Maicon Stihler

Uma Arquitetura de Controle de Uso para Sistemas Distribuídos em Ambientes Colaborativos

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática.

Curitiba
2009

Maicon Stihler

Uma Arquitetura de Controle de Uso para Sistemas Distribuídos em Ambientes Colaborativos

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática.

Área de Concentração: Ciência da Computação

Orientador: Prof. Dr. Altair Olivo Santin

Curitiba

2009

página reservada para a ficha catalográfica

página reservada para a folha de aprovação

Dedicado aos meus pais, Luiz Carlos Stihler e Maria Laurete Stihler.

Agradecimentos

Agradeço aos meus pais, Luiz Carlos Stihler e Maria Laurete Stihler, e minha irmã, Scheila Stihler, pelo suporte durante toda a minha vida. Agradeço também ao meu orientador, Prof. Dr. Altair Olivo Santin, por sua paciência e dedicação, fundamentais para o sucesso deste trabalho. Por fim, agradeço aos meus colegas de mestrado e aos professores do PPGIa que me auxiliaram nesta caminhada, com suas críticas e conselhos.

Resumo

Sistemas distribuídos em ambientes colaborativos exploram tecnologias de comunicação para permitir que recursos geograficamente distribuídos, e pertencentes a domínios de administração distintos, sejam utilizados por múltiplas organizações afim de atingir objetivos em comum. As abordagens tradicionais de gerenciamento de controle de acesso são limitadas, pois não são capazes de avaliar políticas de uso em tempo de execução. Além disso, tais abordagens não se mostram integradas ao gerenciamento dos recursos, dificultando a gestão do consumo de recursos e consequentemente lidando mal com violações das políticas. Neste trabalho propomos a utilização de uma arquitetura de gerenciamento de controle de acesso descentralizada, que está baseada em controle de uso (UCON_{ABC}). O objetivo é suportar avaliações de políticas de autorização dinâmicas – que admitem alterações de atributos de recursos e usuários durante o uso dos recursos. Além disso, propomos uma abordagem para implementação dos mecanismos de controle de uso, integrando-o a um sistema de *Accouting* (gerenciamento de informações) para facilitar a gestão integrada do controle de acesso. Utilizamos uma linguagem de escrita de políticas de uso de fácil compreensão, que é convertida em XACML para ser executada no mecanismo de controle. Realizamos a implementação e avaliação de um protótipo, de modo a mostrar a viabilidade da proposta. Os resultados da avaliação do protótipo podem ser utilizados para ajustar a configuração do mecanismo proposto às características do ambiente onde o mesmo está sendo utilizado.

Palavras-chave: UCON_{ABC}, Controle de Uso, Políticas de Uso, Segurança de Serviços Web.

Abstract

Distributed systems on collaborative environments exploit the communications technologies to permit that geographically distributed resources, which belong to different administration domains, be utilized by many organizations who wish to achieve common goals. The traditional approaches for access control management are restricted, because they are not able to evaluate usage policies at run time. Furthermore, such approaches are not integrated with resource management, making it hard to manage resource consumption and, as a consequence, it becomes troublesome to deal with policy violations. In this work we propose the employment of a decentralized architecture for managing access control, which is based on usage control (UCON$_{ABC}$). The goal is to support dynamic evaluations of authorization policies – admitting resource and user attribute updates during resource utilization. Also, we propose an approach for implementing the usage control mechanism, integrating it with an accounting system (for information management) to ease the integrated access control management. We employ a language for policy writing which is easy to understand, and that is converted to XACML format to be evaluated on the access control mechanism. We implemented and evaluated a prototype, to show the proposal's viability. The prototype evaluation results may be employed to configure the proposed mechanism, fine tuning it to the environment's characteristics where it is being utilized.

Keywords: UCON$_{ABC}$, Usage Control, Usage Policies, Web Services Security.

Sumário

1 Introdução

1.1 Contexto

O desenvolvimento da infra-estrutura de comunicação de dados, mais notadamente da Internet, tem permitido o desenvolvimento de novas formas de cooperação entre indivíduos e organizações (companhias, corporações, etc.). Atualmente, mais do que em qualquer outro momento, os indivíduos e organizações tem a possibilidade de unirem seus recursos e habilidades específicas num ambiente virtual, possibilitando a cooperação para solução de problemas que seriam inviáveis de resolver individualmente.

Sistemas distribuídos em ambientes colaborativos, por congregar indivíduos e organizações distintas e possivelmente sem relacionamentos anteriores, tem a necessidade de estabelecer mecanismos para controlar a interação entre os diversos parceiros e os recursos oferecidos por cada um deles. É de fundamental importância que os acordos estabelecidos entre os parceiros colaboradores sejam respeitados. Do mesmo modo, uma organização que contrata os recursos ou serviços de outros colaboradores, tem interesse em regular como estes serão utilizados por seus usuários.

Diversos trabalhos técnicos tem estudado o problema do controle de acesso em sistemas distribuídos colaborativos. Tais trabalhos, entretanto, somente estudam o problema do controle de acesso, tido como a autorização que acontece antes da concretização do acesso ao recurso, ignoram o controle do recurso no decorrer de sua utilização.

Ciente das limitações dos modelos de controle de acesso tradicionais, Park e Sandhu (PARK; SANDHU, 2004) desenvolveram um modelo de controle de uso chamado UCON$_{ABC}$. Este modelo introduz diversos conceitos fundamentais para superar as limitações dos modelos de controle de acesso tradicionais.

Os principais conceitos são:

- Continuidade dos controles, de modo que os controles possam ser aplicados durante

todo o período de uso do recurso protegido;

- As obrigações, atividades que devem ser executadas para que a utilização do recurso seja concedida ou mantida;
- A mutabilidade de atributos, alterações em atributos referentes a sujeitos e recursos que podem causar a suspensão do direito concedido, durante o exercício do mesmo;
- Condições, mudanças em variáveis independentes de sujeitos e recursos que podem causar a negação do direito de acesso, ou sua revogação durante o exercício do mesmo.

O Controle de uso engloba o conceito tradicional de Autorização, e inclui oBrigações e Condições de uso – por isso o nome UCON$_{ABC}$ (de *Usage Control*, com autorizações, obrigações, e condições). As autorizações verificam se um determinado sujeito possui um determinado direito sobre o recurso requisitado. Condições são restrições de ambiente como localidade de origem de um acesso, data/hora de acesso, etc., que devem ser respeitadas para que o acesso seja permitido. Condições atentam para aspectos independentes do sujeito ou recurso envolvido (e.g. número de processos em execução, horário do dia). As obrigações são ações que devem ser executadas para que o direito seja concedido ou mantido.

O controle de uso preocupa-se com a proteção do recurso durante todo o ciclo de utilização, por isso é aplicável em diversos momentos do uso: antes, durante, e depois do acesso. Além disso existe uma preocupação em refletir imediatamente as mudanças em atributos do sujeito, e do recurso oferecido. O controle de uso implementa o que chamamos de continuidade dos controles e mutabilidade dos atributos. Em função disto, é necessária a avaliação continua durante o acesso, e não apenas antes do mesmo.

1.2 Motivação

O dinamismo dos sistemas distribuídos em ambientes colaborativos apresenta um desafio para arquiteturas de autorização tradicionais. O gerenciamento de usuários e a escrita de políticas tem um alto custo de recursos humanos e gerencial. Em abordagens tradicionais, a entidade provedora de recursos deve, além de gerenciar o próprio serviço provido, lidar com estas tarefas de administração de usuários e políticas, o que torna a sua operação mais dispendiosa.

Em busca de reduzir custos de gerenciamento, uma alternativa comum é o estabelecimento de controles mais genéricos (eg. em nível de grupo, ao invés de individual). No entanto, esta prática pode, por exemplo, introduzir problemas que inviabilizem processos de auditoria adequados.

O dinamismo natural dos sistemas distribuídos em ambientes colaborativos expõe as deficiências dos modelos de controle de acesso tradicionais. Atributos relacionados a usuários, recursos, e o ambiente, podem mudar a qualquer momento. No entanto, o controle de acesso tradicional não pode refletir estas mudanças, potencialmente expondo organizações provedoras a abusos na utilização de seus recursos.

É possível implementar controles mais abrangentes, entretanto, isto costuma envolver o emprego de mecanismos não integrados. Demandando assim a intervenção de administradores de segurança para integrar tais mecanismos, sistemas de monitoração, gerenciamento de credenciais, entre outros, o que muitas vezes resulta em soluções *ad hoc* complexas e de difícil manutenção, além de aumentar custos de gerenciamento e as possibilidades de erro.

Apesar de seu grande potencial para utilização em ambientes como estes, as pesquisas sobre a aplicação de controle de uso em ambientes colaborativos distribuídos ainda são bastante limitadas. As propostas encontradas na literatura não cobrem os aspectos de controle de uso adequadamente, e não contemplam a estrutura do ambiente colaborativo em si, dificultando a sua aplicação prática.

As aplicações não estão preparadas para lidar com as alterações no ambiente que podem levá-las a condição de exceção (e.g. quando um direito é revogado). Em um ambiente de colaboração, é interessante que se forneçam mecanismos que permitam que os participantes tenham o menor número de interrupções de operação possível, para evitar o desperdício de tempo e de informações.

1.3 Objetivos

Este trabalho tem como objetivo propor uma arquitetura flexível e dinâmica para sistemas distribuídos em ambientes colaborativos, utilizando o controle de uso como seu modelo de controle fundamental.

1.3.1 Objetivos Específicos

Os objetivos específicos deste trabalho são:

1. Definir uma arquitetura para ambientes colaborativos que vise o baixo acoplamento entre participantes.
2. Definir um modelo para gerenciamento de usuários e políticas de segurança de baixo custo gerencial e visando integridade de políticas.
3. Definir um modo de permitir que os participantes possam detectar as violações de políticas e corrigir o problema.
4. Propor uma estratégia de aplicação de controle de uso em sistemas distribuídos colaborativos utilizando tecnologias de software livre e amplamente aceitos.
5. Implementar e avaliar um protótipo que permita demonstrar a viabilidade da arquitetura proposta.

1.4 Escopo

Este trabalho tem tanto um enfoque conceitual, discutindo e propondo soluções para uma arquitetura para sistemas distribuídos em ambientes colaborativos, como uma característica de prova-de-conceito, implementando determinados componentes da arquitetura para demonstrar a viabilidade da mesma.

Por se tratar de um trabalho conceitual e prova-de-conceito, nem todos os componentes da arquitetura serão implementados, limitando-se a implementação dos componentes essenciais para demonstrar a aplicabilidade da proposta.

1.5 Organização do Documento

Este documento está organizado da seguinte maneira:

Capítulo 1: Contextualização, motivação, objetivos e o escopo deste trabalho.

Capítulo 2: Uma revisão dos modelos de Controle de Acesso tradicionais, e apresentação do modelo de Controle de Uso – UCON$_{ABC}$.

Capítulo 3: Discussão de trabalhos prévios relevantes para esta proposta.

Capítulo 4: Apresentação da arquitetura proposta.

Capítulo 5: Discussão da implementação do protótipo.

Capítulo 6: Avaliação do protótipo.

Capítulo 7: Conclusões e discussão de trabalhos futuros.

2 Controle de Acesso e Controle de Uso

Neste capítulo abordaremos os conceitos fundamentais para o desenvolvimento deste trabalho: os conceitos de Controle de Acesso e Controle de Uso. A apresentação de tais conceitos nos permitirá demonstrar porque o Controle de Uso é um modelo mais apropriado para os modernos ambientes computacionais.

2.1 Controle de Acesso Tradicional

O controle de acesso é uma parte importante do sistema de segurança, visa limitar as atividades realizadas por usuários legítimos, normalmente utilizando um componente conhecido como monitor de referência (SANDHU; SAMARATI, 1994). O monitor de referência decide se uma determinada requisição é permitida com base em informações contidas em um repositório de políticas e nos atributos dos sujeitos. O repositório de políticas utilizado pelo monitor de referência é gerenciado por um administrador de segurança, o qual estabelece as regras de autorização com base nas políticas de segurança da entidade em questão.

Políticas de segurança podem ser compreendidas como guias que definem, de maneira mais abstrata, como o controle de acesso deve ser feito e como as decisões de autorização são tomadas. A implementação das políticas de segurança é feita através de mecanismos de *hardware* e *software* que traduzem as regras abstratas para operações em nível de mecanismos de segurança. Mecanismos de segurança são os componentes utilizados para implementar os objetivos de uma ou mais políticas de segurança.

Os atributos de autorização do sujeito podem ser obtidos de três modos diferentes, e caracterizam a arquitetura como sendo *push*, *pull*, ou híbrida (LORCH et al., 2003).

- No modo *push* o sujeito[1] tem a responsabilidade de obter todos as evidências que comprovem que o mesmo é possuidor das credenciais necessárias para obter acesso. Existe por isso um primeiro momento em que o sujeito busca todas as credenciais (e.g. *capabilities* ou *tokens*), no segundo momento o sujeito envia as credenciais ao guardião do recurso juntamente com a requisição. O provedor avalia então a validade das credenciais e decide se a requisição é permitida de acordo com as políticas vigentes ou se o pedido deve ser rejeitado.

- O modo *pull* o sujeito simplesmente envia a sua requisição ao guardião do recurso. O guardião do recurso deve então obter todas as credenciais necessárias para a avaliação da política de controle de acesso. Após obter todas as credenciais, a requisição é avaliada de acordo com as políticas e credenciais fornecidas, permitindo ou rejeitando a requisição.

- Uma terceira abordagem é possível, onde uma combinação entre o modo *push* e *pull* são utilizadas, conforme o tipo da credencial. Assim o sujeito pode apresentar apenas algumas credenciais, e o provedor obtém outras credenciais necessárias a avaliação da politica de acesso. Essa abordagem é bastante flexível possibilitando diversas combinações de autorização.

Classicamente, o controle de acesso pressupõe que o usuário esteja devidamente autenticado antes de avaliar suas requisições no monitor de referência. Um sistema de autenticação é responsável por verificar que o usuário está corretamente identificado, com pena de, em caso de falha, negar o pedido de acesso. Em sistemas distribuídos deve-se notar que a identificação deve ser ainda mais abrangente, podendo conter vários qualificadores (usuários, máquinas, domínio), pois uma identificação pode precisar transportar os domínios de autenticação..

É importante a existência de monitores de atividade para possibilitar uma correta auditoria de todas as atividades relevantes realizadas no sistema. A auditoria atua tanto de modo a desestimular possíveis atividades irregulares, pois os usuários sabem que suas atividades são monitoradas, como para auxiliar na identificação de violações as políticas de segurança.

Vale ressaltar que a distinção dos diversos componentes do controle de acesso (políticas, administrador do repositório de políticas, o repositório de políticas, o monitor de

[1]Sujeito pode ser um usuário e/ou programa, e um usuário pode apresentar-se como um ou mais sujeitos em situações distintas. São os sujeitos quem normalmente iniciam as requisições de acesso.

referência), não implica necessariamente numa clara distinção em nível de mecanismo. É possível, por exemplo, que as informações de autorização estejam armazenadas juntamente com o objeto protegido pelo monitor de referência, ao invés de localizadas em uma base centralizada.

As políticas podem ter níveis diferentes de segurança. A definição do nível de segurança da política aplicada vai depender dos requerimentos de segurança do ambiente. Nem sempre uma política excessivamente criteriosa será aconselhável.

2.1.1 Matriz de Acesso

Uma das abstrações mais importantes desenvolvidas na área de controle de acesso é, talvez, a representação de todos os recursos de um sistema sob a forma objetos (SANDHU; SAMARATI, 1994). Deste modo a proteção dos objetos passa a ser crucial e, ao mesmo tempo, simplifica a proteção dos recursos. Observe que isto não inclui a proteção física dos recursos, que deve ser garantida para que as proteções do sistema computacional não sejam contornadas. O conceito formal de matriz de acesso foi introduzido em (LAMPSON, 1974).

Tabela 2.1: Matriz de Acesso

	arquivo1	arquivo2	...	conta1
sujeito1	R	RW	...	consulta
sujeito2	RX	–	...	–
...	...	...	...	...
sujeitoN	RWX	R	...	consulta, saque

A matriz de acesso é um modelo conceitual, representável por uma tabela, onde nas linhas estão os sujeitos e nas colunas estão os objetos. Cada célula contém os direitos de acesso do sujeito da linha sobre o objeto da coluna. Tais direitos de acesso são variáveis conforme o objeto (e.g. leitura e escrita em um arquivo, ou consulta, saque, e depósito em uma conta bancária). A Tabela 2.1 apresenta um exemplo de matriz de acesso. O sujeito1 tem direito de leitura (R) sobre o arquivo1 e leitura e escrita (RW) no arquivo2, enquanto só possui acesso para consulta na conta1. O sujeito2 possui acesso para leitura e execução (RX) no arquivo1 e nenhum direito sobre arquivo2 e conta1. Por fim, o sujeitoN tem acesso completo ao mesmo, acesso para leitura no arquivo2, e direito de consulta e execução de saques na conta1.

É importante notar que sujeitos podem ser representados como objetos também, e

que a matriz de acesso deixa bem clara a distinção entre autenticação e autorização. É função do monitor de referência garantir que somente os direitos representados na matriz de acesso sejam executados no sistema (autorização). A seguir veremos algumas maneiras de implementar a matriz de controle de acesso.

2.1.2 Mecanismos – ACLs e Listas de Competências

A lista de controle de acesso (ACL) equivale a uma implementação da matriz de acesso por colunas. Cada objeto está vinculado a uma lista contendo os sujeitos que possuem direitos sobre o objeto, e quais são esses direitos. Essa abordagem permite a verificação facilitada de todos os sujeitos e direitos relacionados a um objeto, facilita também a revogação de todos os direitos de um objeto, bastando substituir a lista original por uma lista vazia. A utilização de ACLs, no entanto, dificulta a gerência do controle de acesso, pois é necessário consultar todos os objetos do sistema para obter uma visão global, assim como dificulta a revogação de todos os direitos de um sujeito. ACLs podem conter nomes de grupos (conjunto de sujeitos), sendo que os membros do grupo herdam os direitos do mesmo, porém neste caso quando algum membro do grupo faz alguma operação nos objetos do mesmo, não se sabe quem a executou.

Alguns sistemas (e.g. UNIX) tomam uma abordagem simplificada de ACL, onde somente um número reduzido de grupos é utilizado. Isto permite que a ACL seja representada com apenas alguns bits no objeto, simplificando a implementação e reduzindo os custos. De mesmo modo, existem implementações de ACL que permitem controles complexos, utilização de padrões na seleção de sujeitos aos quais o direito se aplica, e definição de quando e como o acesso pode ser requisitado.

Nas listas de Competências (*capabilities*), implementação da matriz de acesso pela linha, cada sujeito tem atrelado a si uma lista contendo todos os objetos sobre os quais possui direitos, e os referidos direitos (SANDHU; SAMARATI, 1994). O ponto positivo de tal abordagem é a facilidade de verificar sobre quais objetos um determinado sujeito possui direitos e que direitos são esses. Por outro lado, verificar que sujeitos possuem direitos sobre um determinado objeto torna-se bastante difícil, é necessário verificar as listas de competências de todos os sujeitos que compõem o sistema.

As listas de competências podem ser utilizadas de modo complementar com outros mecanismos. Uma possibilidade é a combinação de *capabilities* em sistemas distribuídos. Por exemplo, um sujeito executa a sua autenticação em um sistema e recebe do mesmo a sua lista de competências, a partir deste momento ele pode apresentar esta lista para

os demais sistemas que fazem parte do sistema distribuído, cada sistema pode ter suas ACLs locais para, juntamente com a lista de competências, gerenciar mais facilmente o controle de acesso.

2.1.3 Modelos Tradicionais de Controle de Acesso

Nesta seção são apresentados os três modelos de controle de acesso tradicionais utilizados, o modelo discricionário, o modelo mandatório e o modelo baseado em papéis. A utilização de um modelo não exclui a possibilidade de utilizar os outros modelos, pelo contrário, os modelos podem ser utilizados de forma complementar para oferecer um controle de acesso mais adequado (SANDHU; SAMARATI, 1994).

Modelo de Acesso Discricionário – DAC

Controle de acesso discricionário (*discretionary access control* – DAC) é utilizado para restringir o acesso de um sujeito (usuário ou programa) a um subconjunto dos modos de acesso disponíveis para os objetos protegidos. As decisões de acesso são feitas com base na identidade do sujeito, e em regras que especificam as permissões que cada sujeito possui sobre cada objeto do sistema. O controle de acesso pode ser tanto *aberto* como *fechado*, ou híbrido. No sistema aberto, as regras de autorização define explicitamente o que os usuários não podem fazer, tudo que não for explicitamente proibido é autorizado por padrão. No sistema fechado as autorizações são explicitadas e os modos de acesso que não forem explicitamente autorizados estão proibidos. O sistema híbrido permite que sejam definidas regras tanto em termos de permissão como negação de direitos (SANDHU; SAMARATI, 1994).

A flexibilidade do controle de acesso discricionário reside no fato de que um sujeito ou grupo de sujeitos com determinados direitos, pode atribuir direitos a terceiros à sua vontade. Ou seja, o controle é *discricionário* no sentido de que o sujeito pode conceder ou revogar direitos com base em critérios subjetivos. O controle não tem nenhum conhecimento, e não baseia suas decisões na semântica do objeto protegido. Em um mecanismo de DAC o fluxo da informação não pode ser controlado, sendo que um sujeito tem total liberdade para fazer o que quiser com a informação uma vez que a tenha acessado.

A identidade do sujeito é fundamental para o um controle DAC, se houver a possibilidade de realizar ações com a identidade de outro sujeito, então os controles podem ser contornados. Como na maioria dos sistemas, qualquer programa sendo executado em

nome do sujeito herda a identidade do mesmo, um mecanismo de DAC está vulnerável a ataques usando *trojan horses*. Estas e outras questões são estudas em (DOWNS et al., 1985).

Modelo de Acesso Mandatório – MAC

O modelo de controle de acesso mandatório (*mandatory access control* – MAC) baseia-se no nível de habilitação de objetos e de sujeitos. A *classificação* dos objetos define níveis de segurança que representam a sensitividade da informação (SANDHU, 1993). Os sujeitos são categorizados de acordo com seu nível de confidencialidade (*clearance*). No caso mais simples, o nível de segurança é um elemento de um conjunto ordenado. Na área militar, por exemplo, os níveis podem ser formados por Altamente Secreto (AS), Secreto (S), Confidencial (C), Não-Classificado (NS), sendo que $AS > S > C > NS$. Cada nível de segurança domina a si mesmo e a todos os níveis abaixo de si na hierarquia.

A autorização de acesso só é permitida se as regras para operações de leitura e escrita nos objetos do sistema forem satisfeitas. Essas relações podem ser definidas conforme o objetivo (integridade ou confidencialidade) da política de segurança. A seguir serão apresentados brevemente os 2 modelos de MAC mais conhecidos da literatura: Bell–LaPadula e Biba.

O modelo Bell–LaPadula preocupa-se com a confidencialidade (não revelação de informações sensíveis). Considerando que λ corresponde ao nível de segurança do sujeito s e do objeto o, podemos analisar as duas duas propriedades básicas:

segurança simples (no read up): um dado sujeito pode ler informações do seu mesmo nível ou níveis inferiores ($\lambda(s) \geq \lambda(o)$). Isto impede que os usuários não tenham acesso a informações de níveis de segurança mais elevados que o seu.

propriedade estrela (no write down): um sujeito pode somente realizar operações que atendam a relação ($\lambda(s) \leq \lambda(o)$), desta forma, limita-se a possibilidade de fluxo de informações no mesmo nível do sujeito ou níveis superiores, impedindo o vazamento de informações para níveis de segurança abaixo do *clearance* do sujeito.

Uma das críticas ao modelo Bell-LaPadula é que existe uma tendência de que as informações fluam para os níveis de segurança superiores, além disso, existe o problema de escrita cega. Um sujeito, apesar de limitado pela propriedade *no read up*, pode escrever às cegas em níveis superiores, potencialmente danificando informações importantes. Uma

modificação na propriedade estrela pode evitar a danificação de informações, neste caso ($\lambda(s) = \lambda(o)$), limitando a capacidade de escrita do sujeito ao seu próprio nível.

O modelo Biba, por outro lado, preocupa-se basicamente com a integridade das informações. As propriedades do modelo Biba são consideradas o oposto das propriedades do modelo Bell–LaPadula, formando um dualismo. Considerando que ω simboliza o nível de integridade do sujeito e do objeto, podemos formular as seguintes propriedades:

integridade simples: a um sujeito só é permitido ler informações se a seguinte relação for atendida ($\omega(s) \leq \omega(o)$). Isto equivale a dizer que o sujeito só pode obter informações de níveis iguais, ou superiores ao seu. Observa-se então um fluxo de informação para baixo.

propriedade estrela da integridade (no write down): a escrita só será permitida se a seguinte relação for atendida ($\omega(s) \geq \omega(o)$). Complementando a primeira propriedade, o sujeito só tem permissão para escrever em objetos com nível de integridade menor ou igual ao seu.

Modelo de Acesso baseado em Papéis – RBAC

O modelo baseado em papéis (*role-based access control* – RBAC), flexibiliza o gerenciamento do controle de acesso através da adição de um componente que intermedeia sujeitos e permissões, o papel (SANDHU, 1998). O modelo considera a existência de quatro componentes básicos: usuários (o mesmo que sujeitos); papéis; permissões; e sessões;

Usuários podem ser seres humanos ou outros agentes autônomos tais como robôs, agentes de softwares e computadores. Permissões são os direitos de execução de uma ou mais ações sobre objetos do sistema. Os objetos podem representar arquivos, dispositivos (e.g. conexões de rede, bancos de dados). A concessão de permissões varia conforme a semântica do objeto, um banco de dados pode conceder permissões para leitura ou alteração de registros, um sistema operacional pode permitir a execução de um programa ou a impressão de um arquivo, etc.

Os papeis são os intermediários entre os usuários e as permissões. Ao invés de conceder permissões diretamente aos usuários, as permissões são concedidas aos papéis. Papéis representam funções distintas dentro do sistema (corporação), como por exemplo Administrador, Contador, ou Auditor. Cada papel tem o número de permissões mínimas necessárias para a execução correta da sua função, permitindo a implementação do prin-

cípio do menor privilégio – um usuário só deverá ter as permissões que precisa para realizar sua tarefa (BREWER; NASH, 1989).

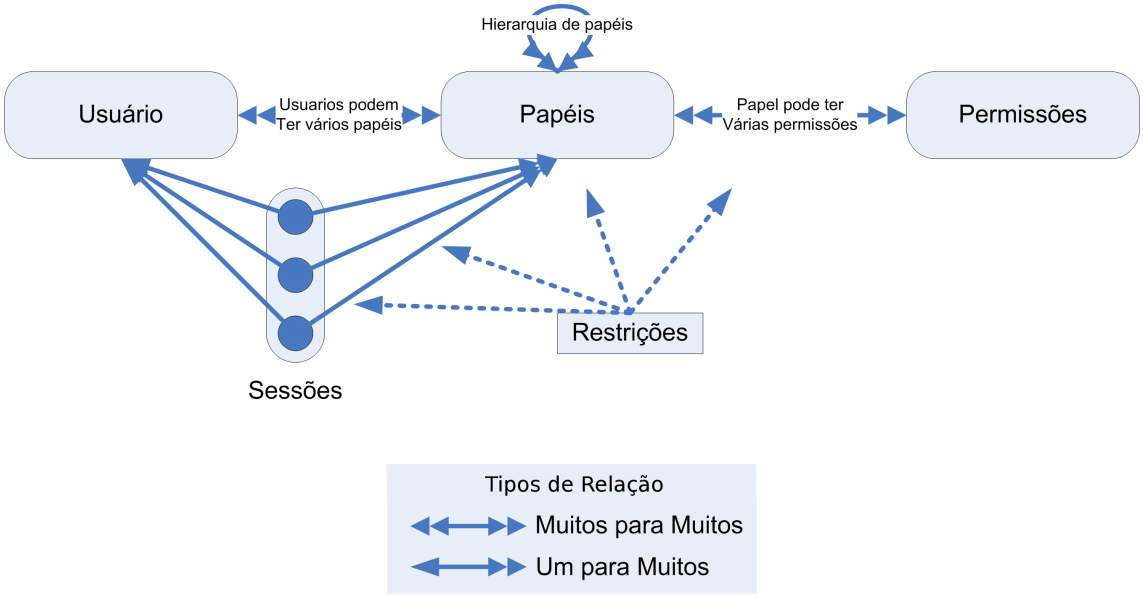

Figura 2.1: Elementos do modelo RBAC (SANDHU, 1998)

Usuários são ligados a um ou mais papéis. Quando um usuário acessa o sistema, estará iniciando uma sessão e, durante esta sessão, pode ter um ou mais papéis ativos. É possível ainda que um usuário mantenha várias sessões ativas paralelamente. Conforme a especificação do modelo, o usuário pode ter ou não o poder de decidir quais papéis ativar num dado momento.

O modelo RBAC permite que sejam impostas restrições na ativação de papéis, de modo a impedir a ativação de papéis com conflitos de interesse (BREWER; NASH, 1989). Os papéis podem ser organizados de maneira hierárquica, gerando uma cadeia de herança de permissões.

A Figura 2.1 apresenta os diversos elementos do modelo RBAC, usuários, papéis, permissões, restrições e sessões. Um usuário pode ter várias sessões abertas, e cada sessão pode ter vários papéis ativados. Um papel pode estar relacionado a muitas permissões, e uma mesma permissão pode estar relacionada a muitos papéis. Um usuário pode estar relacionado a muitos papéis, e um papel pode estar relacionado a vários usuários. Pode haver ainda uma relação de herança de múltiplos papéis. As restrições podem ser aplicadas

a diversas partes do modelo.

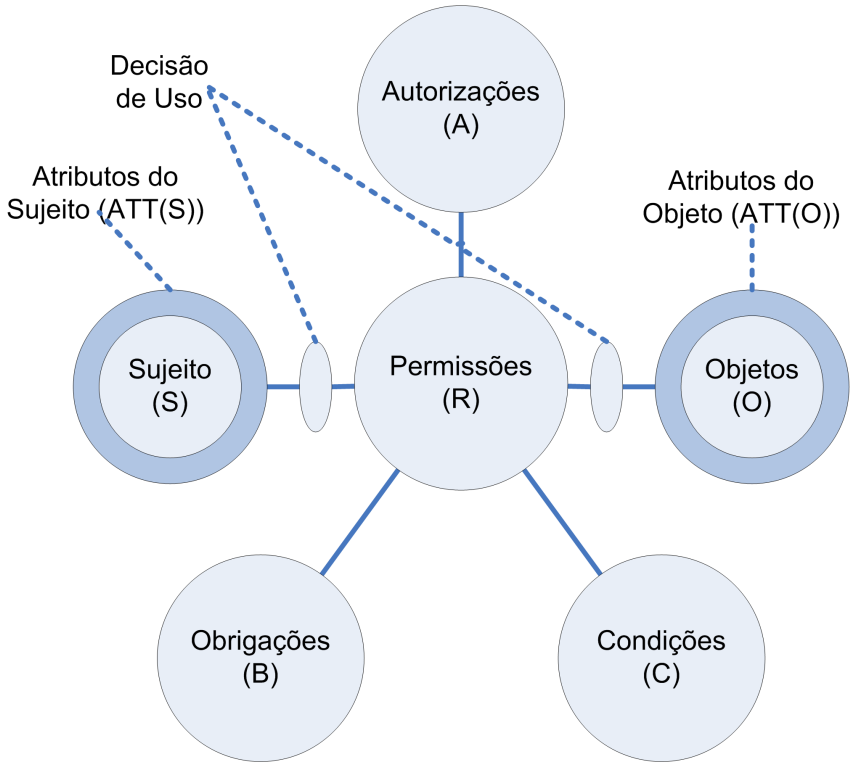

Figura 2.2: Componentes do $UCON_{ABC}$ (PARK; SANDHU, 2004)

A flexibilidade do RBAC reside no fato de que o gerenciamento de permissões não precisa mais ser individualizado. Quando um usuário deixa de ser responsável por uma função no sistema, basta removê-lo do papel, ou caso seja necessário adicionar uma permissão ao papel, basta conceder o direito ao papel ao invés de ter que conceder o direito individualmente para todos os membros associados ao papel.

2.2 Controle de Uso – $UCON_{ABC}$

O conceito de controle de uso, como apresentado em (PARK; SANDHU, 2004), vem de encontro as necessidades de controle de acesso das modernas aplicações e sistemas. Um modelo chamado $UCON_{ABC}$ foi elaborado buscando unificar diversas iniciativas tais como *trust management*, *digital rights management* (DRM), controle de acesso baseado em tarefas, e outros. O modelo $UCON_{ABC}$ é, na verdade, uma família de modelos focados em três fatores de decisão: autorizações (A), obrigações (B), e condições (C).

A Figura 2.2, apresenta os oito componentes básicos deste modelo de controle de uso:

sujeitos e objetos: Tem basicamente o mesmo significado utilizado na literatura de

controle de acesso tradicional.

atributos do sujeito e objeto: referem-se ao sujeito e ao objeto, podendo ser atualizados a qualquer momento, e sempre são considerados nas decisões de autorização. Apesar da identidade do sujeito ser importante, essa não é uma obrigatoriedade no modelo $UCON_{ABC}$, afim de não excluir o controle de acesso em serviços anônimos. Exemplos para atributos de sujeito: identidade, grupos, papéis, lista de capacidades, etc. Para objetos alguns exemplos possíveis são: identificador do proprietário, classes, listas de controle de acesso, entre outros.

Autorizações: são baseadas na avaliação de atributos do sujeito e objeto, e as políticas relacionadas. Diferente dos modelos tradicionais, considera que o acesso tem um caráter finito. A autorização pode ocorrer antes do acesso, mas também pode ser avaliada durante o processo de utilização do objeto.

Obrigações: São exigências que o sujeito deve cumprir para ter o seu acesso liberado ou então mantido. As obrigações podem ocorrer antes do uso (*pre*) ou durante (*ongoing*) o uso. Exemplos de pré-obrigação: fornecer informações pessoais em algum formulário antes de obter acesso a algum documento corporativo. Obrigações durante o acesso pode ser, por exemplo, manter janelas de propaganda abertas durante a utilização de algum serviço.

Condições: São fatores ambientais que influenciam o processo decisório. As condições podem ser avaliadas antes do uso (*pre*) ou durante o uso (*ongoing*). Exemplos de condições: nível de utilização do sistema, hora do dia, status de segurança do sistema, etc.

Permissões: referem-se às políticas de controle de uso em si, e permitem ao sujeito acessar o objeto alvo de um modo particular. A permissão pode variar conforme a semântica do objeto, podendo ser, por exemplo, uma política que permite leitura e escrita, ou consulta de saldo. Funções de decisão de uso (indicadas na figura como *decisão de uso*), utilizam os atributos do sujeito e do objeto, autorizações, obrigações e condições, para avaliar se as políticas (permissões) permite ou negam o acesso requerido pelo sujeito.

O modelo $UCON_{ABC}$ possui mutabilidade de atributos em função do acesso, permitindo a aplicação de políticas de acesso baseadas em histórico como, por exemplo, autorizações baseadas em RBAC.

A atualização dos atributos mutáveis, seja de sujeitos ou de objetos, podem ocorrer antes, durante, ou após a utilização. A alteração dos atributos só pode ser afetada por autorizações e obrigações, as condições do sistema não influem nos mesmos.

A continuidade do controle de acesso é um aspecto fundamental na concepção deste modelo, sendo que os diversos componentes do modelo UCON$_{\text{ABC}}$ podem ser avaliados em diversos momentos do uso, inclusive após o mesmo.

A Tabela 2.2 apresenta uma relação com as combinações dos diversos componentes e possíveis atualizações de atributos. A coluna '0' mostra todos os casos de ocorrências de autorizações, obrigações e condições quando a utilização é imutável. As colunas '1', '2', e '3', informam se existe a possibilidade de atualização de atributos (antes, durante, ou após o uso) com pré-autorização, autorização durante o uso, pré-obrigação, obrigação durante o uso, pré-condições, e condições durante o uso. É importante notar que o modelo não define a periodicidade com que as verificações continuadas devem ser executadas.

Devido à sua expressividade e flexibilidade, o modelo UCON$_{\text{ABC}}$, engloba controles de acesso tradicionais tais como o discricionário, mandatório e baseado em papéis, além de suportar controles modernos tais como *trust management*, DRM, e outros. É neste sentido que o UCON$_{ABC}$ é tido como uma família de modelos, pois cada componente permite diversas configurações e opções. O modelo é focado no processo de controle de acesso e não prevê como se deve realizar a administração e a delegação, por exemplo.

Tabela 2.2: Os 16 Modelos ABC básicos

	0 (imutável)	1 (pré-atual.)	2 (atual. durante)	3 (atual. após)
Pre Autor.	Sim	Sim	Não	Sim
Autor Dur.	Sim	Sim	Sim	Sim
Pre Obrig.	Sim	Sim	Não	Sim
Obrig. Dur.	Sim	Sim	Sim	Sim
Pre Cond.	Sim	Não	Não	Não
Cond. Dur.	Sim	Não	Não	Não

2.2.1 Monitor de Referência

Do ponto de vista de classificação arquitetural o UCON$_{\text{ABC}}$ pode utilizar monitores de referência no lado do servidor (*server-side reference monitor*, SRM), no lado do cliente (*client-side reference monitor*, CRM), ou ambos.

O monitor de referência para controle de uso possui algumas diferenças do monitor

de referência do padrão ISO/IEC 10181–3. A Figura 2.3, apresenta os componentes conceituais do monitor de referência para controle de uso.

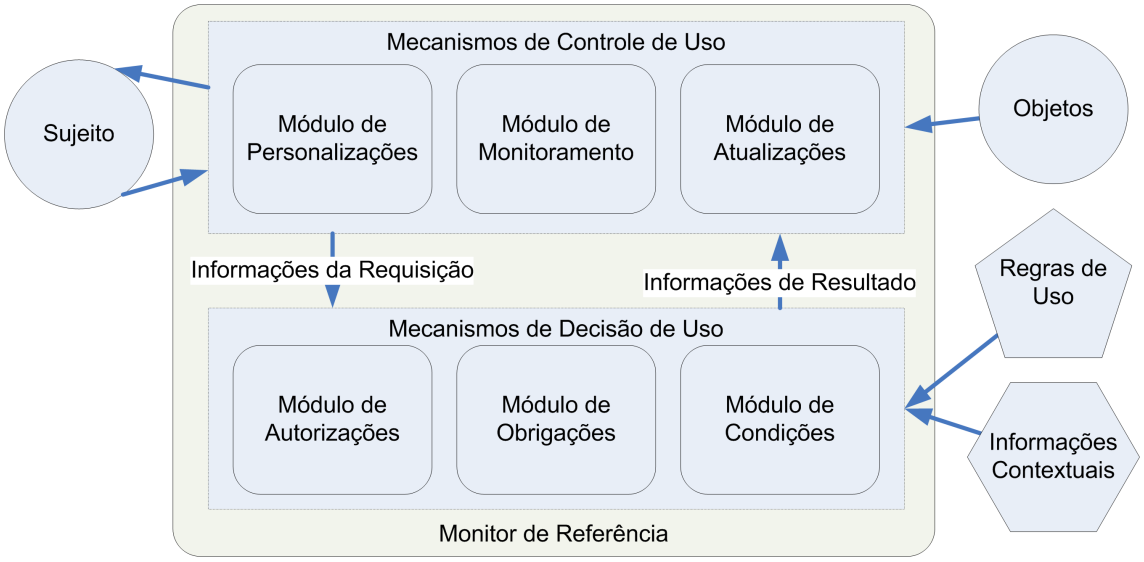

Figura 2.3: Monitor de referência do UCON$_{ABC}$ (PARK; SANDHU, 2004)

O monitor de referência divide-se em duas facilidades, uma para controle de uso, e outra para decisão de uso, e cada uma subdivide-se em alguns módulos. O módulo de autorizações tem a função de avaliar as regras de autorização com base em atributos do sujeito e do objeto, juntamente com outras regras de uso. O módulo de autorizações pode, ainda, retornar meta-dados para personalização dos objetos, operação esta realizada através do módulo de personalizações. O módulo de condições monitora o ambiente para certificar-se de que as condições de uso estão sendo atendidas. O módulo de obrigações verifica se as obrigações foram e/ou estão sendo cumpridas, e para isso pode utilizar o modulo de monitoramento. O módulo de atualizações mantém os atributos de objetos e de sujeitos atualizados.

2.3 Conclusão

Neste capítulo foram apresentados brevemente os modelos de controle de acesso tradicionais, tais como o modelo discricionário (DAC), modelo mandatório (MAC), e o modelo

baseado em papéis (RBAC), juntamente com seus conceitos fundamentais e mecanismos utilizados para sua implementação. Nenhum destes modelos é capaz de avaliar as políticas além do momento da requisição inicial de acesso, o que os caracteriza como controles estáticos. A incapacidade destes modelos para expressar controles mais abrangentes foi o que motivou o desenvolvimento do modelo de controle de uso.

O modelo de controle de uso (UCON_{ABC}) oferece uma maior expressividade, possibilitando a agregação de informações sobre o estado do ambiente, do recurso, e dos sujeitos, na avaliação das políticas, além de viabilizar uma avaliação constante e permanente durante o uso dos recursos protegidos.

Neste capítulo apresentamos também o modelo de controle de uso, UCON_{ABC}, e seus principais conceitos – mutabilidade de atributos, avaliação contínua de políticas, autorizações, obrigações, e condições. UCON_{ABC} apresenta-se como um modelo consideravelmente mais apropriado para as necessidades dos modernos ambientes computacionais.

No próximo capítulo abordaremos alguns trabalhos que visam melhorar o gerenciamento de controle de acesso em ambientes computacionais colaborativos. Alguns dos trabalhos que serão discutidos ainda tem suas raízes em modelos de controle de acesso tradicionais, entretanto fornecem idéias interessantes para o gerenciamento de informações de controle de acesso.

3 Controle de Recursos em Sistemas Colaborativos Distribuídos

Neste capítulo apresentamos os principais trabalhos relacionados ao controle de recursos em ambientes colaborativos distribuídos. Como será apresentado nas próximas seções, os trabalhos dividem-se basicamente em trabalhos que estudam a aplicação de controle de uso em ambientes distribuídos e trabalhos que tentam organizar o gerenciamento das informações de autorização em ambientes distribuídos.

3.1 Controle de Uso para Sistemas Computacionais Colaborativos

Um framework para controle de uso em sistemas computacionais colaborativos foi proposto em (ZHANG et al., 2006), tendo como principais objetivos atender as necessidades de escalabilidade, dinamicidade e controle fino de autorização presentes em ambientes computacionais colaborativos.

Seguindo a metodologia *Objective–Model–Architecture–Mechanism* (OM–AM), existe uma separação em quatro camadas a fim de reduzir a distância entre quais são os objetivos de segurança e como serão atingidos. As camadas *Objective* e *Model* são responsáveis pela definição das políticas de segurança, enquanto as camadas *Architecture* e *Mechanism* especificam os mecanismos de implementação das políticas. Mais informações sobre o método OM–AM podem ser encontradas em (SANDHU, 2000).

Atributos do sujeito podem incluir informações como, por exemplo, grupos aos quais o sujeito pertence, papéis que desempenha dentro da organização virtual, seu nível de *clearance*, além de informações específicas de uma aplicação particular como a quota de utilização de um determinado recurso e grupos de conflito de interesse. Atributos de

objeto podem incluir propriedades como o tipo, proprietário, etc., assim como informações específicas de aplicação tais como estado de utilização, entre outros.

A Figura 3.1 apresenta a framework proposta por Zhang e seus colegas (ZHANG et al., 2006). Tal arquitetura apresenta tipicamente três componentes principais: plataformas de usuário (PU), provedores de recursos (PR), e um repositório de atributos (RA). O repositório de atributos é um serviço centralizado para armazenar os atributos dos sujeitos e dos sistemas. Os atributos de objeto são armazenados no monitor de uso (MU), localizado em cada PR. As autoridades responsáveis por identidade e atributos externos não são incluídas a fim de simplificar a figura.

Uma sessão de uso inicia-se com a requisição do sujeito, gerada na PU e enviada ao PR através de um *proxy* localizado no lado do cliente (passo 1). O *proxy* é responsável por intermediar a comunicação entre a PU e o PR. Os atributos persistentes do sujeito são enviadas por esse ao PR juntamente com a requisição gerada no passo 1.

O *Gate Keeper* intercepta a requisição e passa os atributos persistentes do sujeito ao *Policy Decision Point* (PDP) (passo 2). O PDP é o responsável por decidir se a requisição é permitida ou negada. O PDP então busca os atributos mutáveis armazenadas no RA (passos 3 e 4). No passo 5 o PDP obtém os atributos do objeto. Com base nos atributos e nas políticas aplicáveis, armazenadas localmente num repositório de políticas, o PDP envia a decisão ao *Policy Enforcement Point* (PEP). O PEP é responsável por garantir que a decisão será executada corretamente no ambiente de provedor.

O PDP é responsável, ainda, por atualizar os atributos do sujeito e do objeto como efeito colateral das decisões de uso. Os passos 7 e 8 apresentam a atualização de tais atributos. A cada nova requisição os atributos são verificados no RA e no MU.

A aquisição dos atributos atuais é um requisito essencial para a correta aplicação das políticas de controle de uso. A proposta desta arquitetura é a utilização de uma abordagem híbrida de aquisição conforme o tipo de atributo. Para a aquisição de atributos persistentes é utilizada uma abordagem *push mode*. O sujeito apresenta os atributos ao PDP, como os atributos são imutáveis a avaliação é feita somente uma vez. Objetos também podem ter atributos persistentes mantidos pelo PR. O gerenciamento de tais atributos não é considerado na proposta.

Os atributos mutáveis são obtidos via *pull mode*. O PDP busca os atributos no RA central e MU. Quando os atributos são modificados o PDP é informado da mudança, desencadeando a reavaliação das políticas.

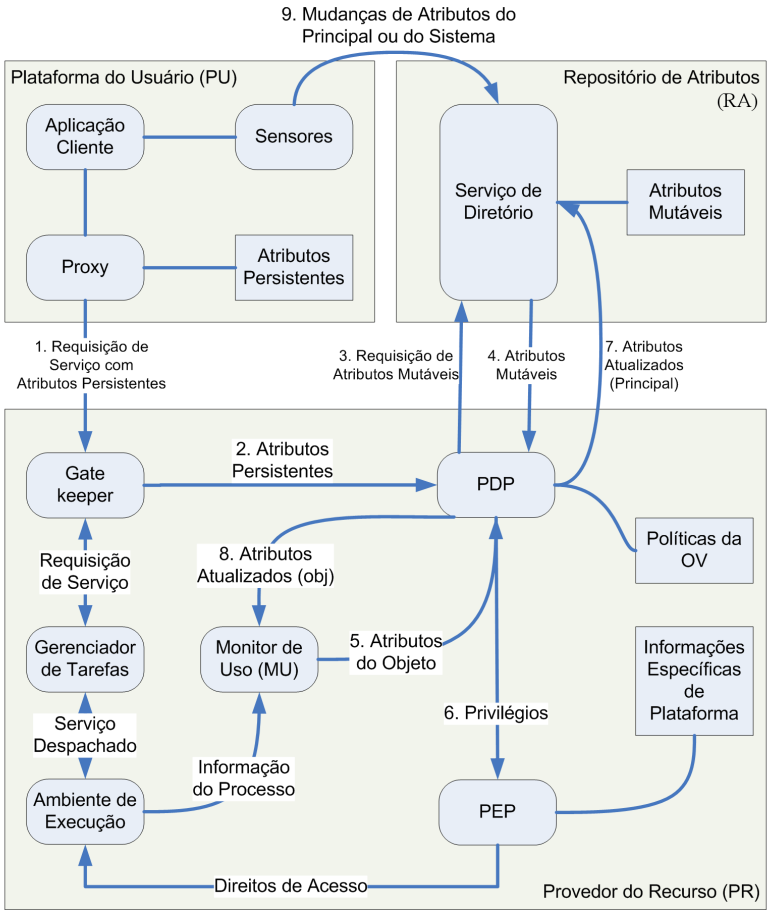

Figura 3.1: Arquitetura de autorização baseada em uso (ZHANG et al., 2006)

A atualização dos atributos do sujeito e do objeto podem ser realizadas antes, durante e depois do uso. As atualizações antes do uso são desencadeadas por uma requisição de acesso. Atualizações durante a sessão de uso podem ser desencadeadas por diversos eventos, como por exemplo temporizadores ou mudanças de estado do sistema. Nas atualizações durante o uso, o PDP é informado da ocorrência de tais eventos pelo MU ou outro mecanismo, atualizando os atributos e reavaliando as políticas para permitir a continuidade da decisão. A finalização da sessão de uso por parte do sujeito, ou através da revogação da permissão pelo PDP, pode desencadear a atualização dos atributos do objeto ou do sujeito, os valores são propagados para o MU e o RA.

Os atributos do sistema, necessários para avaliações de condição, são monitorados e atualizados por componentes funcionais do sistema e não são incluídos na arquitetura proposta. Na Figura 3.1 os "sensores" na PU representam tais componentes funcionais.

A contínua aplicação das políticas é obtida através de um sistema de notificação entre o RA e os diversos PRs. Como a mudança de um atributo por um PR deve ser sincronizada com os outros PRs, o RA funciona como uma ponte para a propagação em

tempo real das alterações. Quando um PR faz uma requisição de atributos ao RA, o RA registra a requisição juntamente com os nomes de atributos. Ao receber uma modificação de atributos, o RA verifica a sua lista de requisições e propaga as mudanças para os PRs envolvidos. Ao receber a notificação da mudança o PDP pode reavaliar a política e informar o PEP da decisão tomada, permitindo ou revogando a permissão de acesso. No caso de atributos de objeto, o responsável por notificar o PDP é o MU, que é implementado localmente.

A autenticidade dos atributos do sujeito dependem da autenticação do sujeito, da ligação entre a sua identidade e seus atributos, e da integridade dos valores dos atributos. Um mecanismo para ligar a identidade e os atributos é proposta em (PARK; SANDHU, 2000), e é aplicável nesta arquitetura. Os atributos mutáveis são mantidos pelo RA, o qual é responsável por garantir a autenticidade e integridade dos mesmos, o que implica em uma relação de confiança entre o RA e os PRs. Para o caso da autenticidade e integridade dos atributos de objeto é possível aproveitar a infra-estrutura local de cada PR para atender esses requerimentos.

O controle de atualizações concorrentes não é discutido na proposta desta arquitetura, mas sugere-se a utilização de mecanismos tradicionais para controle de concorrência.

3.1.1 Implementação do Protótipo

A arquitetura proposta foi implementada na forma de um protótipo baseado em web, permitindo que um grupo de desenvolvedores compartilhem e desenvolvam código fonte de uma aplicação de modo colaborativo a partir de diferentes locais.

O provedor de recurso oferece o serviço de um sistema de controle de versões *Subversion* através de um servidor web *Apache*. O repositório de atributos é implementado com um servidor LDAP. A Figura 3.2 apresenta a arquitetura do protótipo. A comunicação entre a PU, AR e PR é feita utilizando SSL, criando canais seguros com autenticação mútua. O Sensor na PU é um programa que simula a detecção da localização da PU.

O controle de acesso do *Subversion* é baseado em ACL e não faz referência a nenhum atributo mutável. O PEP é implementado na forma de um módulo do servidor *Apache* chamado *mod_authz_ucon*, e faz uma interface entre o servidor e o PDP. A cada requisição recebida para o recurso, o módulo encaminha a requisição para o PDP, e libera o acesso caso o PDP permita.

A representação das políticas foi feita utilizando *extensible access control markup lan-*

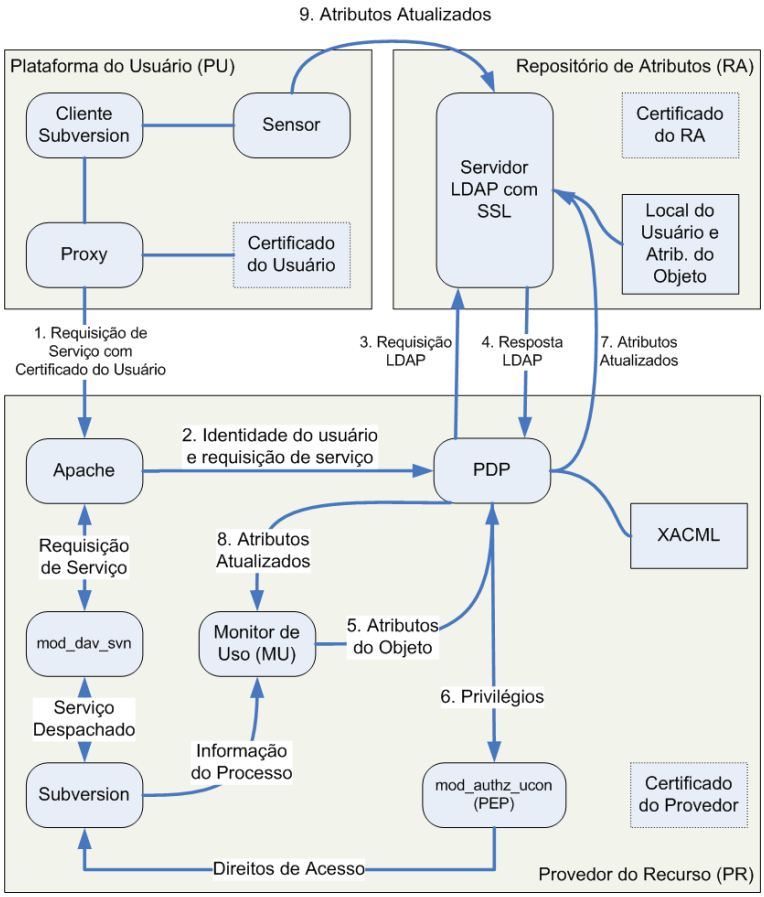

Figura 3.2: Arquitetura do protótipo (ZHANG et al., 2006)

guage (XACML). Esta implementação não representa exatamente uma política abstrata de UCON$_{ABC}$.

O PDP pode identificar a política da organização virtual correta (e.g. organização virtual 1) através da checagem do nome distinto presente no certificado do sujeito. Esta abordagem permite que um PR participe de várias organizações simultaneamente.

Para a autenticidade e integridade dos atributos o RA, MU e PDP utilizam autenticação mútua através de SSL. Consequentemente, o protocolo de autenticação utiliza certificados X.509 para restringir as comunicações a uma organização virtual em particular.

O protótipo foi testado em dois cenários: controle de acesso baseado em localização do sujeito (condições do UCON$_{ABC}$, pois opera sobre atributos de ambiente), e controle de acesso baseado em tarefa (autorizações do UCON$_{ABC}$, pois opera sobre atributos do objeto). No primeiro cenário existem duas organizações virtuais, e as requisições são permitidas ou negadas conforme a localização do sujeito, se estiver nas dependências

de uma organização virtual qualquer requisição para dados da outra organização serão negados. No segundo cenário os diretórios compartilhados podem ser bloqueados por um sujeito para evitar alterações durante testes do código fonte, esse bloqueio é representado como um atributo de objeto. Quando uma requisição é recebida o PDP verifica este atributo, e caso encontre-se ativo o acesso é negado. Todas as políticas são especificadas utilizando XACML.

3.1.2 Considerações

O trabalho propõe uma arquitetura conceitual que abrange autorizações, obrigações e condições. O protótipo implementado, no entanto, somente abrange autorizações e condições. A linguagem utilizada para a especificação das políticas (XACML) precisa ser ampliada, para que possa, assim, representar as obrigações. Tal extensão do XACML foi deixada para trabalhos futuros.

O PDP tem sua função sobrecarregada, sendo que o mesmo tem a responsabilidade de avaliar as políticas e atualizar atributos do sujeito e do objeto. A função do PDP deveria ser única e exclusivamente a de avaliar as políticas, devendo a atualização de atributos ser realizada por outro componente.

O protótipo ainda apresenta uma aplicabilidade pouco genérica. O mesmo foi desenvolvido para o caso específico do compartilhamento de arquivos através do *subversion*, via *apache*. A mutabilidade dos atributos é pouco explorada, já que os únicos atributos mutáveis são a localização do cliente e *locks* de arquivo.

3.2 Controle Contínuo de Uso para Serviços Computacionais em Grid

Em (MARTINELLI; MORI; VACCARELLI, 2005) é proposto um mapeamento dos conceitos do UCON$_{ABC}$ para organizações virtuais baseadas em computação *grid*, mais especificamente para serviços computacionais. O conceito de serviço computacional é um ambiente para execução de aplicações dos sujeitos da organização virtual. As implementações atuais de ambientes de *grid* fornecem somente um controle tradicional dos recursos compartilhados. Visando contribuir para a solução deste problema, Martinelli e seus colegas propuseram um componente para aumentar a segurança em ambientes *grid* através de uma monitoração fina, contínua e baseada em histórico.

A monitoração fina refere-se às interações da aplicação com o ambiente de execução. A política de segurança detalha como a aplicação pode se comportar, definindo todas as operações permitidas. As operações podem incluir alocação de memória, acesso a arquivos, chamadas de sistema, etc.

A execução da aplicação é monitorada do princípio ao fim da execução. Todas as interações entre a aplicação e os recursos compartilhados são monitoradas durante a sua execução. Isto permite a continuidade dos controles e a revogação do direito de execução em qualquer momento em que a política de segurança seja violada. Os controles continuados podem ser influenciados por condições externas à aplicação, como por exemplo o nível de carga do sistema.

Um controle histórico significa que todas as operações relevantes para a avaliação da política de segurança são registrados, desde o princípio da execução. A política pode definir dependências entre operações (obrigações), exigindo que o histórico de execução da aplicação seja analisado para que uma decisão seja tomada. Operações permitidas por padrão e que não são obrigações não são registradas no histórico.

A política de segurança define um ambiente restrito para a execução da aplicação. Qualquer operação não prevista na política é negada por padrão. A política pode ser definida no local do recurso, global (e.g. definida pela organização virtual), ou uma combinação de ambos. É possível aplicar políticas distintas ao mesmo recurso de acordo com o sujeito (e.g. aplicar políticas de acordo com o nível de *confiança* do sujeito). Uma linguagem própria foi desenvolvida para a especificação da política de controle de acesso, em termos de seqüências de chamadas de sistema, com seus parâmetros e resultados.

Um conjunto de limites sobre o uso dos recursos locais, juntamente com um conjunto de regras de comportamento, compõem a política de segurança. Os limites podem representar o tanto de memória ocupado pela aplicação, o tempo de processamento necessário, número de processos ou *threads*, entre outros. As regras de comportamento definem as chamadas de sistema que podem ser invocadas pela aplicação, dependências na ordem de invocação, o valor dos parâmetros da chamada de sistema, e os resultados de tais invocações. Tais regras podem ainda incluir condições, representadas por predicados que devem ser atendidos durante a execução. Alguns limites podem ser derivados da própria requisição de execução.

A Tabela 3.1 apresenta um exemplo de especificação de política de controle de uso. As linhas MAX_CPU_TIME e MAX_MEMORY são limites do tempo de uso do processador e memória (em megabytes). AF_INET, STREAM, TCP, AH, CF e READ são constantes.

...	
MAX_CPU_TIME=100	
MAX_MEMORY=64	
CR:=false	
$[eq(x_1, AF\_INET), eq(x_2, STREAM), eq(x_3, TCP)]$.	(p_1)
$socket(x_1, x_2, x_3, sd)$.	
$[eq(x_5, sd), eq(x_6, AH)]$.	(p_2)
$connect(x_5, x_6, x_7, x_8)$.	
$i([eq(x_9, sd), eq(CR, false)]$.	(p_3)
$send(x_9, x_10, x_11, x_12, x_13)$ or	
$[eq(x_14, sd)]$.	(p_4)
$recv(x_14, x_15, x_16, x_17, x_18))$;	
$[eq(x_20, sd)]$.	(p_5)
$close(x_20, x_21)$	
$[eq(x_1, CF), eq(x_2, READ)]$.	(p_6)
$open(x_1, x_2, x_3, fd)$.	
CR:=true.	
$i([eq(x_5, fd)]$.	
$read(x_5, x_6, x_7, x_8))$;	
$[eq(x_9, fd)]$.	(p_8)
$close(x_9, x_10)$	

Tabela 3.1: Política de Controle de Uso (MARTINELLI; MORI; VACCARELLI, 2005)

CR sinaliza a abertura de um arquivo critico, nomeado em CF. A primeira regra (p_1) verifica se a variável x_1 equivale ao valor de AF_INET e se x_2 e x_3 são STREAM e TCP, respectivamente. Em caso positivo, a aplicação pode chamar a função *socket* com os parâmetros indicados na política. A chamada da função armazena um descritor em *sd*, que será avaliado em p_2, possivelmente liberando a invocação da função *connect*. Em p_3 existe o operador iterativo, a aplicação pode alternar entre as chamadas *send* ou *recv*, desde que CR continue com valor falso. A execução de *send* ou *recv* depende da obrigação em p_3, que pode ser afetado a qualquer momento pela regra em p_6, caso o aplicativo abra o arquivo crítico, CR se torna verdadeiro e *send* e *recv* são proibidos. Apesar de não aparecer neste exemplo, a linguagem de especificação permite ainda que se executem operações paralelas, de forma síncrona ou assíncrona.

3.2.1 Implementação

O protótipo foi implementado como um monitor, ativado por *hooks* inseridos em uma máquina virtual Java (JVM – *Java Virtual Machine*) desenvolvida pela IBM, chamada *jikes Research Java Virtual Machine* (RVM). Os *hooks* consistem em código que envolve as chamadas de sistema realizadas pela máquina virtual, e também chamadas de gerenciamento como manipulação de *threads*, gerenciamento de memória, etc., e que ativa o monitor, chamado *Gmon*, quando uma destas chamadas é executada pela máquina virtual para executar a aplicação. Quando o *Gmon* é ativado a máquina virtual é suspensa, a operação pode então ser avaliada e caso seja permitida, a máquina virtual retorna a execução e a invocação é feita, caso contrário a aplicação é parada e um erro pode ser retornado.

Quando a máquina virtual é iniciada, *Gmon* carrega todas os limites e regras contidos na política de controle de uso, e cria uma representação das regras utilizando *autômatos de segurança* (SCHNEIDER, 2000). Nesta representação, cada nó do autômato é representado um estado possível da aplicação. O vértice entre um nó A e B representa uma chamada de sistema que pode levar a aplicação, quando no estado A, para o estado B. Os vértices possuem ainda predicados agregados que devem ser avaliados para verificar a possibilidade de execução dos seus respectivos vértices. O estado inicial da aplicação é representado pelo nó que não possui nenhum vértice incidente. O estado atual de uma aplicação é considerado como o nó cujo vértice incidente acabou de ser executado, e o estado global da aplicação é o conjunto dos estados de todos os autômatos.

Gmon utiliza esta representação para decidir se uma chamada de sistema é permitida pela política de controle de uso. Uma chamada *sc* é permitida se um dos autômatos inclui ao menos um vértice saindo do estado atual e indo para outro nó, e se os predicados deste vértice foram avaliados como verdadeiros. Após a execução de *sc*, o estado atual aponta para o novo nó. Cada autômato pode possuir mais do que uma instância, potencialmente representado ocorrências separadas do mesmo comportamento.

A máquina virtual incrementada com o *Gmon* é inserida no ambiente de *grid Globus Toolkit* (FOSTER; KESSELMAN, 1997), versão 3 (GT3). Quando o GT3 recebe a requisição para execução de um serviço computacional java na RVM, faz os procedimentos de autenticação e autorização rotineiros e inicia a máquina virtual, passando a requisição para o *Gmon*. Os limites são derivados da requisição, e as regras são carregadas da política, o resto da operação segue o descrito nos parágrafos anteriores.

3.2.2 Considerações

A utilização de uma máquina virtual, apesar de facilitar a implementação dos controles, limita a aplicabilidade do conceito de controle de uso a somente serviços computacionais, especificamente programados para o ambiente de execução da JVM/RVM. Considerando a imensa variedade de recursos que podem ser compartilhados em uma organização virtual, esta opção é limitada.

A linguagem de especificação de políticas de segurança não é tão expressiva quanto preciso para representar todas as políticas da família UCON_{ABC}. Não é mencionado em momento algum a mutabilidade dos atributos do sujeito e do objeto, aliás, não está bem claro os papéis do sujeito e do objeto na política, o que limita sensivelmente sua aplicabilidade para outras situações. Não existe distinção entre controles que são feitos antes, durante, ou após a execução. A utilização de uma linguagem não padronizada possui ainda o problema de dificultar a interoperabilidade, caso outras aplicações precisem processar tais regras.

Se considerarmos o número de chamadas de sistema diferentes, e a variedade na seqüência, que podem ser feitas por uma aplicação, pode-se concluir que a criação de políticas explícitas, baseadas no tipo e ordem de invocações de chamadas de sistema, torna-se uma tarefa computacional bastante custosa. Existe a possibilidade de agrupar chamadas de sistema em operações mais abstratas, no entanto tal recurso não é aprofundado no trabalho.

A abordagem utilizada para lidar com as violações da política de segurança se mostra inflexível, fato este observado pelos próprios pesquisadores em seu artigo. A simples revogação imediata do acesso é inadequada para a maioria dos cenários reais de utilização. É necessário criar mecanismos que permitam o tratamento adequado da violação da política, permitindo que, por exemplo, o serviço continue sendo fornecido em modo degradado, ou que sejam emitidas notificações, ou então que o serviço possa ser revogado mas de modo adequado, permitindo que não sejam produzidas inconsistências em dados importantes. Qual será a punição por violação da política de segurança deve ser algo possivelmente estabelecido nos contratos da organização virtual, e o modo de revogação do acesso deve ser dependente do domínio da aplicação.

3.3 VOMS – Sistema de Autorização para Organizações Virtuais

O sistema *Virtual Organization Membership Service*, proposto em (ALFIERI et al., 2004), visa estruturar a administração de informações de autorização em ambientes de *grid*. Dois conceitos são utilizados no trabalho: organização virtual e provedor de recursos. Uma organização virtual é uma entidade abstrata que reúne sujeitos, instituições e recursos em um mesmo domínio administrativo. Provedor de recurso é uma entidade que oferece algum tipo de recurso (e.g. processadores, redes, armazenamento) a outras partes, segundo acordos firmados entre o provedor e a parte interessada (possivelmente uma organização virtual).

Para simplificar o gerenciamento das informações de autorização, essas são classificadas em duas categorias: informações a respeito da relação do sujeito com a organização virtual (e.g. grupos a que pertence, lista de *capabilities*, papéis, etc.); informações que definem o que um sujeito pode fazer no provedor de recurso, de acordo com sua relação com uma organização virtual específica. O primeiro tipo de informação é gerenciado pela organização virtual, enquanto a segunda categoria é gerenciada localmente no provedor de recursos.

A autorização é baseada em políticas escritas pela organização virtual e nos acordos firmados com os provedores de recurso, que impõe a política de autorização localmente. Uma organização virtual pode ter vários grupos e sub-grupos, e seus sujeitos podem pertencer a qualquer número de grupos. Sujeitos podem ser caracterizados por vários papéis, grupos, e *capabilities*. Restrições podem ser impostas para que Papéis sejam ativados, e *capabilities* sejam utilizadas, em momentos definidos ou em intervalos periódicos. A imposição destes atributos é responsabilidade do provedor do recurso, e está definida nos acordos estabelecidos entre a organização virtual e o provedor, sendo que este tem a autonomia para impor políticas locais conjuntamente com as políticas da organização. Em caso de conflitos entre as políticas a política local tem prioridade.

O sistema VOMS está focado na administração das informações de autorização no escopo da organização virtual. Para isso funciona essencialmente como uma interface com um banco de dados onde todas as informações dos sujeitos estão armazenadas. O sistema é composto basicamente pelos seguintes componentes, conforme apresentados na Figura 3.3:

Servidor de Usuário: recebe requisições de um cliente e retorna informações sobre um

sujeito;

Cliente de Usuário: comunica-se com o servidor, apresenta um certificado do sujeito, e obtém a lista de atributos do sujeito;

Cliente Administrativo: utilizado pelos administradores da organização virtual para gerenciar os dados;

Servidor Administrativo: recebe as requisições do cliente e atualiza o banco de dados.

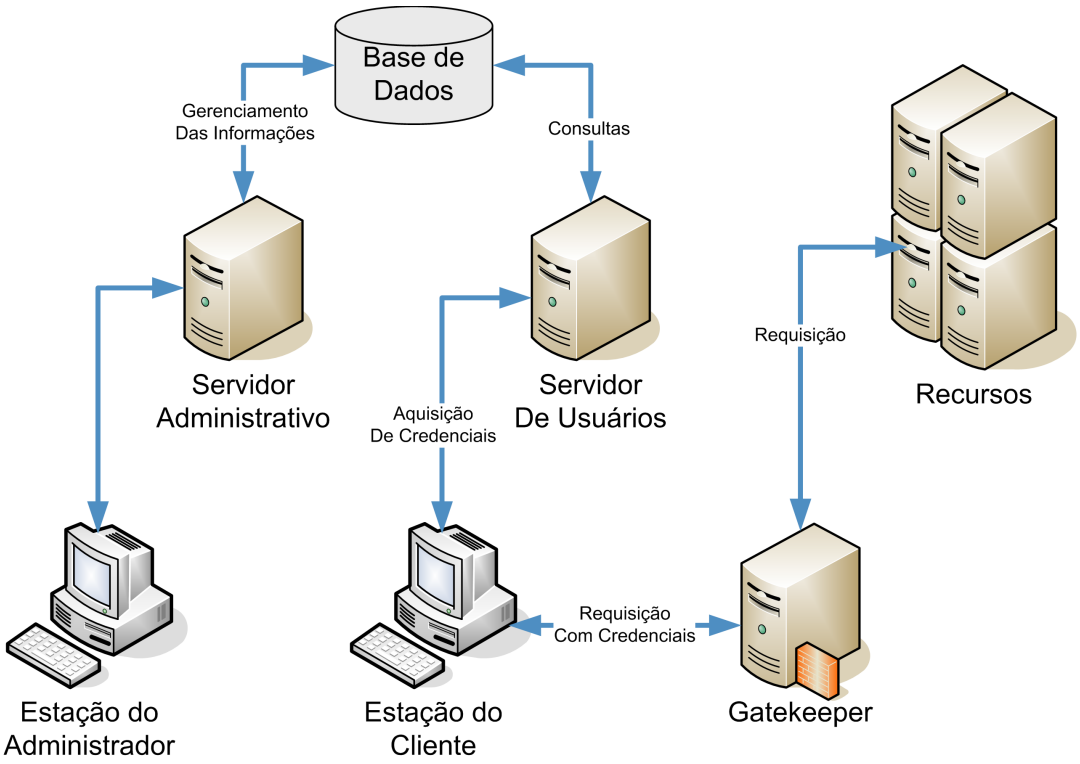

Figura 3.3: Visão conceitual do sistema VOMS

Como VOMS foi desenvolvido dentro do contexto de computação *grid*, utiliza-se dos mecanismos de autenticação e delegação do Globus Toolkit (FOSTER; KESSELMAN, 1997). Mais especificamente, é criado um comando chamado *voms-proxy-init* que cria um certificado de *proxy* para o sujeito, do mesmo modo que *grid-proxy-init* exceto pelo fato de que adiciona as informações de autorização da organização virtual ao certificado gerado. As informações são assinadas pelo servidor VOMS.

O sujeito pode contactar diversos servidores VOMS quanto for necessário. Um *Gate keeper* é o componente no provedor de recurso que avalia as requisições de uso e determina se as mesmas permitidas ou não (monitor de referência). Quando deseja utilizar um recurso o sujeito apresenta o certificado de *proxy* ao *Gate keeper*, o qual verifica a validade

do certificado e as informações da organização virtual. As informações do VOMS são inseridas numa extensão do certificado de *proxy*, podendo ser utilizado em *Gate keepers* que não entendem VOMS, mantendo assim a compatibilidade.

Interfaces de administração são fornecidas para gerenciar o servidor. O servidor pode ser contactado utilizando protocolo *SOAP*. As rotinas do servidor são classificadas em três categorias: *core* – para fornecer funcionalidade básica aos clientes; *admin* – para os métodos administrativos do banco de dados VOMS; e *history* – para agrupar as funcionalidades de registro de eventos e responsabilidade (*accountability*).

A proposta do VOMS contribui para a estruturação do gerenciamento das informações da organização virtual, assim como para resolver a questão de flexibilidade e escalabilidade existente no modelo *pull*. Entretanto, como o sistema foi desenvolvido para trabalhar no contexto do ambiente de *grid* EDG, a definição de políticas e imposição dos controles segue um sistema tradicional, deixando a desejar na questão de *continuidade* dos controles, *mutabilidade* dos atributos, e outros aspectos fundamentais para a aplicação dos conceitos do UCON_{ABC}.

3.4 CAS – Serviço de Autorização Comunitário

Na administração de organizações virtuais três grandes problemas foram identificados: baixa escalabilidade; flexibilidade e expressividade reduzidas; dificuldades para representar a hierarquia das políticas de autorização (PEARLMAN et al., 2002).

A baixa escalabilidade encontrada refere-se à dificuldade crescente de gerenciar sujeitos, grupos, políticas, etc. Nas abordagens em que cada provedor de recurso precisa realizar diversas operações para refletir as mudanças na organização virtual, o custo tende a crescer proporcionalmente ao número de provedores e participantes.

Baixa flexibilidade e expressividade advém do fato de que várias políticas da organização virtual são peculiares e variáveis ao longo do tempo. Aplicar estas políticas de uma maneira distribuída cria dificuldades (e.g. como representar a política localmente, caso o sistema de autorização local seja incapaz de representar tal política). As políticas de autorização podem ser hierárquicas, ou seja, podem haver políticas em nível de organização virtual, em nível de instituição, e em nível de recurso.

Para reduzir os custos de administração de uma organização virtual, aumentar sua escalabilidade, flexibilidade expressividade, e permitir a hierarquização das políticas, um

sistema de autorização comunitário foi proposto, chamado CAS.

O sistema CAS utiliza a infra-estrutura de segurança para *grid* do Globus Toolkit, chamada GSI, como mecanismo para fornecer autenticação e delegação. GSI é um conjunto de bibliotecas e ferramentas de software que permitem a sujeitos e aplicações acessar recursos de forma segura, focado especialmente em autenticação e proteção de mensagens. Com GSI é possível fornecer autenticação *single sign-on*[1] e delegação de credenciais.

Peça fundamental para o mecanismo de *single sign-on* são as credenciais *proxy*. Para criar credenciais *proxy*, o sujeito gera um par de chaves temporárias (do tipo assimétricas) e então gera um certificado, que é então assinado pela chave privada do sujeito (a permanente). Deste modo o sujeito cria um certificado com duração curta vinculado à sua identidade. Quando o sujeito apresenta o certificado *proxy* a um provedor de recurso, este pode verificar a validade do certificado permanente do sujeito, e se o certificado *proxy* foi assinado com a chave privada permanente (prova de autenticidade). É através dos certificados *proxy* que o servidor CAS delega as suas permissões aos sujeitos (as permissões são de propriedade da organização e não de sujeitos específicos). CAS estende este certificado *proxy* para carregar informações de política e, para evitar ambigüidades, chama-o de *certificado proxy restrito*. O GSI permite que o sujeito delegue este certificado a processos agindo em seu nome.

Quando um sujeito GSI acessa algum recurso, o provedor do recurso traduz a identidade GSI para uma identidade local, podendo então aplicar as políticas locais a esta identidade (e.g. restrições de sistema de arquivos do Unix). O Sistema CAS não substitui a autorização local, mas sim permite que as políticas da organização virtual sejam aplicadas à identidade GSI. Como a identidade GSI é a mesma em qualquer local da organização virtual a escalabilidade é mantida mesmo com o aumento do número de recursos.

A Figura 3.4 apresenta uma visão conceitual do CAS. O servidor CAS funciona como uma terceira parte confiável, intermediando as relações de confiança entre os diversos provedores de recurso e os participantes da organização virtual. No servidor são mantidas informações de autoridades certificadoras, sujeitos, servidores e recursos que compõem a organização virtual. Informações de política de autorização também são armazenadas no servidor CAS.

A política especifica quais grupos e/ou sujeitos possuem uma determinada permissão, a qual recurso a permissão se refere, e qual a semântica associada. As permissões são

[1] mecanismo que permite autenticar-se uma única vez e acessar múltiplos recursos distribuídos, eliminando a necessidade de autenticar-se a cada requisição de recurso.

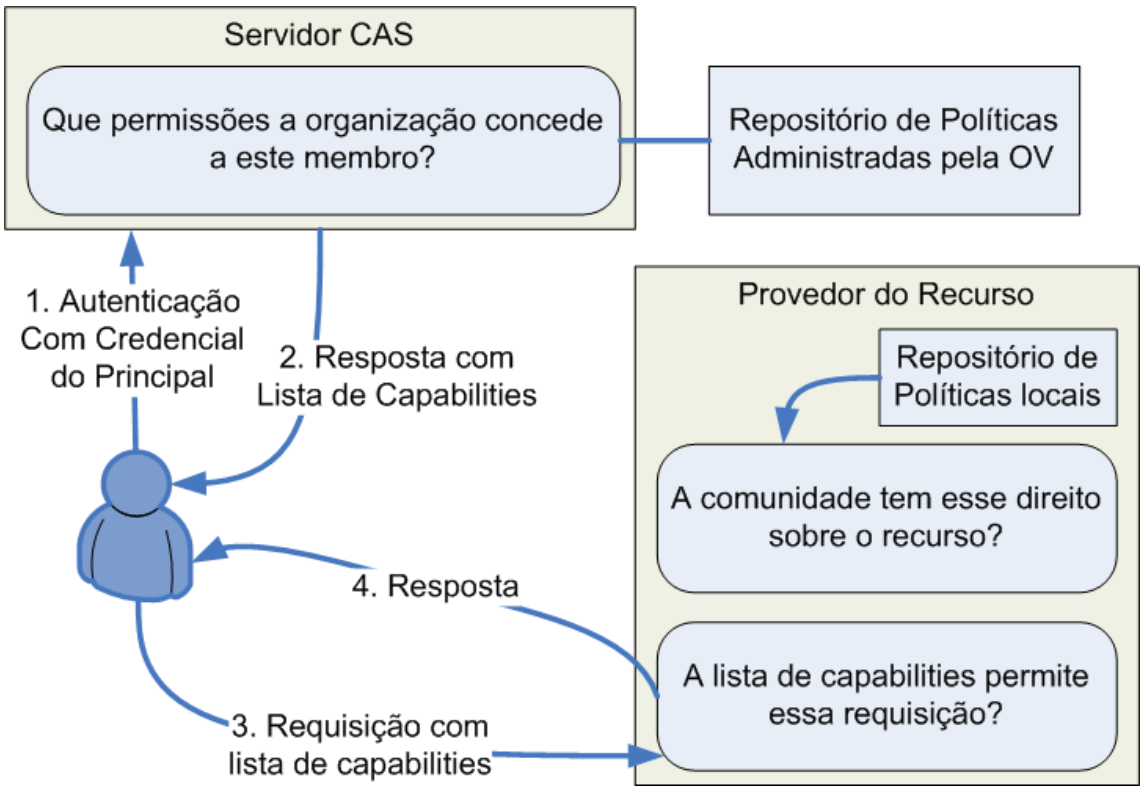

Figura 3.4: Visão conceitual do sistema CAS

formadas por um tipo de serviço e uma ação. Ações são dependentes da semântica do recurso (e.g. ler, escrever, executar), o tipo de serviço define um espaço de nomes onde a ação é definida. Os provedores de recursos podem reconhecer diferentes tipos de serviços, mas o significado das ações de cada tipo de serviço deve ser interpretado da mesma maneira por todos os provedores que o reconhecem.

Quando um sujeito deseja acessar um determinado recurso, primeiro autentica-se no servidor CAS junto com a requisição de permissão para executar uma ação. Se a requisição estiver de acordo com as políticas da organização virtual, uma lista de *capabilities* é fornecida ao sujeito na forma de um certificado *proxy* restrito. De posse do certificado o sujeito pode autenticar-se no provedor do recurso e executar uma ação. Antes de permitir a execução da ação, o provedor do recurso verifica se a lista de *capabilities* está de acordo com as políticas do repositório local, ou seja, se o provedor realmente concedeu tais permissões à comunidade.

Usando esta estrutura, cada sujeito precisa confiar no servidor CAS, que deve confiar nas autoridades certificadoras dos sujeitos. Os produtores precisam confiar apenas no servidor CAS, que por sua vez deve confiar nas autoridades certificadoras dos provedores de recursos. Os provedores concedem direitos ao servidor CAS, que por sua vez delega

seus direitos ao sujeito conforme a organização virtual estipular.

Um servidor CAS pode ser administrado por um único administrador, que gerencia os grupos, sujeitos, recursos, permissões, etc., ou pode também ter diversos aspectos do seu gerenciamento delegados a outros participantes.

3.4.1 Implementação

O servidor CAS foi implementado com base no GSI, tendo ampliado alguns de seus mecanismos. Foi adicionado o recurso de certificados de *proxy* restritos para representar informações de autorização, permitindo um controle fino dos direitos. Uma linguagem para especificação de direitos foi desenvolvida, e um conjunto de bibliotecas e interfaces de programação de aplicação foram criadas.

Os certificados de *proxy* restritos são baseados em certificados X.509. Um mecanismo de hierarquia de *proxy* foi desenvolvido, sendo que cada certificado *proxy* pode pertencer a um grupo *proxy*, informação esta que é adicionada ao certificado e permite avaliações baseadas na identidade do grupo, além da identidade do sujeito.

A linguagem implementada é simples e consiste em uma lista de concessão de direitos. Cada direito é formado por uma lista de nomes de objeto e uma lista de ações que podem ser executadas nos objetos.

Para a avaliação das credenciais de *proxy* foram desenvolvidas bibliotecas que utilizam a *Generic Authorization and Access control* (GAA), uma interface programação de aplicações que permite a integração de módulos para adquirir, ler e avaliar políticas de modo flexível. Algumas alterações foram realizadas no GSI, que usa a interface de programação de aplicação *Generic Security Services API* (GSSAPI). Um protótipo foi implementado com a linguagem Python, com base no Globus Toolkit. Uma aplicação do CAS foi feita no controle de acesso à arquivos, no projeto *Earth System Grid*. A aplicação consistia em vários servidores de FTP contendo informações compartilhadas pelos pesquisadores do projeto. Originalmente a administração dos direitos de acesso era realizada localmente, e não possui expressividade. Com o CAS a administração foi centralizada, algumas alterações foram realizadas no software dos servidores de FTP e nas aplicações cliente, para que pudessem comunicar-se com o servidor CAS e reconhecer as credenciais de *proxy*.

3.4.2 Considerações

Como servidor CAS mantém todas as informações de autorização de modo centralizado, e como os provedores confiam no servidor, caso este tenha sua segurança comprometida, toda a segurança do sistema de autorização fica prejudicada, e qualquer autorização concedida será honrada pelo provedor de recurso, desde que esteja em conformidade com as políticas locais. A única saída, caso descubra-se que o servidor CAS foi comprometido, é revogar todos os direitos concedidos a este nos provedores.

Não existe nenhuma forma de revogação das credenciais *proxy* após a concessão. Numa tentativa de amenizar este problema, as credenciais têm um tempo de vida curto, geralmente de algumas horas. Caso seja necessário suspender os direitos de um sujeito o único modo é removê-lo do servidor central e aguardar a expiração das credenciais. Por isto é possível concluir que, do modo como é proposto, CAS é inadequado para as necessidades modernas de controle de uso, como as focadas no UCON$_{ABC}$.

3.5 Akenti – Autorização com Certificados para ambientes PKI

O sistema de autorização Akenti visa resolver dois problemas básicos inerentes aos métodos tradicionais de autorização em sistemas distribuídos: gerenciamento da identidade dos sujeitos e definição das políticas de acesso por múltiplas autoridades – *stakeholders*. Akenti utiliza infra-estrutura de chave pública (PKI) com certificados X.509 para identificação única em toda a organização virtual, e três categorias básicas de certificados para a parte de autorização (THOMPSON; ESSIARI; MUDUMBAI, 2003).

A modelo conceitual do Akenti é um cenário onde existem vários recursos compartilhados, os quais são acessados por sujeitos através de um *gateway* que faz a imposição das políticas (*policy enforcement point* – PEP). As conexões entre o sujeito e o *gateway* são feitas com protocolo SSL e autenticadas usando certificados X.509.

Stakeholders são os indivíduos que possuem autoridade sobre o recurso compartilhado. São estes indivíduos que definem a política de acesso através de restrições que devem ser atendidas para que a requisição seja permitida. Tais restrições são representadas por certificados assinados, que são armazenados possivelmente junto ao PEP, ou em algum local remoto seguro. Estes certificados informam quais atributos um sujeito necessita para ter o acesso garantido, quem pode criar regras de condições de uso, e quem pode atestar

os atributos do sujeito.

Quando o sujeito tenta acessar um recurso, o PEP contacta o servidor Akenti (*policy decision point* – PDP) que informa quais direitos o sujeito tem sobre o recurso desejado. O PDP coleta todos os certificados relevantes, verifica a assinatura dos certificados para assegurar-se de que foram emitidos por entidades bem conhecidas, e retorna a decisão na forma de um certificado de *capabilities* com os direitos que o sujeito tem sobre o recurso.

O servidor Akenti concentra-se em um modelo de aquisição de informações *pull mode* puro, para permitir que as aplicações utilizem conexões SSL para transportar e verificar os certificados X.509.

Todas as autoridades certificadoras são representadas por seus certificados X.509, os quais contêm informações sobre onde os certificados são publicados e listas de revogação de certificados. A política de autorização do Akenti é expressada em XML armazenado em três categorias de certificados:

Certificados de política: definem as fontes de autoridade para um recurso, são auto-assinados e armazenados com os recursos aos quais se aplicam. Devem conter somente um mínimo de informação afim de reduzir as dificuldades em caso de necessitarem atualizações. Indica as autoridades certificadoras que assinam os certificados X.509 para todos os sujeitos envolvidos e para os *stakeholders* que emitem os certificados de condição de uso. Este certificado indica ainda os locais de publicação dos certificados de condição de uso que se aplicam ao recurso.

Certificados de condição de uso: deve existir pelo menos um destes para cada recurso compartilhado. Representa uma restrição de uso na forma de uma expressão relacional de atributos exigidos do sujeito, e uma lista de autoridades que possam atestar os atributos. As expressões podem utilizar operadores booleanos do tipo &&(e) e ||(ou). Alguns atributos podem representar condições (e.g. system_load < 2). Caso algumas das condições não possam ser avaliadas pelo PDP elas são retornadas ao PEP para que este avalie.

Certificados de atributo: É composto de pares de nome e valor, junto com o nome do sujeito a que se aplicam. São assinados por autoridades de atributo, especificadas nos certificados de condição de uso. Podem ser aplicados a mais de um recurso. A estrutura do certificado é definida em um schema XML e não segue nenhum padrão amplamente conhecido.

A localização dos certificados de condição de uso pode ser múltipla, para aumentar a confiabilidade do serviço, no entanto cada localização deve conter o conjunto completo de certificados de condição de uso. Caso os certificados não sejam encontrados o pedido de acesso é negado. A falta de alguns dos certificados de atributos não acarreta o mesmo problema imediatamente, a ausência de algum dos certificados pode causar a limitação do uso ou a sua negação.

A definição da política de acesso é iniciada por um *stakeholder* com a ajuda de algumas ferramentas para criação dos certificados. Inicialmente é criado um certificado de política raiz para o *realm* (domínio administrativo). Todos os certificados das autoridades certificadoras confiáveis são postos em um diretório seguro. O certificado de política inclui o nome dos possíveis demais *stakeholders*.

Ao menos um certificado de condição de uso deve ser gerado após a criação do certificado de política. Todos os *stakeholders* listados no certificado de política devem criar certificados de condição de uso. Esta tarefa também é auxiliada por ferramentas de software. Os certificados de condição de uso podem ter um tempo de vida, e são armazenados nos locais pré-especificados no certificado de política.

As autoridades listadas nos certificados de condição de uso podem emitir certificados de atributo, formados por uma lista de permissões concedidas a um determinado sujeito e aplicável ao recurso no *realm* específico.

3.5.1 Implementação

A implementação do protótipo do sistema Akenti objetivou atender recursos acessados via web. Foram considerados três modos de implementá-lo: através de scripts CGI que chamariam o servidor Akenti; através de Java servlets ou JSPs que se comunicariam com o servidor; e como um módulo de autorização embutido em um servidor web. A terceira opção foi escolhida por eliminar um nível de indireção e não exigir URLs complexas como nas duas primeiras opções.

O servidor web utilizado para implementação foi o Apache web server, versão 1.3.x. Este servidor permite a implementação de módulos para registro de atividades, autenticação, controle de acesso, etc., com possibilidade de serem embutidos de modo estático ou carregados dinamicamente.

O módulo do sistema Akenti (*mod_akenti*) fornece um mecanismo de autorização como apresentado na seção anterior, podendo ser carregado no servidor no momento da iniciali-

zação. Este módulo trabalha com duas diretivas na configuração do servidor: AkentiConf que indica a localização do arquivo de configuração do mod_akenti; e AkentiResources onde é definido a lista de diretórios que é protegida pelo mod_akenti. AkentiResources pode ser uma lista vazia (para nenhuma proteção), uma lista de diretórios, ou "ALL" para proteger toda a hierarquia do servidor.

As conexões entre o cliente e o servidor web utilizam SSL com a diretiva *FakeBasicAuth*. Neste caso a conexão utiliza o sujeito do certificado X.509 do cliente como nome de sujeito, e a posse do certificado como prova de autenticidade.

Apache encaminha as requisições que coincidem com o AkentiResources ao mod_akenti. O módulo lê os certificados de política, coleta e interpreta os certificados de condição de uso, coleta os certificados de atributo necessários, e retorna a decisão (permitido ou negado) ao servidor web. O mod_akenti pode fazer cache dos certificados para reduzir o tempo na aquisição dos mesmos, no entanto, de acordo com testes realizados, a maior parte do tempo é gasto com o transporte de informações pelo SSL.

3.5.2 Considerações

Akenti apresenta um sistema de condições de uso que semelhante a aquisição de atributos mutáveis apresentada em (ZHANG et al., 2006), no entanto a mutabilidade das condições não é suportada no Akenti.

A avaliação das políticas é, como nos métodos tradicionais, realizada apenas em um momento, sendo que não há forma de revogar o acesso durante o uso. As condições são avaliadas no começo, mas não durante o uso, o que abre possibilidades de violação da política.

O sistema é altamente dependente de uma infra-estrutura de chave pública e não suporta colaborações *ad hoc* sem que haja uma infra-estrutura pré-estabelecida. Além disso, os certificados de atributo utilizados não são padronizados, mas sim utilizam uma linguagem definida pelos pesquisadores do projeto.

Como a definição das políticas é feita por *stakeholders*, não existe um método formalizado para que as mesmas políticas estejam em consonância com os acordos firmados entre provedores de recursos e uma organização virtual, o que pode gerar problemas para a administração da mesma. Neste aspecto a abordagem dos sistemas VOMS e CAS parecem mais interessantes.

3.6 Conclusão

Neste capítulo foram apresentados alguns trabalhos que investigam o problema do controle de acesso e uso em sistemas distribuídos. Os trabalhos estudados dividem-se basicamente em dois grupos, aqueles que estudam unicamente a aplicação do controle de uso em sistemas distribuídos, e aqueles que estudam o gerenciamento de informações de controle de acesso – VOMS, CAS, Akenti.

No primeiro grupo encontramos os trabalhos apresentados em (ZHANG et al., 2006), (PU et al., 2006), e (MARTINELLI; MORI; VACCARELLI, 2005). Apesar destes trabalhos investigarem a questão do controle de uso, deixam a desejar no quesito integração com sistemas de gerenciamento de colaborações dinâmicas, sendo este um dos pontos motivadores do nosso trabalho. No caso da proposta feita por Zhang (ZHANG et al., 2006), o sistema é uma prova de conceito, sua generalização para outros domínios de aplicação se torna difícil devido ao alto acoplamento existente entre o conceito demonstrado e a aplicação de exemplo. Outros trabalhos não possuem implementação (PU et al., 2006), ou possuem um alto custo administrativo (MARTINELLI; MORI; VACCARELLI, 2005).

O segundo grupo de trabalhos discutidos aborda o gerenciamento de informações de controle de acesso (ALFIERI et al., 2004), (PEARLMAN et al., 2002) e (THOMPSON; ESSIARI; MUDUMBAI, 2003). Apesar de todos as propostas buscarem uma distribuição no gerenciamento das informações de controle, os sistemas não foram projetados para operar com sistemas colaborativos dinâmicos. Além disto, seu modo de avaliação das políticas de controle de acesso é baseado nos modelos tradicionais, sendo impossível estabelecer um controle aplicável durante todo o período de uso de uma permissão para operar sobre um objeto.

No capítulo seguinte será apresentada a proposta deste trabalho, com a discussão de como a mesma pretende unir estes dois universos aparentemente distintos (controle de uso e gerenciamento de informações de controle), para fornecer uma arquitetura mais adequada para as necessidades dos modernos ambientes distribuídos de colaboração.

4 *Uma Arquitetura de Controle de Uso para Sistemas Distribuídos em Ambientes Colaborativos*

O conceito de controle de uso amplia a noção tradicional de controle de acesso, abordando aspectos relacionados à mutabilidade de atributos e à continuidade dos controles. Com o modelo UCON_{ABC} é possível obter um nível de controle consideravelmente mais apropriado para ambientes distribuídos colaborativos.

Este modelo, no entanto, foi apenas definido como uma especificação formal, deixando o nível de mecanismo sob a responsabilidade dos implementadores. Este trabalho visa, por isto, investigar a implementação do controle de uso, especificamente para sistemas distribuídos em ambientes colaborativos , utilizando para isso, e na medida do possível, tecnologias padronizadas de modo a ampliar a aplicabilidade da proposta.

4.1 Cenário

Para facilitar a compreensão de nossa proposta, vamos exemplificar as necessidades a serem resolvidas através do cenário apresentado na Figura 4.1. Consideremos a existência de uma companhia, que chamaremos aqui de Consumidor, que necessita contratar um serviço de armazenamento de dados (*Storage*). O Provedor, uma companhia que provê serviços de *Storage*. Entre ambos, uma entidade intermediadora, conhecida como *Broker*.

O *Broker* fornece uma infra-estrutura que permite a negociação e estabelecimento de contratos entre Provedores e Consumidores. O Provedor foca seus esforços em prover serviços, delegando ao *Broker* a responsabilidade por publicar e negociar acesso a esses serviços. Além disso, o *Broker* também mantém serviços de suporte a interação entre Provedores e Consumidores.

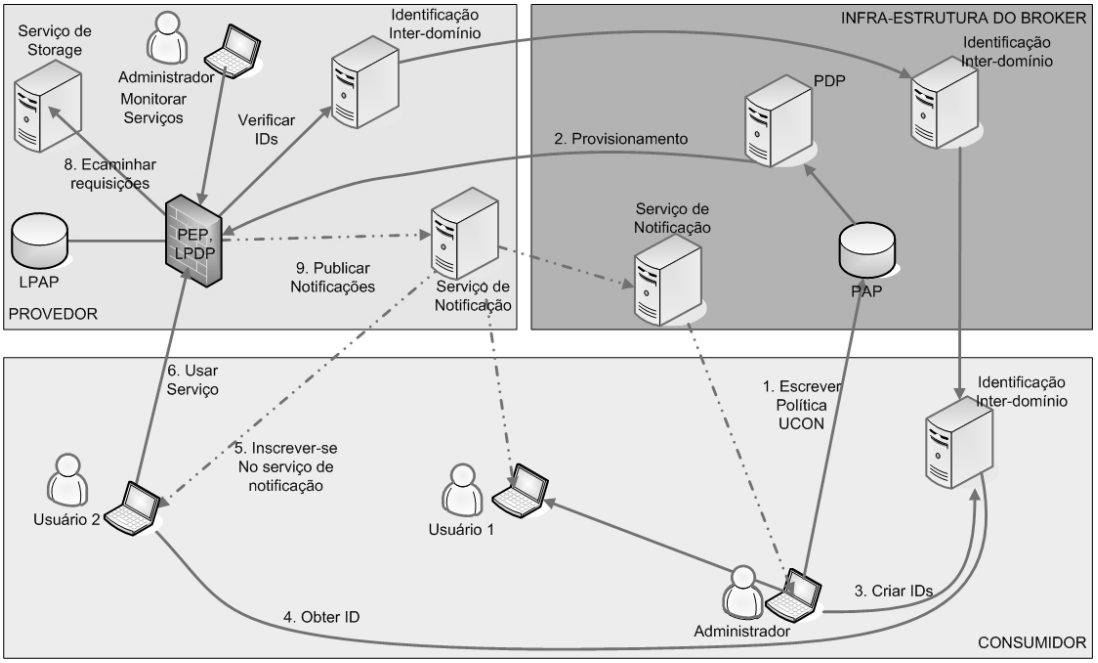

Figura 4.1: Cenário Motivador

O Provedor tem interesse, especificamente, em regular a utilização de seus recursos em nível de serviço, ou seja, não é interesse do provedor gerenciar usuários individualmente. Relacionado a gerência de usuários, está também a gerência de identidades. Num ambiente de colaboração distribuída é importante que os usuários possam efetuar suas tarefas em diversos domínios de segurança. A abordagem mais simples é a criação de contas específicas para cada usuário, nos domínios onde estes devem desempenhar suas atividades. Entretanto, esta abordagem apresenta diversos problemas, tais como alto custo gerencial humano, tendência de criação de brechas de segurança através de contas que deveriam estar desativadas, mas que, talvez por esquecimento, não foram removidas, falta de integração com sistemas de contabilização (*Accounting*) e autorização, etc.

Por isso, um dos primeiros requisitos é a existência de uma maneira de identificar os participantes. Na Figura 4.1, o Provedor, Consumidor e Infra-estrutura do *Broker*, representam um domínio de segurança diferente, e as identidades de um domínio não são automaticamente aceitas nos outros domínios. Sendo assim, deve-se utilizar um sistema de identificação inter-domínio, que permita a transposição de identidades entre domínios diferentes.

Além disso, o Provedor não deve ser responsável pela escrita de políticas de controle de uso em nível de usuário, a fim de reduzir o seu custo operacional. O administrador do Consumidor é quem deve escrever políticas de controle de uso para os recursos contratados. Isto cria um controle em dois níveis, um em nível de serviço que abrange um cliente como

um todo, e outro em nível de usuário (controle de uso), escrito pelo Consumidor e aplicado a cada indivíduo.

No passo 1, o administrador do Consumidor escreve políticas de controle de uso no repositório de políticas do *Broker*, representado na Figura 4.1 como o servidor PAP. Esta política serve para configurar a infra-estrutura de controle do Provedor (PEP/LPDP), através de um mecanismo de provisionamento (MOORE et al., 2001), no passo 2 da Figura 4.1. Após serem provisionadas, as políticas são armazenadas no LPAP do Provedor.

Considerando que o Consumidor desconhece à infra-estrutura interna do Provedor, e que não tem acesso aos meios para integrar diversas fontes de informação para serem usadas na avaliação de suas políticas, faz-se necessária a disponibilização de um sistema de contabilização, integrado ao mecanismo de escrita de políticas de controle de uso. Deste modo, o Consumidor pode utilizar diversas fontes de informações providas pelo Provedor para escrever políticas de controle de uso para os recursos contratados. Estas informações ficam disponíveis através de uma interface existente no PAP do *Broker*.

É importante para o Consumidor que suas políticas sejam avaliadas com regularidade, garantindo assim que as políticas de controle de uso não sejam violadas durante a utilização dos recursos contratados. A implementação dos mecanismos para avaliação de políticas apresenta um dos maiores desafios para aplicação do controle de uso, como veremos nos próximos capítulos.

Antes que os usuários do Consumidor possam acessar os recursos contratados, é necessário que o administrador do Consumidor crie credenciais de identificação para os mesmos, através do serviço de identificação inter-domínio (passo 3, Figura 4.1). Os usuários recebem suas credenciais no passo 4, Figura 4.1.

Ao aplicarmos o conceito de controle de uso neste ambiente, tornamos possível a interrupção em tempo real do uso de um recurso, para usuários que estejam violando as políticas de uso. Tais interrupções, ainda que produzidas no intuito de garantir o respeito das políticas de uso, podem causar prejuízos indesejados aos usuários que não perceberam a violação das políticas vigentes.

Ainda na Figura 4.1, a fim de evitar tal situação, o usuário do Consumidor deve estar inscrito no serviço de notificação (passo 5). O usuário pode então enviar requisições ao provedor (passo 6). A infra-estrutura de controle do provedor verifica as credenciais do usuário (evento *Verificar IDs*), avalia as políticas de nível de serviço e de controle de uso, e toma as ações cabíveis (eg. liberar o uso, passo 8, ou publicar uma notificação de

violação, passo 9). O administrador do Provedor pode se concentrar em monitorar o bom andamento do provimento dos serviços.

Além disso, o Consumidor e o Provedor necessitam de um meio flexível de interação entre si. O Provedor deseja publicar os serviços oferecidos, de modo que possam ser pesquisados e contratados pelo Consumidor. O Consumidor precisa ter a privacidade de seus usuários preservada, recursos para gerir os usuários, interfaces padronizadas para acessar os serviços, e assim por diante.

Baseados nestas informações, podemos derivar a seguinte lista de requerimentos para uma arquitetura de controle de uso adequada a este ambiente:

- Mecanismos de publicação, busca, e contratação de serviços.
- Um mecanismo para criação de políticas de nível de serviço.
- Mecanismos que permitam ao Consumidor produzir políticas de controle de uso para seus usuários, com uma linguagem fácil.
- Um mecanismo de gerenciamento de informações de uso, integrado com os mecanismos de controle e escrita de políticas.
- Mecanismos para provisionar ao Provedor as políticas escritas na infra-estrutura do *Broker*.
- Um mecanismo para permitir que os usuários e administradores sejam notificados das violações de políticas.
- Mecanismos que forneçam a transposição de identidades entre diferentes domínios de segurança.
- Mecanismos de controle capazes de implementar a semântica de controle de uso.

4.2 Arquitetura Proposta

Visando atender os requerimentos especificados na seção 4.1, foi elaborada a arquitetura apresentada na Figura 4.2. Nas seções seguintes serão apresentadas as entidades da arquitetura (*Broker*, Provedor, e Consumidor), e suas interações. Ressaltamos que está arquitetura é baseada numa concepção de serviços, ou seja, cada componente expõe

uma interface que pode ser invocada remotamente através de um protocolo de comunicações padronizado, favorecendo o baixo acoplamento entre as entidades da arquitetura e integração entre plataformas heterogêneas do ambiente distribuído.

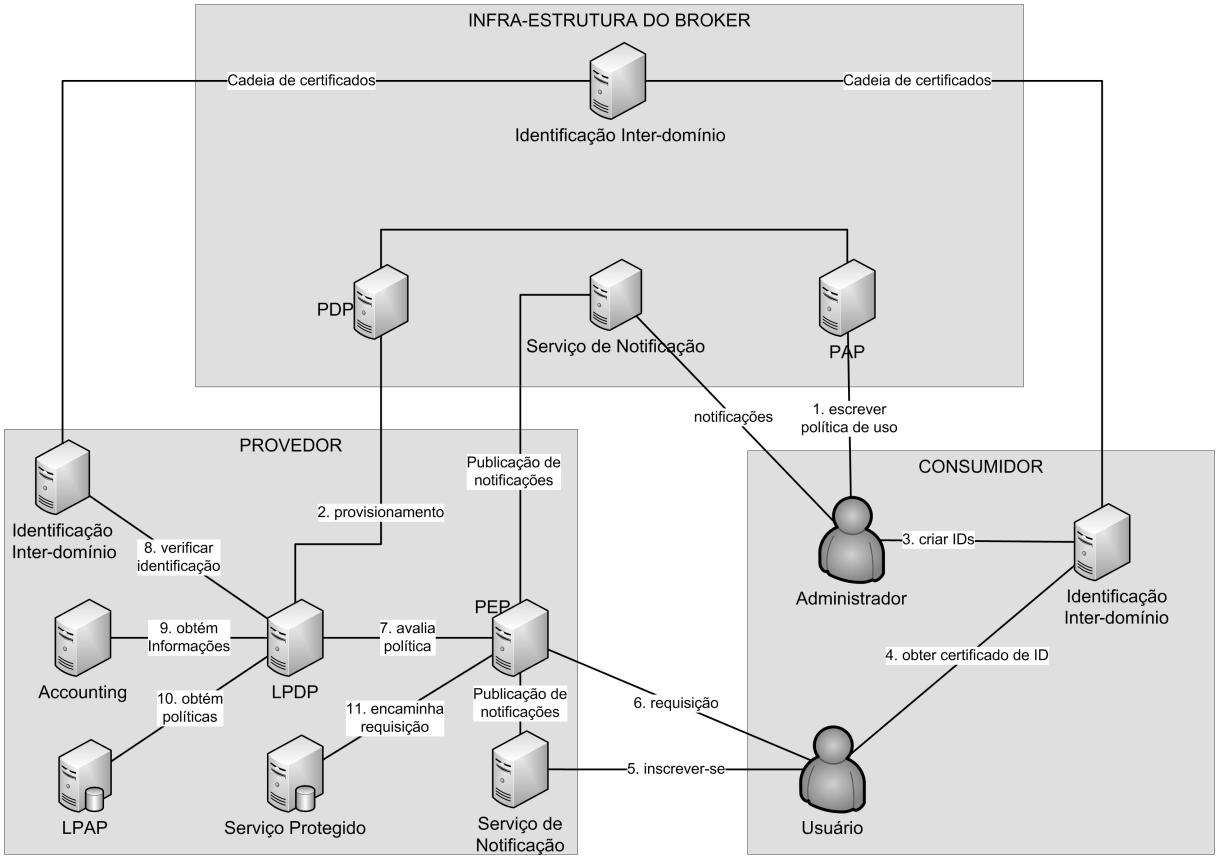

Figura 4.2: Visão geral da arquitetura proposta

4.2.1 Infra-estrutura do Broker

A infra-estrutura de intermediação (*Broker*), é responsável por estabelecer os contratos entre companhias Consumidoras e Provedoras. Por este motivo, deve disponibilizar um conjunto de funcionalidades que permitam a publicação, busca, e contratação de serviços. Como vemos na Figura 4.2, o *broker* possui: componentes para transposição de identidades, ou identificação inter-domínio; o PAP (*policy administration point*, ponto de administração de políticas); PDP (*policy decision point*, ponto de avaliação de política); e o serviço de notificação.

A primeira função desta infra-estrutura é permitir a publicação de serviços para serem contratados por Consumidores. Assumimos que esta infra-estrutura permite que o Provedor forneça à infra-estrutura do *Broker*, as informações correspondentes ao serviço a ser negociado. Através de um mecanismo de delegação, o Provedor é capaz de garantir

que não será negociada uma quantidade de serviço maior do que o informado ao *Broker*. Caso a infra-estrutura do *Broker* seja comprometida, o Provedor pode facilmente comprovar a quantia de serviço informado por delegação, e assim recusar tentativas abusivas de utilização dos recursos contratados.

Ao estabelecer um contrato com o Consumidor, o *Broker* delega uma parte, ou a totalidade, da quantidade de serviço contratado e originalmente delegado pelo Provedor. Este processo resulta em dois elementos: uma credencial para o Consumidor; e um conjunto de políticas de nível de serviço.

A credencial do Consumidor tem como funções liberar o acesso do mesmo à interface de administração do PAP, garantir o acesso ao serviço contratado, e atuar como parte do sistema de identificação inter-domínio.

A política de nível de serviço é destinada à configurar os controles do Provedor. As políticas de *nível de serviço* tem um caráter mais genérico, não sendo aplicada a usuários individuais, mas sim ao Consumidor em nível de contratação do serviço. Esta política é armazenada no PAP da infra-estrutura do *Broker*, sendo posteriormente transferida para o Provedor.

4.2.2 Domínio do Provedor

O Provedor possui a infra-estrutura para prover o serviço contratado, assim como diversos mecanismos para permitir o controle de utilização do mesmo. Na Figura 4.2, vemos que existe um ponto de aplicação de políticas (PEP), um serviço de notificação em nível de usuário, um ponto de avaliação de políticas local (LPDP), um repositório de políticas local (LPAP), um sistema de monitoração de informações de uso (*Accounting*), e parte do sistema de identificação inter-domínio.

As políticas de nível de serviço e de controle de uso são fornecidas pelo *Broker* através do mecanismo de provisionamento. O Provedor somente se concentra em gerenciar os serviços providos e a infra-estrutura que controla o acesso ao serviço contratado (Serviço Protegido, Figura 4.2).

4.2.3 Domínio do Consumidor

O domínio do Consumidor, por sua vez, é composto basicamente pelo sistema de identificação inter-domínio, seus usuários, e seus administradores, Figura 4.2.

Administradores são responsáveis por criar identidades (credenciais de identificação) para seus usuários, escrever as políticas de controle de uso, e monitorar a utilização do serviço contratado através do recursos oferecidos pelo provedor do serviço. Os administradores recebem notificações de violações tanto em nível de serviço como em nível de controle de uso (em nível de usuário). Porém, quando uma notificação é enviada ao administrador do Consumidor, significa que o usuário foi notificado e não conseguiu sair da condição de violação.

Os usuários interagem com o serviço contratado através de aplicações que se comunicam com pontos de acesso pré-determinados, localizados no Provedor. Todos os usuários devem se registrar no serviço de notificação localizado no Provedor, para que possam receber notificações de violações das políticas de controle de uso quando estiverem utilizando o recurso contratado.

4.2.4 Sistema de identificação inter-domínio

O sistema de identificação inter-domínio é peça fundamental para permitir a fácil colaboração em ambientes arbitrariamente complexos. Como apresentado na Figura 4.2, o sistema de identificação inter-domínio está presente nos três domínios da arquitetura proposta – Provedor, Consumidor, e infra-estrutura do *Broker*.

Optamos por utilizar um modelo de confiança baseado em fonte certificada da identidade do usuário. Queremos dizer com isso que se a identidade do usuário é endossada por uma entidade conhecida e confiável, então a identidade do usuário também é confiável. Por este motivo, a infra-estrutura intermediadora funciona como uma ponte entre o Consumidor e o Provedor.

O Consumidor recebe uma credencial da infra-estrutura do *Broker* ao contratar um serviço. Esta credencial é assinada pela infra-estrutura do *Broker*. O administrador do Consumidor passa então a emitir credenciais de identificação assinadas com a sua própria credencial, endossada pelo *Broker*.

Quando um usuário tenta autenticar-se no Provedor para utilizar um serviço, é possível seguir a cadeia de assinaturas (endosso) e certificar-se de que a identidade é confiável, pois o Provedor confia na assinatura da infra-estrutura do *Broker*.

4.2.5 Mecanismos de delegação

O Provedor pode utilizar certificados contendo atributos que indiquem as quantias de recurso negociáveis pelo *Broker*. Seguindo este princípio, o Provedor cria um certificado com os atributos referentes ao recurso que deseja negociar, e delega-o ao *Broker*. O *Broker*, por sua vez, pode delegar novos certificados contendo parte, ou a totalidade, dos valores contidos no certificado original, a organizações Consumidoras, no momento da contratação de um recurso. O Consumidor pode então apresentar este certificado ao Provedor, para que este permita a utilização do recurso contratado.

No momento em que o Consumidor apresenta o certificado ao Provedor, este pode contabilizar os valores contidos nos atributos do certificado, e nos atributos dos demais certificados referentes ao mesmo recurso, e comparar com os valores originalmente delegados ao *Broker*, de modo a validar os atributos. Percorrendo a cadeia de delegação de certificados e, através de mecanismos de sumarização de direitos, é possível identificar se o certificado transporta mais direitos do que realmente foram delegados inicialmente pelo Provedor. Um exemplo de mecanismo de delegação similar é encontrado em SPKI (ELLISON, 1999; ELLISON et al., 1999) e SDSI (RIVEST; LAMPSON, 2007).

4.2.6 PAP, LPAP, e Provisionamento

A infra-estrutura do *Broker* possui um repositório global de políticas de uso e um de nível de serviço, o PAP (Figura 4.2). Uma versão correspondente existe no Provedor, o LPAP (local PAP), contudo o LPAP somente possui as políticas correspondentes ao Provedor e Consumidor envolvidos, enquanto que o PAP pode conter políticas aplicáveis a inúmeros Provedores e Consumidores, que utilizem seus serviços.

O PAP e LPAP são sincronizados através de um sistema de provisionamento. Por provisionamento entende-se o processo de configuração do LPAP, de acordo com as políticas contidas no PAP e específicas do Provedor em questão. Quando os controles do Provedor estão sendo inicializados, uma mensagem é enviada ao PDP da infra-estrutura do *Broker*, solicitando as políticas adequadas. Estas políticas são recuperadas do PAP, e enviadas ao Provedor, sendo armazenadas no LPAP.

As avaliações de políticas ocorrem localmente no Provedor, ao invés de solicitar avaliações de políticas ao PDP da infra-estrutura do *Broker* a cada pedido de acesso. Eventualmente, o Provedor pode receber requisições para as quais ainda não foram provisionadas as políticas adequadas. Neste caso uma nova mensagem é enviada à infra-estrutura do

Broker para que provisione as novas políticas, mantendo assim a sincronia entre os repositórios. Além disso, o provisionamento pode ocorrer por iniciativa da infra-estrutura do *Broker*, quando há uma alteração nas políticas armazenadas no PAP.

Esta estratégia proporciona uma operação com maior autonomia para o Provedor, reduzindo o custo associado com o envio repetitivo de mensagens à infra-estrutura do *Broker*, tradicional dos modos de operação dos mecanismos de avaliação de políticas de controle de acesso. O Provedor torna-se mais resistente a faltas temporárias nas comunicações com a infra-estrutura do *Broker*, podendo avaliar políticas localmente enquanto as comunicações não são restabelecidas.

4.2.7 Sistema de notificação

O sistema de notificação (Figura 4.2) busca melhorar a experiência dos usuários durante a utilização dos recursos. Através deste sistema é possível enviar notificações aos usuários alertando-os para violações de políticas, revogações do direito de uso, etc.

O sistema também serve para propagar notificações de violações aos administradores das organizações envolvidas. Neste caso, as notificações são referentes às violações em nível de serviço e em nível de controle de uso. O sistema de notificação existente na infra-estrutura do *Broker* atende aos administradores, enquanto o sistema de notificação existente no domínio do Provedor atende aos usuários, e somente propaga notificações referentes às políticas de controle de uso.

Devido à importância desta funcionalidade, decidimos por tornar a utilização do mesmo uma obrigação. Sendo assim, todo usuário que deseja utilizar um recurso, deve inscrever-se no sistema de notificação, e verificar periodicamente o mesmo para certificar-se de que não existem notificações pendentes – optamos por uma abordagem *pull* pois em determinados casos o cliente pode estar impedido de receber notificações diretamente, sendo mais adequado que o mesmo as busque no provedor. Para comprovar a sua conformidade com esse requerimento, os usuários recebem um *token* que serve como prova de sua inscrição no sistema de notificação.

Outra obrigação requer que ao menos um administrador esteja conectado ao sistema de notificação, quando seus usuários estiverem utilizando o recurso contratado. Isto se mostra essencial, pois a ocorrência de violações em nível de serviço podem acarretar penalidades contratuais e, no caso de violações em nível de controle de uso, em alguns casos, o administrador é a única entidade com poderes para corrigir a situação.

4.2.8 Infra-estrutura de monitoração – Accounting

A infra-estrutura de monitoração, representada na Figura 4.2 como o sistema de *Accounting*, não pode ter sua importância negligenciada. O modelo UCON$_{ABC}$ prevê a possibilidade de avaliação de diversos atributos, tanto de usuários, como de recursos, e de ambiente. Por este motivo, é importante que o Provedor disponibilize mecanismos para monitorar tais atributos.

Nesta proposta, a infra-estrutura de monitoração oferece uma interface de acesso a diversas informações referentes aos serviços providos, aos usuários que os utilizam, e ao ambiente em que estão inseridos. Todas estas informações devem ser fornecidas de modo transparente. Os atributos providos pela infra-estrutura de monitoração são utilizados para a escrita das políticas de controle de uso, sem que para isso o Consumidor precise conhecer os mecanismos utilizados pelo Provedor para ter acesso aos mesmos.

A infra-estrutura de monitoração funciona de modo autônomo e permite que componentes da infra-estrutura do Provedor a invoque, a fim de obter informações relevantes para o controle em nível de uso e em nível de serviço. Isto é essencial para a implementação do mecanismo de controle de uso. Além disso, os atributos monitorados pelo sistema de *Accounting* são exportados para a interface de escrita de políticas de controle de uso, existente no PAP.

4.2.9 Operação do PAP

A administração de políticas de controle de uso, na Figura 4.2, utiliza o PAP como repositório. Através de uma interface o administrador do Consumidor pode escrever políticas de controle de uso para os seus usuários. Esta interface tem acesso ao conjunto de atributos disponibilizados pela infra-estrutura de monitoração do Provedor, e armazena as políticas no PAP.

Objetivando manter uma maior compatibilidade com o modelo UCON$_{ABC}$, optamos por criar uma interface que permita a escrita da política sob a forma de predicados. Cada predicado é classificado como sendo uma autorização, obrigação, ou condição, e pode ser aplicado antes ou durante o uso de um recurso.

Na Figura 4.3 podemos observar um fragmento de política de controle de uso baseada em predicados derivados da proposta UCON$_{ABC}$. Este tipo de política não é concebido para ser interpretado por mecanismos computacionais de controle, mas sim para ser mais

```
********************** Controles do tipo pre **********************
01 verifyGroup( user )
02        ( user.group eq ‘‘Developers’’ )
03 verifyRight( user )
04        contains( user.permissions, ‘‘Write’’ )
05 isSubscribed( user )
06        sts.isValid( user.token )
*********************** Controles do tipo ongoing (durante) ******
07 verifyQuota( user )
08        (( service.quotaOrg( user.OrgID ) le (50))
09            and
10            ( service.quotauser( user.ID) lt 10 )
11         )
12         or
13         ( service.quotauser( user.ID ) lt 5 )
14 verifyTimeShift( user )
15        ( env.now ge user.startTS )
16           and
17        ( env.now le user.endTS )
18 verifyToken( user )
19        sts.isValid( user.token )
```

Figura 4.3: Exemplo de política escrita pelo Consumidor

facilmente compreendida por humanos.

O primeiro predicado (linhas 01 e 02) é aplicado sobre o objeto *user*, que representa o usuário tentando efetuar alguma ação no sistema, onde é avaliado se o grupo do usuário é igual à *Developers*.

O predicado das linhas 03 e 04 verifica se o usuário possui permissão do tipo *Write*, enquanto que o predicado das linhas 05 e 06 invoca um serviço (STS) para verificar se o *token* obtido do sistema de notificação é válido.

A linha 07 apresenta um predicado mais complexo. No corpo do predicado existe um operador lógico do tipo *or*, linha 12. O primeiro operando (linhas 08 à 11) é o resultado do operador lógico *and*. Caso a quota utilizada pela organização Consumidora não ultrapasse 50 gigabytes, e o usuário não tenha consumido sua quota máxima de 10 gigabytes, o uso é permitido. Caso contrário o primeiro operando do operador *or* é falso, então é avaliado se o usuário não atingiu 5 gigabytes de quota. Caso ambos os operandos retornem falso, o predicado é avaliado como falso.

Nas linhas 14 à 17 apresentamos um predicado que verifica, como condição, se o usuário está utilizando o recurso dentro de seu período da jornada de trabalho. E nas linhas

18 a 19 outro predicado verifica, como obrigação, se o *token* do sistema de notificações continua válido.

Ao terminar a escrita das políticas de predicados, através da interface do PAP, o administrador do Consumidor pode desencadear o processo de conversão (*parser*) para políticas de configuração dos mecanismos computacionais de controle.

Como é sabido, UCON_{ABC} possui autorizações, obrigações e condições, que podem ser aplicadas antes ou durante o uso de um recurso. Por este motivo, a interface do PAP divide a tarefa de escrita de políticas de controle conforme seu tipo e momento de avaliação (antes ou durante). Esses predicados são então convertidos em políticas de mecanismo correspondentes a cada um dos controles mencionados. A forma de avaliação destes mecanismos será estudada na seção 4.2.11

4.2.10 Ponto de aplicação de política - PEP

O ponto de aplicação de política, ou PEP, (*policy enforcement point*), é responsável por honrar o resultado da avaliação das políticas de controle de uso e de nível de serviço (Figura 4.2). O PEP intercepta requisições destinadas ao recurso protegido, avalia se a requisição representa o início de uma nova sessão de utilização ou se faz parte de uma sessão pré-existente. No caso de ser uma nova sessão, o PEP envia uma requisição de avaliação de política ao LPDP. Caso a requisição pertença a uma sessão já estabelecida, o PEP consulta a última avaliação do LPDP e, caso as políticas de controle de uso continuem sendo respeitadas, encaminha a requisição para o recurso protegido.

Apesar de funcionalmente distinto, o PEP está localizado juntamente com o recurso protegido. O PEP solicita regularmente a reavaliação das políticas aplicáveis a cada sessão de uso ativa. Esta funcionalidade é importante pois o PEP não solicita avaliação de políticas a cada interação com o usuário, reduzindo o custo de operação do controle de uso. No entanto, quando a sessão já está estabelecida e o usuário tenta acessar o recurso, suas credenciais são verificadas para confirmar se a sessão não entrou em estado de violação após a última avaliação periódica de políticas de controle de uso.

Outra funcionalidade importante do PEP é invocar o serviço de notificação para enviar mensagens de violação para o usuário e administradores, contendo informações relativas à violação ocorrida. Essas notificações são consideradas parte das atividades de aplicação das políticas em nossa proposta.

4.2.11 PDP, LPDP e Suporte ao Controle de Uso

O ponto de avaliação de políticas local (LPDP, Figura 4.2) é o componente responsável por decidir se uma determinada requisição de acesso é permitida, de acordo com as políticas de nível de serviço e de controle de uso. O PDP é o ponto de avaliação de políticas global, sendo localizado no *Broker*. Entretanto, o PDP somente atua atendendo requisições de provisionamento de políticas dos LPDPs.

As políticas são avaliadas em seqüência, primeiro as de nível de serviço e, em seguida, as de controle de uso. Caso a primeira avaliação recuse o acesso, nenhuma outra avaliação é realizada, e a requisição é negada.

O primeiro aspecto importante do avaliador de políticas é que o mesmo não avalia políticas anteriormente ao acesso, ele é disparado por chamadas periódicas do PEP, sendo a única exceção o início de uma sessão de uso. Ao receber uma requisição pertencente à uma nova sessão de uso o LPDP, através de seu gerenciador de contexto, obtém os atributos relevantes junto à infra-estrutura de monitoração e serviço de identificação inter-domínio.

A cada avaliação, os atributos são recuperados com os valores atuais. O resultado das avaliações é enviado ao PEP, o qual toma as ações necessárias para aplicar a decisão. Este sistema de avaliação de políticas permite que obtenhamos uma avaliação continuada sem, no entanto, incorrer em custos demasiado altos de processamento devido à necessidade de avaliação continua exigida pelo UCON_{ABC}.

As políticas geradas através da interface do PAP são convertidas, por um *parser*, em políticas correspondentes a cada controle (A,B,C) e fase da utilização (*pre* ou *ongoing*). Por isso, o LPDP possui um sub-componente que é responsável por identificar quais políticas são aplicáveis dentro de um determinado contexto (o gerenciador de contexto). A Figura 4.4 apresenta um fragmento de código referente à esse sub-componente.

O gerenciador de contexto executa o pseudo código da Figura 4.4 de modo ordenado até que as avaliações cheguem ao fim, ou uma violação ocorra. As palavras *catch* simbolizam os pontos onde são invocadas as funções que lidam com violações de políticas (e.g. retorno da mensagem de violação ao PEP).

Caso a sessão esteja sendo iniciada (linha 01), primeiro são executadas possíveis atualizações de atributos (linha 02). As pre-autorizações, pre-condições, e pre-obrigações são executadas seqüencialmente (linhas 03 a 05), caso nenhuma delas retorne falso, a avaliação da política retorna uma permissão de acesso, e a requisição é permitida.

```
01 if( session.state == ‘‘new’’ ) {
02     perform( getPreUpdates( request ) )
03     try ( evaluate ( getPreAuthorizations( request ) ) catch ()
04     try ( evaluate ( getPreConditions( request ) ) catch ()
05     try ( evaluate ( getPreObligations( request ) ) catch ()
06 } else {
07     perform( getOngoingUpdates( request ) )
08     try ( evaluate ( getOngoingAuthorizations( request ) ) catch ()
09     try ( evaluate ( getOngoingAuthorizations( request ) ) catch ()
10     try ( evaluate ( getOngoingAuthorizations( request ) ) catch ()
11 }
```

Figura 4.4: Fragmento de código do gerenciador de contexto

O mesmo se dá quando estão sendo avaliadas as políticas aplicáveis durante a sessão de uso (linhas 07 a 11). As palavras *evaluate* indicam o ponto em que são recuperadas as políticas aplicáveis nos repositórios locais, e invocados o monitor de referência, juntamente com as informações da requisição *request*. Os objetos *request* contém todas as informações referentes à uma sessão de utilização (e.g. identificação de usuário, estado da sessão, nome da organização, serviço sendo utilizado, identidade da sessão, etc).

Observamos que o comportamento do PEP pode variar diante de violações de políticas. Pode-se optar por configurar o PEP para que o mesmo forneça um período de tempo para que o usuário tente corrigir a situação, antes de interromper a sessão de uso totalmente, ou o PEP pode ser configurado para interromper a sessão unilateralmente. Na interface do PAP é possível configurar informações adicionais sobre as acões de aplicação de políticas.

4.3 Conclusão

Neste capítulo foi apresentado um cenário motivador no qual a proposta pode ser aplicada, porém a proposta não está limitada ao mesmo. Foram apresentados também os requerimentos e os detalhes da proposta, visando mostrar a aplicação do modelo UCON_{ABC} em sistemas colaborativos distribuídos.

As abordagens de aplicação de controle de uso apresentadas nos trabalhos relacionados, de modo geral apresentam baixa capacidade de generalização. A implementação proposta por Zhang e seus colegas (ZHANG et al., 2006), por exemplo, apesar de explorar UCON_{ABC} em sistemas colaborativos, não apresenta o caráter integrado desta proposta, além de seus mecanismos de controle possuírem alto acoplamento. A arquitetura proposta neste capítulo elimina estas deficiências e propõe mecanismos mais adequados para

colaborações em sistemas distribuídos.

Nos próximos capítulos discutiremos aspectos de implementação de um protótipo desenvolvido para avaliar a viabilidade da proposta, assim como sua avaliação e trabalhos futuros.

5 *Protótipo*

Neste capítulo será apresentada a implementação do protótipo, e as tecnologias utilizadas em seu desenvolvimento. A aplicação escolhida para a implementação do protótipo foi o serviço de armazenamento de dados, dada a sua adequação para avaliar os aspectos relevantes da proposta. Para efeitos de avaliação, o protótipo implementa autorizações, obrigações, e condições do tipo *pre* e *ongoing*.

5.1 Tecnologias utilizadas

Optamos por utilizar o máximo de especificações publicamente reconhecidas e patrocinadas por organizações de boa reputação, tais como a OASIS, W3C, IETF, etc. Além disso foram utilizados um grande número de bibliotecas e ferramentas de software amplamente disponíveis, evitando assim a duplicação de esforços.

A linguagem de programação escolhida para servir de base à implementação do protótipo foi Java 1.6 (GOSLING, 1996). Esta linguagem baseia-se em uma máquina virtual, disponível em diversas plataformas, além de dispor de um grande número de bibliotecas, cobrindo boa parte das funcionalidades necessárias.

Um dos fatores mais importantes considerados durante a concepção da proposta, foi a redução do acoplamento (dependência funcional) entre entidades da arquitetura. Por este motivo a escolha de serviços web se mostra bastante apropriada. De pouco adiantaria a redução da dependência entre entidades se, na eventualidade da mudança de uma das implementações remotas, fosse necessário a modificação de todas as implementações que pudessem vir a interagir com esta.

A arquitetura de serviços web busca promover a segurança, escalabilidade, extensibilidade, e inter-operabilidade (GOTTSCHALK et al., 2002). Para isso utiliza-se de serviços que são independentes da plataforma utilizada, aplicando interfaces de invocação bem definidas, permitindo que diferentes implementações ofereçam a mesma funcionalidade

de modo transparente. A Figura 5.1 apresenta a pilha das principais especificações e protocolos utilizados.

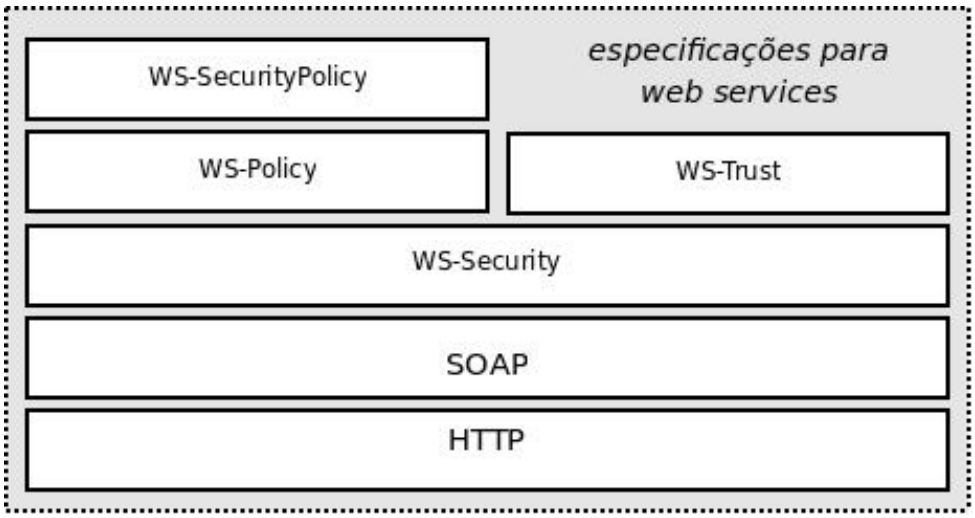

Figura 5.1: Protocolos e especificações para serviços web

Serviços web estão baseados na especificação XML (*Extensible Markup Language*) (W3C, 2006). A XML é uma linguagem para marcação extensível, utilizada na descrição de dados de maneira neutra em relação à plataforma, podendo ser interpretada em diversos ambientes. É baseada em mensagens de texto, organizadas por uma estrutura de *tags* hierárquica e bem definida.

O protocolo, na base da Figura 5.1, é o *Hypertext Transfer Protocol* (HTTP). Conforme a especificação diz, trata-se de um protocolo de nível de aplicação para sistemas de informação de hipermídia, distribuídos e colaborativos, sendo genérico, e sem informação de estado (FIELDING et al., 1999). Este protocolo pode ser utilizado para outras tarefas além do transporte de hipertexto. Este é o protocolo mais utilizado para o transporte de mensagens entre serviços web, que possuem sua própria especificação.

Acima do protocolo HTTP, encontra-se o protocol *Simple Object Access Protocol* (SOAP). Este protocol, baseado em XML, especifica o formato das mensagens trocadas entre serviços web. Pode operar em modo síncrono (*request/response*) ou assíncrono. Neste protocolo estão especificados formatos de cabeçalho, envelope, etc. A especificação SOAP, entretanto, não define o formato das informações trocadas entre os serviços (BOX et al., 2000).

A framework SOAP Apache Axis2 (The Apache Foundation, 2007a) opera sobre uma implementação de servidor HTTP fornecido pelo Apache Tomcat (The Apache Foundation, 2007b). O Apache Axis2 fornece um grande número de funcionalidades que reduzem

a necessidade de lidar com os detalhes de baixo nível na geração de mensagens SOAP, especificação de interface de serviço, etc.

Como as mensagens trocadas entre os serviços podem operar em ambiente de alto risco (e.g. Internet), a segurança fim-a-fim das mensagens é um requisito primordial. Pensando nisso, adotamos a especificação WS-Security (LAWRENCE; KALER, 2004), que baseia-se nas especificações prévias *XML Digital Signature*, assinatura digital para documentos XML (EASTLAKE; REAGLE; SOLO, 2002), e *XML Encryption*, criptografia para documentos XML (EASTLAKE et al., 2008), provendo mecanismos para garantir a privacidade, integridade, e autenticidade das mensagens.

Para facilitar a utilização de mecanismos de segurança, aplicamos também as especificações WS-SecurityPolicy (LAWRENCE; KALER, 2007a), que baseia-se na especificação mais genérica WS-Policy (Schlimmer, J. (Editor), 2006) para definir políticas que permitam aos serviços a negociação automática dos mecanismos de segurança a serem utilizados. O estabelecimento de relações de confiança entre as entidades envolvidas é facilitado com a especificação WS-Trust (LAWRENCE; KALER, 2007b).

Um módulo, chamado Rampart, é oferecido como parte do Apache Axis2 para prover as funcionalidades de WS-Security, WS-Policy, WS-SecurityPolicy, e WS-Trust. As chaves criptográficas utilizadas são armazenadas em *keystores* JKS, um formato definido para a plataforma Java. A integração com servidores de chaves não está disponível neste modulo, sendo que optamos por não nos concentrar nesta funcionalidade por considerá-la fora do escopo deste trabalho.

Para a escrita de políticas optamos pela utilização da *eXtensible Access Control Markup Language* (XACML) (GODIK; MOSES, 2002). Esta especificação define uma forma de escrever políticas de controle de autorização com uma marcação baseada em XML.

As principais entidades da especificação XACML são apresentadas na Figura 5.2, a especificação, no entanto, define esta figura como *não normativa*. Pode-se sumarizar a função de cada entidade da seguinte maneira:

PAP: repositório de políticas e conjuntos de políticas, disponibilizadas ao PDP, para a utilização em um ambiente específico (*target*).

Access Requester: envia requisições de acesso ao PEP.

PEP: envia requisições de avaliação de políticas ao *context handler*, em seu formato nativo, possivelmente contendo atributos relativos à requisição. Honra a decisão da

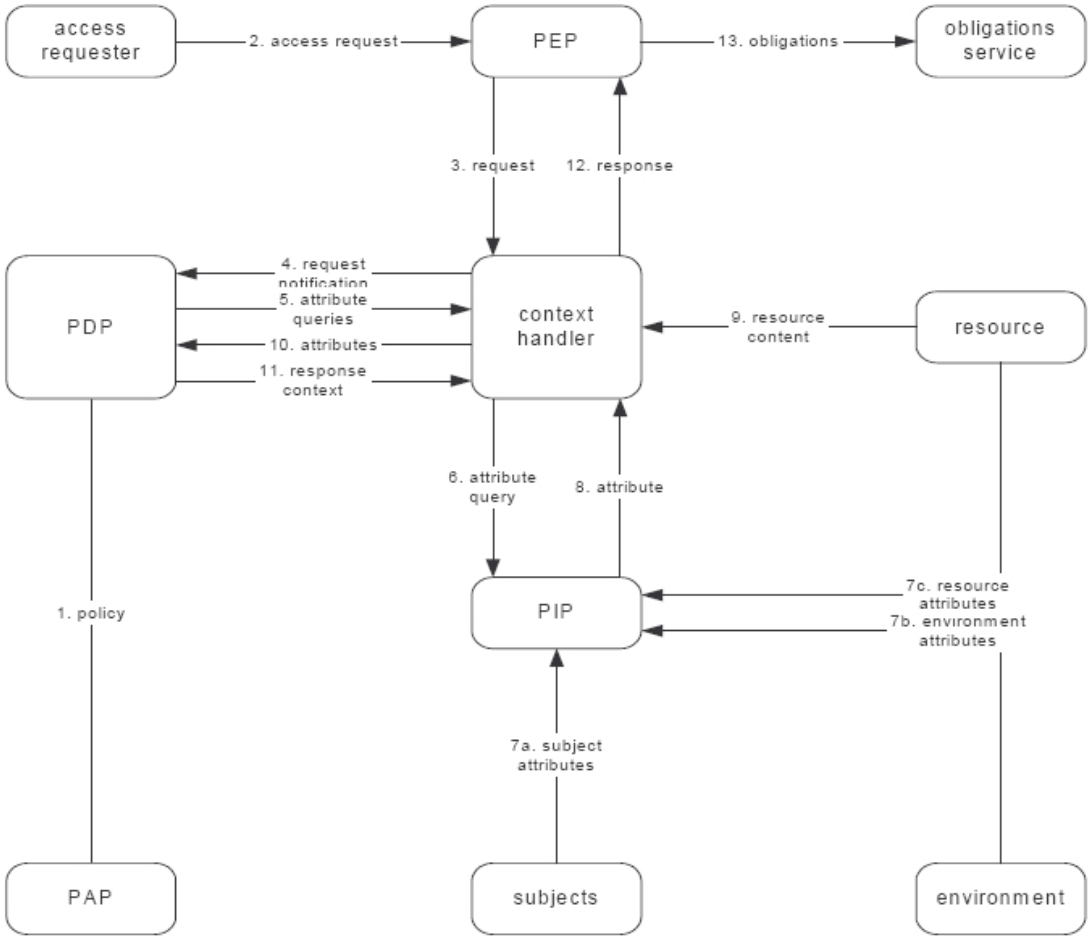

Figura 5.2: Fluxo de dados do modelo XACML (GODIK; MOSES, 2002)

avaliação de política feita no PDP ou LPDP.

PIP: fornece informações sobre sujeitos, recursos, e ambientes ao *context handler*.

Context Handler: constrói requisições de decisão em XACML, contento informações contextuais, e as envia ao PDP. Além disso, traduz respostas XACML para o formato nativo do PEP.

PDP: identifica qual politica se aplica a uma determina requisição, e retorna o resultado da avaliação de autorização (*Permit*,*Denied*) ao *context handler*.

O módulo definido como *Obligations* é utilizado pelo PEP para executar as ações recebidas nas decisões de políticas. Tais ações são chamadas de <Obligation>, em XACML, não devendo, contudo, serem confundidas com as obrigações mencionadas no modelo UCON_{ABC}.

Uma política XACML possui *targets* que identificam a qual sujeito, recurso, e ação, ela se aplica. O *target* pode ser definido num escopo mais amplo, afetando todas as regras que

não possuírem um *target* próprio, ou pode ser definido para cada regra individualmente.

A especificação prevê a utilização de operadores lógicos (i.e. *and*, *or*, *if*, etc.) para a construção de regras complexas de acesso. Além disso existem diversos algoritmos de combinação de avaliação de política que permitem usar esquemas do tipo *deny overrides* ou *ordered deny overrides*, entre outros.

Para a criação e avaliação de políticas e requisições XACML utilizamos a biblioteca Sun XACML (Sun Microsystems, Inc., 2007). Esta biblioteca fornece basicamente toda a especificação XACML 1.1, permitindo a fácil construção de PDPs, a manipulação de documentos XACML como objetos Java, etc. Esta biblioteca contudo não implementa componentes como o *context handler*, sendo que este componente é parte do protótipo e vai além do previsto no modelo de fluxo de dados apresentado na Figura 5.2.

5.2 Arquitetura do Protótipo

A arquitetura do protótipo é apresentada na Figura 5.3. Para fins de avaliação limitamos o protótipo aos aspectos mais importantes do controle de uso, e focamos nos aspectos envolvidos no domínio do consumidor e do provedor.

Para a avaliação do protótipo, somente as partes consideradas fundamentais para o controle de uso em nível de usuário foram implementadas. As partes relativas ao nível de serviço não foram implementadas (e.g. controle e notificação em nível de contratos).

Para a autenticação de usuários, representando o serviço de identificação, e também para proteção das mensagens, foram executadas experiências utilizando uma infra-estrutura de chave pública baseada em certificados X.509. O provedor configura seus serviços para confiarem nas chaves assinadas pelo consumidor.

A cifragem das mensagens também foi implementada utilizando WS-Security, o que permite um nível maior de segurança fim-a-fim, diferente de outros mecanismos que somente cifram a comunicação de modo ponto-a-ponto, permitindo a ocorrência de brechas de segurança antes da aplicação final.

5.2.1 Plataforma do Consumidor

Na plataforma do Consumidor, existe basicamente a aplicação cliente. Esta aplicação é responsável por enviar cargas de dados ao serviço de storage, e verificar as notificações existentes para o usuário (Figura 5.3, passos 1b e 1a, respectivamente).

A aplicação cliente cria duas *threads*, uma responsável por verificar notificações periodicamente, e outra que envia dados para o serviço de *storage*. Quando a *thread* das notificações encontra uma notificação para o cliente, uma *flag* é ativada para que a *thread* dos dados saiba que uma exceção ocorreu na utilização do serviço.

A *thread* de envio de dados verifica as *flags* de exceção sempre antes de tentar invocar o serviço de *storage*. Caso encontre uma *flag* ativada, a *thread* executa um *checkpoint* da tarefa sendo realizada, apresenta a mensagem da notificação ao usuário, e desliga o serviço de modo a não perder informações do trabalho já realizado.

Este mecanismo permite que, juntamente com o estabelecimento de um conjunto de notificações bem definidos, um certo nível de inteligência seja embutido na aplicação do cliente para tratar de violações da política de uso sem a intervenção do usuário.

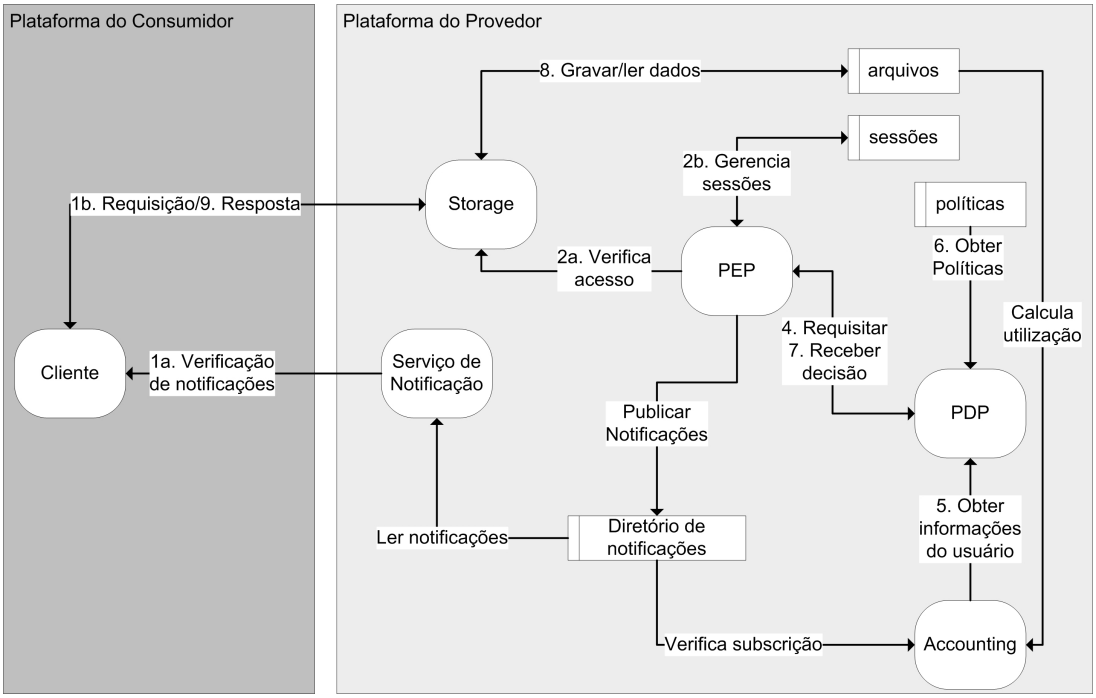

Figura 5.3: Visão geral das entidades do protótipo

5.2.2 Plataforma do Provedor

O provedor possui quatro serviços web: Serviço de notificação, *Storage*, *Accounting*, e PDP. O PEP funciona como um processo que roda em *background*. A aplicação cliente apenas interage com o *Storage* e o Serviço de notificação, estando os demais serviços ocultos para o mesmo.

Storage

O *Storage*, ao receber uma requisição, verifica inicialmente se existe uma sessão SOAP já existente para esta requisição. As sessões SOAP são mantidas através de *cookies*, e servem basicamente para distinguir invocações paralelas executadas pelo mesmo usuário.

Ao detectar a não existência de uma sessão, um identificador único (ID) é criado e armazenado no *cookie*. Uma solicitação de avaliação é realizada de modo síncrono, ou seja, o *Storage* aguarda o retorno do PEP. Em caso de recusa de acesso nesta primeira invocação do PEP, o *Storage* retorna uma mensagem de erro para o usuário e destrói a sessão SOAP.

Caso uma sessão SOAP já exista, o *Storage* verifica uma *flag* indicadora de decisão de uso. Esta *flag* é criada pelo PEP. Caso a *flag* de recusa esteja ativa, o serviço aguarda um momento pre-determinado após o qual a *flag* é novamente verificada. Caso a recusa continue ativada, uma mensagem de violação é retornada ao usuário e a sessão SOAP é destruída, caso contrário, um arquivo é criado no diretório de sessões do PEP, sinalizando que a sessão continua ativa. Este recurso permite que a aplicação do usuário tente corrigir a violação de política e que, desta maneira, a utilização do serviço não precise ser revogada.

Nos casos em que a avaliação de políticas retorna uma permissão, os dados do usuário são armazenados numa estrutura de arquivos com uma hierarquia do tipo *nome da organização/identificador do usuário*. A transferência dos dados entre a aplicação cliente e o serviço se dá através de anexos binários no envelope SOAP, em conformidade com as especificações.

PEP

O PEP funciona como um processo que verifica regularmente as sessões de uso existentes, solicitando a avaliação das políticas de uso aplicáveis a cada uma delas. As sessões são mantidas como diretórios nomeados com o ID da sessão SOAP. Este diretório é criado na primeira interação da aplicação cliente com o serviço de *Storage*. A comunicação entre o PEP e o serviço de *Storage* se dá através do sistema de arquivos.

Quando uma sessão necessita de reavaliação de política, algumas informações básicas são coletadas pelo PEP (e.g. nome de usuário, momento da requisição, nome da organização consumidora), que cria uma requisição XACML e envia ao PDP. Ao receber a resposta, verifica-se o tipo. Se for uma permissão, um arquivo é criado no arquivo de sessão correspondente, de modo que o *Storage* possa saber que o acesso continua permitido,

este arquivo faz o papel da *flag* mencionada na sessão anterior.

Caso o PDP retorne uma recusa, o PEP recupera o motivo da recusa contido na resposta do PDP. Um arquivo é então criado em outro diretório, monitorado pelo serviço de notificações, função esta que corresponde à publicação de notificações. Um arquivo é criado indicando ao *Storage* que a sessão está em suspenso, o PEP então aguarda um tempo pré-configurado e tenta uma nova avaliação da política. Caso o resultado continue como recusado, um arquivo é criado no diretório sinalizando que a sessão foi realmente revogada. Deste modo o *Storage* pode fechar sua própria sessão SOAP e encerrar o uso.

A regularidade com que as políticas de uso são avaliadas é uma das principais dificuldades na implementação do controle de uso. Idealmente a verificação deveria ser em tempo real ou, no caso do *Storage*, a cada pacote de dados recebido da aplicação cliente. Esta abordagem, no entanto, acarretaria uma sobrecarga no serviço de PDP, e uma conseqüente degradação na velocidade de resposta do serviço utilizado pela aplicação cliente.

Diferentemente da abordagem adotada em trabalhos prévios, optamos por um PEP que solicita a avaliação de políticas em intervalos de tempo pré-definidos, pois não é possível implementar o principio da continuidade, definido pelo UCON_{ABC}, em avaliações efetivamente ininterruptas, assim é necessária a discretização do período de avaliações. Com isso reduzimos consideravelmente o impacto sobre a velocidade de resposta do serviço provido e o *overhead* do sistema, assim como o impacto sobre o PDP. Isto causa, contudo, um comprometimento no tempo de detecção das violações de política.

Como o processo do PEP pode precisar solicitar a reavaliação de muitas sessões, optamos por implementar a lógica de invocação do PDP como *multithread*, como mostra o fragmento de código da Figura 5.4. Para cada diretório de sessão encontrado (linha 02), o PEP verifica se a sessão já está registrada para avaliação (linha 03), em caso negativo, uma *thread* de trabalho é criada, passando o identificador de sessão como parâmetro (linha 04). A *thread* é então iniciada (linha 05), executando as tarefas necessárias para invocar o PDP.

Os administradores, tanto do Consumidor como do Provedor, devem estar cientes disto, no momento de estabelecer a periodicidade de avaliação das políticas, de modo que os "abusos" decorrentes destas detecções tardias estejam dentro de limites aceitáveis. Como veremos no capítulo relativo a avaliação do protótipo, mais de um fator pode influenciar na maneira como esta margem de erro se comporta.

```
01 while { shutdown.equals(false) }; do
02  for i in sessions/*; do
03        if(!sessions.hasKey(i)); do
04           thr = new WorkerThread(i);
05           thr.start();
06        end if;
07      end for;
08 end while;
```

Figura 5.4: Fragmento de código do PEP

A limpeza do diretório de sessões é efetuada após o término da mesma, seja por pedido direto do usuário, por inatividade por tempos prolongados, ou por revogação do acesso. Um arquivo é criado no diretório da sessão, quando uma destas condições é detectada, indicando que o mesmo deve ser removido. A *thread* de trabalho responsável pela sessão a ser removida, identifica o arquivo, e toma as providências para removê-lo fisicamente, feito isso, a *thread* também apaga as referências no registro de sessões e libera seus recursos, para que possa ser finalizada.

Serviço de Notificação

O serviço de notificação basicamente recebe requisições de leitura dos usuários do Consumidor. Quando o usuário interage com o serviço, um arquivo é criado registrando o momento em que o usuário acessou o serviço. Através deste arquivo, é verificado posteriormente se o usuário está realmente cumprindo sua obrigação de verificar notificações regularmente.

As notificações são obtidas a partir do sistema de arquivos, em uma estrutura de diretórios que é composta pelo nome da organização consumidora, tendo como sub-diretórios o identificador dos usuários. Após a leitura e envio da notificação ao usuário, os arquivos de notificações são removidos de modo a não serem enviados repetidamente ao usuário.

Accouting

O serviço de *Accounting*, na arquitetura conceitual chamado de infra-estrutura de monitoração, é responsável por fornecer informações ao PDP, as quais serão utilizadas na avaliação de políticas. De certo modo, este serviço pode ser comparado ao PIP encontrado no modelo proposto na especificação XACML.

Neste protótipo, o serviço recupera basicamente três informações: a quantidade de

espaço em disco utilizado pelo usuário, a quantidade de disco utilizada pela organização consumidora em sua totalidade, e o último momento em que o usuário foi avistado acessando o serviço de notificação.

Para calcular a quantidade de disco utilizado, o serviço percorre a estrutura de arquivos do serviço de *Storage* e faz a consolidação dos dados. O momento de utilização do serviço de notificação é verificado através do *timestamp* do arquivo criado previamente por este serviço. Desta maneira, sempre que as informações de *Accounting* são solicitadas, elas serão sempre as mais recentes possíveis.

Criação das políticas em XACML

Na Seção 4.2.9 mencionamos a criação de uma política baseada em predicados, a qual deve ser convertida em um formato compreensível por mecanismos computacionais. A seguir será mostrada a maneira como uma política é convertida em seu equivalente XACML, isto é, o *parser* gerador de XACML.

A linguagem XACML não é baseada em predicados, contudo, é possível utilizar suas funcionalidades para criar estruturas que representam o comportamento desejado. A idéia consiste em identificar os operadores lógicos que atuam na implementação do predicado da política de uso, e então converte-lo para uma função XACML correspondente.

O comportamento do avaliador de políticas fornecido pela biblioteca XACML é de avaliar a política e, em caso de acesso negado, simplesmente retornar a decisão, sem especificar em qual regra o acesso foi invalidado. Por este motivo optamos por representar cada predicado UCON_{ABC} como sendo uma única política XACML. Isto permite saber exatamente em que predicado a avaliação parou e, desta maneira, retornar uma notificação condizente para a aplicação cliente.

A Figura 5.5 mostra um predicado que simplesmente verifica se o sujeito *user* possui quota em disco disponível. A implementação do predicado é composta por uma expressão lógica (linhas 02 à 07).

Se observarmos a Figura 5.5, em sua linha 06, perceberemos que existe um operador básico do tipo *or*. Em XACML existe uma função equivalente chamada *function:or*, apresentada na Figura 5.6, na linha 02.

O primeiro operando do *or*, é outra expressão lógica, desta vez um operador *and*, Figura 5.5, linha 03, que tem como correspondente XACML a função *function:and*, Figura 5.6, linha 03. O primeiro operando do predicado é operador relacional “menor-ou-

```
01 verifyQuota( user )
02        (( service.quotaOrg( user.OrgID ) le (50))
03            and
04            ( service.quotauser( user.ID) lt 10 )
05         )
06         or
07         ( service.quotauser( user.ID ) lt 5 )
```

Figura 5.5: Predicado verifyQuota

igual" (*le*), verificando se a quota consumida é menor que 50 megabytes. O teste *le* é convertido para a função *function:integer-less-than-orequal*. Neste caso *service.quotaOrg(user.OrgID)* é um identificador de atributo, que é convertido para o seu nome de mecanismo em XACML (i.e. service.quotaOrg.userOrgID, Figura 5.6, linha 06).

Processo semelhante ocorre com as expressões nas linhas 04 e 07. Como pode se perceber, o resultado na Figura 5.6 é extensamente declarativo, sendo que ainda foram omitidos diversos aspectos sintáticos para facilitar a visualização dos detalhes mais importantes.

A política XACML resultante consiste em um cabeçalho contendo um *target* no escopo mais amplo, utilizado para definir à qual combinação de usuário, ação, e recurso, a mesma se aplica. O corpo da política consiste em uma regra permissiva, como a apresentada na Figura 5.6, e outra negativa que é considerada a regra padrão. O algoritmo padrão utiliza uma lógica *permit overrides*, ou seja, se a regra permissiva for verdadeira, então o acesso é permitido. Caso contrário, a avaliação resulta em uma negação por padrão.

A *tag* <Obligation> é utilizada para colocar o conteúdo da notificação que deve ser retornada à aplicação cliente, em caso de violação da política. Deste modo, se a avaliação da política resultar em uma negação, o PEP terá uma mensagem adequada para publicar no serviço de notificação.

Devido a complexidade inerente à linguagem XACML, a utilização de uma linguagem de escrita de políticas simplificada, como a apresentada na Figura 5.5, pode reduzir consideravelmente o trabalho dos administradores de políticas de controle de uso, além de reduzir a possibilidade de ocorrerem erros de sintaxe, exceto nos casos em que o próprio *parser* for implementado incorretamente. Para avaliar o protótipo as políticas XACML foram convertidas manualmente.

```
01 <Rule RuleId="verifyQuota" Effect="Permit">
02  <Condition FunctionId="function:or">
03   <Apply FunctionId="function:and">
04    <Apply FunctionId="function:integer-less-than-orequal">
05     <Apply FunctionId="function:integer-one-and-only">
06      <ResourceAttributeDesignator AttributeId="service.quotaOrg.userOrgID"
07                                   DataType="integer"/>
08     </Apply>
09      <AttributeValue DataType="integer">
10       50
11      </AttributeValue>
12    </Apply>
13    <Apply FunctionId="function:integer-less-than">
14     <Apply FunctionId="function:integer-one-and-only">
15      <SubjectAttributeDesignator AttributeId="service.quotaUser"
16                                  DataType="integer"/>
17     </Apply>
18     <AttributeValue DataType="integer">
19      10
20     </AttributeValue>
21    </Apply>
22   </Apply>
23   <Apply FunctionId="function:integer-less-than">
24    <Apply FunctionId="function:integer-one-and-only">
25     <SubjectAttributeDesignator AttributeId="service.quotaUser"
26                                 DataType="integer"/>
27    </Apply>
28    <AttributeValue DataType="integer">
29     5
30    </AttributeValue>
31   </Apply>
32  </Condition>
33 </Rule>
```

Figura 5.6: Fragmento de XACML

PDP

O PDP foi desenvolvido de modo a ampliar o funcionamento normal de um PDP XACML comum, considerando que estes não possuem a semântica UCON$_{ABC}$. Para isso, internamento o PDP foi dividido em dois componentes básicos: um gerenciador de contexto, e um avaliador de política (O PDP XACML propriamente dito).

O gerenciador de contexto é a parte que recebe as requisições do PEP. Ao receber uma requisição do PEP, o gerenciador de contexto recupera os dados de identificação da requisição. Inicialmente ele invoca o serviço de *Accounting* para obter as informações de *accounting* pertinentes à requisição sendo avaliada. Após receber estas informações, o

próximo passo é recuperar as políticas aplicáveis.

Estas políticas estão armazenadas numa hierarquia de diretórios sob a forma *organização consumidora/ID/estagio/tipo*, onde *organização consumidora* é a organização à qual o usuário pertence, ID é o código identificador (credencial de identificação) do usuário, *estágio* é um valor que pode ser *pre* ou *on*, e *tipo* representa a família do predicado (A = autorizações, B = obrigações, C = condições).

Seguindo a lógica sugerida na Figura 4.4, o gerenciador de contexto passa à avaliação de políticas. Primeiro executa uma listagem no diretório de autorizações e, para cada política encontrada, uma instância de PDP XACML é criada, e a requisição é avaliada. Caso o retorno seja uma permissão, passa-se à avaliação da próxima política, destruindo a instância antiga do PDP e criando-se uma nova e reconfigurada instância.

O estágio da requisição vem do PEP, o que permite ao PDP distinguir em qual subdiretório procurar a política. Caso nenhuma das avaliações retorne uma negação, o gerenciador de contexto segue percorrendo os diretórios envolvidos até o fim da hierarquia. Caso contrário, a mensagem de negação do PDP é retornada para o PEP, que se encarregará de gerar a notificação, e configurar as *flags* do serviço de *Storage*. O PDP é um objeto *stateless* que é destruído após cada invocação.

5.3 Conclusão

Este capítulo apresentou as tecnologias utilizadas no desenvolvimento do protótipo, assim como a sua estruturação, em termos de componentes e suas interações. Discutimos também algumas decisões de projeto que, como veremos no capítulo a seguir, podem impactar consideravelmente no desempenho do sistema.

Possivelmente a mais importante decisão de projeto refere-se ao tempo de reavaliação das políticas, o que exige a aceitação de que existe a possibilidade de uma janela de tempo em que as políticas podem ser violadas. Esta periodicidade deve ser adaptada conforme as necessidades da aplicação específica, de modo a encontrar a melhor relação entre a velocidade na detecção das violações e o impacto sobre o desempenho do sistema. O próximo capítulo apresentará os testes realizados para avaliar o protótipo, juntamente com uma discussão dos resultados obtidos, permitindo uma melhor compreensão deste tipo de questionamento.

6 Avaliação do protótipo e resultados

Para a realização dos testes de avaliação foram utilizados dois *hosts* conectados por uma rede *ethernet* de 1 gigabit. O *host* do cliente é composto de um sistema operacional Linux, *kernel* versão 2.6.24-22, com 2 gigabytes de memória RAM, e processador de 1.8 gigahertz. O *host* do provedor é baseado em sistema operacional Windows XP Professional, com Service Pack 2, processador de 2.2 gigahertz CORE 2 DUO, e 3 gigabytes de memória RAM.

Todas as comunicações realizadas entre os componentes do provedor foram realizadas dentro do mesmo *host*, não sofrendo o impacto do tempo de transmissão de rede. Esta abordagem foi adotada para simplificar o ambiente de testes, contudo, os componentes se comunicam em parte utilizando invocações de serviços web, o que os credencia à serem executados em *hosts* separados sem que muitas mudanças sejam efetuadas. As comunicações efetuadas entre a aplicação cliente e o provedor são sempre via serviços web.

Para a avaliação do sistema a segurança da transmissão de dados foi desabilitada devido a complexidade da configuração dos componentes, a qual não está automatizada. A cifragem total das mensagens deve ser utilizada somente onde for necessário, pois seu uso pode representar um impacto considerável sobre a performance do sistema, o que não é objeto de consideração de nossa avaliação.

A implementação do protótipo e o estudo dos resultados obtidos durante sua avaliação possibilitou a publicação do artigo "*Framework de Controle de Uso para Coalizões Dinâmicas de Negócios*" (STIHLER; SANTIN; MARCON Jr., 2008), na XXXIV Conferência Latinoamericana de Informática (CLEI'2008), assim como o aceite do artigo "*Distributed Usage Control Architecture for Business Coalitions*", no *IEEE International Conference on Communications 2009* (ICC'2009) – *Communication and Information Systems Security* (CISS).

6.1 Testes realizados e resultados obtidos

Quatro testes foram realizados separadamente para avaliar o protótipo implementado. O primeiro teste consistiu na avaliação de performance do serviço de *Storage* para atender um único cliente, com variações no tamanho do pacote de dados transmitidos. O segundo teste mediu o impacto do número de predicados sobre o tempo de resposta do PDP. No terceiro teste, avaliamos a capacidade do PDP atender múltiplas requisições em paralelo. Por fim, no quarto teste avaliamos o tempo de resposta do serviço de *Accounting*. Em todos os experimentos, foi observado um desvio padrão abaixo de 5%.

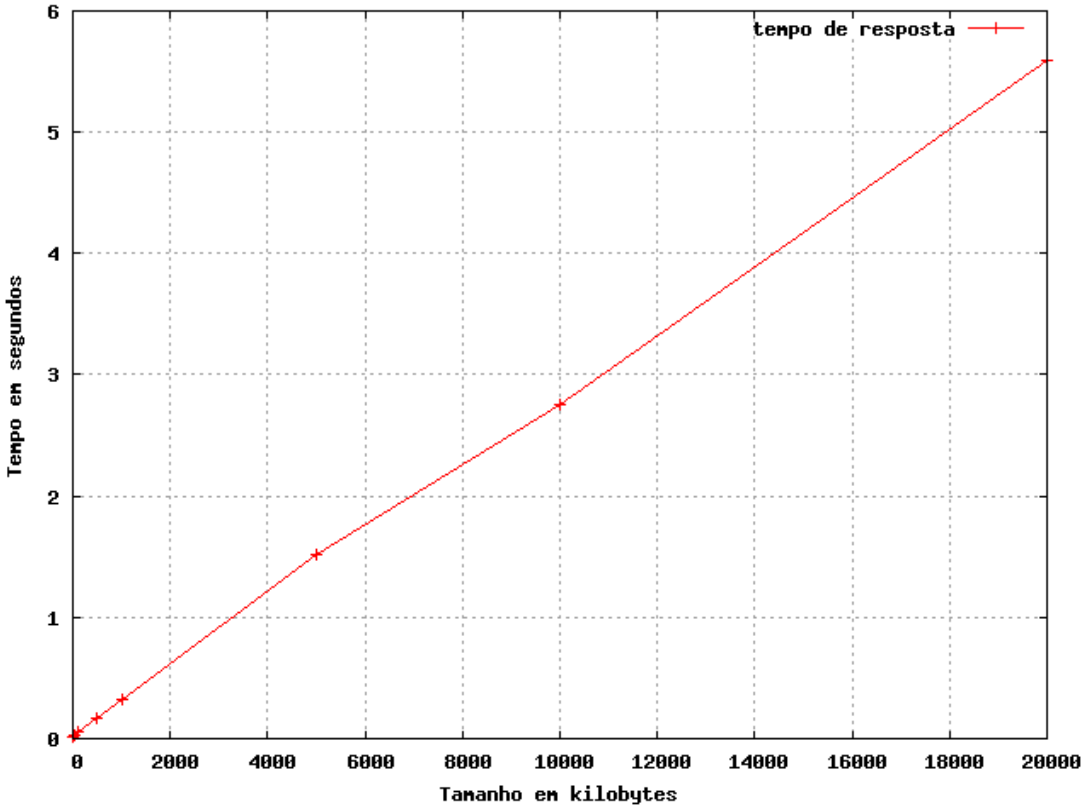

Figura 6.1: Impacto do tamanho dos pacotes no serviço de Storage

No teste de avaliação do impacto do tamanho dos pacotes no desempenho do serviço de *Storage*, foram executadas 50 requisições consecutivas, e calculada a média dos tempos de resposta. A Figura 6.1 apresenta os resultados deste resultado, iniciando em pacotes com 10 kilobytes (kb), seguindo para 50 kb, 100 kb, 500 kb, 1000 kb, 5000 kb, 10000 kb, e 20000 kb.

Como era esperado, o tamanho do pacote tem um grande impacto no tempo de resposta. Este parâmetro deve ser ajustado conforme a função do serviço. O tamanho do pacote irá influenciar diretamente na margem de erro do sistema no momento de detecção

de violações. Um tamanho de pacote maior pode beneficiar o rendimento do sistema, pois apesar do tempo das respostas ser maior, o tempo gasto para manipulação de mensagens é menor. No entanto, isso também significa que um erro na detecção da violação incorrerá uma quantidade consideravelmente maior de dados sendo armazenados de maneira ilegal.

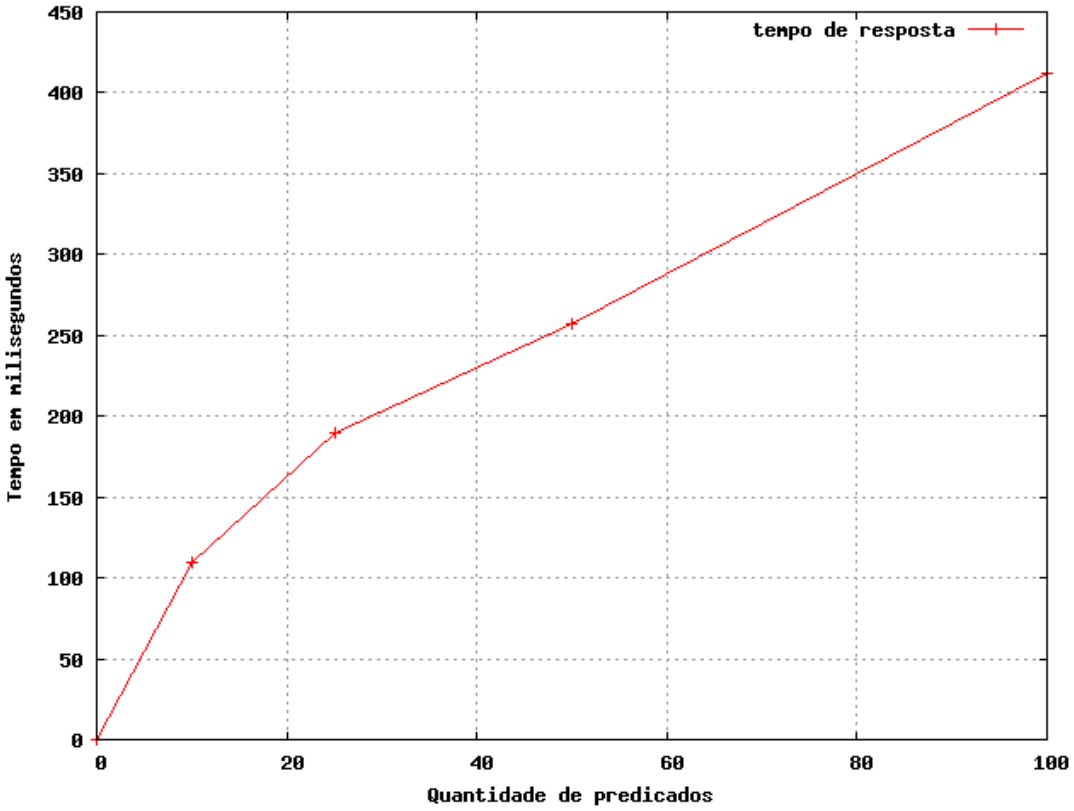

Figura 6.2: Impacto do número de predicados

A Figura 6.2 apresenta os resultados do segundo teste. Como podemos ver, o tempo de avaliação aumenta de numa proporção de 4/10 em relação à quantidade de predicados a serem avaliados. Para este teste utilizamos predicados compostos, como aquele apresentado na Figura 5.6.

A evolução dos tempos, no entanto, não é diretamente proporcional. Consideramos uma política de 100 predicados como sendo bastante complexa, já que devem ser escritas por um administrador humano, sendo desnecessária a verificação do impacto de milhares de predicados.

O tempo de resposta deste teste é importante para sabermos o tempo mínimo entre as reavaliações de políticas. Não adianta configurar o PEP para solicitar reavaliações de políticas com maior rapidez que a resposta média do PDP, pois o mesmo terá de ficar aguardando de qualquer maneira. Considerando que no primeiro teste obtêmos um tempo médio para 100 kb em torno de 500 milisegundos (T1), e o tempo para 100 predicados de

cerca de 400 milisegundos (T2), pode-se deduzir que um tempo mínimo razoável seria de 1 segundo (arrendondamento de T1+T2).

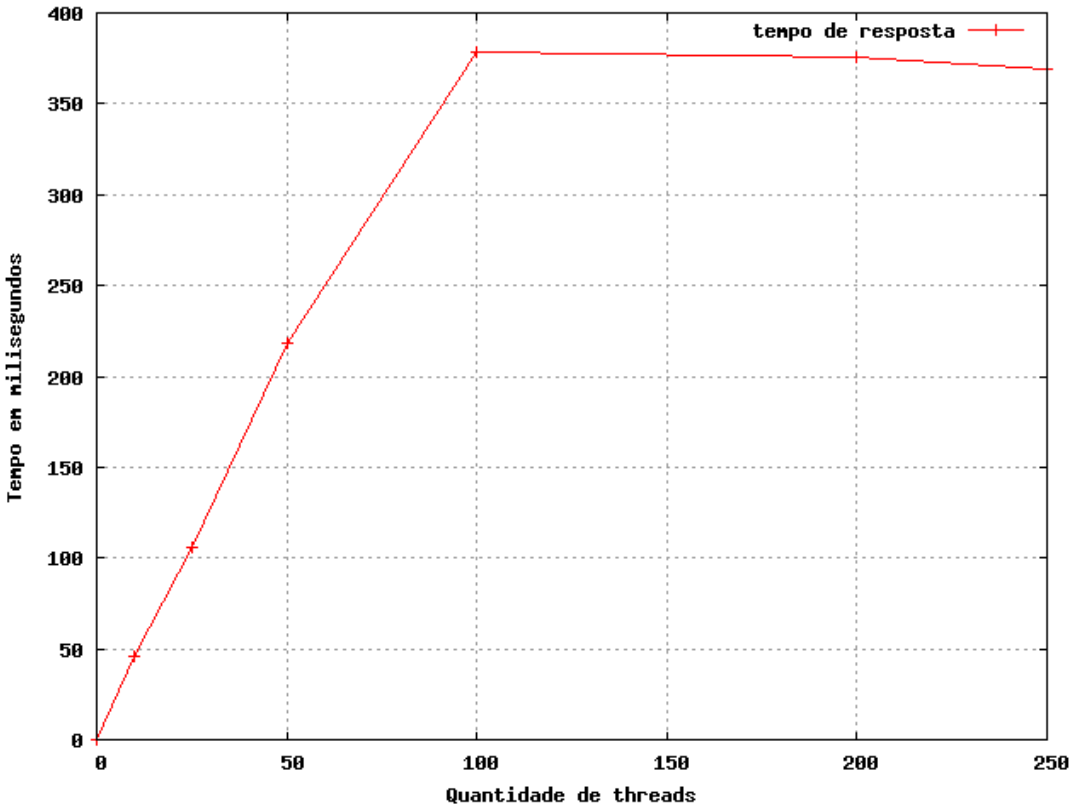

Figura 6.3: Impacto do número de *threads* na resposta do PDP

No terceiro teste, Figura 6.3, verificamos o tempo de resposta do PDP para responder à diversas *threads* de trabalho do PEP. As invocações são feitas como serviços web. Neste tempo de resposta, somente um predicado composto é avaliado pelo PDP.

A análise do gráfico mostra que a partir de 100 *threads* o tempo de resposta tende a estabilizar, inclusive com uma pequena melhora na faixa de 200 a 250 *threads*. Esta melhora provavelmente decorre do processadores dual core do *host* do Provedor, entretando tal hipótese deve ser investigada com mais profundidade.

A Figura 6.4 apresenta o resultado do quarto teste. Neste teste o serviço calculou o consumo de espaço em disco utilizados pela organização Consumidora e individualmente para cada usuário. O serviço é invocado através de uma chamada de serviços web. No sistema de arquivos havia somente um arquivo de 1 gigabyte de tamanho.

Como se pode observar na Figura 6.4, o tempo de resposta é bastante influenciado pelo número de *threads*. Entretando os tempos de resposta são razoavelmente baixos, estando abaixo de 100 milisegundos para atender 100 *threads*. Diferente dos testes realizados com

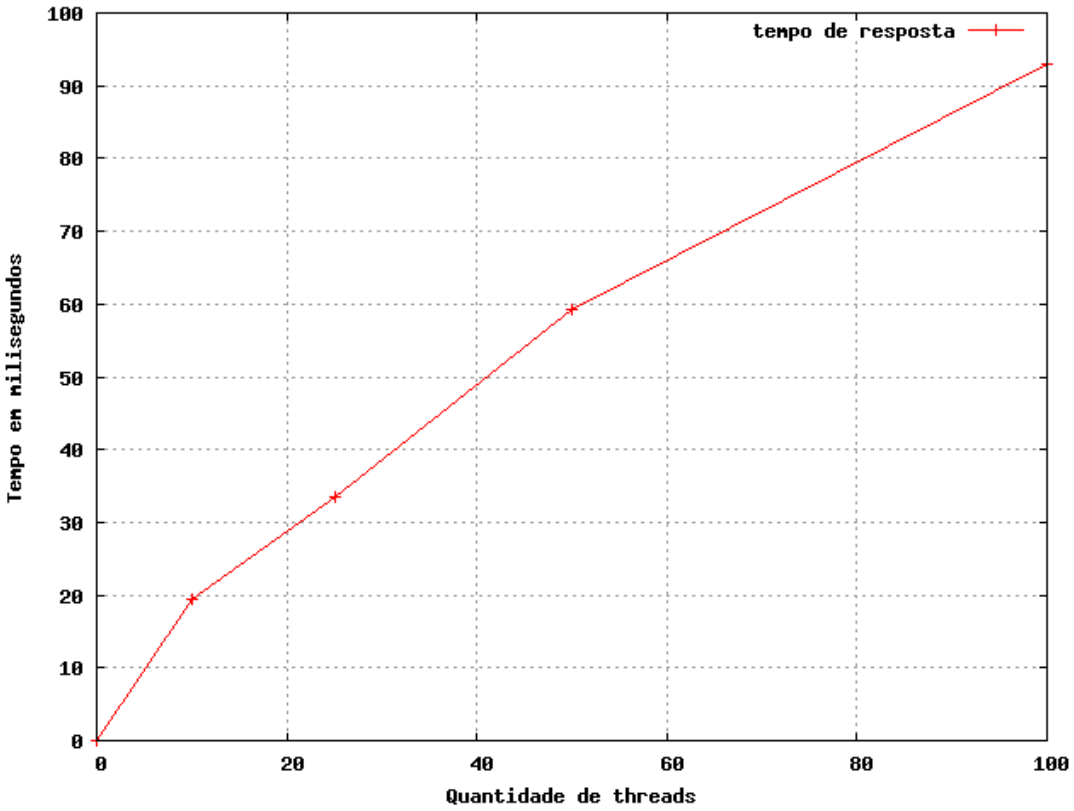

Figura 6.4: Impacto do número de *threads* na resposta do serviço de *Accounting*

o PDP, neste teste não foi possível realizar chamadas para mais de 100 *threads* pois o sistema do servidor apresenta instabilidades no gerenciamento das mesmas. Uma das possíveis hipóteses para este comportamento deve-se ao curto tempo entre as invocações, não havendo tempo para o sistema reciclar as *threads* finalizadas.

Com base nestes dados podemos ampliar o cálculo anterior do tempo mínimo de intervalo de revalidações, considerando o número de conexões em paralelo esperadas. Por exemplo, para pacotes de 100 kb (500 milisegundos), soma-se o tempo de resposta correspondente aos 100 predicados (400 milisegundos), mais o tempo de resposta para requisições em paralelo ao PDP, por exemplo 100 requisições (380 milisegundos), e o tempo de resposta do serviço de *Accounting* (95 milisegundos), o que resultado num tempo T3 aproximado de 1,3 a 1,5 segundos. Conhecendo este valor podemos evitar que a *thread* do PEP fique bloqueada por longos períodos esperando a resposta do PDP.

6.2 Margem de erro

Como mencionado já neste trabalho, os administradores devem decidir qual a margem de erro aceitável para a utilização do sistema, sem impactar demasiadamente de modo

negativo o uso do sistema. Com base no resultado dos testes anteriores é possível realizar algumas discussões sobre este tema. Observamos que o sistema de *Storage* trabalha com fragmentação de pacote em nível de aplicação.

O tempo final de resposta para o usuário segue aproximadamente a soma da média do tempo de resposta do serviço de *Storage*, da invocação do PDP, invocação do serviço de *Accounting*, e número de predicados envolvidos na política. Considerando um pacote mínimo de dados de 5 kb, pode-se concluir que o controle mais fino possível consome no mínimo um tempo igual a T3 = 1,5 segundo. O que não proporciona um bom controle se considerarmos o número de pacotes que serão gravados antes da detecção da violação. Contudo, o tamanho dos pacotes é pequeno, logo a quantidade de dados armazenada em violação seria pequena. Depende do administrador deliberar se, dentro do contexto da aplicação, esta margem de erro é aceitável ou não.

Em casos de grande proporção, imaginemos o serviço de *Storage* trabalhando com quotas de vários gigabytes, o controle possui margem de erro baixa. Trata-se, neste caso, de considerar qual o custo se deseja impor ao usuário final. Suponhamos uma quota de 5 gigabytes. Se o usuário estiver utilizando um pacote (fragmento) de 20 megabytes, ele levará cerca de 4.7 minutos para atingir a sua quota. Se utilizar um pacote de 5 megabytes, o usuário irá levar aproximadamente 5.15 minutos para atingir a quota. Considerando que as políticas sejam reavaliadas a cada 30 segundos, com pacotes de 20 megabytes poderá haver um estouro de quota de aproximadamente 109 megabytes. Utilizando pacotes de 5 megabytes, este estouro passa a ser aproximadamente de 99.3 megabytes. Quanto menor o pacote de dados, mais fino o controle, ainda que o tempo de revalidação seja alto.

Cabe ao administrador decidir se é preferível impactar o tempo final para o usuário para reduzir a margem de erro, ou aumentar a velocidade de resposta para o usuário com o prejuízo da precisão na aplicação das políticas. Os dados obtidos através destes testes são, contudo, apenas relevantes para o cenário do serviço de *Storage*, sendo que outros cenários requerem mais testes para que se possa obter um melhor *insight* sobre como decidir qual o melhor intervalo de tempo de reavaliação de políticas.

6.3 Conclusão

Neste capítulo foram apresentados os testes de avaliação do protótipo e os resultados obtidos. A análise dos dados permitiu um melhor entendimento de como o sistema se comporta, e como atingir um controle mais próximo do ideal. Além disso, os testes

serviram para mostrar que o sistema tem viabilidade prática, com tempos de resposta razoáveis para situações de uso real.

No próximo capítulo apresentaremos as conclusões do presente trabalho.

7 Conclusões e Trabalhos Futuros

O trabalho desenvolvido mostrou a viabilidade da proposta de controle de uso de UCON_{ABC} para recursos em sistemas colaborativos distribuídos. A arquitetura proposta oferece um gerenciamento de políticas mais razoável, dividindo as responsabilidades entre as partes envolvidas (consumidores e provedores). Devido à sua natureza dinâmica, e aos resultados obtidos na avaliação do protótipo, mostramos que UCON_{ABC} é bastante adequado a ambientes colaborativos distribuídos, e viável em termos de implementação e utilização.

Na parte conceitual, esta proposta apresentou idéias para facilitar a tarefa do administrador de sistemas, como a interface para escrita de políticas em uma linguagem de alto nível, o sistema de notificações que o mantém informado das violações existentes, permitindo que o mesmo tome as medidas corretivas, além da redução da tarefa de escrita de políticas, agora divida entre as partes.

A implementação do protótipo mostrou que é possível utilizar UCON_{ABC} sem a necessidade de extensões à especificação XACML. Com a utilização de mecanismos de software com a semântica do controle de uso, é possível utilizar XACML sem problemas.

O desacoplamento da utilização do recurso protegido do processo responsável por requisitar as reavaliações de políticas, permitiu uma melhora no desempenho percebido pelo usuário final. Em abordagens altamente acopladas, o usuário teria que arcar com reavaliações de políticas a cada interação com o serviço.

A implementação do protótipo também deixou clara a necessidade de se estabelecer uma relação entre o grau de controle do sistema, sua margem de erro, e o impacto que isso causa para o usuário final. Como mostramos, é tarefa do administrador definir qual o melhor intervalo de reavaliação de políticas. Caso exista a necessidade de um controle em tempo real, deve-se ter em mente que o custo de operação se torna bastante alto, podendo inviabilizar sua utilização na maioria das situações.

Além disso, o protótipo utilizou, em sua maior parte, especificações e tecnologias am-

plamente conhecidas e aceitas. A utilização da arquitetura de serviços web permitiu uma redução do acoplamento das aplicações, facilitando assim a implementação em ambientes heterogêneos.

Por estes motivos, consideramos que a proposta apresentada e avaliada neste trabalho, possui um conjunto de vantagens qualitativas que a diferencia dos demais trabalhos realizados anteriormente relacionados a este tema.

Como trabalhos futuros consideramos que protótipo deve ser melhorado para ampliar sua capacidade de generalização, e para implementar os mecanismos de atualização de atributos do tipo *pre* e *ongoing*. Para isso, um aspecto importante é remover os mecanismos de controle de uso para uma camada independente da camada de aplicação, de modo a realizar o controle sem necessidade de alterações no código da aplicação. Uma possível alternativa para isso, é a utilização de interceptadores localizados em nível de protocolo *SOAP*, de modo que as atividades de controle sejam realizadas antes da aplicação tomar conhecimento da tentativa de acesso.

Vislumbra-se também a implementação de um *parser* capaz de interpretar as regras escritas no formato de predicados, e gerar automaticamente as políticas em XACML. Formalizar a linguagem de escrita de predicados. Além disso, estudar e melhorar a implementação da biblioteca XACML utilizada para avaliar as políticas. Mais especificamente, implementar um PDP *multi-thread* para tirar proveito de múltiplos processadores.

Sugere-se ainda o estudo de cenários mais complexos como, por exemplo, um sistema de arquivos distribuído, ou serviços web compostos. Para isso será necessário aprofundar o estudo do sistema de *Accounting*, assim como uma maneira de manter a consistência na aplicação das políticas entre os diversos *hosts* que participam no provimento do serviço. Definir um protocolo que permita descrever as interações entre as entidades envolvidas.

Referências Bibliográficas

ALFIERI, R.; CECCHINI, R.; DELL'AGNELLO, L.; FROHNER, Á.; GIANOLI, A.; LöRENTEY, K.; SPATARO, F. Voms, an authorization system for virtual organizations. *Lecture Notes In Computer Science*, p. 33–40, 2004.

BOX, D.; EHNEBUSKE, D.; KAKIVAYA, G.; LAYMAN, A.; MENDELSOHN, N.; NIELSEN, H. F.; THATTE, S.; WINER, D. May 2000. Disponível em: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.

BREWER, D. F. C.; NASH, M. J. The chinese wall security policy. *IEEE Symposium on Research in Security and Privacy*, p. 206–214, 1989.

DOWNS, D. D.; RUB, J. R.; KUNG, K. C.; JORDAN, C. S. Issues in discretionary access control. *Proceedings of the 1985 IEEE Symposium on Security and Privacy*, p. 208–218, 1985.

EASTLAKE, D.; REAGLE, J.; SOLO, D. *XML-Signature Syntax and Processing*. 2002. Disponível em: <http://www.w3.org/TR/xmldsig-core/>.

EASTLAKE, D.; REAGLE, J.; SOLO, D.; HIRSCH, F.; ROESSLER, T. *XML Signature Syntax and Processing (Second Edition)*. 2008. Disponível em: <http://www.w3.org/TR/xmldsig-core/>.

ELLISON, C. *SPKI Requirements*. IETF, set. 1999. RFC 2692 (Experimental). (Request for Comments, 2692). Disponível em: <http://www.ietf.org/rfc/rfc2692.txt>.

ELLISON, C.; FRANTZ, B.; LAMPSON, B.; RIVEST, R.; THOMAS, B.; YLONEN, T. *SPKI Certificate Theory*. IETF, set. 1999. RFC 2693 (Experimental). (Request for Comments, 2693). Disponível em: <http://www.ietf.org/rfc/rfc2693.txt>.

FIELDING, R.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, P.; BERNERS-LEE, T. *Hypertext Transfer Protocol – HTTP/1.1*. IETF, jun. 1999. RFC 2616 (Draft Standard). (Request for Comments, 2616). Updated by RFC 2817. Disponível em: <http://www.ietf.org/rfc/rfc2616.txt>.

FOSTER, I.; KESSELMAN, C. Globus: A metacomputing infrascrtucture toolkit. *Intl. J. Supercomputer Applications*, v. 11, n. 2, p. 115–128, 1997.

GODIK, S.; MOSES, T. *OASIS eXtensible Access Control Markup Language (XACML) Specification 1.1*. 2002. Disponível em: <http://www.oasis-open.org/committees/xacml/repository/>.

GOSLING, J. *The Java Language Environment*. Sun Microsystems, May 1996. Disponível em: <http://java.sun.com/docs/white/langenv/>.

GOTTSCHALK, K.; GRAHAM, S.; KREGER, H.; SNELL, J. Introduction to web services architecture. *IBM Systems Journal*, v. 41, n. 2, 2002.

LAMPSON, B. W. Protection. *SIGOPS Oper. Syst. Rev.*, v. 8, n. 1, p. 18–24, 1974. ISSN 0163-5980.

LAWRENCE, K.; KALER, C. *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*. 2004. Disponível em: <http://docs.oasis-open.org/wss/v1.1/>.

LAWRENCE, K.; KALER, C. *WS-SecurityPolicy 1.2*. 2007. Disponível em: <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html#_Toc161826495>.

LAWRENCE, K.; KALER, C. *WS-Trust 1.3*. 2007. Disponível em: <http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html>.

LORCH, M.; ADAMS, D. B.; KAFURA, D.; KONENI, M. S. R.; RATHI, A.; SHAH, S. The prima system for priviledge management, authorization and enforcement in grid environments. *Proceedings of the 4th International Workshop on Grid Computing*, 2003.

MARTINELLI, F.; MORI, P.; VACCARELLI, A. Towards continuous usage control on grid computational services. *Proceedings of the The International Conference on GRID Networking and Services 2005*, 2005.

MOORE, B.; ELLESSON, E.; STRASSNER, J.; WESTERINEN, A. *Policy Core Information Model*. IETF, feb 2001. RFC 3060. (Request for Comments, 3060). Disponível em: <http://www.ietf.org/rfc/rfc3060.txt>.

PARK, J.; SANDHU, R. Biding indentities and attributes using digitally signed certificates. *Proceedings of the Annual Computer Security Applications Conference*, 2000.

PARK, J.; SANDHU, R. The UCON$_{ABC}$ Usage Control Model. *ACM Transactions on Information and System Security*, ACM, New York, NY, USA, v. 7, n. 1, p. 128–174, 2004. ISSN 1094-9224.

PEARLMAN, L.; WELCH, V.; FOSTER, I.; KESSELMAN, C.; TUECKE, S. A community authorization service for group collaboration. *Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks*, p. 50–59, 2002.

PU, F.; SUN, D.; CAO, Q.; CAI, H.; YANG, F. Pervasive computing context access control based on ucon$_{abc}$ model. *iih-msp*, IEEE Computer Society, p. 689–692, 2006.

RIVEST, R. L.; LAMPSON, B. *SDSI - A Simple Distributed Security Infrastructure*. 2007. Disponível em: <http://theory.lcs.mit.edu/ rivest/sdsi10.html>.

SANDHU, R. Lattice-based access control models. *IEEE Computer*, v. 26, n. 11, p. 9–19, 1993.

SANDHU, R. Engeneering authority and trust in cyberspace: The om-am and rbac way. *Proceedings of the 5th ACP Workshop in Role-Based Access Control (RBAC-00), 25.-26. July 2000, New York, NY*, ACM Press, p. 111–119, 2000.

SANDHU, R.; SAMARATI, P. Access control: Principles and practice. *IEEE Communications Magazine*, v. 32, n. 9, p. 40–48, 1994. ISSN 0163–6804.

SANDHU, R. S. Role-based access control. Academic Press, v. 48, 1998.

Schlimmer, J. (Editor). *Web Services Policy 1.2 - Framework (WS-Policy)*. 2006. Disponível em: <http://www.w3.org/Submission/WS-Policy/>.

SCHNEIDER, F. Enforceable security policies. *ACM Transactions on Information System Security*, v. 3, n. 1, p. 30–50, 2000.

STIHLER, M.; SANTIN, A. O.; MARCON Jr., A. L. Framework de Controle de Uso para Coalizões Dinâmicas de Negócios. *Conferência Latinoamericana de Informática - CLEI*, 2008.

Sun Microsystems, Inc. *Sun XACML Implementation*. 2007. Disponível em: <http://sunxacml.sourceforge.net/>.

The Apache Foundation. *Apache Axis2 Web services engine*. 2007. Disponível em: <http://ws.apache.org/axis2/>.

The Apache Foundation. *Apache Tomcat*. 2007. Disponível em: <http://tomcat.apache.org/>.

THOMPSON, M. R.; ESSIARI, A.; MUDUMBAI, S. Certificate-based authorization policy in a pki environment. *ACM Trans. Inf. Syst. Secur.*, ACM Press, New York, NY, USA, v. 6, n. 4, p. 566–588, 2003. ISSN 1094-9224.

W3C. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. Sep 2006. Disponível em: <http://www.w3.org/TR/REC-xml/>.

ZHANG, X.; NAKAE, M.; COVINGTON, M. J.; SANDHU, R. A usage-based authorization framework for collaborative computing systems. *Proceedings of the eleventh ACM symposium on Access control models and technologies*, p. 180–189, 2006.