bertino, E., Samarati, P., and Jajodia, S. (1993). Authorizations in relational database management systems. In *CCS '93: Proceedings of the 1st ACM conference on Computer and communications security*, pages 130–139, New York, NY, USA. ACM Press.

[Bertino et al. 1997] Bertino, E., Samarati, P., and Jajodia, S. (1997). An extended authorization model for relational databases. *Knowledge and Data Engineering*, 9(1):85–101.

[Bondy and Murty 1976] Bondy, J. A. and Murty, U. S. R. (1976). *Graph Theory with Applications*. American Elsevier.

[Brewer and Nash 1989] Brewer, D. F. C. and Nash, M. J. (1989). The chinese wall security policy. In *Proc. IEEE Symposium on Security and Privacy*, pages 162–163.

[Cover 2007a] Cover, R. (2007a). Extensible access control markup language (XACML). WWW page. `http://xml.coverpages.org/xacml.html`. Acessado em Janeiro de 2007.

[Cover 2007b] Cover, R. (2007b). Security assertion markup language (SAML). WWW page. `http://xml.coverpages.org/saml.html`. Acessado em Janeiro de 2007.

[Covington et al. 2000] Covington, M., Moyer, M., and Ahamad, M. (2000). Generalized role-based access control for securing future applications. In *23rd National Information Systems Security Conference (NISSC)*, Baltimore, MD, USA.

[Giuri and Iglio 1997] Giuri, L. and Iglio, P. (1997). Role templates for content-based access control. In *RBAC '97: Proceedings of the second ACM workshop on Role-based access control*, pages 153–159, New York, NY, USA. ACM Press.

[Griffiths and Wade 1976] Griffiths, P. P. and Wade, B. W. (1976). An authorization mechanism for a relational data base system (abstract). In Rothnie, J. B., editor, *SIGMOD Conference*, page 51. ACM.

[Hulsebosch et al. 2005] Hulsebosch, R. J., Salden, A. H., Bargh, M. S., Ebben, P. W. G., and Reitsma, J. (2005). Context sensitive access control. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 111–119, New York, NY, USA. ACM Press.

[Jajodia et al. 2001] Jajodia, S., Samarati, P., Sapino, M. L., and Subrahmanian, V. S. (2001). Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 26(2):214–260.

[Kumar et al. 2002] Kumar, A., Karnik, N., and Chafle, G. (2002). Context sensitivity in role-based access control. *SIGOPS Oper. Syst. Rev.*, 36(3):53–66.

[Negrello and Wainer 2006] Negrello, F. and Wainer, J. (2006). Revogação com downgrade. In *Proceedings of the Sixth Symposium on Information and Computer System Security (SBSeg'2006)*, Santos, SP, Brazil.

[Niezette and Stevenne 1992] Niezette, M. and Stevenne, J.-M. (1992). An efficient symbolic representation of periodic time. In *Proc. of the ISMM International Conference on Information and Knowledge Management CIKM-92*, pages 161–168, Baltimore, MD.

[Park and Sandhu 2004] Park, J. and Sandhu, R. (2004). The $UCON_{ABC}$ usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174.

[Rabitti et al. 1991] Rabitti, F., Bertino, E., Kim, W., and Woelk, D. (1991). A model of authorization for next-generation database systems. *ACM Trans. Database Syst.*, 16(1):88–131.

[Ruan and Varadharajan 2004] Ruan, C. and Varadharajan, V. (2004). A weighted graph approach to authorization delegation and conflict resolution. In Wang, H., Pieprzyk, J., and Varadharajan, V., editors, *ACISP*, volume 3108 of *Lecture Notes in Computer Science*, pages 402–413. Springer.

[Samarati and di Vimercati 2001] Samarati, P. and di Vimercati, S. D. C. (2001). Access control: Policies, models, and mechanisms. In Focardi, R. and Gorrieri, R., editors, *Foundations of Security Analysis and Design*, LNCS 2171. Springer-Verlag.

[Shen and Dewan 1992] Shen, H. and Dewan, P. (1992). Access control for collaborative environments. In *CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pages 51–58, New York, NY, USA. ACM Press.

[Siewe et al. 2003] Siewe, F., Cau, A., and Zedan, H. (2003). A compositional framework for access control policies enforcement. In *FMSE '03: Proceedings of the 2003 ACM workshop on Formal methods in security engineering*, pages 32–42, New York, NY, USA. ACM Press.

[U. Bussolati and Martella. 1993] U. Bussolati, M. F. and Martella., G. (1993). A conceptual framework for security system: the action-entity model. In *Proc. 9th IFIP World Conference*, pages 127–132.

[Wainer et al. 2007] Wainer, J., Kumar, A., and Barthelmess, P. (2007). DW-RBAC: A formal security model of delegation and revocation in workflow systems. *Inf. Syst.*, 32(3):365–384.

[Woo and Lam 1993] Woo, T. Y. C. and Lam, S. S. (1993). Authorizations in distributed systems: A new approach. *Journal of Computer Security*, 2(2-3):107–136.

[Yao et al. 2001] Yao, W., Moody, K., and Bacon, J. (2001). A model of OASIS role-based access control and its support for active security. In *SACMAT*, pages 171–181.

[Zhang et al. 2003a] Zhang, L., Ahn, G.-J., and tseng Chu, B. (2003a). A rule-based framework for role-based delegation and revocation. *ACM Trans. Inf. Syst. Secur.*, 6(3):404–441.

[Zhang et al. 2003b] Zhang, X., Oh, S., and Sandhu., R. (2003b). PBDM: A flexible delegation model in rbac. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies.*