Políticas de Segurança Obrigatórias: Bell e Lapadula no CORBAsec

Carla Merkle Westphall[1], Joni da S. Fraga[2], Carlos B. Westphall[1] e Silvia C. S. Bianchi[1]

Universidade Federal de Santa Catarina - UFSC

[1]INE - LRG (Laboratório de Redes e Gerência)
[2]DAS - LCMI (Laboratório de Controle e Microinformática)
e-mails: [1] {carla, westphal, silvia}@lrg.ufsc.br, [2] fraga@lcmi.ufsc.br
Campus Universitário - Trindade - Florianópolis - SC - Brasil
C.P. 476 - CEP 88040-900

Abstract – This paper proposes extending the CORBA security model to make possible the use of mandatory policies in distributed applications. The *Bell & Lapadula* model is adopted to define the mandatory controls in the authorization scheme *JaCoWeb*, through a policy service designated as *PoliCap*. Our mandatory control is carried out on the level of ORB, on the client side, preventing, in unauthorized accesses, the emission of the corresponding requisition, the associated processing on the server and also, the generation of new requests through this unauthorized processing. Practical experiments and related work are also presented.
Key words – Security, Mandatory policies, CORBAsec.

Resumo – Este artigo propõe a extensão do modelo de segurança do CORBA para possibilitar o uso de políticas obrigatórias em ambientes de aplicações distribuídas. O modelo *Bell e Lapadula* é adotado para definir os controles obrigatórios no esquema de autorização *JaCoWeb*, a partir de um serviço de política designado *PoliCap*. Nosso controle obrigatório é efetuado em nível de ORB, no lado do cliente, prevenindo em acessos não autorizados: a emissão da requisição correspondente, o processamento associado no servidor e ainda, a geração de novas requisições a partir deste processamento não autorizado. As experimentações práticas e a confrontação com a literatura correspondente são apresentadas neste texto.
Palavras-chave – Segurança, Políticas Obrigatórias, CORBAsec.

1. Introdução

O aumento da disponibilidade de redes como a Internet nos últimos anos tem sido determinante para que empresas públicas ou privadas tenham seus fluxos de informações trafegando em redes de escala mundial. A descoberta de informações sigilosas muitas vezes pode representar o comprometimento da própria existência destas organizações. Nestes ambientes de larga escala é fundamental o uso de suportes para a programação distribuída que forneçam mecanismos garantindo, além da interoperabilidade entre recursos heterogêneos, a segurança dos fluxos de informações nestas aplicações. O *CORBAsec*, modelo de segurança proposto nas especificações *CORBA* pela OMG (***O**bject **M**anagement Group*) [1], preenche estas necessidades de segurança para informações e aplicações de objetos distribuídos, em ambientes de larga escala, quando implementado e administrado corretamente.

O esquema de autorização *JaCoWeb* (*http://www.lcmi.ufsc.br/jacoweb/ e http://www.lrg.ufsc.br/~carla/secpoli/*), foi desenvolvido dentro desta perspectiva, usando conceitos e interfaces *CORBAsec*. O *JaCoWeb* corresponde a uma estrutura de controle de acesso baseada em dois níveis de controle: um global e outro local, nas estações dos objetos de aplicação. O serviço de política *PoliCap* constitui o primeiro nível de verificação e atua em tempo de ligação entre cliente e servidor. O *PoliCap*, fornece o gerenciamento centralizado de

objetos de política (políticas discricionárias nas primeiras versões [2, 3]), em um domínio de segurança de aplicações de objetos distribuídos. As nossas propostas no *JaCoWeb* suprem as carências identificadas no *CORBAsec* quanto ao gerenciamento de objetos de política [4]. Um segundo nível de controle de acesso baseado em mecanismo de *capabilities* é realizado em tempo de execução da aplicação, refletindo as políticas do *PoliCap* e completando, portanto, as verificações em ambos os lados - no cliente e no servidor - nas invocações de método.

Este artigo descreve nossa experiência na extensão do esquema de autorização *JaCoWeb* para implementar controles de políticas obrigatórias. Usamos como base na construção de políticas obrigatórias o modelo *Bell e Lapadula* (*BLP*) [7, 8]. O *BLP* é um modelo tradicional de controles obrigatórios em sistemas informáticos que — muito embora exista um grande número de críticas em relação ao mesmo [6, 9, 10, 11, 12] — ainda atualmente motiva um grande número de trabalhos na literatura, no sentido de diminuir suas limitações [13, 14, 15]. As extensões do *JaCoWeb* que definem a implementação deste modelo, além de permitir a necessária síntese das características discricionárias e obrigatórias em ambientes de aplicações de larga escala, usam abstrações do *CORBAsec*, embora estas especificações não definam interfaces para o uso de políticas obrigatórias.

Na seção 2 deste artigo, é descrito o modelo CORBA de segurança. A seção 3 apresenta o modelo Bell e Lapadula, usado para incorporar funcionalidades obrigatórias no CORBAsec. Na seção 4 o esquema de autorização *JaCoWeb-Obrigatório* é apresentado, definindo as entidades que compõem o modelo, as regras do esquema de autorização, como os rótulos de segurança evoluem no sistema e de que forma os objetos da política obrigatória são definidos e utilizados no ambiente distribuído. Resultados de implementação são mostrados na seção 5 e alguns trabalhos correlatos são descritos na seção 6.

2. Modelo CORBA de Segurança

O modelo CORBA de segurança é uma especificação aberta para segurança em sistemas de objetos distribuídos [1, 2, 3]. O modelo CORBA de segurança relaciona objetos e componentes de quatro níveis de um sistema (figura 1): o *nível de aplicação*; o *nível de middleware*, que é formado pelos objetos de serviço *COSS* (***C****ommon* ***O****bject* ***S****ervices* ***S****pecification*), pelos serviços ORB e pelo núcleo do ORB. Os serviços ORB e os objetos de serviço *COSS* implementam a segurança de objetos distribuídos no nível de *middlewar*e. O *nível de tecnologia de segurança* corresponde aos serviços subjacentes de segurança. O *nível de proteção básica* compreende o sistema operacional e o *hardware*.

As políticas de segurança no CORBAsec são descritas na forma de *atributos de segurança* dos objetos servidores do sistema (*atributos de controle*) e dos clientes (*atributos de privilégio*). As políticas de autorização são representadas no objeto *DomainAccessPolicy*; um exemplo é mostrado na tabela 1. Este objeto contém as permissões concedidas aos principais para a invocação de operações no sistema *CORBA* seguro, com base nos seus atributos de privilégio. Apenas os direitos *g* (*get*), *s* (*set*), *m* (*manage*) e *u* (*use*) são previstos na especificação *CORBAsec*, embora seja possível definir outras famílias e tipos de direitos mais adaptados à aplicação em mãos. Para simplificar a administração, o *DomainAccessPolicy* agrega principais usando seus atributos de privilégio. Alguns tipos de agrupamento são *group* (grupo) e *role* (papéis desempenhados pelos principais no sistema).

O objeto *RequiredRights* define que para a invocação de cada operação na interface de um objeto seguro, alguns direitos são necessários ou requeridos (*atributos de controle*). Conforme pode ser visto no exemplo da tabela 2, os direitos requeridos são atribuídos para *interfaces* (uma classe no modelo CORBA) e não para *instâncias* (objetos), ou seja, todas as instâncias de uma interface terão sempre o mesmo conjunto de direitos requeridos.

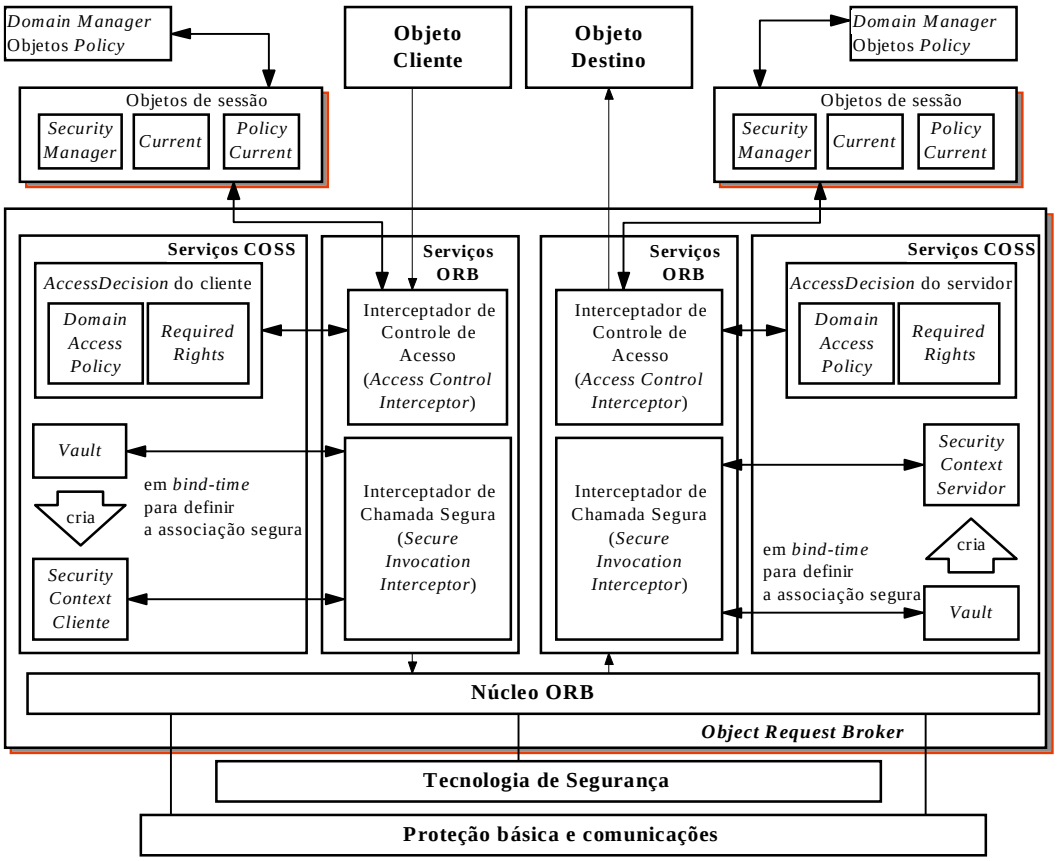

Figura 1. Modelo CORBA de Segurança.

Atributo de Privilégio	Direitos Fornecidos (*Granted Rights*)
role:gerente_banco	corba: gs--
role:gerente_banco	corba: g---
role:caixa	corba: g--u

Tabela 1. Objeto *DomainAccessPolicy*.

Direitos Requeridos (*Required Rights*)	Operação	Interface
Corba:g	Ver_Saldo	Renda_Fixa
Corba:gs	Depositar	Renda_Fixa
Corba:g--u	Depositar	Conta_corrente

Tabela 2. Objeto *RequiredRights* para as interfaces Renda_Fixa e Conta_corrente.

O objeto *AccessDecision* é responsável por decidir se a invocação de uma operação de um determinado servidor deve ser permitida ou não. Esta decisão de acesso depende dos atributos de privilégio e dos atributos de controle. A lógica dessa decisão de acesso é deixada em aberto pela especificação CORBAsec, e é dependente do contexto do sistema e do tipo de política utilizada (discricionária, obrigatória, baseada em papéis, ou outra). Por exemplo, a política discricionária definida na tabela 1 fornece a um principal *caixa*, os direitos requeridos - *g* e *u* - para executar a operação *Depositar* da interface Conta_corrente.

Os *objetos de sessão* - *SecurityManager*, *PolicyCurrent* e *Current* - armazenam informações sobre o contexto corrente de segurança como referências aos objetos *RequiredRights* e *AccessDecision* (*SecurityManager*), referências também de objetos de política usados no estabelecimento de associações seguras (*PolicyCurrent*) e ainda, as credenciais do principal obtidas na autenticação da sessão (*Current*). Na figura 1, os objetos *Vault* e *SecurityContext*, por sua vez, são participantes no estabelecimento de *associações seguras*, que garantem confidencialidade e/ou integridade às mensagens trocadas entre cliente

e servidor.

A especificação CORBAsec define dois tipos de interceptadores[1] que atuam durante uma invocação de método. O primeiro é o *interceptador de controle de acesso*, que atua em nível mais alto, desviando a invocação para efetivação do controle de acesso, e o segundo é o *interceptador de invocação segura*, que atua em nível mais baixo, fornecendo integridade e confidencialidade às mensagens trocadas entre cliente e servidor. Estes interceptadores são criados durante o *binding* entre cliente e servidor, e atuam de maneira diferente em momentos distintos de uma invocação de método. No *binding*, o interceptador de controle de acesso é responsável por instanciar o objeto *AccessDecision* e atualizar a sua referência no objeto *SecurityManager*. O objeto *AccessDecision*, por sua vez, deve disponibilizar o objeto de política *DomainAccessPolicy* que atuará nas invocações do principal, inserindo sua referência no objeto *PolicyCurrent*, e também localizar o objeto *RequiredRights* da interface servidora desejada, atualizando sua referência no objeto *SecurityManager*. Em tempo de decisão de acesso, o interceptador de controle de acesso invoca a operação *access_allowed* do objeto *AccessDecision*, que é responsável por autorizar ou não a invocação, obtendo os direitos fornecidos (contidos em *DomainAccessPolicy*) e comparando-os com os direitos requeridos (contidos em *RequiredRights*).

3. Modelo de Segurança Bell e LaPadula

O *modelo Bell e LaPadula* acrescenta controles de acesso *obrigatórios* aos controles de acesso discricionários, impedindo o fluxo de informações de níveis de segurança mais altos (mais privilegiados) para os níveis de segurança mais baixos [7, 8, 12, 14, 16]. Um usuário é autorizado a manipular uma informação se possui o direito ao acesso correspondente e se é habilitado ao nível de classificação da informação.

No *BLP* cada sujeito (entidade ativa que executa o acesso) e objeto do sistema deve ter associado um *nível de segurança* que envolve um *nível de sensibilidade* e um *compartimento*. O nível de segurança f_s de um sujeito define uma sensibilidade normalmente identificada como *habilitação*. O nível de sensibilidade em um objeto, por sua vez, é designado como *classificação* no seu nível de segurança f_o. No *BLP*, a *habilitação de um sujeito* e a *classificação de um objeto* assumem os níveis definidos na classificação do Departamento de Defesa americano (*DoD)*: (não classificado, confidencial, secreto ou ultra secreto). O *compartimento* no nível de segurança delimita os espaços em que valem os acessos dos sujeitos sobre os objetos. O nível de segurança de um sujeito (f_s) representa o nível máximo de segurança das informações que o sujeito pode consultar. Na dinâmica do modelo BLP, uma segunda etiqueta é associada a um sujeito: é o nível de *segurança corrente*, notado por f_c. O nível corrente representa o mais alto nível das informações consultadas pelo sujeito no sistema; este último nível flutua, portanto, com a evolução do sistema. O modelo Bell e LaPadula define duas propriedades básicas [7]:

- *Propriedade simples*: conhecida como *propriedade-ss*, que delimita os acessos de leitura de um sujeito só a objetos cujos níveis sejam *dominados* pelo seu nível de segurança, isto é, $f_o \leq f_s$.
- *Propriedade estrela*: também conhecida como *propriedade-**, define que um sujeito pode ler somente objetos *dominados* pelo seu nível corrente de segurança e pode escrever em objetos que *dominem* seu nível corrente de segurança, isto é, f_c (s) ≤ f_o (*o*).

Considerando, por exemplo, que o rótulo corrente de um sujeito f_C = SECRETO e que ele deseja acessar um objeto o_1, que possui f_{O1} = CONFIDENCIAL, para realizar uma cópia

[1] Os serviços ORB são implementados por *interceptadores*, que são objetos logicamente interpostos na seqüência de invocação entre um cliente a um servidor. Cada serviço *COSS* de segurança é associado a um interceptador, que tem a finalidade de desviar a chamada de maneira transparente, ativando assim o objeto de serviço associado.

desse arquivo. A propriedade-* impõe que a classificação da cópia seja superior ou igual a SECRETO, mesmo que as informações ali contidas sejam originalmente de classificação confidencial. Ao longo do tempo no *BLP*, as informações tendem a *subir* no reticulado formado pelos rótulos de segurança, recebendo classificações sucessivamente maiores. Este fenômeno, conhecido como *superclassificação da informação* [16, 17], é resultado das restrições impostas pela propriedade-*, e representa um dos principais problemas do modelo BLP. Algumas técnicas são sugeridas na literatura para tratar ou suavizar a superclassificação das informações neste modelo [7, 10, 14, 17, 18, 19, 20].

3.1. Técnicas para evitar a superclassificação

A noção de *processos confiáveis* (*trusted processes*) foi introduzida como implementação do conceito de sujeitos de confiança [10]. Em [14] são definidos *objetos confiáveis* — identificados como *objetos sem estado* — que, a exemplo dos sujeitos confiáveis sugeridos em [20], têm associados *intervalos de confiança* [L_{min} , L_{max}]. L_{min} representa o nível de segurança *mínimo* dos dados que o objeto sem estado pode *escrever*. L_{max} corresponde ao nível de segurança *máximo* que pode ser *lido* pelo objeto sem estado.

Um objeto *sem estado* é um objeto que não guarda nenhum dado de aplicativo na memória entre duas ativações sucessivas[2] [14]. Seu estado (ou o estado da aplicação) é reiniciado a cada invocação do mesmo. Esta propriedade dos objetos sem estado é muito importante pois significa que duas requisições sucessivas que acessam o mesmo objeto sem estado *não podem concretizar um fluxo de informação* por meio deste objeto.

Processos servidores do sistema em geral são objetos *sem estado*. Um exemplo de objeto sem estado é um servidor de arquivos, que responde às requisições de montagem de sistemas de arquivos sem guardar nenhuma informação sobre um atendimento prévio na execução da próxima requisição. A distinção de objetos com estado e sem estado possibilita obter um esquema de autorização menos restritivo do que o modelo original do *BLP*: alguns sujeitos podem agora ler e escrever em objetos de forma mais flexível do que a propriedade-* tradicional possibilitaria [14].

Em [17], Woodward explica que o problema da superclassificação é ocasionado porque, em implementações tradicionais de sistemas baseados no *BLP* [21], os sujeitos e objetos *herdam o nível de segurança do seu criador*. Levando em consideração que um objeto recém criado não contém dado algum, associar o nível de segurança do seu criador imediatamente não representa o nível de sensibilidade real das informações contidas neste objeto naquele momento. O autor sugere *inicializar* qualquer objeto recém criado com f_O = *NÃO-CLASSIFICADO*. Durante a execução de quaisquer operações subseqüentes, o nível de segurança do objeto deve "flutuar" para representar o nível de sensibilidade das informações que são armazenadas em seu interior.

A modificação de rótulos de segurança feita de forma dinâmica pelo próprio sistema, que evita a superclassificação da informação usando as técnicas mencionadas acima, habilita um sistema a atender uma variedade de requisitos de segurança muito maior do que os modelos onde os rótulos são estáticos [14, 17, 19]. Entretanto, essa modificação deve ser realizada de forma restrita para não provocar o aparecimento de fluxos ilegais de informação.

---

[2] A origem do conceito dos objetos sem estado é fundamentada no conceito de *Object Reuse*, introduzido pelo TCSEC em [22]. *Object reuse* é definido como a reutilização, feita por um sujeito, de algum meio de armazenamento temporário (setor de disco rígido, página de memória, etc), que armazenou um ou mais dados de um objeto de aplicação. Para ser reutilizado de forma segura, nenhum dado residual pode ficar disponível ao novo sujeito, quando do reuso deste objeto (recurso) de armazenamento.

4. A Proposta *JaCoWeb*-Obrigatório

O objetivo da proposta *JaCoWeb*-Obrigatório é estender as interfaces existentes no CORBAsec, usando o modelo Bell e Lapadula, para incorporar políticas obrigatórias, inexistentes no padrão atual [1].

Nosso trabalho, a exemplo de [14], usa os *objetos sem estado* e os intervalos de confiança associados com as requisições para evitar a superclassificação, suavizando o aspecto restritivo da propriedade-*. Usamos também a técnica da sensibilidade inicial, assumindo o valor NÃO-CLASSIFICADO e rótulos flutuantes sugeridos em [17]. Entretanto, nossa proposta de controles obrigatórios está inserida em um ambiente de objetos distribuídos acessados a partir de *browsers* Web. A nossa solução utiliza interceptadores do CORBAsec que permitem uma verificação transparente e também flexível. A *concretização dos controles obrigatórios* é realizada no lado do cliente, antes que a requisição inicie a comunicação com o objeto servidor. Controles no lado do cliente permitem a existência de uma definição clara da regra que trata as mensagens de retorno, questão esta que normalmente é tratada de maneira informal [14, 25]. A implementação desta regra no nosso esquema não provoca bloqueios de retornos de requisições já processadas pelo servidor da aplicação.

4.1 O Serviço de Política *PoliCap*

O *PoliCap* é um serviço de políticas para objetos distribuídos cujas invocações são reguladas segundo o modelo CORBAsec [23, 24]. O *PoliCap* foi desenvolvido no contexto do projeto *JaCoWeb* e centraliza em tempo de ligação (*binding*) a implementação do primeiro nível de controle de acesso do esquema de autorização sobre as invocações de método no sistema de objetos distribuídos.

O *PoliCap* possibilita o gerenciamento centralizado de objetos de política em um domínio de segurança, suprindo a carência na especificação *CORBAsec* quanto ao gerenciamento de objetos de política. Segundo a especificação, as políticas de segurança são disponibilizadas através dos objetos *DomainAccessPolicy* e *RequiredRights*. Aplicações administrativas interagem com o *PoliCap* para gerenciar estes objetos. Por sua vez, aplicações operacionais ou objetos COSS interagem com o serviço de política *PoliCap* para obter, no *binding*, as políticas e os direitos necessários aos controles em tempo de execução sobre invocações em um objeto servidor de aplicação. A idéia é que, no *binding*, o serviço de política forneça versões locais dos objetos *DomainAccessPolicy* e *RequiredRights* que contenham os atributos de privilégio e de controle apropriados a um principal e a uma determinada interface. A decisão de acesso é, então, efetuada com base nestas instâncias locais dos objetos *DomainAccessPolicy* e *RequiredRights*. O *PoliCap* deve também gerenciar a política obrigatória em um contexto *CORBAsec*. Maiores detalhes sobre o *PoliCap* podem ser encontrados em [23, 24].

4.2 *JaCoWeb-Obrigatório*: *Bell e Lapadula* no CORBAsec

A proposta *JaCoWeb-Obrigatório*, seguindo o modelo *BLP*, define: as entidades que compõem o nosso modelo; a forma de atribuir rótulos de segurança a essas entidades; as regras do esquema de autorização *JaCoWeb-Obrigatório*; e ainda, a forma de estabelecer e usar uma política obrigatória através dos serviços *PoliCap*. As entidades que fazem parte do nosso modelo são: *sujeitos*, *objetos* e *requisições* (figura 2). Um sujeito representa uma entidade que provoca execuções de operações sobre os objetos. Os objetos, destino das operações executadas pelos sujeitos, podem possuir ou não um estado associado.

No modelo é atribuído um *rótulo de segurança* a todo sujeito ou objeto do sistema; nestas atribuições são distinguidos objetos com estado e sem estado. Na nossa proposta, as requisições também recebem rótulos de segurança, uma vez que as mesmas transportam informações sensíveis entre sujeitos e objetos. Cada rótulo de segurança da proposta é

materializado nas estruturas do modelo de controle de acesso do *CORBAsec*. A figura 2 ilustra os diferentes rótulos de nossa proposta.

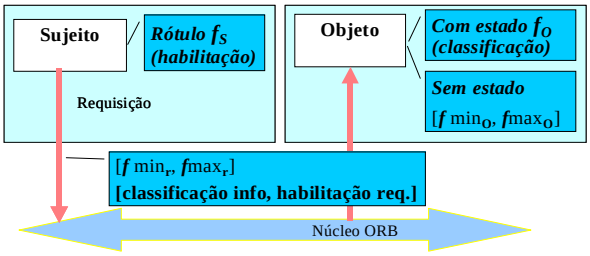

Figura 2. Rótulos no *JaCoWeb-Obrigatório*.

4.2.1 Rótulos de Sujeitos

A cada sujeito é atribuído um único rótulo de segurança f_s. Esse rótulo representa a sua *habilitação* (*clearance*) no sistema (figura 2). Para incorporar a habilitação de sujeitos na estrutura do CORBAsec, é incluída na lista de acesso representada pelo objeto *DomainAccessPolicy* uma coluna de dados chamada *Clearance*. O conteúdo de *Clearance* pode assumir um dos seguintes valores: NÃO-CLASSIFICADO (valor numérico 1), CONFIDENCIAL (valor 2), SECRETO (valor 3), ULTRA-SECRETO (valor 4). Neste sentido, o objeto *DomainAccessPolicy* segue a representação da tabela 3, onde é associada a habilitação (ou nível de segurança) SECRETO ao sujeito*gerente_do_banco*.

Atributo de Privilégio	Granted Rights	*Clearance*
Role: gerente_do_banco	corba: gs--	3

Tabela 3. Objeto *DomainAccessPolicy* com a inclusão da coluna *Clearance*.

4.2.2 Rótulos de Objetos

Rótulos de segurança são atribuídos a objetos *com estado* e objetos *sem estado*. Um *rótulo único* f_0 é associado a um objeto *com estado O*. Esse rótulo corresponde à classificação do objeto que representa o seu nível de segurança fixo que não pode ser modificado durante o tempo de vida do objeto.

Um *intervalo de confiança* [$fmin_0$, $fmax_0$] [3], onde $f\text{min}_0 \leq f\text{max}_0$, é associado com os objetos *sem estado O*. O rótulo $f\text{max}_0$ representa o nível de segurança *máximo* das informações contidas no objeto sem estado *O*. Da mesma forma, o rótulo $f\text{min}_0$ representa o nível de segurança *mínimo* do cliente para escrever neste objeto sem estado (figura 2). Nossa proposta é representar um *intervalo de confiança* usando dois dígitos: o primeiro dígito representa o nível $f\text{min}_0$ e o segundo dígito representa o nível $f\text{max}_0$. Considerando a representação dos níveis de segurança NÃO-CLASSIFICADO, CONFIDENCIAL, SECRETO e ULTRA-SECRETO por valores numéricos (1, 2, 3 e 4, respectivamente), o intervalo [CONFIDENCIAL, SECRETO] será dado então pelo valor 23.

Considerando que as informações sobre os objetos servidores de aplicação no CORBAsec estão contidas no objeto *RequiredRights*, para associar rótulos de segurança a esses objetos, a proposta inclui um novo elemento no objeto *RequiredRights* chamado *Classification* (ver tabela 4). Essa extensão consiste em uma coluna que contêm valores numéricos compostos por três dígitos numéricos: o primeiro dígito representa o *modo de acesso* do método considerado (1 – leitura, 2 – escrita, 3 – leitura/escrita); o segundo e terceiro dígitos representam a *classificação* dos objetos com estado e sem estado. Se o objeto

[3] A definição do intervalo de confiança deve considerar as informações que podem fluir através do objeto sem estado.

é *com estado*, o segundo dígito é sempre zero e o nível de classificação é representado pelo terceiro dígito. Um objeto *sem estado* teria nos valores numéricos de sua classificação definindo o seu intervalo de confiança. Na tabela 4 a operação *Ler_valor_conta* corresponde a um modo de acesso *leitura* (primeiro dígito é igual a 1) e a interface *Conta_bancária* a um objeto com estado de nível de classificação NÃO-CLASSIFICADO (valor 01). Na mesma tabela, *Imprime_saldo* da interface *Impressora_I1* é uma operação de leitura/escrita (valor igual a 3) de um objeto sem estado com intervalo de confiança igual [NÃO-CLASSIFICADO, CONFIDENCIAL] (valor de 12 do segundo e terceiro dígitos).

Required Rights	**Operação**	**Interface**	***Classification***
Corba:g---	Ler_valor_conta	Conta_bancária	101
Corba:gs--	Imprime_saldo	Impressora_I1	312

Tabela 4. Objeto *RequiredRights* com a inclusão da coluna *Classification*.

4.2.3 Rótulos de Requisições

Cada *requisição* no sistema transporta informações que podem ser modificadas pelos diferentes objetos acessados [25]. No nosso modelo, um rótulo de segurança é também atribuído a uma requisição. Para cada requisição é definido um *intervalo de confiança* [$f\min_r$, $f\max_r$] onde o elemento $f\min_r$ representa a classificação da informação contida na requisição (a informação que flui da requisição para um objeto)[4] e $f\max_r$ corresponde à *habilitação* da requisição. O nível de habilitação é inicializado com o nível corrente de segurança do principal que ativa a requisição e representa a classificação máxima das informações que podem ser lidas por esta requisição (as informações que devem fluir de objetos para a requisição). Esses dois elementos compõem então o rótulo de segurança da requisição: *[classificação da informação, habilitação da requisição]***.** A figura 2 ilustra este rótulo.

O conceito de *rótulos flutuantes* [17] é usado em nosso modelo durante a execução de uma requisição. O rótulo de uma requisição, conforme o objeto servidor desta requisição, não necessariamente será o mesmo da resposta ou de uma nova requisição gerada a partir do processamento associado. Suponha um usuário com habilitação SECRETO, interagindo com o objeto com estado *O*1 de classificação CONFIDENCIAL (02) e com o objeto sem estado *O*2 de *intervalo de confiança* [CONFIDENCIAL, SECRETO] (23). O usuário dispara uma requisição r_1, que acessa *O*1 para executar um dos seus métodos. Essa execução pode produzir o envio de uma nova mensagem, a requisição r_2, que é rotulada levando em consideração as informações de r_1 e do *O*1. Seguindo as premissas introduzidas no modelo, o rótulo inicial da requisição r_1 terá o valor 13, definindo a classificação como NÃO-CLASSIFICADO (primeira ativação de requisição), e o valor SECRETO como sua habilitação (espelhando a *clearance* do usuário que invoca a operação no objeto *O*1). O método executado em *O*1 invoca outro método no objeto *O*2. A requisição *r2* que transporta informações sensíveis de *O*1 para o objeto *O*2, é então rotulada como [CONFIDENCIAL, SECRETO], já que a classificação das informações transportadas é a mesma do objeto requisitante.

A codificação de rótulos no modelo considerado usa a estrutura do *Request* CORBA que contém um campo de rótulo de segurança na requisição. Por exemplo, o rótulo de *r1* do exemplo anterior será representado pelo número 13 e o rótulo de r_2 ocupa o campo do *Request* com o valor 23. A estrutura do *Request* CORBA, além de transportar campos adicionais relacionados com as *capabilities*, deve transportar o rótulo de segurança da requisição conforme mostra a Figura 3.

[4]A classificação da primeira ativação de uma requisição criada por um usuário é inicializada com o nível de segurança = NÃO-CLASSIFICADO, já que não carrega nenhum dado sensível, seguindo a sugestão de [17].

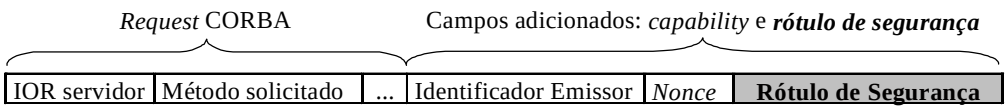

Figura 3. Estrutura do *Request* CORBA com o campo do rótulo de segurança.

4.3 As regras do esquema de autorização obrigatório no projeto *JaCoWeb*

As regras do esquema de autorização definem a evolução do modelo e como são feitos os controles obrigatórios. Para as nossas implementações foram definidas seis regras que determinam a evolução dos rótulos de segurança no ambiente. Na apresentação das regras do esquema de autorização, serão feitas referências seguidas aos modos de acesso de leitura e de escrita. O *acesso de leitura* sobre *O*, um objeto com estado, corresponde a toda execução de método que provoca um fluxo de informação do estado de *O* para a requisição de execução do método. De igual forma, o *acesso de escrita* representa toda execução de um método de *O* que provoca um fluxo de informação da requisição para o estado do objeto. O *acesso de leitura-escrita* define uma troca de informações nos dois sentidos entre a requisição que executa o método e o estado do objeto *O*. Nesta proposta requisições de serviço e respostas do modelo cliente-servidor são representadas por uma única entidade conceitual que chamamos de requisição (ver figura 4).

Também antes da apresentação das regras, temos que fazer algumas considerações sobre as propriedades a serem mantidas nos dois tipos de objetos identificados no nosso modelo: objetos com estado e objetos sem estado.

Objeto com estado

As regras do modelo devem servir para os objetos com estado manterem as propriedades do modelo Bell e Lapadula: a propriedade-ss e a propriedade-*.

– *Propriedade-ss (simples)*: Uma requisição de *leitura r* sobre um objeto com estado *O* é autorizada se e somente se $f_O \leq f\max_r$. A requisição tem autorização para ler um objeto se sua habilitação domina a classificação do objeto.
– *Propriedade-* (estrela)*: Uma requisição de *escrita r* sobre um objeto com estado *O* é autorizada se e somente se $f\min_r \leq f_O$. Como a escrita é um fluxo de informação unicamente da requisição para o objeto, é necessário que classificação da informação carregada pela requisição seja dominada pelo rótulo do objeto, para que a escrita se efetue sem fluxo de informações ilegais.

Para que estas propriedades sejam garantidas, é associado a cada método de um objeto com estado um atributo que indica qual *modo de acesso* é realizado sobre o objeto na execução do método: acesso de *leitura*, de *escrita* ou de *leitura-escrita*. Esse atributo foi incluído no objeto *RequiredRights* do CORBAsec, conforme explicado na seção 4.2.2.

Objeto sem estado

Uma requisição *r* pode acessar um objeto sem estado *O* (invocando um método qualquer de *O*) se e somente se a intersecção de $[f\min_O, f\max_O]$ e $[f\min_r, f\max_r]$ não é vazia. Ou seja, as seguintes condições devem ser verificadas:

- $fmin_O \leq fmax_r$: a *requisição* está habilitada a *receber* informações armazenadas em *O*. É uma releitura da propriedade-ss para objetos sem estado, inviabilizando fluxos ilegais.
- $fmin_r \leq fmax_O$: o *objeto O* está habilitado a *receber* informações da requisição.

4.3.1 A flutuação dos rótulos

Neste subitem, são descritas as seis regras que lidam com o controle de autorização de uma *requisição r* invocando um método no objeto *O*. As quatro primeiras regras descritas abaixo são derivadas das propriedades do modelo Bell e Lapadula original, mas a quarta regra

leva em consideração técnicas para suavizar a superclassificação de informações. A quinta regra está relacionada com os objetos sem estado, e a regra seis trata as mensagens de retorno das invocações.

A evolução dos rótulos adaptada ao contexto CORBAsec considera a requisição *r* representada nas estruturas de um *Request* CORBA. Os rótulos de segurança usados nas verificações estão associados ao sujeito (rótulo presente no objeto *DomainAccessPolicy*), ao objeto (rótulo presente no objeto *RequiredRights*) e à requisição (rótulo presente no *Request* CORBA).

As informações que são geradas a partir do processamento ativado por uma *requisição r*, serão transportadas na *resposta* (retorno ao cliente, figura 4.1) ou em uma *nova requisição* a outro objeto (requisição aninhada, figura 4.2). Tanto a resposta quanto a nova requisição são representadas como uma *requisição r'* no nosso modelo. O rótulo associado a esta *requisição r'* é gerado em função do rótulo da *requisição r* e do *objeto O* invocado.

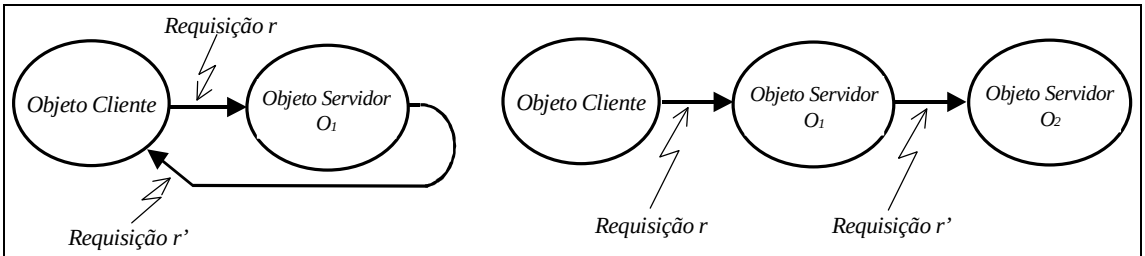

Figura 4.1. Requisição de saída como resposta ao cliente

Figura 4.2. Requisição de saída como invocação de método aninhado

Figura 4. Requisições de entrada e requisições de saída no *JaCoWeb-Obrigatório.*

<u>*Se o objeto O é um objeto com estado:*</u>

- *Regra 1 (R1)* : Se o método *m* corresponde a um acesso de *leitura* e se $f_O \leq f\max_r$, o acesso é *autorizado com restrição*. O $f\min_{r'}$ que representará a execução da requisição será ajustado para o valor $max(f\min_r , f_O)$. Quando $f_O \leq f\min_r$, o acesso *é autorizado sem restrição* porque $max(f\min_r , f_O) = f\min_r$.
- *Regra 2 (R2)* : Se o método *m* corresponde a um acesso de *escrita* e se $f\min_r \leq f_O$, o acesso *é autorizado sem restrição.*

A *regra R1* descreve a evolução das requisições que efetuam um acesso de leitura sobre um objeto com estado. Se uma *requisição r* efetua um acesso de leitura sobre um objeto, cuja classificação domina a classificação da informação veiculada pela *requisição r* (representada pelo elemento inferior do rótulo da *requisição r*), então o elemento inferior do rótulo da *requisição r'* deve ser ajustado/aumentado para a classificação do objeto. Em contrapartida, uma *requisição r* não pode ler um objeto com estado cuja classificação domina a habilitação da *requisição r*. A *regra R2* representa a propriedade estrela do modelo *Bell e Lapadula* quando da escrita em um objeto com estado: O acesso é recusado quando a classificação do objeto é dominada pelo rótulo inferior da *requisição r*, e o acesso é concedido de forma irrestrita quando a classificação do objeto domina o elemento inferior do rótulo da *requisição r*. Em um acesso de escrita, a classificação da informação transportada pela *requisição r'* de saída não varia em relação ao de *r*, pois um acesso de escrita consiste unicamente de um fluxo de informação da *requisição r* para o objeto.

- *Regra 3 (R3)* : Se o método *m* corresponde a um acesso de *leitura-escrita* e se f_O se situa no intervalo [$f\min_r$, $f\max_r$], o acesso *é autorizado com restrição* e $f\min r_{r'}$ deve ser ajustado para o valor max($f\min_r$, f_O). A restrição provém da operação de leitura, a escrita entretanto é autorizada sem restrição. No caso onde $f_O = f\min_r$, o acesso *é autorizado sem restrição.*

- *Regra 4 (R4)* : *Criação de objetos com estado* : Um sujeito tipicamente cria um objeto pela emissão de uma *requisição r* do tipo "Crie instância" para a classe do objeto a ser criado. A fim de evitar o aparecimento de canais cobertos, e suavizar a superclassificação da informação [17], o objeto *"filho"* deve possuir, como rótulo inicial, um valor igual a pelo menos a classificação da requisição *r* que gerou essa criação (elemento inferior do rótulo de *r*). Assim, o objeto com estado é criado com $f_O = f\min_r$.

Se o objeto O é um objeto sem estado:

Os objetos sem estado são criados por aplicações administrativas ou autorizadas pelo administrador de segurança do sistema. É responsabilidade deste administrador a definição dos intervalos de confiança associados aos objetos sem estado. A regra que segue define as condições de autorização para acessos nestes objetos:

- *Regra 5 (R5)* : Se $f\min_O \leq f\max_r$ e se $f\min_r \leq f\max_O$ o acesso *é autorizado com restrição* (intersecção não vazia dos intervalos de confiança). As informações geradas a partir do processamento ativado serão transportadas na *requisição r'* de saída com o rótulo dado por [*max* ($f\min_r$, $f\min_O$) , *min* ($f\max_r$, $f\max_O$)].

As condições da regra 5 implicam que a *requisição r* só será executada se houver intersecção entre os intervalos de confiança de *r* e do objeto sem estado *O*, respectivamente. Na verdade esta regra, descreve a evolução (ou "flutuação") do intervalo de confiança nos fluxos de requisições atravessando objetos sem estado. Se uma requisição *r* de entrada acessa um objeto sem estado, cujo elemento inferior do rótulo domina a classificação da informação contida na *requisição r* (elemento inferior do rótulo da *requisição r*), então o elemento inferior do rótulo da *requisição r'* deve ser elevado, assumindo o valor do elemento inferior do rótulo do objeto. Isso significa de fato que o nível corrente das informações transportadas pela *requisição r'* deve ser ajustado ao nível mínimo do objeto sem estado. Se o objeto sem estado tem o elemento superior do rótulo dominado pela habilitação da *requisição r* (elemento superior do rótulo da *requisição r*), então a habilitação da *requisição r'* deve ser diminuída de acordo com o elemento superior do rótulo do objeto, para permitir o fluxo de informações para a *requisição r'*.

Quando o intervalo [$f\min_r$, $f\max_r$] está incluso no intervalo [$f\min_O$, $f\max_O$], o acesso é autorizado sem restrição de ajuste da *requisição r'*, já que: *max*($f\min_r$, $f\min_O$) = $f\min_r$ e *min*($f\max_r$, $f\max_O$) = $f\max_r$.

Retorno das requisições executadas:

Uma questão bastante importante a ser considerada nos controles obrigatórios diz respeito às mensagens de retorno das invocações [25, 26]. Ou seja, as requisições de saída *r'* quando são *replies* do servidor ao cliente (emissor de *r*, figura 4.1) devem estar sujeitas a controles no sentido de evitar fluxos indevidos. A regra 6 apresenta a nossa abordagem no tratamento da questão do retorno ao cliente dos resultados do processamento no servidor.

- *Regra 6 (R6) : Condições de retorno.*
  Requisições r' podem ser geradas a partir de processamentos executados por objetos sem estado e por objetos com estado:
  - O *objeto sem estado* não conserva informações de requisições e, se a invocação de um método de um objeto sem estado *O* (*requisição r*) é autorizada com ou sem restrição, a

mensagem de retorno (*r'*) será necessariamente autorizada: o intervalo de confiança da *requisição r'* é forçosamente incluso no intervalo de confiança do seu objeto chamador (o objeto servidor sem estado).

- *Em objetos com estado*, uma *requisição r*, invocando uma operação de *leitura* ou de *leitura-escrita*, sobre um objeto *O*, gera uma mensagem de retorno *r'* que provoca uma *escrita* no objeto emissor da requisição *r* (o objeto cliente). É necessário controlar essa escrita da mesma forma que a chamada de um método de escrita de um objeto qualquer. A *requisição r'* de retorno tem acesso de escrita sem restrição ao objeto emissor de *r* caso cumpra uma das seguintes condições:
  - Se o emissor de *r* for um objeto com estado: $fmin_{r'} \leq f_{\text{OBJETO EMISSOR}}$
  - Se o objeto cliente (emissor de *r*) for um objeto sem estado: $fmin_{r'} \leq fmax_{\text{OBJETO\_EMISSOR}}$

Caso nenhuma dessas condições seja satisfeita para a *requisição r'*, o acesso de retorno é negado (a escrita no cliente emissor de *r* é negada).

4.4 Gerenciamento e Controles de Políticas Obrigatórias no *JaCoWeb-Obrigatório*

Aplicações administrativas, executadas pelo administrador do sistema, cooperam para o estabelecimento de uma política obrigatória a ser armazenada no *PoliCap*.

As IDLs (***I**nterface **D**efinition **L**anguage*) dos objetos *DomainAccessPolicy* e *RequiredRights* devem ser modificadas para comportar os controles obrigatórios correspondentes. Ao objeto *DomainAccessPolicy* são acrescentados os seguintes métodos: *grant_clearance*, que insere o nível de habilitação do sujeito; *revoke_clearance*, que remove o nível de habilitação do sujeito; *replace_clearance*, que substitui a habilitação do sujeito e; *get_clearance*, que consulta a habilitação de um sujeito. Ao objeto *RequiredRights*, são acrescentados os seguintes métodos: *set_classification*, que insere a classificação do método do objeto e; *get_classification*, que consulta a classificação do método do objeto.

O próprio administrador do ambiente, executando uma aplicação administrativa, define a *habilitação* dos sujeitos usando os métodos *grant_clearance*, *revoke_clearance*, *replace_clearance* e *get_clearance*, do objeto *DomainAccessPolicy* e, especifica a *classificação* das operações existentes na interface de cada um dos objetos, usando a operação *set_classification*, do objeto *RequiredRights*.

Em *bind time* (*tempo de ligação*) a política obrigatória, juntamente com a discricionária, é verificada de forma global, através da interceptação de controle de acesso no lado do cliente que desvia o controle para o *PoliCap*. Uma vez verificada as condições de política discricionária, são montados os objetos locais no lado do cliente. Ao montar os objetos locais, a política obrigatória passa a ser representada nos objetos *DomainAccessPolicy* e *RequiredRights* locais, através das colunas *Clearance* e *Classification*.

Em *access decision time* (*tempo de decisão de acesso*), localmente no lado do objeto cliente, o objeto *AccessDecision* realiza o controle obrigatório, implementando as regras R1, R2, R3, R4, R5 e R6. Para implementar cada uma das regras, são usadas as operações para consulta dos rótulos de segurança, propostas para os objetos *DomainAccessPolicy* (*get_clearance*) e *RequiredRights* (*get_classification*). Depois da execução da regra apropriada, o controle discricionário da invocação é executado, no lado do cliente, gerando uma *capability*. Além dos valores associados a esta *capability*, o *Request* CORBA deve transportar mais um campo, cujo valor é o rótulo de segurança da requisição (seu intervalo de confiança). O objeto *AccessDecision*, no lado do servidor, faz a verificação da *capability* relativa ao controle discricionário, e extrai o rótulo de segurança correspondente ao intervalo de confiança da requisição nas verificações obrigatórias do lado do servidor em relação às

operações aninhadas.

O ajuste de rótulos da *requisição r'* (mensagens de retorno ou novas requisições), na nossa proposta, é realizado ainda no lado do cliente pelo objeto *AccessDecision*, antes que ocorra o fluxo de informações da *requisição r* para o lado do servidor. O mesmo procedimento é adotado para a regra R6 onde as suas condições sobre as mensagens de retorno são verificadas ainda no lado do cliente. Este é o caso de uma requisição *r*, correspondendo a um acesso de leitura ou de leitura-escrita que terá autorização negada, ainda no lado do cliente, caso as condições da R6 não sejam verificadas. As verificações das regras no lado do cliente, simplificam os controles obrigatórios associados às mensagens de retorno e prevê somente o fluxo de requisições autorizadas no ambiente.

5. Resultados de Implementação

Um protótipo da implementação do projeto *JaCoWeb-Obrigatório* foi desenvolvido (figura 5). Uma aplicação exemplo constituída de um sistema bancário composto por um objeto servidor CORBA e uma aplicação cliente Java foi construída para testar as implementações deste protótipo. O objeto servidor CORBA foi desenvolvido com a ferramenta *JacORB 1.3* (*www.jacorb.org*), um ORB Java *free*, e a aplicação cliente foi implementada com a ferramenta *JDK 1.3*. O objetivo dessa implementação foi de concretizar uma política de controle de acesso obrigatória baseada nas estruturas definidas no *CORBAsec*. O experimento implementa políticas obrigatórias, realizando a verificação de controle de acesso no lado do cliente. O protótipo está limitado a um único domínio de nomes. A figura 5 sintetiza as funcionalidades implementadas no protótipo.

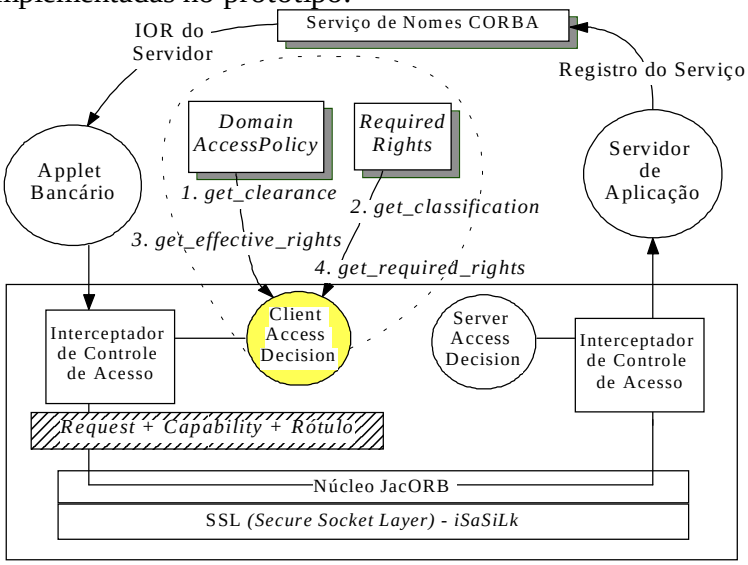

Figura 5. Estrutura do Protótipo Implementado.

Dentre os objetos implementados nesse protótipo (além da aplicação bancária e do servidor da aplicação) estão os objetos *ClientAccessDecision*, *ServerAccessDecision*, *DomainAccessPolicy* e *RequiredRights* do modelo CORBA de segurança. Esses objetos usam outros serviços CORBA (COSS) como o serviço de nomes. O objeto *ClientAccessDecision* é responsável pela validação dos pedidos de acesso aos métodos do objeto servidor, interagindo com os objetos *DomainAccessPolicy* e *RequiredRights*. O método *access_allowed* do objeto *ClientAccessDecision* implementa as regras que definem a evolução dos rótulos de segurança presentes na política obrigatória do ambiente. Depois de realizar os controles obrigatórios (passos 1 e 2, figura 5), realiza os controles discricionários normais de acordo com o modelo CORBAsec (passos 3 e 4, figura 5).

Os objetos *DomainAccessPolicy* e *RequiredRights* são criados estaticamente na máquina do cliente neste protótipo. A implementação usa como mecanismos de desvio os *interceptors* presentes no JacORB. No protótipo, o interceptador de controle de acesso no lado do cliente invoca o objeto *ClientAccessDecision*, para realizar o controle obrigatório e discricionário. Já no lado do servidor, o *interceptador de controle de acesso* invoca o objeto *ServerAccessDecision* que realiza as verificações no lado do servidor.

Um dos experimentos realizados no protótipo do sistema bancário, foi uma requisição de *leitura*, feita pelo objeto com estado *Conta_Simples*, com $f_O = 3$, sobre um objeto com estado designado *Conta_Especial* com $f_O = 4$. O usuário que acessou o objeto *Conta_simples* possui $f_s = 4$. A *requisição r* disparada por *Conta_Simples* tem como rótulo o intervalo [3,4]. Aplicando a regra R1, o acesso de leitura é autorizado com restrição, isto é, $fmin_{r'}$ deve ser ajustado para o valor 4. Dessa forma, a requisição *r'* tem como rótulo [4,4], o que provocaria pela regra R6 (figura 6) um bloqueio na mensagem de retorno ao cliente *Conta_Simples*, já que $fmin_{r'} > f_{Conta\_Simples}$. Como o ajuste de rótulos é realizado no lado do cliente no nosso sistema, essa operação de leitura não é executada já que a mensagem de retorno associada seria bloqueada.

```
public static String Regra_seis()
   {   ...
      if (nova_requisicao == "555")
         System.out.println("Retorno Recusado");
      else
         if ((acesso == leitura)||(acesso == leitura_escrita)||(estado_objeto == com_estado))
         {
          if (objeto_emissor == com_estado)
           if(minimo <= fobjeto_emissor)
            System.out.println("Requisição Aprovada");
           else
            nova_requisicao = "555";
          if (objeto_emissor != com_estado)
           if(minimo <= fmax_objeto_emissor)
            System.out.println("Requisição Aprovada");
           else
            nova_requisicao = "555";
         }
         return nova_requisicao;    }
```

Figura 6. Regra 6 executada pelo objeto *ClientAccessDecision*.

No protótipo, foram utilizadas as funções criptográficas IAIK-JCE e o pacote iSaSiLk 3.03 que implementa o SSL v3 em Java (http://jcewww.iaik.at/products/isasilk/index.php). Toda a negociação e uso do SSL é executada de forma transparente pelo iSaSiLk.

6. Trabalhos Relacionados

O trabalho desenvolvido por [13] propõe uma idéia inicial, bastante simplificada, da inclusão de políticas de controle de acesso obrigatórias no CORBAsec. O trabalho de [14] apresenta um esquema de autorização obrigatório, usando, para evitar a superclassificação de informações, as técnicas dos objetos sem estado e dos intervalos de confiança associados aos processos confiáveis. Este trabalho usa uma estrutura de programação orientada a objetos genérica, e define oito regras no seu esquema de autorização obrigatório. Sua abordagem não possui regras para a criação dos objetos e para tratar as mensagens de retorno em um esquema obrigatório.

Outros trabalhos relacionam o uso do modelo Bell e Lapadula com a programação orientada a objetos [14, 25, 26]. O trabalho de [26] define um algoritmo de "filtro" de mensagens, onde cada requisição tem o seu nível de "*clearance*" verificado junto ao rótulo de segurança associado ao objeto servidor. Sua abordagem considera rótulos de segurança associados aos objetos como um todo, e não a cada um dos seus métodos. A proposta de [25] considera um ambiente orientado a objetos construído sobre um sistema operacional

obrigatório tradicional. No trabalho de [15], os autores propõem uma forma de implementar modelos obrigatórios com RBAC (*Role-Based Acess Control*).

A proposta de política de segurança baseada no modelo Bell e Lapadula e a respectiva extensão do CORBASec preenche a lacuna destas especificações da OMG no que se refere a políticas de autorização obrigatória. Comparando o trabalho proposto com a literatura existente, podemos verificar que a proposta *JaCoWeb-Obrigatório* trata as políticas obrigatórias no ambiente CORBAsec. Sugere e usa técnicas para evitar a superclassificação das informações inerente ao modelo Bell e Lapadula.

Nosso trabalho, a exemplo de [14], usa os objetos sem estado e os intervalos de confiança associados com as requisições para evitar a superclassificação. Entretanto, nossa proposta de controles obrigatórios está inserida em um ambiente de objetos distribuídos acessados a partir de *browsers* Web. A nossa solução utiliza interceptadores do CORBAsec que permitem uma verificação transparente e também flexível. Uma das diferenças da nossa proposta, em contraste com outros trabalhos da literatura mencionada [14, 25], é a *concretização dos controles obrigatórios* realizados *no lado do cliente*, antes que a requisição inicie a comunicação com o objeto servidor. Isto possibilitou a existência de uma definição clara da regra que trata as mensagens de retorno, questão esta que normalmente é tratada de maneira informal. A implementação desta regra no nosso esquema não provoca bloqueios de retornos de requisições já processadas, como pode ocorrer em [14, 25, 26]. Nosso trabalho define seis regras que mostram a maneira como os rótulos de segurança evoluem no sistema, usando técnicas para evitar a superclassificação de informações e levando em consideração um ambiente de implementação.

7. Conclusões

Um dos objetivos desse trabalho é mostrar que este esquema de autorização, montado a partir de COTS (***C**ommercial-**O**ff-**T**he-**S**helf*), pode ser um meio de expressão de modelos como o Matriz de Acesso e Bell e Lapadula em ambientes heterogêneos e de larga escala. Neste artigo é introduzida a nossa experiência com esquema de autorização *JaCoWeb-Obrigatório*, que define as entidades e funcionalidades que fazem parte dos controles obrigatórios: os rótulos de segurança associados, as regras do esquema de autorização, o estabelecimento e uso de políticas obrigatórias a partir do *PoliCap*. Este esquema de autorização obrigatório possibilita a modelagem e a concretização da política de segurança obrigatória em ambientes que usam as especificações *CORBAsec*, contribuindo de forma significativa na implantação de políticas de segurança obrigatórias em ambientes distribuídos.

Nosso esquema de autorização aproveita as boas características do *PoliCap*, para estabelecer políticas discricionárias relacionadas com as políticas obrigatórias do ambiente. Para implementar as políticas obrigatórias, não foi necessária a inclusão de numerosas estruturas de dados no ambiente. Tivemos simplesmente a inclusão de um campo no objeto *DomainAccessPolicy* (habilitação do sujeito) e uma coluna no objeto *RequiredRights* (classificação do método do objeto).

Uma avaliação geral do *JaCoWeb*, incluindo os controles obrigatórios, foi feita tomando como base a metodologia de testes dos Critérios Comuns (*http://www.commoncriteria.org/cc/cc.html*). Informações sobre estas avaliações estão disponíveis em [23, 24] e (*http://www.lrg.ufsc.br/~carla/FinalReport.html*).

A existência dos controles obrigatórios, sobre os quais os usuários não têm controle, auxiliam a controlar as violações de segurança existentes em aplicações comerciais complexas que executam em sistemas de larga escala. As políticas obrigatórias permitem estabelecer regras incontornáveis que asseguram a proteção eficaz contra a descoberta de informações.

REFERÊNCIAS

[1] Object Management Group. Security Service:v1.5, *OMG Document Number 00-06-25*, June 2000.

[2] Carla Merkle Westphall and Joni da Silva Fraga. Authorization Schemes for Large-Scale Systems based on Java, CORBA and Web Security Models. *ICON 99 - The IEEE International Conference on Networks*, September 28 to October 1, pp. 327-334, 1999, Brisbane-Queensland, Australia.

[3] Rafael Rodrigues Obelheiro, Joni da Silva Fraga, Carla Merkle Westphall. Controle de Acesso Baseado em Papéis para o Modelo CORBA de Segurança. *In: XIX SBRC*, 2001, Florianópolis, p.869-884.

[4] OMG, Security Domain Membership Management Service, *orbos/01-07-20*, 2001.

[5] C. I. Dalton and J. F. Griffin. Applying Military Grade Security to the Internet. *In: Proceedings of the 8th Joint European Network Conference*, p. 711-1 – 711-8, Edinburgh, 1997.

[6] Q. Zhong and N. Edwards. Security Control for COTS Components. *IEEE Computer*, v. 31, n. 6, p. 67-73, June 1998.

[7] D. Elliot Bell e Leonard J. LaPadula. Security Computer Systems: Unified Exposition and Multics Interpretation. *MITRE Tech. Report MTR-2297 Rev. 1*, MITRE Corporation, Bedford, MA, March 1976.

[8] Leonard J. LaPadula and D. Elliot Bell. Mitre Technical Report 2547 - Volume II. *Journal of Computer Security*, v. 4, issue 2/3, p. 299-323, 1996.

[9] David Clark and David Wilson. A Comparison of Commercial and Military Computer Security Policies. *In: Proceedings of the 1987 IEEE Symposium on Security and Privacy*, p. 184-194, Oakland, CA, May 1987.

[10] C. E. Landwehr, C. L. Heitmeyer and J. McLean. A Security Model for Military Message Systems. *ACM Transactions on Computer Systems*, v. 9, n. 3, p. 198-222, August 1984.

[11] John McLean. Reasoning About Security Models. *In: Proceedings of the 1987 IEEE Symposium on Security and Privacy*, p. 123-131, Oakland, California, April 1987.

[12] Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, v. 26, n. 11, p. 9-19, November 1993.

[13] Günter Karjoth. Authorization in CORBA Security. *In: Proceedings of the Fifth ESORICS*, LNCS, p. 143-158, Springer-Verlag, Berlin Germany, September 1998.

[14] Vincent Nicomette and Yves Deswarte. An Authorization Scheme for Distributed Object Systems. *In: Proc. of the IEEE Symp. on Research in Security and Privacy*, p. 21-30, Oakland, California, 1997.

[15] Sylvia Osborn, Ravi S. Sandhu and Qamar Munawer. Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies. *ACM TISSEC*, v.3, n.2, May 2000.

[16] C. E. Landwehr. Formal models for computer security. *ACM Computing Surveys*, v. 13, n. 3, p. 247-278, September 1981.

[17] J. P. L. Woodward. Exploiting the Dual Nature of Sensitivity Labels. *In: Proc. of the IEEE Symposium on Research in Security and Privacy*, p. 23-30, Oakland, California, 1987.

[18] S. Foley. A Kernelized Architecture for Multilevel Secure Application Policies. *In: Proceedings of the European Symposium on Research in Computer Security*, Louvain-la-Neuve, Belgium, Sep. 1998.

[19] S. Foley, L. Gong and X. Qian. A security model of dynamic labeling providing a tiered approach to verification. *In Proc. of the 1996 IEEE Symp. on Security and Privacy*, p. 142-153, Oakland, May 1996.

[20] D. Elliot Bell. Secure Computer Systems: A Network Interpretation. *In: Proceedings of the 2nd Annual Computer Security Application Conference*, p. 32-39, Vienna, Virginia, USA, 1986.

[21] L. J. Fraim. SCOM: A Solution to the Multilevel Security Problem. *IEEE Computer*, v. 16, n. 7, p. 26-34, July 1983.

[22] Department of Defense Trusted Computer System Evaluation Criteria. *DoD 5200.28-STD*, Dec. 1985.

[23] Carla Merkle Westphall, Um Esquema de Autorização para a Segurança em Sistemas Distribuídos de Larga Escala, *Tese de Doutorado*, Curso de Pós-Graduação em Engenharia Elétrica, UFSC, Dez. 2000.

[24] Carla Merkle Westphall, Joni da Silva Fraga, Michelle Silva Wangham, Rafael R. Obelheiro, Lau C. Lung. PoliCap - Proposal, Development and Evaluation of a Policy Service and Capabilities for CORBA Security. Aceito para publicação no congresso *IFIP/SEC 2002 - 17th IFIP International Conference on Information Security* (http://www.sec2002.eun.eg/), 7 a 9 de Maio 2002, Cairo, Egito.

[25] J. K. Millen and T. F. Lunt. Security for object-oriented database systems. *In: Proceedings of the 1992 IEEE Symposium on Security and Privacy*, p. 260-272, Oakland, California, May 1992.

[26] S. Jajodia and B. Kogan. Integrating an object-oriented data model with multilevel security. *In: Proceedings of the 1990 IEEE Symp. on Security and Privacy*, p. 76-85, Oakland, California, May 1990.