Uma Biblioteca para Segurança de Aplicações CORBA

Stephany Martins , Nélio Cacho , Thaís Batista

[1]Departamento de Informática e Matemática Aplicada (DIMAp)
Universidade Federal do Rio Grande do Norte (UFRN)
Campus Universitário - Lagoa Nova - 59072-970 - Natal, RN

stephany@ppgsc.ufrn.br, cacho@lcc.ufrn.br, thais@ufrnet.br

Abstract. *This paper presents LOrbSec - a library that provides security functions by abstracting the CORBA Security Service (CORBASec) and by including new functionalities: the association of domain names to entities that access the system and a mechanism for verifying certificates. Besides, LOrbSec allows dynamic insertion/removal of security functions in objects that compose an application.*

Resumo. *Este artigo tem como objetivo apresentar LOrbSec - uma biblioteca que disponibiliza funções de segurança abstraindo o serviço de segurança de CORBA (CORBASec) e incluindo novas funcionalidades: a associação de nomes de domínios à entidades que acessam o sistema e um mecanismo para verificação de certificados. Além disso, LOrbSec permite inserção/remoção dinâmica de funções de segurança nos objetos que compõem uma aplicação.*

1. Introdução

O desenvolvimento de aplicações distribuídas utilizando componentes CORBA (*Common Object Request Broker Architecture*) [OMG, 1997] tem sido uma tendência atual uma vez que CORBA é uma especificação aberta que oferece suporte à interoperabilidade entre componentes heterogêneos, transparência de localização e independência de plataforma. Além disso, o modelo de desenvolvimento baseado em componentes promove o reuso de componentes podendo possibilitar a redução do custo e do tempo de desenvolvimento.

Para aplicações que necessitam de segurança como comércio eletrônico, aplicações bancárias, telecomunicações e gerenciamento de redes, o serviço de segurança de CORBA (CORBASec) [Lang and Schreiner, 2002, Blakley, 2000, OMG, 2001] oferece um conjunto de objetos e funções de segurança que podem ser usados para proteger componentes e mensagens trocadas entre os componentes em um ambiente CORBA. O serviço de segurança de CORBA introduz diversos conceitos, objetos e funções que juntos podem oferecer suporte a vários requisitos de segurança das aplicações. As funções de segurança oferecidas pelo CORBASec abrangem autenticação, controle de acesso, auditoria e não-repúdio.

Devido à diversidade de conceitos, objetos e funções que são definidos pelo CORBASec para prover segurança às aplicações, a utilização deste serviço é bastante complexa. Para usá-lo é necessário definir diversos objetos que dão suporte à segurança da aplicação e realizar várias invocações a funções para habilitar o serviço e prover a segurança necessária à aplicação. Isto significa que um esforço expressivo é necessário para incorporação de segurança em uma aplicação CORBA. O programador tem de dividir sua atenção entre a implementação da funcionalidade da aplicação e a provisão dos aspectos de segurança. Além disso, aplicações baseadas em componentes CORBA são compostas por um conjunto de componentes que podem estar distribuídos em

um ambiente de rede. Tais componentes distribuídos podem pertencer a diferentes domínios, cada um com suas políticas de segurança específicas. Portanto, gerenciar a segurança neste contexto é uma tarefa complexa.

Tal problema agrava-se mais quando é necessário gerenciar segurança em aplicações inerentemente dinâmicas. Estas aplicações podem necessitar de freqüentes reconfigurações dinâmicas, ou seja, modificações em tempo de execução que consistem na inserção e remoção de objetos que fazem parte da aplicação. Portanto, é necessário que os aspectos de segurança também possam ser modificados dinamicamente.

Além da complexidade, a especificação CORBASec apresenta dois outros problemas. O primeiro é não definir requisitos para manipulação, obtenção e validação de certificados. O segundo é não relacionar as políticas de controle de acesso com o conceito de domínio.

Neste artigo apresentamos uma biblioteca, chamada LOrbSec, que abstrai as complexidades inerentes à utilização do serviço de segurança de CORBA além de adicionar um mecanismo de verificação de certificados e uma estratégia para o relacionamento consistente entre políticas de controle de acesso e domínios. LOrbSec foi desenvolvida com o propósito de dar suporte à segurança de aplicações distribuídas baseadas em componentes CORBA no contexto de um ambiente de configuração e reconfiguração de aplicações distribuídas chamado LuaSpace [Batista, 2000]. LOrbSec adiciona as funcionalidades do CORBASec ao LuaSpace estendendo LuaOrb [Cerqueira et al., 1999], um *binding* entre a linguagem Lua [Ierusalimschy et al., 1996] e CORBA que permite acesso a objetos CORBA via Lua.

Este artigo está estruturado da seguinte forma: a seção 2 discute, brevemente, os conceitos básicos de LuaSpace e do serviço de segurança de CORBA. A seção 3 descreve LOrbSec e sua aplicabilidade em um estudo de caso. A seção 4 comenta alguns trabalhos relacionados. A seção 5 contém as considerações finais.

2. Conceitos Básicos

2.1. LuaSpace

LuaSpace é um ambiente para desenvolvimento de aplicações distribuídas baseadas em componentes CORBA que segue o modelo de programação orientado à configuração. LuaSpace integra a plataforma CORBA com a linguagem Lua e oferece um conjunto de ferramentas baseadas em Lua com funções estratégicas para facilitar o desenvolvimento de aplicações baseadas em componentes e para promover reconfiguração dinâmica. Em LuaSpace a aplicação é escrita em Lua e pode ser composta por componentes implementados em qualquer linguagem que tenha o *binding* para CORBA. Componentes, scripts e *glue code* são os elementos que formam a aplicação em LuaSpace.

Lua é uma linguagem interpretada, com um sistema de tipos dinâmico: variáveis não tem tipo, apenas valores estão associados à tipos. A linguagem possui algumas características não convencionais como: funções são valores de primeira classe; *arrays associativos* (chamados *tabelas*) são a única facilidade de estruturação de dados; *tag methods* é o mecanismo mais genérico para reflexão. *Tag methods* podem ser especificados para serem chamados em situações nas quais o interpretador Lua não tem como proceder.

LuaOrb é uma das principais ferramentas de LuaSpace. LuaOrb é um *binding* entre Lua e CORBA, baseado na Interface de Invocação Dinâmica de CORBA (DII), que oferece acesso dinâmico a componentes CORBA da mesma forma que se faz acesso a objetos Lua. O acesso a objetos CORBA é realizado de forma transparente. LuaOrb também usa a Interface de Esqueleto

Dinâmico (DSI) de CORBA para permitir instalação dinâmica de novos objetos em um servidor em execução. Para usar um componente CORBA é necessário, primeiro, criar um *proxy* Lua usando a função `createproxy` de LuaOrb. Esta função cria um objeto Lua que representa um objeto CORBA. Através do mecanismo de *tag methods*, as operações aplicadas sobre o *proxy* são tratadas por LuaOrb que as transforma em operações sobre componentes CORBA.

As demais ferramentas de LuaSpace não são apresentadas nesse artigo uma vez que não são relevantes no contexto desse trabalho.

2.2. Segurança em CORBA

CORBASec define diferentes funções de segurança e interfaces para serem usadas por desenvolvedores e administradores. As funções abrangem autenticação, autorização e controle de acesso, além de comunicação segura, não-repúdio e administração das políticas de segurança. Estas funções podem ser usadas por aplicações cientes ou não-cientes de segurança. Nas aplicações cientes, os métodos que viabilizam as funções do CORBASec estão misturados ao código da aplicação. Isto não acontece nas aplicações não-cientes, onde as funções de segurança são aplicadas através de arquivos de configuração que são passados como parâmetro no momento da execução da aplicação.

Para facilitar a utilização do CORBASec pelas aplicações cientes de segurança, as funções foram distribuídas em dois níveis: Nível 1 e Nível 2. No primeiro nível o ORB provê apenas métodos de obtenção de atributos de segurança. No segundo nível, além das funcionalidades do nível 1, são fornecidas funcionalidades como controle de acesso, auditoria e não-repudio.

A Figura 1 resume a distribuição das interfaces no CORBASec. A descrição hierárquica dessa Figura representa a dependência de cada módulo com o seu superior. Na hierarquia mais alta está o módulo *Security*, responsável por definir os tipos de dados usados por todos os demais módulos do serviço de segurança. Abaixo de *Security*, há os módulos *SecurityLevel1*, com sua função de obtenção de atributos de segurança, e *SecurityLevel2*, responsável por definir os objetos relacionados à comunicação e autenticação. Em um nível inferior há o *SecurityAdmin*, responsável por definir os objetos de administração, entre eles os relacionados à controle de acesso e auditoria; o *NRService*, responsável por definir os objetos relacionados ao não-repúdio; e o *SecurityReplaceable*, responsável por definir os objetos fornecidos pelo ORB que podem ser substituídos por implementações do usuário.

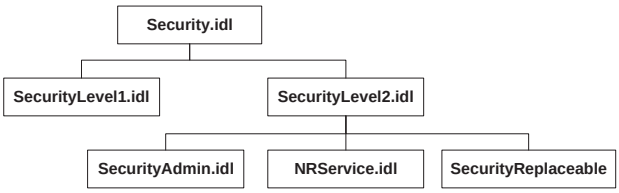

Figura 1: Hierarquia de interfaces no CORBASec

Nas subseções seguintes serão detalhadas as principais funções oferecidos pelo CORBASec: autenticação, controle de acesso e auditoria.

2.2.1. Processo de Autenticação

O processo de autenticação envolve uma série de conceitos, alguns deles com sentidos bastante gerais. Para facilitar o entendimento, mostraremos um sistema bancário que trabalha de forma segura sob o modelo cliente/servidor.

A Figura 2 exibe os integrantes do sistema. O primeiro elemento é o cliente do banco que telefona para o número do *call center* (0800 999) identificando-se com seu número de conta/agência e senha para obter o saldo de sua conta corrente. O outro elemento cliente é um objeto executando em um servidor de uma empresa contábil, que identifica-se através de um certificado digital reconhecido pelo banco e produz a folha de pagamento de seus clientes. O servidor bancário é o elemento central neste sistema. Para que seus clientes confiem nos seus dados é necessário que ele também tenha uma identidade. Neste caso ele tem duas identidades: o número do *call center*, que é uma referência para usuários humanos, e um certificado digital, que é uma referência para entidades de software.

Observando o exemplo sob o prisma da especificação CORBASec, todas as entidades que utilizam-se de identificadores (como número conta/agência e senha, certificado digital ou número de *call center* para acessar o sistema) são definidos como *Principais*. No exemplo, o cliente do banco, o servidor do banco e o servidor da empresa contábil são todos *Principais*. O processo de identificação dos *Principais* chama-se *Autenticação do Principal* e como resultado desse processo obtém-se um objeto *Credencial*. O objeto *Credencial* reúne informações relativas à identidade do *Principal* além de informações relacionadas à seus atributos de segurança, como tipo de conta (Pessoa Física ou Jurídica) que poderá ser usado para conceder ou negar determinados tipos de financiamentos bancários.

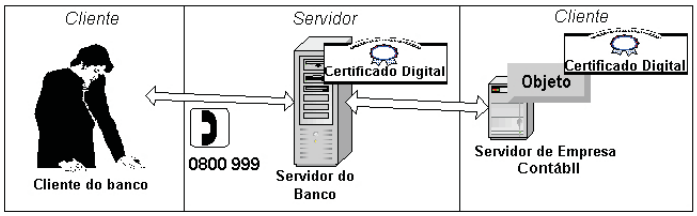

Figura 2: Sistema de informação bancaria

Portanto, quando o cliente (*Principal*) confia no número do *call center* (identidade do banco) e o acessa informando (processo de *Autenticação do Principal*) seu número de conta/agência e senha, ele obtém um objeto *Credencial*, que é guardado no objeto *Current*. Este objeto representa o contexto de execução atual e mantém informações de segurança dos objetos (cliente e alvo) envolvidos na comunicação. Neste momento o cliente estabeleceu um canal de comunicação com o servidor bancário e pode enviar e receber seus dados.

Certificados digitais foram citados no exemplo como elementos de identificação. No CORBASec são suportadas as seguintes plataformas de certificação digital: Kerberos, SPKM (*Simple Public-Key GSS-API Mechanism*), CSI-ECMA e SSL. No entanto, a especificação CORBASec não define requisitos para a obtenção, gerenciamento e validação de certificados de qualquer uma das plataformas citadas. Portanto, os certificados são utilizados, no contexto do CORBASec, apenas para criação de *Credenciais* de *Principais* com seus respectivos atributos de segurança.

2.2.2. Controle de Acesso

Após a autenticação, o sistema pode necessitar controlar quais operações poderão ser acessadas pelos seus clientes. Para ter esse controle é necessário a utilização das funções relacionadas à política de controle de acesso. O controle de acesso no CORBASec é realizado à nível de operação. Ou seja, à medida que as requisições são recebidas o servidor verifica, usando informações do objeto *Current*, se aquele *Principal* pode ou não executar a operação solicitada.

Para tomar essa decisão o servidor avalia o objeto *Credencial* do cliente, presente no objeto *Current*, e as políticas de acesso aplicadas ao objeto alvo. O objeto *Credencial* reúne informações de identidade e atributos de segurança. Através dos atributos de segurança pode-se obter os atributos de privilégio que são usados para o controle de acesso. Estes atributos servem para especificar quais os privilégios de um *Principal* dentro do sistema. Os atributos mais usados são: Identidade de acesso dos Principais (*AccessId*), Grupo (que normalmente reflete um departamento de uma organização - *GroupId* e o Grupo Primário ao qual um *Principal* pertence (*PrimaryGroup*).

Após a avaliação dos atributos de privilégio, o sistema verifica as políticas de acesso. Os direitos da política de acesso são representados pelas seguintes letras: g (*get*), s (*set*), u (*use*), e m (*manage*). Essas letras determinam os direitos garantidos (*Granted Rights*) e os direitos requisitados (*Required Rights*) para realizar uma operação .

O processo de avaliação utiliza-se dos combinadores *SecAnyRight* e *SecAllRights* para determinar se os *direitos garantidos* são suficientes em relação aos *direitos requisitados*. Desse modo, na utilização do *SecAnyRight*, uma operação é realizada quando qualquer um dos direitos (g,s,u,m) em *direitos garantidos* está também presente nos *direitos requisitados*. Utilizando-se *SecAllRights*, uma operação é realizada quando todos os direitos presentes nos *direitos requisitados* estão também presentes no *direitos garantidos*.

Para facilitar a aplicação das políticas de controle de acesso, a especificação CORBASec agrupa os objetos com políticas semelhantes em grupos chamados de *domínios (Domains)*. Quando um objeto é criado, ele se torna automaticamente membro de um ou mais domínios, estando sujeito às políticas de segurança desses domínios. Um domínio pode ter sub-domínios que representem a estrutura de uma organização.

2.2.3. Auditoria

Através do serviço de Auditoria é possível registrar os eventos relevantes ocorridos no sistema. Os eventos são registrados num canal de auditoria que pode ser um arquivo, um banco de dados ou o *log* do sistema.

Os eventos são registrados após o processo de avaliação que determina se o evento é de interesse ou não. Para avaliar o evento, inicialmente o serviço verifica o tipo de evento ocorrido. Um evento pode ter um dos seguintes tipos: *AuditPrincipalAuth* (processo de autenticação), *AuditInvocation* (invocação de operação), *AuditPolicyChange* (mudança da política de segurança), etc. Em seguida, o serviço avalia uma lista de condições chamada de Seletores (*Selectors*). Essa lista pode ser avaliada de duas formas. Na primeira, utiliza-se o *Combinador SecAllSelectors* que exige a validação de todas as condições da lista. Na segunda, utiliza-se o combinador *SecAnySelector* que exige a validação de pelo menos uma das condições da lista. Estas condições podem possuir os seguintes valores: nome da interface do objeto alvo, nome da operação, sucesso ou falha, dia da semana da chamada da operação, etc.

Para demonstrar a utilização das funções de auditoria no CORBASec, a Figura 3 ilustra um programa que audita, em um arquivo *severlog.txt*, todos os eventos de invocação(*AuditInvocation*) que possuem interface de nome *IDL:Hello:1.0* e operação igual a *hello_world*.

Por limitações de espaço, a Figura 3 omite o código necessário para iniciar o ORB e criar a hierarquia de domínios. As linhas 1 a 4 contêm a definição do canal de auditoria para o arquivo *serverlog.txt*. Em seguida, obtém-se uma referência do objeto *PolicyCurrent* (linhas 6 a 15), necessário para a obtenção do objeto *AuditPolicy*. As linhas 17 a 23 contêm a definição de um

AuditEventTypeList com o valor *Security::AuditInvocation*, para auditar por invocação. As linhas 25 a 30 incluem a definição de um *SelectorValueList* para auditar apenas as operações *hello_world* da interface *IDL:Hello:1.0* . Na linha 32 o combinador *SecAllSelectors* é definido. Finalmente, na linha 33 são aplicadas as políticas de auditoria ao objeto *AuditPolicy*.

```
1  CORBA::Object_var sm = orb->resolve_initial_references("SecurityManager");
2  SecurityLevel2::SecurityManager_var security_manager = SecurityLevel2::
       SecurityManager::_narrow(sm);
3  SecurityLevel2::AuditDecision_ptr auditdes = security_manager->
       audit_decision();
4  CORBA::Boolean audit = auditdes->create("file","serverlog.txt");
5
6  CORBA::Object_var policy_current_obj = orb->resolve_initial_references("
       PolicyCurrent");
7  SecurityLevel2::PolicyCurrent_var policy_current = SecurityLevel2::
       PolicyCurrent::_narrow(policy_current_obj);
8  assert(!CORBA::is_nil(policy_current));
9
10 CORBA::PolicyTypeSeq policy_types;
11 policy_types.length(1);
12 policy_types[0] = Security::SecTargetInvocationAudit;
13 CORBA::PolicyList * policies = policy_current -> get_policy_overrides(
       policy_types);
14 CORBA::Policy_ptr policy = (*policies)[0];
15 SecurityAdmin::AuditPolicy_ptr audit_policy = SecurityAdmin::AuditPolicy::
       _narrow(policy);
16
17 Security::AuditEventTypeList events;
18 events.length(1);
19 Security::ExtensibleFamily family;
20 family.family_definer = 0;
21 family.family = 12;
22 events[0].event_family = family;
23 events[0].event_type = Security::AuditInvocation;
24
25 Security::SelectorValueList selectors;
26 selectors.length(2);
27 selectors[0].selector = Security::Operation;
28 selectors[0].value <<= "hello_world";
29 selectors[1].selector = Security::InterfaceName;
30 selectors[1].value <<= "IDL:Hello:1.0";
31
32 Security::AuditCombinator audit_combinator =  Security::SecAllSelectors;
33 audit_policy->set_audit_selectors("",events,selectors,audit_combinator );
```

Figura 3: Exemplo do uso de auditoria no CORBASec

Como ilustra a Figura 3, uma simples auditoria pode requisitar várias linhas de código com a construção de toda uma seqüência de passos. Para permitir a utilização, de forma mais simples, da auditoria e das demais funções do CORBASec apresentaremos na próxima seção a biblioteca LOrbSec que, além de facilitar o uso de CORBASec, também acrescenta novas funcionalidades.

3. LOrbSec

LOrbSec é uma biblioteca que dá suporte ao desenvolvimento de aplicações seguras baseadas em objetos CORBA. Um dos seus objetivos é permitir que programadores utilizem o serviço de segurança CORBA sem a necessidade de conhecer detalhes da especificação. Ou seja, LOrbSec visa prover uma abstração sobre este serviço de forma que não seja necessário a criação de várias estruturas de dados, obtenção de várias referências a objetos, muitas chamadas às diversas funções para a realização de uma tarefa ou mesmo conhecimento de detalhes sobre o funcionamento e como os objetos do CORBASec relacionam-se.

Outro objetivo da biblioteca LOrbSec é oferecer um método para verificação de certificados baseado no uso da tecnologia SSL uma vez que, originalmente, a especificação do serviço de segurança CORBA não define suporte para verificação de certificados X.509.

LOrbSec também implementa o relacionamento entre as políticas de controle de acesso e os domínios, facilitando a administração e garantindo mais uma restrição no controle de acesso de Principais.

3.1. Arquitetura

A Figura 4 ilustra a parte da arquitetura do LuaSpace relevante para situar a biblioteca LOrbSec. Em LuaSpace o script de configuração de uma aplicação pode ser escrito diretamente no console Lua ou armazenado em um arquivo. O script é submetido ao interpretador Lua que o executa fazendo as chamadas à cada diferente componente da arquitetura para tratar comandos.

LuaOrb é acionado nos casos de chamadas que fazem acesso a objetos CORBA e faz o mapeamento entre a chamada Lua e a chamada correspondente na plataforma CORBA. LuaOrb também é invocado quando são feitas chamadas para instalação dinâmica de objetos Lua em um servidor remoto. Quando o interpretador Lua executa uma chamada de funções de LOrbSec, ele invoca a implementação da função, disponível na biblioteca LOrbSec. Através de LuaOrb, LOrbSec faz chamadas a objetos, serviços e repositórios CORBA. LOrbSec possui também autonomia de comunicar-se diretamente com outras bibliotecas que não fazem parte do ambiente LuaSpace, como a API SSL, para dar suporte à funcionalidades não fornecidas pelo CORBASec.

LOrbSec permite o uso de funções do CORBASec através da Interface de Invocação Dinâmica (DII) de CORBA, permitindo inserção/remoção dinâmica de funções e políticas de segurança nos objetos que compõem uma aplicação. Como a implementação CORBA (MicoSec [Schreiner and Lang, 2000]) utilizada trabalha apenas com interfaces estáticas, LOrbSec implementou a interação do MicoSec com a interface dinâmica.

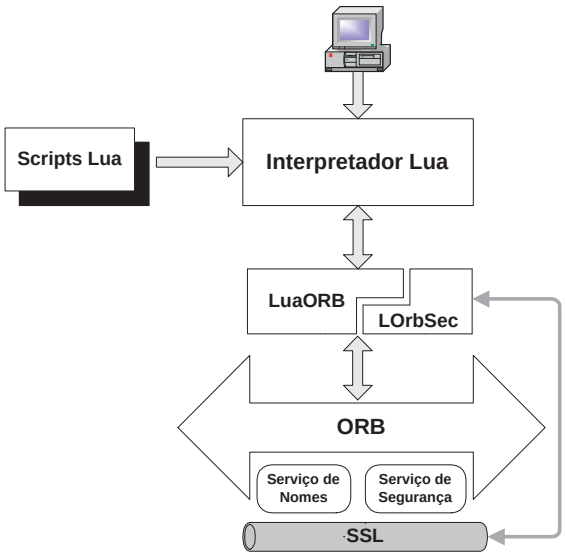

Figura 4: Arquitetura de LuaSpace

3.2. Funcionamento

Assim como a especificação do CORBASec, LOrbSec também foi dividida em dois níveis de funcionalidade. O primeiro representado pela classe LOrbSecLevel1 e o segundo por LOrbSecLevel2.

A classe LOrbSecLevel1 implementa o método *getTargetAttributes()*, além do método *getAttributes()*, referente ao nível 1 do CORBASec. Ambos os métodos retornam atributos

de segurança. No entanto, o primeiro obtém atributos de objetos remotos. Para utiliza-los invoca-se *LOrbSecLevel1:getTargetAttributes("AccessId",readIOR("./server.ref"))* ou *LOrbSecLevel1:getAttributes("AccessId")*. A diferença entre os dois é que, no primeiro, além de se informar o nome do atributo que se deseja obter, é necessário também informar a referência(IOR) do objeto remoto.

A classe LOrbSecLevel2 implementa os métodos relacionados à autenticação, controle de acesso e auditoria. Descreveremos cada um desses elementos nas sub-seções seguintes.

3.3. Autenticação em LOrbSec

O processo de autenticação na biblioteca LOrbSec é realizado pelas funções descritas na tabela 1. O método *create_certificate* é responsável por criar os certificados. Ele recebe como parâmetro as informações do proprietário (nome, e-mail, cidade, empresa, setor, etc) e retorna a referência do objeto que será usado para chamar os métodos *get_certificate()* e *get_key()*.

Tabela 1: Métodos para autenticação

Nome e parâmetros dos métodos	Descrição
create_certificate();	Criar certificados.
get_certificate();	Obter o certificado.
get_key();	Obter a chave do certificado.
authenticate();	Realizar a autenticação.
authentication_state();	Obter o estado da autenticação.
mechanism_type();	Obter o tipo de mecanismo gerado.

A Figura 5 mostra como criar certificados utilizando LOrbSec. Na linha 1 o método *create_certificate* cria o certificado e retorna a referência do objeto (*Cert*) que será usado para invocar os demais métodos. Nas linhas 2 a 4, através do método *get_certificate()* o certificado é obtido e armazenado no arquivo *ContabilCert.pem*. Nas linhas 5 a 7 a chave do certificado é obtida e armazenada no arquivo *ContabilKey.pem* através do método *get_key()*.

```
1 Cert = LOrbSecLevel2:create_certificate("BR","RN","Natal","Contabil Ltda.",
      "Faturamento","Gerente","gerente@contabil.com")
2 writeto("ContabilCert.pem")
3 write(Cert:get_cert())
4 writeto()
5 writeto("ContabilKey.pem")
6 write(Cert:get_key())
7 writeto()
```

Figura 5: Criando Certificados na LOrbSec

A criação de certificados é opcional uma vez que uma aplicação pode obter seus certificados por outras fontes, por exemplo, diretamente da API SSL. No entanto, a autenticação é requisito básico para que uma aplicação opere num ambiente seguro. Para dar suporte a autenticação, LOrbSec disponibiliza a função *authenticate()*. Esta função recebe como parâmetro o par certificado/chave e fornece a referência do objeto que será usada para chamar os métodos *authentication_state* e *mechanism_type()* que retornam, respectivamente, o *status* da autenticação e o mecanismo utilizado na autenticação. Portanto, para autenticar um *Principal* pode-se usar a chamada *Result = LOrbSecLevel2:authenticate("ContabilCert.pem","ContabilKey.pem")*. Para obter o status da autenticação pode-se usar *Result:authentication_state()*.

Com a utilização da biblioteca LOrbSec, todos os processos de autenticação que executam no ambiente LuaSpace passam por um módulo interno à biblioteca chamado de *Control Certificate*. Este módulo adiciona ao CORBASec a funcionalidade de verificação dos certificados,

uma vez que tal aspecto não foi definido na especificação do serviço de segurança CORBA. Portanto, quando o método *authenticate()* é invocado, o *Control Certificate* verifica, através da API SSL, as seguintes informações do par certificado/chave: data de validade, se a chave pública pertence ao certificado e se a autoridade certificadora emitiu o certificado. O sucesso no processo de verificação permite a continuação do processo de autenticação; uma falha interrompe o processo de autenticação e retorna um erro indicando o motivo da falha.

3.4. Controle de acesso no LOrbSec

Tabela 2: Métodos de Controle de Acesso

Nome e parâmetros dos métodos	Descrição
create_domain(domain_name);	Definir hierarquia de domínios.
set_required_rights(domain_name, idl_name, operation_name, combinator, required_rights);	Definir os direitos requeridos.
get_required_rights(idl_name, operation_name);	Obter os direitos requeridos.
grant_rights(sec_attr,condition, grant_rights,domain_name);	Definir os direitos garantidos.
get_rights(sec_attribute_type,condition);	Obter os direitos garantidos
replace_rights(sec_attr,condition,grant_rights,domain_name);	Substituir os direitos garantidos.
revoke_rights(sec_attr,condition,grant_rights,domain_name);	Revogar os direitos garantidos

Para aplicar as políticas de acesso, LOrbSec disponibiliza os métodos da tabela 2. A implementação de uma política depende inicialmente da construção da hierarquia de domínios, seguida pela distribuição, nos devidos domínios, dos objetos que possuem restrição de acesso e, por fim, a atribuição dos direitos aos usuários.

A hierarquia de domínios permite agrupar, em cada domínio, as operações que serão realizadas por um determinado grupo de usuários. Por exemplo, a Figura 6 ilustra um exemplo de uma hierarquia de domínios. O domínio *Banco* representa a aplicação bancária. Esta aplicação possui dois grupos de usuários, representados pelos sub-domínios *Clientes* e *Funcionários*. O grupo *Funcionários* possui ainda os sub-domínios *Gerentes* e *Caixas*. Para cada um desses domínios e sub-domínios deve-se determinar quais métodos estarão disponíveis. No exemplo, o domínio *Gerente* poderia possuir operações relacionadas à administração das contas (criar_conta, fechar_conta, etc). O domínio *Cliente* poderia possuir apenas as operações de movimentação de conta (depositar, sacar, etc). Portanto, a idéia é dividir os domínios de acordo com os níveis de funcionalidade da aplicação.

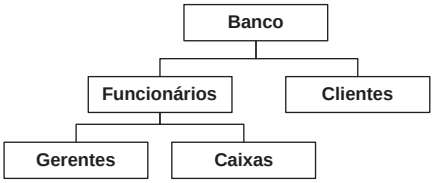

Figura 6: Hierarquia de Domínios num Sistema Bancário

Na biblioteca LOrbSec domínios são criados através do método *create_domain(domain_name)* que recebe como parâmetro uma *string* contendo o nome do domínio. O *domain_name* é representado de forma semelhante ao sistema de arquivo UNIX, onde o domínio raiz (*root*) é sempre (/) e chamadas como *create_domain("/Banco/Funcionarios/Gerentes"), create_domain("/Banco/Funcionarios/Caixas")* e *create_domain("/Banco/Clientes")* criam uma hierarquia semelhante à descrita na Figura 6.

Depois da construção da hierarquia de domínios é necessário definir quais as operações que irão requerer direitos para sua execução e associá-las à seus devidos domínios. Para

isso deve-se usar o método *set_required_rights()*. Este método recebe como parâmetro o nome do domínio, o nome da interface, o nome da operação, um *combinador* e os direitos requisitados. Pode-se então chamar *set_required_rights("/Banco/Funcionários/Gerentes", "IDL:Bank:1.0", "change_password","SecAllRights","sg")* para que o método *change_password* da interface *IDL:Bank:1.0* possa ser invocado apenas se todos (*SecAllRights*) os direitos requisitados (*sg*) tenham sua respectiva correspondência nos direitos garantidos do objeto *Credencial* do cliente. Um método pode estar presente em vários domínios por isso a importância do parâmetro *domain_name*. Por exemplo, a chamada *set_required_rights("/Banco/Clientes", "IDL:Bank:1.0", "change_password","SecAllRights","sgm")* disponibiliza para o domínio */Banco/Clientes* o método *change_password* para todos os clientes que possuirem direitos garantidos *sgm*.

A atribuição dos direitos garantidos é realizada pelo método *grant_rights()*. Este método recebe como parâmetro o atributo de privilégio, uma condição para o atributo de privilégio, os direitos garantidos e um nome de domínio. Este último parâmetro é opcional, uma vez que ele foi adicionado pela biblioteca LOrbSec pois não é disponibilizado pela especificação do CORBASec. Portanto, pode-se invocar *grant_rights("PrimaryGroupId","Customers","sg")* para garantir os direitos *sg* para os usuários que possuírem em suas chaves X.509 o *PrimaryGroupId* com valor *Customers*. Esses usuários, após essa invocação, poderão chamar todos os métodos de todos os domínios que requerem apenas os direitos *sg*. Isso acontece porque essa chamada é semelhante à chamada *grant_rights("PrimaryGroupId","Customers","sg", "/")*. Como os domínios seguem uma hierarquia, quando se aplica um direito à um elemento de nível mais alto, como "/", esse direito é propagado para os demais sub-domínios. Portanto, considerando as duas últimas chamadas *set_required_rights()*, um usuário com *PrimaryGroupId* igual a *Customers* poderia invocar indevidamente o método *change_password* do domínio */Banco/-Funcionarios/Gerentes*, uma vez que ele possui os direitos (*sg*) requeridos pelo método daquele domínio. Neste caso não haveria grandes problemas, pois ao invés de usarmos o método *change_password* do domínio */Banco/Clientes* usaríamos o seu correspondente no domínio */Banco/Funcionarios/Gerentes*. No entanto, nem sempre as situações são favoráveis, uma vez que se tivesse sido invocado *set_required_rights("/Banco/Funcionários/Gerentes", "IDL:Bank:1.0", "conceder_emprestimo","SecAllRights","sg")*, os usuários com *PrimaryGroupId* igual a *Customers* poderiam também invocar, indevidamente, o método *conceder_emprestimo*.

Para a especificação CORBASec a solução é criar novas letras, além das que já foram inicialmente definidas(s,g,u,m). Essa solução pode resolver o problema, mas dependendo do tamanho do sistema, pode torná-lo difícil de adiministrar. Para resolver esse problema, LOrbSec acrescentou ao método *grant_rights()* o parâmetro *domain_name*. Com isso o administrador pode garantir direitos para apenas um grupo(*Domain*) de objetos. Portanto, para evitar que os usuários com *PrimaryGroupId = Customers* tenham acesso aos objetos do domínio */Banco/Funcionários/Gerentes* deve-se chamar *grant_rights("PrimaryGroupId","Customers","sg", "/Banco/Clientes")*.

Além de garantir direitos, o administrador pode revogá-los ou substituí-los usando os métodos *revoke_rights()* e *replace_rights()*, respectivamente. Ambos possuem os mesmos parâmetros do método *grant_rights* com diferença apenas na funcionalidade. Considerando o último *grant_rights* invocado, podería-se revogar o direito g(*get*) de modo que o usuário mantivesse apenas o direito s(*set*). Isso seria feito chamando *revoke_rights("PrimaryGroupId","Customers","g", "/Banco/Clientes")*. Para substituir, numa única chamada, os direitos *sg* por u(*use*), invoca-se a função *replace_rights("PrimaryGroupId","Customers","u", "/Banco/Clientes")*.

Para dar suporte às políticas de acesso, LOrbSec disponibiliza

get_required_rights(idl_name, operation_name) e *get_rights(sec_attribute_type,condition)*. O primeiro método retorna os direitos requeridos para execução de uma operação (*operation_name*) na interface (*idl_name*). O segundo método retorna os direitos garantidos aos usuários com atributos de privilégio (*sec_attribute*) que tenham a condição (*condition*) satisfeita.

3.5. Auditoria no LOrbSec

A biblioteca LOrbSec oferece os métodos descritos na tabela 3 para registrar os eventos relevantes.

Tabela 3: Métodos de auditoria no LOrbSec

LOrbSec	Descrição
create_auditchannel(storage_type, details_name);	Criar canal de auditoria (tipo de armazenamento).
set_audit_selectors(audit_event_type,object_type, audit_combinator, selectors);	Definir seletores que determinam quando um registro de auditoria será gerado;

No processo de auditoria é necessário determinar onde os eventos serão registrados e, em seguida, determinar quais as condições para que um evento seja registrado.

A Figura 7 ilustra o mesmo exemplo do uso de auditoria da Figura 3, no entanto, nesse exemplo utiliza-se a biblioteca LOrbSec. Na primeira linha a chamada *canalfile = create_auditchannel("file","/root/logsitema.txt")* cria um canal do tipo *file* que irá armazenar os eventos no arquivo */root/logsitema.txt*. No CORBASec, os eventos são registrados em *canais de auditoria*. O método *create_auditchannel* cria um canal recebendo como parâmetros o tipo do canal (*storage_type*) que pode ter o valor *file*(um arquivo comum), *syslog*(arquivo de *log* do sistema) ou *db*(um banco de dados) e os detalhes do mecanismo determinado em *details_name*. Na linha 2 são definidas, através de uma tabela Lua, quais as condições (*seletores*) que o evento deverá satisfazer para ser registrado. A tabela contém as seguintes condições: *Operation = hello_word e InterfaceName = IDL:Hello:1.0*. Na linha 3, o método *set_audit_selectors()* é invocado. Este método recebe como parâmetro o tipo de evento (*audit_event_type*), um tipo de objeto (*objetc_type*), um *combinador* e os *seletores*. No exemplo, a chamada *set_audit_selectors("AuditInvocation","*","SecAllSelectors", seletores)* determina que serão registradas todas as invocações(*AuditInvocation*), de todos os tipos de objetos(*) que satisfazem todas as condições(*SecAllSelectors*) presentes na tabela *seletores*.

```
1 canalfile = LOrbSecLevel2:create_auditchannel("file","/root/logsistema.txt"
      )
2 seletores = {{name = "Operation",value = "hello_word"},{name = "
      InterfaceName", value = "IDL:Hello:1.0"}}
3 canalfile:set_audit_selectors("AuditInvocation", "*", "SecAllSelectors",
      seletores)
```

Figura 7: Utilizando Auditoria no LOrbSec

3.6. Estudo de Caso

Nesta seção discutiremos um estudo de caso que explora as funcionalidades de autenticação e controle de acesso do LOrbSec. Para isso utilizaremos o sistema bancário descrito na sub-seção 2.2.1 e mostraremos apenas o código relacionado ao servidor bancário e ao objeto da empresa contábil.

A Figura 8 exibe o código do servidor bancário. Nesse código, o servidor realiza o processo de autenticação utilizando seu certificado (*ServCert.pem*) juntamente com sua chave (*ServKey.pem*). Em seguida, cria uma hierarquia de domínios que tem como objetivo oferecer

tratamentos distintos para os clientes Pessoa Jurídica e Pessoa Física. Nas linhas 4 e 6, o servidor distribui os métodos pelos domínios e determina seus respectivos direitos requeridos. Como a empresa contábil tem um contrato especial com o banco ela recebe, nas linhas 7 e 8, os direitos *sg* para todos os métodos que estejam abaixo do domínio */Banco/Clientes*. Para os clientes comuns, na linha 9, são disponíveis apenas os métodos do domínio */Banco/Clientes/Fisica*. Finalmente, nas linhas 11 a 14, o servidor disponibiliza o objeto *Bank* da interface *IDL:Bank:1.0* e salva seu IOR no arquivo *bank.ref*.

```
1   SecurityLevel2:authenticate("ServCert.pem","ServKey.pem")
2   LOrbSecLevel2:create_domain("/Banco/Clientes/Juridica")
3   LOrbSecLevel2:create_domain("/Banco/Clientes/Fisica")
4   LOrbSecLevel2:set_required_rights("/Banco/Clientes/Fisica","IDL:
        Bank:1.0","depositar","SecAllRights","s");
5   LOrbSecLevel2:set_required_rights("/Banco/Clientes/Fisica","IDL:
        Bank:1.0","obter_saldo","SecAllRights","g");
6   LOrbSecLevel2:set_required_rights("/Banco/Clientes/Juridico","IDL:
        Bank:1.0","criar_conta","SecAllRights","sg");
7   cert = "/C=BR/ST=RN/L=Natal/O=Contabil Ltda./OU=Faturamento/CN=
        Gerente/Email=gerente@contabil.com"
8   LOrbSecLevel2:grant_rights("AccessId",cert,"sg","/Banco/Clientes")
9   LOrbSecLevel2:grant_rights("PrimaryGroupId","ClientesPessoaFisica",
        "sg","/Banco/Clientes/Fisica")
10
11  source_server = lo_createservant(Bank,"IDL:Bank:1.0")
12  writeto("bank.ref")
13  write(source_server:_get_ior())
14  writeto()
```

Figura 8: Código do servidor bancário

Na Figura 9, o objeto da empresa contábil realiza o processo de autenticação e verifica, na linha 2, se o processo obteve sucesso. Numa situação de sucesso, uma referência do objeto Bank é criada e usada para chamar o método *criar_conta*, que cria uma conta com código ”142”, com senha ”awt”para o cliente ”Senhor José”. O método *criar_conta* pode ser chamado porque o certificado apresentado pelo objeto da empresa contábil coincide com o *AccessId* exigido pelo servidor para garantir os direitos *sg*, necessários para invocação do método. Pelo mesmo motivo, o método *depositar* pode ser chamado para depositar R$240,00 na conta de *”Senhor José”*.

```
1   AuthStatus = SecurityLevel2:authenticate("ContabilCert.pem","
        ContabilKey.pem")
2   if (AuthStatus:authentication_state() == "success") then
3           Bank = lo_createproxy(readIOR("./bank.ref"),"IDL:Bank:1.0")
4           Bank:criarconta(142,"awt","Senhor José")
5           Bank:depositar(142,240)
6   end
```

Figura 9: Código do objeto da empresa contábil

4. Trabalhos Relacionados

Na literatura existem alguns trabalhos relacionados à extensão dos serviços definidos pelo padrão CORBASec. Entretanto, não identificamos trabalhos cujo enfoque seja facilitar o uso do serviço nem a incorporação do mecanismo de segurança em um ambiente de desenvolvimento de aplicações distribuídas e reconfiguráveis.

O trabalho [Westphall et al., 2002] abrange o uso de políticas de segurança obrigatórias no CORBASec baseadas no modelo Bell e Lapadula. O modelo de políticas obrigatórias está inserido no JaCoWeb, e explora um serviço de políticas denominado Policap. Além disso, o trabalho promove o controle de acesso aplicado no lado do cliente usando políticas discricionárias

e obrigatórias. Embora a descrição de políticas obrigatórias não esteja no escopo de LOrbSec, em termos de controle de acesso, LOrbSec inclui a associação de nomes de domínios a Principais, de forma a permitir que Principais tenham acesso a objetos que estão no mesmo domínio ao qual pertencem.

[Lampinen, 1999] apresenta uma estratégia para utilização de certificados SPKI para autorização em aplicações distribuídas baseadas em CORBA. Similarmente, LOrbSec oferece um mecanismo de verificação de certificados, no entanto, não provê suporte à delegação. O mecanismo de verificação de certificados de LOrbSec oferece maior flexibilidade pois pode funcionar sob diferentes *middlewares* e em outros ambientes de desenvolvimento, não limitando-se à CORBA e LuaSpace.

5. Conclusões

Esse artigo apresentou o serviço de segurança de CORBA (CORBASec), discutiu as complexidades relacionadas com o uso de tal serviço e apresentou uma biblioteca - LOrbSec - que abstrai tal complexidade e oferece funcionalidades adicionais: verificação de certificados e associação de nomes de domínios à principais. As funcionalidades providas por LOrbSec compreendem autenticação, controle de acesso, auditoria e verificação de certificados. Um estudo de caso de uma aplicação bancária segura ilustrou a aplicabilidade do LOrbSec.

LOrbSec oferece um conjunto de funções que minimiza o esforço de programação, evitando que o programador tenha de conhecer detalhes do serviço de segurança e que o código da aplicação possua diversas chamadas ao serviço que são alheias ao domínio da aplicação. Isto exime o desenvolvedor da necessidade de conhecer todas as operações dos objetos que implementam as funcionalidades do CORBASec e também do inter-relacionamento entre eles, para tornar aplicações CORBA seguras.

LOrbSec possibilita a inclusão de segurança em aplicações inerentemente dinâmicas uma vez que permite a inserção/remoção dinâmica de funções e políticas de segurança nos objetos que compõem uma aplicação.

A idéia apresentada neste trabalho, de proporcionar aos programadores que usam o ambiente LuaSpace uma camada de software em um nível mais alto de abstração para tornar viável o uso do serviço de segurança de CORBA, não se restringe simplesmente ao serviço de segurança mas a qualquer outro serviço CORBA que apresente complexidade na utilização. Uma vez que LuaSpace visa desenvolvimento de aplicações com simplicidade e reuso de componentes, é necessário proporcionar aos desenvolvedores ferramentas poderosas e fáceis de usar bem como habilitação automática dos serviços CORBA. Em [Batista et al., 2002] são apresentados mecanismos para seleção dinâmica de objetos distribuídos que exploram uma camada de abstração sobre os serviços de Nomes e de Trading de CORBA. Similarmente, em [Cacho and Batista, 2003] é apresentada uma biblioteca que abstrai o uso do serviço de notificação de CORBA e implementa funcionalidades adicionais para garantia de Qualidade de Serviço (QoS).

Em termos de trabalhos futuros, pretende-se inclui em LOrbSec um mecanismo de gerenciamento de certificados incluindo delegação bem como verificação de certificados emitidos por algumas outras entidades emissoras de certificados. Pretende-se também incluir um mecanismo para gerenciamento de políticas de forma a evitar o estabelecimento de políticas conflitantes.

O ambiente apresentado está operacional em uma plataforma Linux 2.2.17 (Red Hat 8), usando Lua versão 4 e LuaOrb versão 2.0. A implementação CORBA utilizada é a MICO e a implementação do serviço de segurança CORBA é a MICOSec.

Referências

Batista, T., Morais, J. N., Gadelha, M., and Teixeira, W. (2002). Seleção Dinâmica de Objetos Distribuídos no Ambiente LuaSpace. In *Anais do 20o. Simpósio Brasileiro de Redes de Computadores (SBRC2002)*, volume II, pages 703–718, Búzios - RJ.

Batista, T. (2000). *LuaSpace: Um Ambiente para Reconfiguração Dinâmica de Aplicações Baseadas em Componentes*. PhD thesis, PUC-Rio.

Blakley, B. (2000). *CORBA Security - An Introduction to Safe Computing with Objects*. Addison Wesley.

Cacho, N. and Batista, T. (2003). Suporte para Comunicação Assíncrona com QoS em Aplicações CORBA. In *Anais do 21o. Simpósio Brasileiro de Redes de Computadores (SBRC2003)*, volume I, pages 347–362, Natal - RN.

Cerqueira, R., Cassiano, C., and Ierusalimschy, R. (1999). Dynamic Component Gluing Across Different Componentware Systems. In *In International Symposium on Distributed Objects and Applications (DOA´99)*, pages 362–371, Edinburgh, Scotland.

Ierusalimschy, R., Figueiredo, L. H., and Celes., W. (1996). Lua - an extensible extension language. *Software: Practice and Experience*, 26(6).

Lampinen, T. (1999). Using SPKI certificates for authorization in CORBA based distributed object-oriented systems. In *Proceedings of the Third Nordic Workshop on Secure IT Systems Nordsec'99)*, Sweden.

Lang, U. and Schreiner, R. (2002). *Developing Secure Distributed Systems with CORBA*. Artech House Inc.

OMG (1997). The Common Object Broker Architecture and Specification. Technical Report formal/97-12-11, OMG.

OMG (2001). Security Services Specification, version 1.7. Technical report, Object Managment Group. Available at `http://www.omg.org/technology/documents/formal/omg_security.htm`.

Schreiner, R. and Lang, U. (2000). *MICOSec User´s Guide*. MICOSec Group (Object Security), , edition. available at http://www.micosec.org.

Westphall, C. M., Fraga, J., Westphall, C. B., and Bianchi, S. (2002). Políticas de Segurança Obrigatórias: Bell e Lapadula no CORBASec. In *Anais do 20o. Simpósio Brasileiro de Redes de Computadores (SBRC2002)*, Búzios - RJ.