Implementação de políticas UCON em um núcleo de sistema operacional

Rafael Coninck Teigão[1], **Carlos Maziero**[1], **Altair Santin**[1]

[1]Programa de Pós-Graduação em Informática
Pontifícia Universidade Católica do Paraná (PUCPR)
Curitiba – PR – Brasil

{teigao,maziero,santin}@ppgia.pucpr.br

Resumo. *O controle de uso vai além do controle de acesso tradicional e trata os aspectos relacionados à mutabilidade dos atributos e validação contínua das restrições de uso. O modelo UCON$_{\mathrm{ABC}}$ estabelece um ferramental formal básico para tratar as novas necessidades de segurança e sistemas de controle de uso. Como esse modelo foi apenas descrito por uma especificação formal, este trabalho o implementa na forma de uma gramática LALR(1). A gramática proposta é então utilizada para representar cenários de controle de acesso e uso comuns, demonstrando sua expressividade e utilidade. Por fim, a gramática é implementada em um mecanismo de controle de uso de arquivos no núcleo de um sistema operacional aberto.*

Abstract. *Usage control goes beyond traditional access control, addressing its aspects related to attribute mutability and continuous validation of usage restrictions. The UCON$_{\mathrm{ABC}}$ model establishes an underlying formal framework to deal with the new needs of security and usage control systems. As that model was only described by a formal specification, this paper implements it as an LALR(1) grammar. The grammar is then used to represent common access and usage control scenarios, showing its expressiveness and usefulness. Ultimately, the grammar is implemented in a file usage control mechanism in the kernel of an open operating system.*

1. Introdução

Os mecanismos de controle de acesso clássicos são insuficientes para a atual variedade de conteúdos digitais e utilização de recursos computacionais. Por exemplo, a gerência de direitos digitais (*Digital Rights Management* - DRM) introduz a necessidade de um controle de uso que vai além da simples concessão de permissões de acesso. Esta limitação também está presente na utilização e armazenamento de dados coletados de vários sujeitos, como informações médicas de pacientes de um hospital. O comércio eletrônico de ítens digitais traz a necessidade de verificar se algumas restrições adicionais estão sendo cumpridas, como a aceitação de um contrato de uso para um *software* ou restrições de tempo em uma transação comercial.

O conceito de controle de uso (*Usage Control* - UCON), introduzido por Park e Sandhu [Park and Sandhu 2003], define verificações de atributos/requisitos (*e.g.* permissões de um usuário para visualizar um filme) durante a utilização dos direitos concedidos

sobre um objeto, e considera a noção de *mutabilidade*, ou seja, as modificações desses atributos durante o uso, em função de ações do usuário ou consumo de créditos, por exemplo. Além disso, os autores apresentam a noção de *dependência dos atributos* de informações externas ao sistema de decisão (e que não são consideradas em modelos clássicos de controle de acesso).

O modelo de controle de uso UCON_{ABC} [Park and Sandhu 2004] é baseado nos conceitos de **A**utorização, o**B**rigações e **C**ondições que suportam as novas necessidades em controle de uso, suportando ainda os controles de acesso clássicos. Baseado na especificação formal descrita em [Zhang et al. 2004], este trabalho propõe uma gramática para descrever políticas UCON. Esta gramática foi concebida para ser suficientemente expressiva na representação de atributos e regras de autorização, obrigação e condição. Sua expressividade será demonstrada através de alguns exemplos de controle de uso.

O propósito desta gramática é permitir a um administrador associar atributos a usuários e requisitos a objetos. Além disso, permitir impor regras que usem estes atributos e requisitos para a tomada de decisão de uso ou acesso e conseqüentemente alterar valores de atributos em tempo de execução. A definição da gramática deve evitar a possibilidade de construir regras ambíguas; cada declaração deve possuir uma semântica única e não-ambígua. A viabilidade de uso da gramática proposta foi comprovada através de sua integração a um mecanismo de controle de uso de arquivos em um sistema operacional COTS, no caso o *OpenBSD*. No protótipo, as principais chamadas de sistema responsáveis pelo acesso e uso dos arquivos em disco foram vinculadas a atributos, obrigações e condições impostas pela gramática proposta.

O restante deste artigo está organizado da seguinte forma: a seção 2 traz uma descrição das principais características do modelo UCON_{ABC}; a seção 3 traz a definição da gramática proposta. Exemplos utilizando a linguagem representada por esta gramática são descritos na seção 4. A seção 5 descreve o protótipo implementado e sua avaliação. A seção 6 apresenta os trabalhos relacionados. Finalmente, a seção 7 sumariza este artigo e apresenta trabalhos futuros.

2. O Modelo UCON_{ABC}

Os mecanismos de controle de acesso atuais relacionam os atributos do objeto e do usuário para tomar decisões de acesso, mas estes atributos são normalmente específicos do mecanismo a que pertencem e na maioria das vezes podem ser alterados apenas por ações administrativas. O UCON_{ABC} [Park and Sandhu 2004, Park and Sandhu 2003] introduz a mutabilidade de atributos, o que fornece, através do histórico de uso, a possibilidade de influenciar eventos atuais ou futuros. É possível relacionar decisões de uso baseadas nos atributos que são alterados em cada acesso (por exemplo, em um sistema baseado em crédito: com cada novo acesso, o crédito do usuário é decrementado em algum valor; quando o usuário fica sem crédito, seu uso é negado). Atributos são também independentes de mecanismo: *Controle de Acesso Discricionário* (DAC), *Controle de Acesso Mandatório* (MAC) e *Controle de Acesso Baseado em Papéis* (RBAC), entre outros, podem ser implementados no UCON_{ABC} sem mudar as propriedades dos atributos.

Os principais controles oferecidos pelo modelo UCON_{ABC} são:

Autorização: conjunto de predicados funcionais avaliados para decidir se um usuário pode exercer um direito sobre um objeto. Estes predicados englobam os modelos

clássicos de controle de acesso, como o Controle Discricionário (DAC), Mandatório (MAC) e Baseado em Papéis (RBAC).

oBrigações: predicados funcionais que tratam ações que devem ser executadas antes ou durante o uso, para que um usuário possa exercer seus direitos sobre o objeto. Por exemplo, para um usuário conseguir executar um *software* novo, terá de aceitar uma licença de uso.

Condições: são restrições que consideram fatores ambientais ou do sistema, independentes dos usuários e objetos. As condições dependem apenas do estado do sistema, e não dos atributos do usuário ou do objeto, como definido por Park e Sandhu [Park and Sandhu 2004].

A avaliação das regras pode ser executada antes (*pre*) ou durante (*on*) o uso/acesso[1]. A atualização de atributos pode ser realizada antes (1), durante (2) ou após (3) o uso, ou pode nunca acontecer (0). É importante notar que, uma vez que as condições são informações do ambiente (*i.e.* o sistema que as está verificando não tem influência em seus valores), não há atualização de atributos relacionados às mesmas.

Os predicados do UCON_{ABC} podem ser avaliados para se obter direito de uso sobre um objeto ou para mantê-lo. Quando o usuário ainda não detém direito de uso, um conjunto de predicados deve ser avaliado **antes** do uso, constituindo uma avaliação `pre`. Assim, tem-se avaliações `preA`, `preB` e `preC`, relativas a Autorização, oBrigações e Condições. Da mesma forma, quando se avalia o direito de permanecer utilizando um recurso, a avaliação acontece **durante** o uso, sendo então chamada de `on`: `onA`, `onB` e `onC`. A figura 1 representa essa continuidade na tomada de decisão.

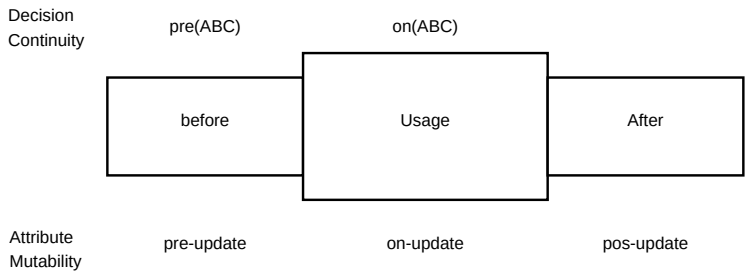

Figura 1. Momentos de Avaliação dos Predicados [Park and Sandhu 2004]

Como pode ser visto na figura 1, existem momentos diferentes para a verificação e também para a atualização de atributos. Atributos do recurso e do sujeito podem ser imutáveis (`0`), ou modificados antes (`1`), durante (`2`) ou após (`3`) o uso (*e.g.* $\texttt{onA}_0$, $\texttt{onA}_1$, $\texttt{onA}_2$ e $\texttt{onA}_3$). A combinação dos momentos de avaliação com os de atualização gera os 16 modelos centrais do UCON_{ABC}, indicados na tabela 1.

Para avaliações `pre`, não existe atualização durante o uso. Como não há avaliações neste momento (`on`), qualquer atributo alterado somente seria utilizado na próxima avaliação, antes do próximo uso, fazendo com que uma atualização `2` tenha o mesmo efeito de uma atualização `3` (após o uso). Nota-se, também, que os predicados de condição não atualizam atributos, existindo apenas $\texttt{preC}_0$ e $\texttt{onC}_0$. Isso acontece porque as condições são avaliadas apenas em função do sistema, e não dos atributos.

[1]Apesar de autorização e condições se relacionarem a controle de acesso e obrigações a controle de uso, nós evitamos diferenciar os termos *uso* e *acesso* para manter o texto simples.

Tabela 1. Os 16 modelos centrais do UCON$_{\mathrm{ABC}}$

	0 (imutável)	1 (*pre-update*)	2 (*on-update*)	3 (*pos-update*)
preA	•	•	–	•
onA	•	•	•	•
preB	•	•	–	•
onB	•	•	•	•
preC	•	–	–	–
onC	•	–	–	–

3. A gramática proposta

O objetivo da gramática proposta é aproximar-se ao máximo da descrição formal do UCON, sem perder a habilidade de implementá-la em um sistema real. Além disso, a gramática deve ser simples, permitindo uma implementação eficiente (rápida). Gramáticas podem ser descritas usando a notação EBNF (*Extended Backus-Naur Form*), compostas por símbolos terminais e não-terminais. Não nos preocupamos aqui com símbolos não-terminais porque pouco ajudam na compreensão da gramática proposta.

3.1. Critérios adotados

Para que a construção das políticas de controle seja simples e eficiente, a linguagem proposta deve seguir alguns critérios, descritos a seguir:

- A linguagem definida pela gramática não pode ser ambígua e deve expressar claramente os predicados funcionais para autorização, obrigação e condição.
- O acesso deve apenas ser permitido após a avaliação de todas as regras retornar verdadeiro. Cada regra terá a habilidade de completamente negar uma requisição de uso se seu retorno for falso, mas se não houver pelo menos uma regra que explicitamente negue o uso, o acesso deve ser permitido. Assim, um conjunto vazio de regras não tem o poder de negar um uso (a regra padrão é permiti-lo). Este posicionamento também foi utilizado em [Woo and Lam 1992].
- Deve ser possível adicionar novas regras de controle sem mudar os atributos de usuários, quando os atributos necessários já estiverem presentes.
- O processo de avaliação deve ser eficiente, pois o tempo utilizado para tomar uma decisão não deve prejudicar a interação do sistema com o usuário (*i.e.* o tempo de tomada de decisão tem que ser semelhante ao tempo dos mecanismos de controle de acesso atuais).
- O analisador léxico deve ser implementável e ter uma fácil integração com as aplicações existentes (como sistemas operacionais).

3.2. Representação de atributos, obrigações e condições

É importante notar que o avaliador das regras da gramática deve ser incorporado em um sistema real, então deve ser eficiente. Este requisito introduz algumas limitações como controles para garantir a integridade das bases de políticas, por exemplo. Outras limitações são inerentes à tradução do modelo formal para uma implementação real. A forma como atributos, obrigações e condições são representados reflete estas limitações: na linguagem, atributos são implementados como variáveis inteiras ou *strings*:

variáveis inteiras são utilizadas para guardar valores numéricos, como hora do dia, utilização de disco, etc.

Tabela 2. Símbolos de variáveis e valores

Símbolo	Tipo	Descrição
$*nome*	inteiro *string*	representa uma variável.
dígitos	inteiro	Um valor inteiro constante.
string_1 $\cdots$ *string_n*	*string*	Um conjunto constante de *strings*.
o$slot	inteiro	Palavra-chave para acessar o valor de um *slot* (obrigação).
c$*nome*	inteiro	Representa uma condição (informação externa).

variáveis *string* são utilizadas para representar conjuntos de palavras, como os grupos ou papéis relacionados a um usuário.

As variáveis são representadas pelo símbolo `$` seguido de um nome (*e.g.* `$nome`). Quando uma variável é criada, um valor deve ser-lhe atribuído, e o tipo deste valor define o tipo da variável. Por exemplo, se uma variável recebe um valor inteiro, ela será sempre tratada como inteiro. Obrigações e condições têm necessidades especiais para os dados que serão armazenados nas variáveis que as representam. Como obrigações e condições tratam informações externas ao sistema, há uma questão de segurança a ser considerada, para que seja evitado executar código fornecido pelo usuário durante suas avaliações.

A linguagem permite associar operadores em seqüência. Por exemplo, é possível testar se o valor de uma variável é maior que a diferença de duas variáveis: `$credit > $cost - $discount`.

Obrigações são apresentadas como *slots*, repositórios de dados que podem ser preenchidos por um programa externo e lidos pelo avaliador. Estes *slots* são acessados utilizando a palavra-chave `o$slot` seguida de um número (*e.g.* `o$slot 37`). Similarmente, uma palavra-chave é utilizada para acessar informações sobre condições. Apenas condições pré-programadas estão disponíveis, como hora do dia (`c$time`), quantidade de CPU utilizada (`c$cpu_used`), quantidade de memória livre (`c$free_mem`) e espaço livre em uma partição do disco (`c$free_disk`). Esta condições específicas foram selecionadas para serem implementadas porque se relacionam fortemente como nosso trabalho atual, a implementação do *enforcer* em um sistema operacional para o controle de uso dos recursos locais.

3.3. Símbolos terminais

Os símbolos terminais da gramática consistem em nomes de variáveis, palavras-chave, valores de constantes e operadores. Como foi mostrado no início desta seção, a representação de alguns aspectos do modelo foi tratada com limitações. As seguintes tabelas apresentam os símbolos terminais: a tabela 2 mostra os símbolos relacionados às variáveis e aos valores, enquanto a tabela 3 introduz os operadores e suas funcionalidades.

4. Utilização da gramática

Políticas são expressas por combinações de atributos, constantes de inteiros e *strings*, obrigações, condições e os operadores que os relacionam. Cada usuário possui apenas um arquivo contendo seus atributos, não tendo políticas associadas a si. Por outro lado, cada objeto é associado a um arquivo de atributos e três arquivos de políticas, conforme indicado na figura 2.

Tabela 3. Símbolos dos operadores

Operador	Tipo	Funcionalidade
`=`	Atribuição	Atribui o valor à direita para a variável à esquerda.
`== != < > <= >=`	Comparação	Realiza comparações entre operandos.
`& |`	Lógico	Operadores lógicos AND e OR.
`size`	Conjunto	Retorna o número de elementos em um conjunto.
`+ - * /`	Aritmético	Operações aritméticas básicas entre operandos numéricos.
`+ *`	Conjuntos	União e interseção entre operandos conjuntos.
`(`*expressão*`)`	Precedência	A expressão interna deve ser avaliada antes da parte externa.
`#`	-	Inicia um comentário (até o final da linha).

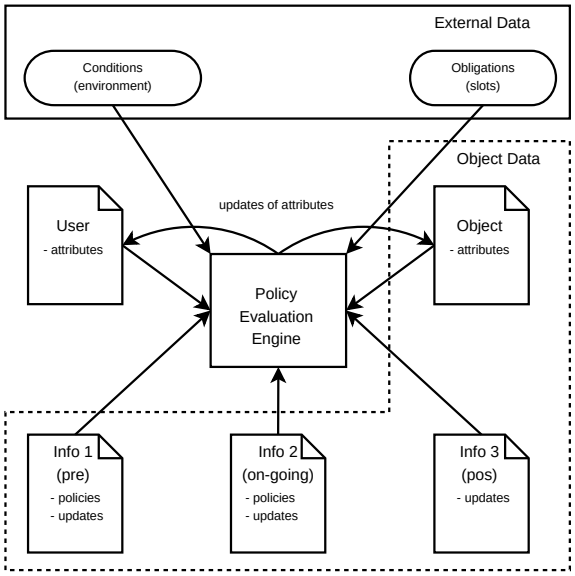

Figura 2. Arquivos utilizados pelo *engine*

- O arquivo de atributos contém a inicialização dos atributos e deve ser processado antes dos outros arquivos.
- O primeiro arquivo de políticas contém as políticas *pre* e as atualizações *pre*.
- O segundo arquivo contém as políticas e atualizações *on*.
- Similarmente, o terceiro arquivo contém as atualizações *pos*.

Uma política de autorização trivial seria: `$usr_id == $owner_id`. Essa declaração deve ser colocada dentro do primeiro arquivo do objeto (*pre*), e define que o acesso será permitido se a identificação do usuário for igual ao proprietário do objeto.

Cada um dos três arquivos deve ser lido e avaliado seqüencialmente. Se uma regra retornar `falso`, então o usuário terá seu pedido de uso negado e o restante do arquivo não será avaliado. Se o arquivo utilizado for de uma política *pre* ou *on*, e esta política negar o acesso, pelo menos o arquivo contendo as atualizações *pos* é avaliado, garantindo que os atributos serão corretamente atualizados.

A seguir serão descritos alguns exemplos de políticas definidas utilizando a linguagem proposta na gramática. Ao comparar estes exemplos com aqueles encontrados em [Park and Sandhu 2004], percebe-se que a a gramática proposta consegue exprimir as políticas definidas formalmente na proposta original do modelo UCON.

4.1. DAC com ACL usando $\mathbf{UCON_{preA_0}}$

O controle de acesso Discricionário [Lampson 1974] com Listas de Controle de Acesso é um mecanismo de controle de acesso simples de implementar usando esta linguagem. Os atributos do usuário contêm apenas sua identificação: `$usr_id = 5456`. O *enforcer* irá passar o direito requisitado (`$right`) como um inteiro, 0 para leitura e 1 para escrita. O arquivo de atributos do objeto deve ser:

```
$obj_perm_read  = 1549 4334 5456    # usuários com permissão de leitura
$obj_perm_write = 4456 5456 7896    # usuários com permissão de escrita
```

O arquivo com as políticas *pre* contém apenas as linhas a seguir. Elas definem que, se um usuário está requisitando uma permissão de leitura ou escrita, sua identificação deve estar na lista `$obj_perm_*` respectiva.

```
( $right == 0 & size ($usr_id * $obj_perm_read)  != 0 ) |
( $right == 1 & size ($usr_id * $obj_perm_write) != 0 )
```

4.2. Políticas MAC usando $\mathbf{UCON_{preA_0}}$

Um controle de acesso mandatório como [Bell and LaPadula 1976] pode ser implementado tratando-se o nível de acesso como um atributo do usuário e a classificação como um atributo do objeto. O arquivo do usuário contém apenas seu nível de acesso (`$clearance`). O arquivo de atributos do objeto contém sua classificação (`$classif`). No acesso de leitura, o nível de acesso deve ser maior ou igual à classificação; no acesso em escrita, o nível de acesso deve ser menor ou igual à classificação. O conteúdo do arquivo *pre* define essa política:

```
( $right == 0 & ($clearance >= $classif) ) |
( $right == 1 & ($clearance <= $classif) )
```

4.3. Controle de uso com obrigação usando $\mathbf{UCON_{onB_0}}$

O usuário deve manter uma janela com publicidade aberta enquanto algum direito sobre um dado objeto é exercido. Um programa externo, possivelmente aquele que controla a janela, irá atualizar um *slot* de obrigações indexado pela identificação do usuário. O arquivo contendo os atributos do usuário possui apenas sua identificação (`$usr_id`). O objeto possui apenas um arquivo, contendo as políticas *on*, que será analisado sempre que uma *syscall* é invocada para exercer algum direito sobre o objeto:

```
o$slot $usr_id == 1
```

O *slot* indexado pela identificação do usuário é criado pelo sistema quando o usuário requisita um direito sobre o objeto. O programa externo que controla a janela de publicidade escreve 1 no *slot* logo que a janela é aberta, e escreve 0 quando for fechada.

4.4. Limitação de acessos usando $\mathbf{UCON_{preA_{13}preC_0}}$

Um dado objeto pode ser acessado por 10 usuários simultaneamente entre 8 e 18 horas e por 20 usuários após às 18 horas. Usuários que já estão acessando o objeto não terão seus acessos revogados quando o horário mudar para 8 horas, mas nenhum novo usuário é admitido até haver disponibilidade. O arquivo de atributos do objeto contém o número de acessos simultâneos aceito e o horário de início e fim do período de redução da quantidade de usuários:

```
$users     = 0
$max_day   = 10
$max_night = 20
$day_start = 8
$day_end   = 18
```

O arquivo com as políticas *pre* controla o número de usuários simultâneos e atualiza a variável controlando este número:

```
( ( (c$time > $day_start) & (c$time < $day_end) ) &
    ($users < $max_day) ) |
( ( (c$time < $day_start) | (c$time > $day_end) ) &
    ($users < $max_night) )
$users = $users + 1
```

Nota-se que na última linha, se o usuário tem seu acesso permitido, então o número de usuários é atualizado. Se a primeira regra falha, a análise do arquivo é interrompida e nenhuma atualização é executada. Deve existir também um arquivo com as políticas *pos*, para decrementar o número de usuários quando algum deles parar de acessar o objeto:

```
$users = $users - 1
```

4.5. Controle baseado em papéis

Escolheu-se apresentar um exemplo de implementação de Controle de Acesso Baseado em Papéis [Sandhu et al. 2000], porque acredita-se que RBAC não apenas contém muito do que é esperado de controles de Autorização, mas também por sua crescente importância como tecnologia de controle de acesso. Aqui será apresentado somente um exemplo envolvendo RBAC plano (*flat* ou nível 1), mas a gramática também permite a representação de políticas RBAC hierárquicas e restritas, que não são exemplificadas aqui por restrições de espaço.

Três dos requisitos do RBAC plano são parte dos atributos do usuário:

```
$roles = director manager teller
$active_roles = manager teller
```

Estas duas variáveis fornecem uma atribuição usuário-papel muitos-para-muitos (`$roles` é um grupo de papéis, e cada papel pode ser atribuído a vários usuários), o suporte à revisão da relação usuário-papel (obtido por consulta ao conteúdo de `$role`) e um usuário pode ativar vários papéis simultaneamente (`$active_roles` representa um grupo de papéis). O controle de acesso é implementado como uma combinação dos atributos do objeto e das políticas. Um único atributo do objeto é necessário:

```
$required_roles = teller manager
```

As políticas *pre* devem cuidar para que pelo menos um dos papéis necessários esteja presente na lista de papéis do usuário, e atualizar os papéis ativos para adicionar este papel:

```
size ($required_roles * $roles) != 0
$active_role = $active_role + ($required_roles * $roles)
```

5. Implementação da gramática proposta

A solução proposta foi implementada em um protótipo, dividido em um *enforcer* e um avaliador de políticas (*engine*). A avaliação de uma política consiste em atribuir um valor semântico às regras que associam requisitos a objetos e decidir se os atributos do sujeito cobrem os requisitos do objeto. Em caso positivo, o pedido de uso é permitido, do contrário é negado. Se qualquer regra falha, o *engine* retorna `falso` ao *enforcer*, que deve tomar as ações para revogar permissões ativas (se houver alguma) e evitar o acesso ou uso deste objeto por este sujeito.

O modelo de controle de uso $\mathrm{UCON}_{\mathrm{ABC}}$ é genérico e pode ser aplicado a diversos contextos. A título de exemplo, para a construção do protótipo, foi escolhido como contexto o controle de uso de arquivos de um sistema operacional de código aberto. Assim, o monitor de referência de controle de uso deve se interpor às chamadas de sistema que permitem o acesso a arquivos.

5.1. Arquitetura do protótipo

O *enforcer* é o ponto de entrada do sistema. Toda chamada de sistema relevante o chamará, e o *engine* será consultado, junto com o direito requisitado, para a tomada de decisão. O *engine* irá avaliar cada regra e o direito requisitado para decidir quanto ao pedido de uso. Quando o *enforcer* receber a decisão do *engine*, será tomada a ação necessária para pemitir ou negar o uso. A figura 3 ilustra este processo.

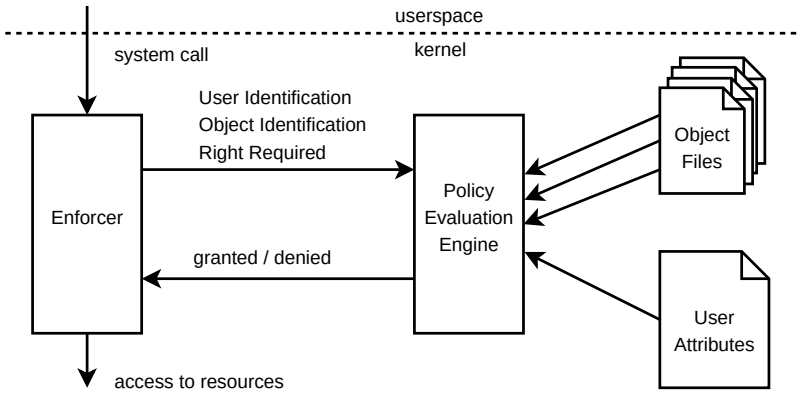

Figura 3. Interação *Enforcer/Engine*

O protótipo foi implementado utilizando a linguagem de programação C e o gerador de analisadores sintáticos *Bison*, para a definição de uma gramática LALR(1); o *enforcer* foi implementado através da interceptação das principais chamadas de sistema (*syscalls*) utilizadas em operações em arquivos: `open`, `close`, `read` e `write` [2]. Verificações e atualizações de atributos *pre* foram executadas durante a chamada `open`, e ações *on* foram executadas nas chamadas `read` e `write`. Atualizações *pos* foram realizadas na chamada `close` ou quando uma requisição de uso for negada, durante a verificação das funções anteriores. Esta solução é consistente com o modelo apresentado em [Zhang et al. 2004]. A implementação do *enforcer* ocupou aproximadamente 500 linhas de código em C; o *engine* tem 35 construções e 60 estados.

[2]Cabe ressaltar que chamadas de sistema mais complexas, como `mmap` (para mapear arquivos em memória) não foram consideradas no protótipo.

O sistema operacional escolhido para servir de base de implementação foi o U*NIX OpenBSD *4.0-current (Oct 4)*, por possuir um núcleo relativamente pequeno e código-fonte bem comentado, além de ter um ótimo histórico de segurança. O ponto de entrada do sistema foi posicionado na função `syscall` do núcleo do OpenBSD, responsável pelo tratamento das *system calls*, antes que a chamada de sistema acionada seja executada. Neste ponto, faz-se a avaliação das políticas de uso e, caso qualquer política falhe, o tratamento de erro da chamada de sistema é executado, retornando o erro `EACCES` (Erro de Acesso).

A função `gucon_evaluate`, chamada por `syscall`, recebe como parâmetro a estrutura da chamada de sistema e o usuário que está executando o processo. Esta função encontra o *inode* e o *device number* vinculados ao arquivo em que a *syscall* deverá ser aplicada e procura pela política adequada no diretório do objeto: `/var/gucon/obj/`*devnumber*`/`*inode*. Os atributos do objeto se encontram no mesmo diretório que as políticas; os atributos do usuário podem ser encontrados em `/var/gucon/usr/`*usuario*. O diretório do objeto contém também os arquivos que servem como *slot* de usuários, utilizados nas obrigações.

Caso o sistema não encontre políticas associadas a um dado arquivo, o sistema implementado recorre aos mecanismos nativos do sistema de arquivos, ao invés de simplesmente retornar um erro. Assim, quando não é possível avaliar uma política UCON, o núcleo utiliza o mecanismo DAC existente. Essa estratégia evita ter de associar políticas UCON a todos os arquivos do sistema.

5.2. Experimentos realizados

Para testar o funcionamento do sistema, foi definido um cenário de acesso/uso simultâneo sobre um arquivo de música no formato MP3, com tamanho de 5 MBytes, a ser reproduzido pelo *mpg123*, um aplicativo simples de reprodução de áudio com interface em modo texto. Por questões operacionais, o desenvolvimento e os experimentos foram realizados em um ambiente virtualizado, usando o emulador de hardware *QEmu* como suporte para a execução do sistema operacional. Os atributos definidos para o arquivo MP3 são:

```
$obj_maxusers  = 10     # quantidade máxima de usuários simultâneos
$obj_currusers = 0      # quantidade atual de usuários simultâneos
$obj_maxcpu    = 30     # % máxima de CPU utilizada (Condition)
$obj_slotvalue = 5      # valor arbitrário para Obligation
$obj_groups    = USERS ADMINS # grupo de usuários autorizados
```

Por outro lado, o atributo do usuário é:

```
$usr_group = USERS              # grupo do qual este usuário faz parte
```

As política *pre*, *on* e *pos* foram assim definidas:

```
# pre policy
size ($obj_groups * $usr_group) >= 1      # o grupo do usuário é autorizado?
$obj_currusers <= $obj_maxusers           # o máximo não foi atingido?
$obj_currusers = ($obj_currusers + 1)     # incrementa o número de usuários

# on policy
$obj_maxcpu <= c$cpu_used        # utilização de CPU no sistema é aceitável?
$obj_slotvalue >= o$slot 1       # valor do slot 1 para este usuário é válido?

# pos policy
$obj_currusers = $obj_currusers - 1       # decrementa o número de usuários
```

O valor semântico da variável `$obj_slotvalue` não é definido aqui, pois este valor é irrelevante para o sistema. Apenas o administrador do arquivo pode definir o objetivo deste valor, que poderia ser, por exemplo, o número de minutos que o usuário já gastou em uma janela de outro *software* que atualiza o valor do *slot* (representando uma obrigação).

Inicialmente foram criadas situações especiais para verificar se as políticas estão realmente sendo respeitadas. A primeira modificação foi a do valor do *slot*, envolvendo a política *on*. Sempre que o *slot* era modificado para ser maior que o valor armazenado em `$obj_slotvalue`, esta política falhava e o usuário perdia o seu acesso ao arquivo. O *software* que acessava o arquivo MP3 (*mpg123*) tratou normalmente o erro `EACCES`.

Em seguida, foi gerada uma série de acessos concorrentes, variando entre 1 e 15 usuários simultâneos. A cada novo usuário que abria ou fechava o arquivo, as políticas *pre* e *pos* corretamente modificavam os atributos do objeto e, no caso da política *pre*, a permissão de uso do objeto era bloqueada quando o número máximo de usuários (`$obj_maxusers`) era atingido.

Para analisar o desempenho do sistema, verificou-se quantas vezes as chamadas de sistema relacionadas a arquivos foram invocadas a cada reprodução do arquivo MP3. Enquanto `open` e `close` foram invocadas 4 vezes cada, a chamada `read` foi invocada 18922 vezes. Assim, neste cenário, apenas a avaliação da política *on* tem impacto efetivo no desempenho do sistema, pois apenas esta é continuamente avaliada.

A figura 4 apresenta o tempo gasto pelo núcleo (*systime*) para uma reprodução do arquivo, em função do número de regras na política *on*. Pode-se observar que, a cada nova regra adicionada a essa política, o tempo gasto pelo núcleo aumenta em cerca de 4 segundos. Em outras palavras, cada nova regra na política *on* aumenta em cerca de 200 μs o tempo de cada execução da chamada de sistema `read`. Os tempos apresentados são valores médios obtidos em 100 execuções.

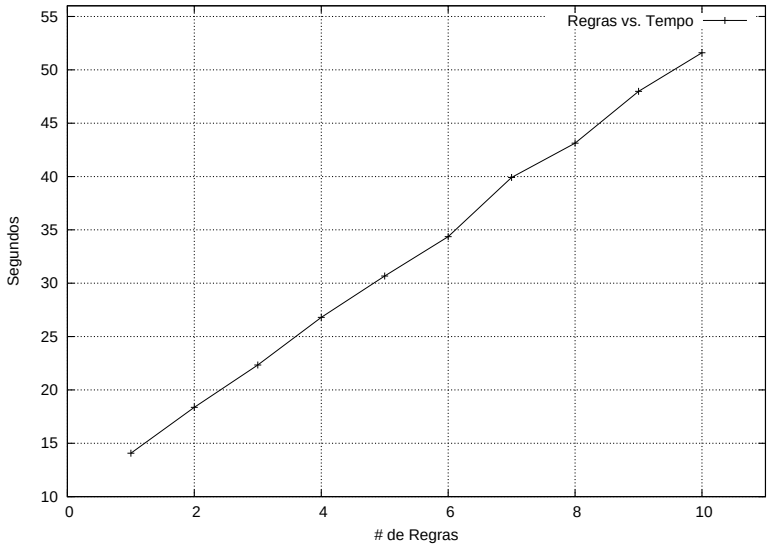

Figura 4. Relação Regras vs. Tempo para 18922 chamadas

5.3. Discussão dos resultados

Os testes realizados mostram que a gramática proposta e sua implementação podem ser utilizadas no contexto de um sistema operacional real para prover os recursos oferecidos pelo UCON_{ABC}, mesmo com um impacto significativo no desempenho do sistema. Este impacto poderia ser bastante reduzido caso fossem implementadas técnicas de *cache* de regras, para evitar ter que lê-las constantemente do disco, ou, em escala menor, se fosse criado um formato binário para armazenar as políticas, para reduzir o tamanho do *parser* e aumentar sua eficiência. Todavia, essas otimizações estão além do escopo atual deste trabalho.

No entanto, mesmo considerando o impacto no desempenho, fica claro que os conceitos utilizados para a representação de Atributos, Obrigações e Condições satisfazem a descrição do modelo UCON_{ABC}, em especial, o conceito de *slots* para representar Obrigações.

Outra constatação que consideramos significativa foi perceber, nos diversos testes realizados com outros aplicativos, como editores de texto, e mesmo com o decodificador de MP3 utilizado nos experimentos, que a grande maioria dos programas trata de forma adequada o erro `EACCES` retornado pela função `syscall` quando de uma negação de acesso/uso por não-cumprimento de alguma política. Esse comportamento facilita a implantação de modelos e políticas mais complexos (como os apresentados neste artigo).

6. Trabalhos relacionados

Há várias linguagens propostas para solucionar diferentes necessidades em segurança e privacidade. O *Enterprise Privacy Authorization Language* (EPAL) [May 2004], por exemplo, tenta unificar as regras que controlam como informações sigilosas são tratadas em diferentes sistemas, através da criação de um mecanismo universal para a descrição das políticas de privacidade.

Em *Modular Authorization and Administration* [Wedde and Lischka 2004], os autores propõem uma linguagem modular de autorização para suportar autorização distribuída entre grupos administrativos cooperantes, baseado principalmente no RBAC. Woo e Lam [Woo and Lam 1992] também tentam resolver este problema de autorização distribuída ao introduzir uma linguagem para codificar requisitos de autenticação, que eles chamam de "bases de políticas". O artigo [Woo and Lam 1998] avança nesse conceito, ao apresentar a sintaxe formal e a semântica para uma linguagem chamada *Generalized Access Control List* (GACL) para representar políticas de autorização baseadas em lista de controle de acesso (*Access Control List* - ACL).

Como a GACL está limitada aos mecanismos baseados em ACL, Ryutov e Neuman [Ryutov and Neuman 2000] apresentam uma linguagem de política que permite representar diversos modelos de controle (como ACL, *capabilities*, baseado em *lattice* e RBAC) e uma linguagem genérica de escrita de políticas de autorização e controle de acesso (GAA API) para facilitar a integração de autenticação e autorização.

Existem, também, duas implementações de uso específico do UCON_{ABC}. A primeira aplica o modelo em um sistema de integração de negócios (B2B) [Camy et al. 2005]; a segunda estende o CORBASec para integrar algumas das funcionalidades do UCON_{ABC} [Higashiyama et al. 2005]. Além destes, três outros trabalhos

descrevem arquiteturas usando o UCON para controle de dados: [Pretschner et al. 2006], os autores descrevem a utilização do modelo UCON, sobretudo o conceito de obrigações, para o controle de privacidade e proteção de direitos de propriedade intelectual em um sistema no qual os provedores de informação (bancos de dados) estão distribuídos com relação aos clientes (consumidores da informação); já os autores de [Syalim et al. 2006] utilizam o UCON para definir um modelo visando garantir a confidencialidade de dados acessados em um Provedor de Serviço de Banco de Dados (DSP); em [Kim and Thuraisingham 2006] foram utilizados os modelos RBAC e UCON para prover controle de acesso/uso em tempo real para um ambiente de objetos distribuídos.

Assim como o modelo $\mathrm{UCON}_{\mathrm{ABC}}$ se diferencia dos modelos tradicionais de controle de acesso ao contemplar as necessidades atuais de controle de uso continuado e de privacidade, como a utilização de um recurso baseada em crédito do usuário ou gerência de direitos digitais (DRM), a linguagem representada pela gramática aqui apresentada difere das linguagens citadas acima ao possibilitar expressar uma abordagem de controle mais abstrata.

7. Conclusão e trabalhos futuros

Cada um dos cinco requisitos descritos na seção 3.1 foram tratados no nosso protótipo:

- A ambigüidade foi evitada ao definir uma gramática LALR(1) e a eficiência na avaliação das regras é obtida ao utilizar um número reduzido de construções na gramática.
- Ao separar as regras e os atributos em arquivos diferentes, é fácil adicionar uma nova política simplesmente editando um arquivo. Não é necessário modificar os atributos do usuário para criar uma nova regra, quando os atributos a serem utilizados já estão presentes.
- Como a semântica associada a cada construção da gramática retorna imediatamente um valor falso sempre que uma regra falha, qualquer regra tem a capacidade de totalmente negar um pedido de uso, independentemente do valor semântico associado às outras regras.

Excluindo-se as limitações relacionadas à representação de obrigações e condições, não é difícil implementar o poder do modelo $\mathrm{UCON}_{\mathrm{ABC}}$ em um sistema real. Há outros exemplos [Camy et al. 2005, Higashiyama et al. 2005] de implementação do $\mathrm{UCON}_{\mathrm{ABC}}$, mas eles não são tão completos quanto o aqui apresentado. Acreditamos que a existência de uma gramática, como a apresentada neste artigo, e um avaliador eficiente poderão ajudar a adoção do modelo $\mathrm{UCON}_{\mathrm{ABC}}$ como uma solução poderosa para controle de acesso, DRM e proteção de dados.

A gramática proposta foi integrada a um sistema operacional de código aberto. Ainda existem alguns problemas que devem ser resolvidos, como a criação e gerência dos *slots* de obrigação, que foram tratados de forma superficial em nosso protótipo, mas que são essenciais para o funcionamento correto em um sistema real. Outros aspectos a aprofundar incluem a realização de experimentos em plataformas e cenários reais, a pré-compilação das políticas para tornar mais eficiente sua avaliação e a agregação de novos operadores à linguagem, para aumentar sua capacidade de expressão. Atualmente trabalhamos no controle de uso de outros recursos do sistema operacional, como conexões de rede, tempo de processador e memória RAM.

Referências

Bell, D. and LaPadula, L. (1976). Secure computer systems: Unified exposition and Multics interpretation. Technical report, MITRE Corporation, Massachusetts, USA.

Camy, A., Westphall, C., and Righi, R. (2005). Aplicação do modelo $UCON_{ABC}$ em sistemas de comércio eletrônico B2B. In 5^{th} *Brazilian Symposium on Information Security and Computing Systems (SBSeg)*. in Portuguese.

Higashiyama, M., Lung, L., Obelheiro, R., and Fraga, J. (2005). JaCoWeb-ABC: Integração do modelo de controle de acesso $UCON_{ABC}$ no CORBASec. In 5^{th} *Brazilian Symposium on Information Security and Computing Systems (SBSeg)*.

Kim, J. and Thuraisingham, B. (2006). Dependable and secure TMO scheme. In *IEEE Intl Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pages 133–140.

Lampson, B. W. (1974). Protection. *SIGOPS Operating System Review*, 8(1):18–24.

May, M. J. (2004). Privacy system encoded using EPAL 1.2. Technical report, University of Pennsylvania.

Park, J. and Sandhu, R. (2003). Usage control: A vision for next generation access control. In Gorodetsky, V., Popyack, L. J., and Skormin, V. A., editors, *Intl Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security*, volume 2776 of *LNCS*, pages 17–31. Springer.

Park, J. and Sandhu, R. (2004). The $UCON_{ABC}$ usage control model. *ACM Transactions on Information and System Security*, 7(1):128–174.

Pretschner, A., Hilty, M., and Basin, D. (2006). Distributed usage control. *Communications of the ACM*, 49(9):39–44.

Ryutov, T. and Neuman, C. (2000). Representation and evaluation of security policies for distributed system services. In *DARPA Information Survivability Conference Exposition*, Healton Head, South Carolina.

Sandhu, R., Ferraiolo, D., and Kuhn, R. (2000). The NIST model for role-based access control: towards a unified standard. In *ACM workshop on Role-based access control*, pages 47–63.

Syalim, A., Tabata, T., and Sakurai, K. (2006). Usage control model and architecture for data confidentiality in a database service provider. *IPSJ Digital Courier*, 2:621–626.

Wedde, H. F. and Lischka, M. (2004). Modular authorization and administration. *ACM Transactions on Information and System Security*, 7(3):363–391.

Woo, T. Y. C. and Lam, S. S. (1992). Authorization in distributed systems: a formal approach. In *IEEE Symposium on Research in Security and Privacy*, pages 33–51.

Woo, T. Y. C. and Lam, S. S. (1998). Designing a distributed authorization service. In *IEEE INFOCOM*, pages 419–429.

Zhang, X., Park, J., Parisi-Presicce, F., and Sandhu, R. (2004). A logical specification for usage control. In *ACM symposium on Access control models and technologies*.