MACA CLIENTE – GUIA DO PROGRAMADOR

Gustavo Motta

Versão: 3.2.2

São Paulo, setembro de 2005

APRESENTAÇÃO

O objetivo deste manual é mostrar como utilizar os serviços de autenticação de usuário e controle de acesso fornecidos pelo servidor MACA CS. Inicialmente são apresentados o modelo de controle de acesso adotado pelo MACA e a arquitetura de software onde foi implementado, enfatizando os serviços de autenticação e controle de acesso do serviço de segurança do CORBA/OMG. Posteriormente, dois tutoriais ilustram passo-a-passo como utilizar os serviços do MACA CS em Java e em Delphi. Ao final, esperamos que os programadores destas linguagem possam utilizar estes serviços autonomamente.

www.**incor**.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

SUMÁRIO

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

www.**incor**.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

LISTA DE FIGURAS

www.**incor**.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

1 Introdução

O objetivo do MACA (*Middleware* de Autenticação e Controle de Acesso) é prover os serviços de autenticação de usuário e controle de acesso para aplicações legadas ou em desenvolvimento, independente de plataforma e linguagem de programação, através de uma API padronizada.

O MACA implementa um modelo de autorização contextual para o controle de acesso baseado em papéis (CABP) definido pelo NIST. O CABP tem características adequadas para definição e administração viável de políticas de acesso, particularmente em aplicações corporativas emergentes, que demandam um controle com granularidade fina para um grande número de usuários e recursos.

A arquitetura de software do MACA baseia-se em padrões de processamento aberto e distribuído a fim de alcançar interoperabilidade e portabilidade para segurança. Adota o serviço de diretórios LDAP como uma base de informações de gerenciamento de segurança (BIGS); o CORBA *Security Service* (CSS) para autenticação de usuários e o serviço de decisão para acesso a recursos (RAD – *Facility*), do CORBA *horizontal facilities*, como soluções de *middleware* para autenticação de usuários e solicitação de autorizações de acesso, respectivamente, por parte das aplicações clientes. Esta solução viabiliza a administração da política de autorização e o controle de acesso de modo unificado e consistente, a partir de diferentes sistemas, em plataformas e linguagens de programação distintas, mas de forma padronizada.

Três componentes principais integram esta arquitetura: o servidor de autenticação e controle de acesso (MACA CS); o módulo para administração da política de autorização de acesso e gerência de contas de usuários (MACA AD); e aplicações clientes que solicitam autorizações de acesso e autenticação de usuários. Este manual descreve de forma prática como utilizar o MACA AD para configuração de uma política de autorização de acesso. A fim de facilitar seu uso, as subseções seguintes descrevem sucintamente os conceitos básicos de controle de acesso e do CABP, bem como a arquitetura do software do MACA.

1.1 Conceitos Básicos de Controle de Acesso

O controle de acesso vai limitar as ações ou operações que um usuário legítimo de um sistema computacional pode realizar, com base nas autorizações aplicáveis ao mesmo no momento do acesso. Uma autorização estabelece as permissões[1] de acesso que um sujeito[2] pode exercer em um determinado recurso computacional. Procura restringir o uso não autorizado dos recursos computacionais para evitar violações de segurança. A restrição do que pode ou não ser acessado é imposta diretamente às ações do usuário.

[1] Os termos direitos de acesso, privilégios, permissões e autorizações são usados neste texto indistintamente.
[2] Um sujeito pode ser um usuário humano ou algum agente autônomo que atua em benefício deste.

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

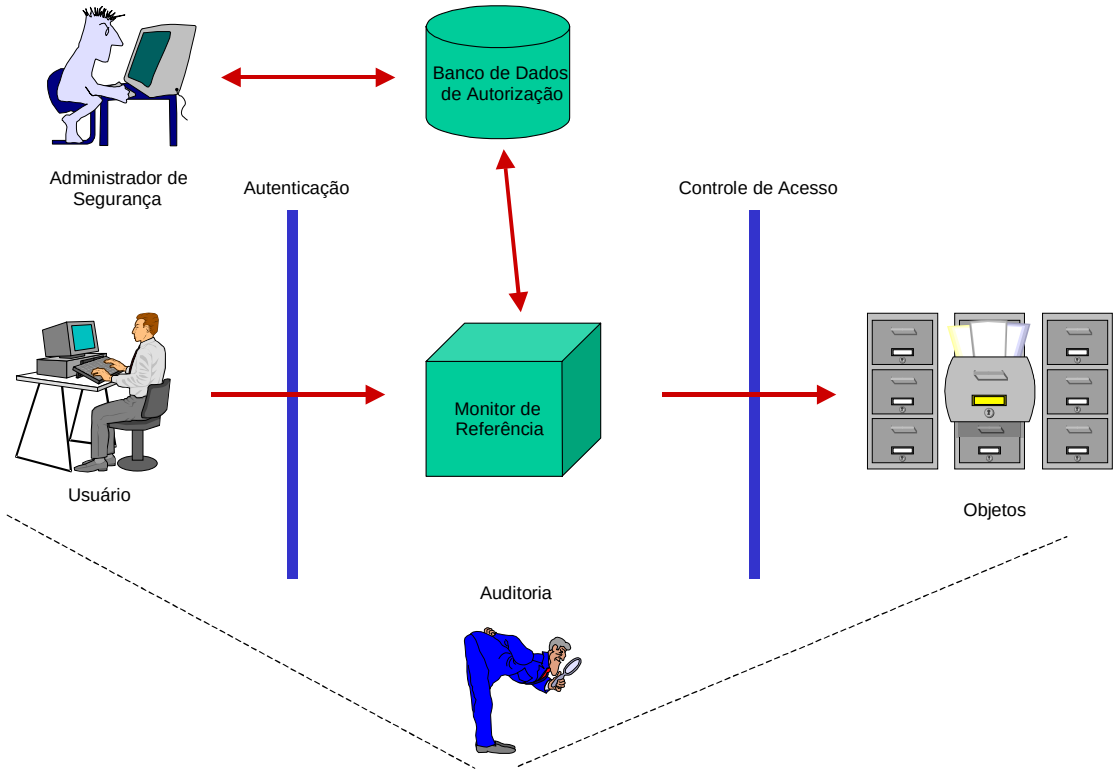


Figura 1 – Esquema do controle de acesso e outros serviços de segurança

Em geral, o controle de acesso exige a autenticação prévia do usuário que deseja usar os objetos protegidos (Figura 1). A autenticação estabelece a identidade do usuário para o sistema de segurança, tipicamente através de um identificação pessoal (nome para *login*) e senha. Quando necessária uma autenticação mais robusta, pode-se adicionalmente empregar cartões de identificação, certificados digitais ou dados biométricos do usuário. O controle de acesso é imposto pelo monitor de referência que intermedeia todas as tentativas do usuário de acessar os objetos protegidos. O monitor de referência consulta uma base de dados de autorização para determinar se um usuário que deseja realizar uma operação tem a permissão ou não realizá-la. A auditoria registra dados sobre a atividade no sistema e os analisa para descobrir violações de segurança e diagnosticar suas causas. O administrador de segurança é o responsável pela configuração e gerência das autorizações, de acordo com política de controle de acesso adotada.

Dentre os modelos de controle de acesso existentes, destacam-se o controle de acesso discricionário (CAD), o controle de acesso compulsório (CAC) e o controle de acesso baseado em papéis (CABP).

1.1.1 Controle de Acesso Discricionário

O CAD controla o acesso a um objeto protegido com base na identidade do usuário (ou grupos de usuários) e nas autorizações que especificam, para cada objeto, através de uma lista de controle de

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

acesso, os privilégios que cada usuário (ou grupo) tem (e. g., ler, escrever, executar, etc.). O proprietário do objeto pode, a seu arbítrio, conceder ou revogar privilégios de acesso para outros usuários (ou grupos), o que torna este enfoque bastante flexível e amplamente utilizado em sistemas operacionais e SGBDs. Entretanto, por sua natureza assimétrica e descentralizada, o CAD dificulta a administração dos privilégios de acesso que um usuário possui. É preciso examinar a lista de controle de acesso de cada objeto num sistema para determinar os privilégios de acesso de um usuário. Outra dificuldade ocorre quando se pretende remover parcial ou totalmente os privilégios de um usuário. Além de ser necessário visitar todas as listas de controle de acesso, o proprietário do objeto deve conceder o privilégio de modificação da lista para quem administra a política de controle de acesso. Assegurar isto em larga escala numa organização que tem um grande número de usuários e sistemas distribuídos e heterogêneos não é trivial.

1.1.2 Controle de Acesso Compulsório

O CAC confina o fluxo da informação numa única direção numa hierarquia de acesso. Está baseado na classificação de sujeitos e objetos. Por exemplo, um usuário da classe *secreto* pode ler qualquer objeto de uma classe igual ou inferior a sua, mas quando grava informações lidas ou criadas, estas só podem ser de uma classe igual ou superior a dele. Informações sensíveis lidas por uma pessoa da classe *secreto* não podem ser passadas para pessoas de uma classe de acesso menos privilegiada. No CAD, este tipo de controle não pode ser realizado. Já o CAC provê mais segurança para os dados, lidando com requisitos de segurança mais específicos, tais como uma política de controle do fluxo da informação. No entanto, implementar mecanismos que satisfaçam este modelo é uma tarefa difícil. Ademais, por ser excessivamente rígido, projetado para uso em ambientes militares, não possui a flexibilidade para suportar as situações excepcionais necessárias para um adequado controle de acesso às informações em ambientes corporativos.

1.1.3 Controle de Acesso Baseado em Papéis

O CABP permite regular o acesso dos usuários aos recursos protegidos com base nos papéis que eles exercem numa organização. Os papéis denotam funções que descrevem a autoridade e a responsabilidade concedidas a um usuário para o qual um papel foi associado. Neste caso, autorizações não são associadas diretamente a usuários, mas sim a papéis, de acordo com as atribuições pertinentes. Papéis são associados a usuários segundo as funções que eles exercem. Por exemplo, num hospital, se um usuário é um médico e tem o cargo de diretor clínico, ele terá os papéis *Médico* e *Diretor Clínico* associados. Consequentemente, seus direitos de acesso são os definidos para estes papéis, de acordo com a necessidade de saber/fazer inerente a autoridade e responsabilidade de cada papel.

Ademais, o CABP favorece a administração da política de acesso, pois permite coloca-la na perspectiva e um modelo organizacional. Usuários podem facilmente ser remanejados de um papel para outro e novas autorizações podem ser concedidas para papéis, refletindo as necessidades da

www.**incor**.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

organização. Como privilégios não são concedidos diretamente para usuários, mas a papéis, a rotatividade de pessoal tem um baixo impacto na administração da política de autorização, que é realizada de forma unificada através de papéis administrativos. Neste caso, os procedimentos para remoção de privilégios ou bloqueio de contas de acesso quando o vínculo de um usuário com a organização se encerra podem ser feitos com facilidade. Isto é particularmente útil em organizações onde o número de usuários com vínculo temporário não é desprezível, como no caso dos hospitais escola.

Como o CABP é viável identificar, a partir de um usuário, todas as suas autorizações e, por outro lado, a partir das autorizações, identificar todos os usuários. A administração da política de acesso pode ser unificada em papéis administrativos, não cabendo necessariamente esta atividade ao proprietário do objeto. Outro aspecto interessante é que o CABP é politicamente neutro, podendo suportar os modelos CAC ou CAD, dentre outros. Assim, o CABP permite uma administração unificada da política de acesso, mas com flexibilidade.

O Padrão para CABP do NIST

O padrão NIST para CABP (Figura 2) possui quatro conjuntos de entidades principais: *U* (usuários), *P* (papéis), *A* (autorizações) e *S* (sessões). Especifica que uma autorização é um relacionamento *n* para *m* entre os recursos protegidos (objetos) e respectivas formas de acesso (operações), mas deixa em aberto a representação de usuários, papéis, objetos e operações, bem como a interpretação de autorizações, cabendo estas tarefas a modelos mais detalhados. Estas entidades possuem os seguintes relacionamentos: usuário-papel *UP*; papel-autorização *PA*; hierarquia de papéis *HP* e sessões. As relações *UP* e *PA* especificam associações *n* para *m* entre usuários e papéis; e entre papéis e autorizações, respectivamente. *HP* define uma relação de ordem parcial entre papéis, dispondo-os em hierarquias a fim de melhor representar as linhas de autoridade e responsabilidade de uma organização. Uma sessão se relaciona com um único usuário por vez, mas permite que ele assuma (ative) múltiplos papéis simultaneamente, desde que estes papéis estejam associados ao usuário na relação *UP*. Por outro lado, um usuário pode ter várias sessões ao mesmo tempo.

Aos relacionamentos do padrão, podem-se estabelecer restrições para minimizar as chances de fraude ou dano acidental pela demasiada concentração de poder numa única pessoa. Uma restrição típica é limitar o número máximo de papéis de um usuário. Outra é a *separação de responsabilidades* (SR), que distribui a responsabilidade para realização de uma ação por múltiplos usuários, de modo que uma pessoa não seja poderosa o suficiente para efetuá-la sem um conluio. A SR é definida através de papéis mutuamente exclusivos, tanto na relação *UP*, quanto na relação *PA*. Em *UP*, dois ou mais papéis mutuamente exclusivos não podem ter usuários em comum associados. Já em *PA*, define-se a separação de responsabilidades proibindo-se a associação de uma mesma autorização a

www.**incor**.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

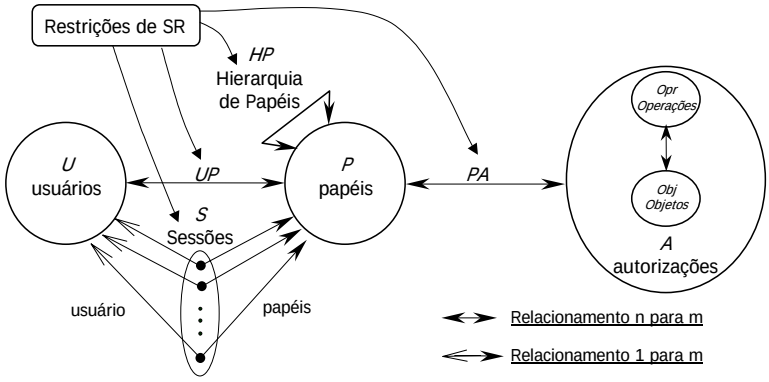


Figura 2 – Padrão NIST de referência para CABP(Fonte: NIST)

papéis mutuamente exclusivos. A idéia é adotá-la para reduzir a possibilidade de um usuário assumir papéis onde ocorram conflitos de interesse. Quando a restrição é imposta no momento em que estas relações são estabelecidas, ela é denominada de *separação de responsabilidades estática* (SRE). A *separação de responsabilidades dinâmica* (SRD) ocorre quando potenciais conflitos de interesse são detectados no momento em que um usuário tenta ativar mais de um papel simultaneamente, independente das sessões que abriu. A SRD admite um usuário possuir vários papéis conflitantes, desde que não sejam ativados ao mesmo tempo.

1.2 Arquitetura de Software do MACA

A arquitetura onde os componentes do MACA estão implementados é apresentada na Figura 3. É um modelo cliente-servidor multicamada com os seguintes componentes principais: um servidor LDAP, encarregado de manter a base de gerência de informações de segurança; um servidor de segurança (MACA CS), encarregado de oferecer serviços de autenticação de usuário, de decisão de acesso a recursos, dentre outros; e finalmente as aplicações cliente que requisitam estes serviços de segurança.

Nesta arquitetura, as aplicações cliente são protegidas pela *intranet* da instituição que as mantém. Isto é, a *intranet* limita o acesso apenas para terceiros confiáveis através de conexões seguras (uma rede privada virtual, por exemplo). Entretanto, tanto dos acessos externos, quanto os internos, são regulados pela política de segurança armazenada na BIGS e estabelecida pela instituição mantenedora das aplicações.

1.2.1 Base de Informações de Gerência de Segurança

A BIGS mantém num servidor LDAP configurações de segurança, tais como, autorizações de acesso, papéis, representações dos recursos protegidos e dos usuários, dados para autenticação, relacionamentos usuários-papéis, papéis-autorizações, etc. Todo acesso à BIGS deve ser realizado através do protocolo LDAP sobre TLS (*Transport Layer Security*) para assegurar confidencia-

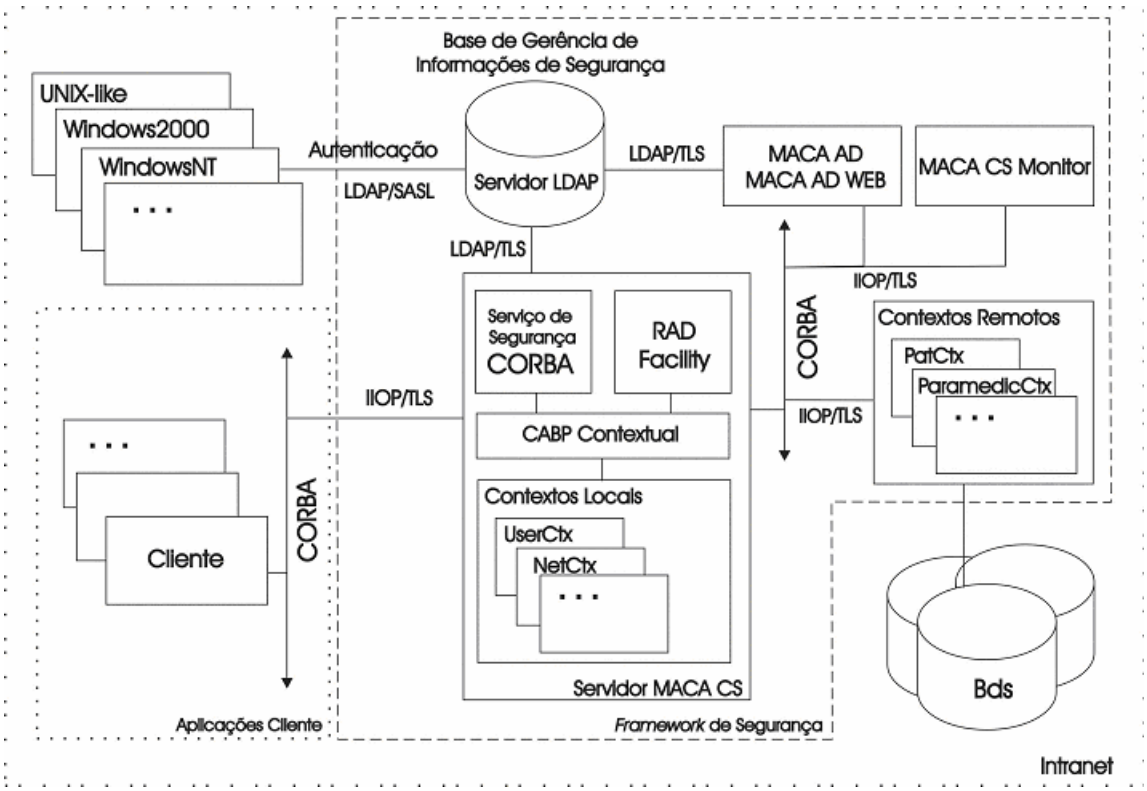


Figura 3 – Arquitetura de Software do MACA

lidade e integridade na comunicação. Adicionalmente, TLS pode assegurar autenticação mútua entre clientes LDAP e o servidor LDAP. Ademais, autenticação via sistemas operacionais e gerenciadores de bancos de dados mais populares podem efetuadas com segurança junto ao servidor LDAP através do protocolo SASL. Esta solução viabiliza a autenticação unificada de usuários numa organização, independente de sistema operacional ou aplicação.

1.2.2 MACA AD e MACA AD Web

A administração da política de segurança é realizada na BIGS com o MACA AD (Figura 3) e apenas por usuários privilegiados através de interações seguras e com autenticação e controle de acesso adequados. De modo similar, o MACA AD WEB permite a gerência de contas de usuários via *web* com flexibilidade para definição dos atributos armazenados em cada conta no LDAP. O servidor LDAP deve ser protegido fisicamente e todos os acesso não locais devem ser desabilitados. A localização unificada das políticas de segurança facilita a administração, mas introduz uma sobrecarga adicional, visto que todos os clientes LDAP usam um único servidor. Ademais, um servidor central introduz um único ponto de falha e abre oportunidades de ataques para colocá-lo fora de serviço. Como LDAP é um serviço de diretório distribuído e a maioria das implementações disponíveis têm mecanismo de réplica automático, é viável construir um servidor logicamente centralizado e tolerante a falhas, embora fisicamente distribuído e redundante.

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

1.2.3 MACA CS

Cabe ao servidor de segurança (Figura 3) oferecer autenticação, autorização e controle de acesso às aplicações clientes, dentre outros serviços de segurança. O RAD – *Facility* oferece interfaces padronizadas que permitem o controle de acesso detalhado, ao nível da aplicação, mas de uma forma em que a lógica do controle de acesso é separada da lógica da aplicação, com transparência em relação ao modelo de decisão efetivamente implementado. Este *framework* é adequado para suportar o modelo de autorização do MACA, pois prevê o tratamento dos fatores dinâmicos que influenciam a lógica de autorização e possibilita a combinação de diferentes políticas de controle de acesso. O Serviço de Segurança CORBA oferece interface padrão para autenticação de usuários e faz o controle de acesso transparente para as operações definidas nos objetos CORBA. O módulo "CABP Contextual" implementa o modelo de autorização do MACA, sendo utilizado como política de autorização de acesso pelo CSS e pelo RAD – *Facility*. Todas as interações entre objetos CORBA cliente e o servidor de segurança devem ocorrer via IIOP (*Internet Inter-ORB Protocol*) sobre TLS.

Serviço de Autenticação do Usuário

A fim de que o usuário possa usar recursos protegidos para os quais ele tenha privilégios de acesso, é necessário que ele antes seja autenticado (Figura 4). O CSS usa o conceito de *principal* (ou sujeito) para representar usuários humanos ou entidades computacionais que são registrados e autenticados no ambiente CORBA protegido. Um principal pode ser autenticado de várias maneiras, sendo a forma mais comum para humanos com o uso de uma senha. O *User Sponsor* é um código de responsabilidade da aplicação que implementa a interface com o usuário para um processo de autenticação, através da invocação de operações padronizadas definidas em interfaces do CSS. Representa o ponto de entrada para o usuário num sistema seguro, por exemplo, um diálogo de *login* e senha. Quem realmente efetua a autenticação é o objeto *Principal Authenticator*, para o qual o CSS especifica interfaces visíveis para o *User Sponsor* (parte da aplicação), embora não especifique nenhum método de autenticação específico. A interface IDL (*Interface Definition Language*) CORBA da operação que efetua a autenticação é a seguinte:

```
interface PrincipalAuthenticator {

  AuthenticationStatus authenticate(
    in AuthenticationMethod method,
    in SecurityName security_name,
    in Opaque auth_data,
    in AttributeList privileges,
    out Credentials creds,
    out Opaque continuation_data,
    out Opaque auth_specific_data
  );

...

};
```


www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

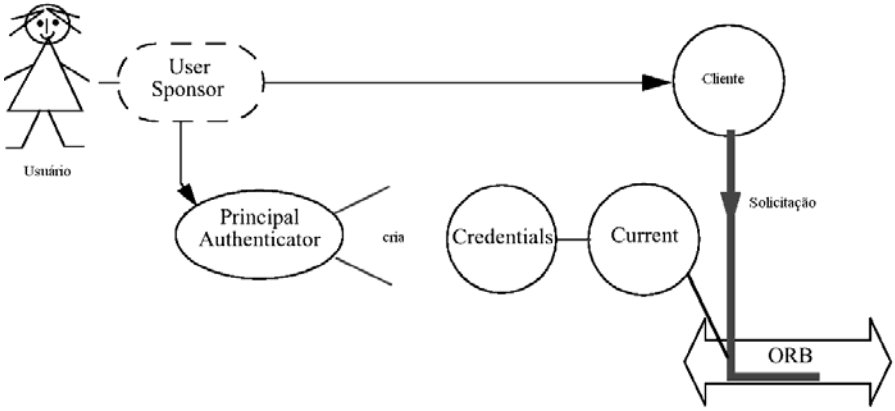


Figura 4 – *Use Case* para autenticação do usuário (Fonte: OMG)

O parâmetro `method` especifica o método de autenticação (e.g., senha simples), `security_name` o nome de *login* do usuário, `auth_data` a senha (ou outras informações que provem sua identidade) e `privileges` os atributos de segurança requeridos pelo usuário (e. g., o papel inicial que pretende assumir), todos parâmetros de entrada do método. Na execução, o método instancia um objeto com as credenciais (`creds`) do principal, que correspondem a uma sessão do usuário do modelo de CABP do MACA. A credencial armazena os atributos de segurança do usuário, que incluem sua identidade, papéis que está assumido, prazo de validade, dentre outras informações. Tanto a autenticação, quanto o preenchimento das credenciais do usuário, são efetuados com informações provenientes da BIGS (Figura 3). Os dois últimos parâmetros de retorno são usados em mecanismos mais sofisticados de autenticação, envolvendo múltiplos estágios. A resposta `AuthenticationStatus` indica se o usuário foi ou não autenticado. O objeto *Current*, que representa o contexto atual de execução de um principal, tanto do lado do cliente, quanto do lado do servidor.

Serviço de Controle de Acesso

No RAD *Facility*, a lógica da autorização é encapsulada pelo serviço de segurança, externo à aplicação cliente (Figura 3). A seqüência das interações entre os objetos cliente e alvo com o RAD *Facility* (Figura 5) é listada a seguir:

1. Um objeto cliente invoca uma operação da interface implementada pelo objeto alvo;
2. Enquanto processa a invocação, o objeto alvo solicita uma autorização do ADO (*Access Decision Object*), objeto que implementa as funções de autorização de acesso, através da chamada do método `access_allowed()` do ADO. O ADO é um objeto servidor CORBA e está executando no servidor MACA CS (Figura 3);

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

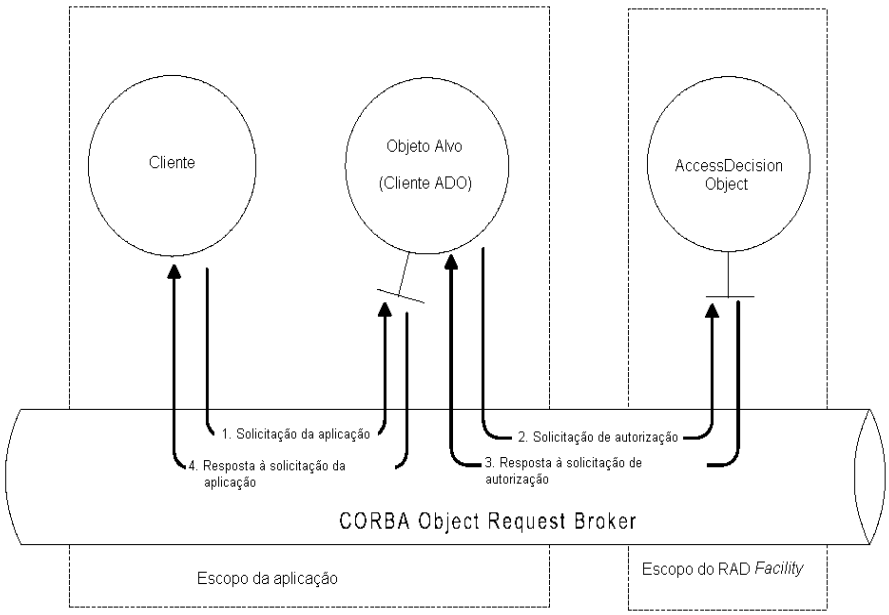


Figura 5 – Interações entre cliente, objeto alvo e o RAD *Facility* (Fonte: OMG)

3. O ADO consulta objetos que são internos ao RAD *Facility* e acessa a BIGS para obter informações de autorização, a fim de efetuar a decisão de autorização do acesso, retornando um valor booleano;
4. O objeto alvo, após receber a decisão de autorização, é o responsável pela imposição da decisão. Se o acesso for autorizado pelo ADO, o objeto alvo executa a operação solicitada e retorna o resultado para o objeto cliente. Caso contrário, o objeto alvo pode retornar resultados parciais ou pode levantar uma exceção para o cliente.

Nota-se que a aplicação (denominado cliente ADO) precisa fazer somente uma invocação para o serviço de autorização a fim de obter a decisão. Os únicos parâmetros que um cliente ADO passa são os três seguintes: o nome do recurso a acessar (resource_name), o nome da operação invocada (operation) e um conjunto autenticado de atributos de privilégios do principal para o qual a aplicação está servindo (attribute_list). A interface IDL CORBA abaixo especifica o método access_allowed que é chamado:

```
interface AccessDecision {
  boolean access_allowed(
    in ResourceName resource_name,
    in Operation operation,
    in AttributeList attribute_list
  )
...
};
```


www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

Um cliente RAD sempre obtém autorizações de uma mesma instância do ADO. Ainda, é o contrato entre a aplicação e o RAD *Facility* que garante a imposição das políticas de controle de acesso adotadas numa instituição.

1.2.4 Implementação

Os módulos MACA CS e MACA AD foram implementados em Java e o MACA AD WEB foi desenvolvido em Java/JSP. O MACA AD WEB permite a um administrador de contas criar, consultar, atualizar e remover contas usuários. O MACA AD permite a um administrador de autorizações estabelecer políticas de acesso através da definição de papéis, recursos e autorizações. O módulo "CABP Contextual" implementa o serviço de autenticação de usuários com *login* e senha, o gerenciador de sessões de usuários e o serviço de autorização de acesso. Ainda provê um interpretador para regras para autorização contextuais. Tais regras permitem que a política de acesso seja estabelecida com base em variáveis ambientais que denotam informações sobre usuário corrente, data/hora e local do acesso, e outras que podem ser livremente programadas e incorporadas para especificação de políticas de autorização mais complexas. Os contextos locais são implementados em Java e compartilham o mesmo espaço de endereçamento do módulo "CABP Contextual". Contextos remotos são acessados por clientes Java/CORBA via IIOP sobre TLS. Contextos são bibliotecas dinâmicas carregadas em tempo de execução através do mecanismo de extensão de Java.

www.**incor**.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

2 Tutorial – Usando o MACA CS em JAVA

Este tutorial mostra como utilizar os serviços de autenticação e autorização de acesso do MACA CS em programas cliente JAVA.

2.1 Requisitos

Antes de iniciar este tutorial, baixe o pacote de APIs para clientes da versão escolhida do MACA no *site* http://maca.sourceforge.net/index.html#Downloads e o descompacte com o seguinte comando no LINUX:

```
gunzip -c maca_cliente_VERSÃO.tgz | tar xf -
```

Caso esteja no MS-Windows, utilize o utilitário WinRar, disponível no *site* http://www.rarlab.com/, para descompactá-lo. Em seguida, verifique a lista abaixo:

- Java 2 Software Development Kit – J2SDK – versão 1.4.x ou superior instalado;
- Servidor MACA CS instalado e iniciado (remota ou localmente);
- Acesso aos seguintes arquivos fornecidos no diretório recém descompactado:
  - \maca_cliente_VERSAO\maca_cliente_java\maca_jacorb_stubs-VERSAO.jar
  - \maca_cliente_VERSAO\maca_cliente_java\jacorb.jar
  - \maca_cliente_VERSAO\maca_cliente_java\jacorb.properties
  - \maca_cliente_VERSAO\maca_cliente_java\orb.properties
  - \maca_cliente_VERSAO\maca_cliente_java\MacaCliente.java

2.2 Utilizando o Serviço de Autenticação

O programa “MacaCliente.java” apresenta um exemplo bem simples de como efetuar uma autenticação. Para ver como funciona, faça o seguinte:

1. Na sua estação de trabalho, crie um subdiretório denominado **maca_cliente** e copie os arquivos **maca_jacorb_stubs-3.2.2.jar**, **jacorb.jar**, **jacorb.properties**, **orb.properties** e **MacaCliente.java** nele;
2. Com o editor de texto de sua preferência, abra o arquivo **MacaCliente.java**, localize o trecho de código abaixo

```
readIORs("http://127.0.0.1");
```

e troque o *string* `"http://127.0.0.1"` pelo *string* contendo o nome do servidor onde o servidor MACA CS está ativado, por exemplo, `"http://ldap.datasus.gov.br”`;

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

3. Agora, localize o trecho de código abaixo

```
pa.authenticate(CLEAR_TEXT_PWD,     //Médoto de autenticação
                "nome.sobrenome",   //Nome de login do usuário
                "maca".getBytes(), //Senha
                attrs,              //Atributos de segurança requeridos
                credHolder,         //Retorna a credencial criada para o
                                    //em caso de sucesso
                contData,           //Valor de retorno não usado
                authEspec);         //Valor de retorno não usado
```

   e modifique o nome do *login* do usuário "`nome.sobrenome`" para o seu *login* e a senha "`maca`" para a sua senha;

4. Salve o arquivo e abra um *prompt* do DOS no subdiretório que foi criado;

5. Agora vamos compilar o programa executando o seguinte comando:

   `javac -classpath ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar -d ./ MacaCliente.java`

6. Para testá-lo, basta executar o comando abaixo:

   `java -cp ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar;./ tutorial.MacaCliente`

7. Uma autenticação bem sucedida apresentará a saída abaixo:

```
Autenticação realizada com sucesso!
Sessão encerrada!
```

2.2.1 Entendendo o programa "MacaCliente.java"

Para utilizar o serviço de autenticação em CORBA, a primeira tarefa a realizar é ler os IORs (*Interoperable Object Reference*) que referenciam os objetos no servidor MACA CS. Neste caso, o IOR do objeto servidor de autenticação é publicado via HTTP num arquivo denominado "maca_cs.iors", juntamente os IORs de outros objetos servidores. Após sua recuperação, o IOR é transformado num objeto CORBA cliente correspondente ao objeto CORBA servidor. Atenção, é imperativo que os IORs usados sejam obtidos da mesma chamada à função `readIORs`. Recomenda-se ler os IORs imediatamente antes de usá-los, para que não fiquem desatualizados quando o servidor MACA CS for reinicializado. A segunda tarefa é ler o arquivo de propriedades do `orb.propreties` para inicialização do ORB do JAVA, presente no pacote `org.omg.CORBA`, que deve ser importado no programa cliente. A variável correspondente ao ORB deve preferencialmente ser estática e ser a única desta classe, isto é, ser um *singleton*. Ao final, antes de sair do programa, o ORB deve ser finalizado. Caso se deseje utilizar o serviço de nomes do CORBA (opcional – não problema se o serviço de nomes não for encontrado), basta modificar as propriedades `org.omg.CORBA.ORBInitialPort` e `org.omg.CORBA.ORBInitialHost` para incluir a porta e o nome do servidor onde se localiza o serviço de nomes CORBA. Note que estas propriedades devem ser as mesmas usadas pelo MACA CS,

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

de modo que o mesmo serviço de nomes seja usado, tanto pelo servidor MACA CS, quanto pelo cliente. Os trechos de código correspondentes a esta etapa são mostrados em destaque abaixo:

```
package tutorial;

import java.io.*;
import java.net.*;
import maca_cs.*;
import java.util.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.PortableServer.*;
import org.omg.maca_cs.Security.*;
import org.omg.maca_cs.SecurityLevel2.*;
import org.omg.maca_cs.DfResourceAccessDecision.*;
import org.omg.CosNaming.NamingContextExtPackage.*;

public class MacaCliente {

  //Método de autenticação baseado em senha simples
  static final int CLEAR_TEXT_PWD = 0;

  static ORB orb;
  static NamingContextExt rootContext;
  static String  paIOR = "";
  static String adoIOR = "";
...
public static void main(String[] args) {
    try {
      //Ler via HTTP os IORs do serviços de autenticação e de
      //controle de acesso junto ao MACA CS, isto é, no
      //servidor onde o MACA CS foi instalado.
      readIORs("http://127.0.0.1");
      //Carrega as propriedades iniciais do ORB.
      FileInputStream st = new FileInputStream("./orb.properties");
      Properties orbProps = new Properties();
      orbProps.load(st);
      System.getProperties().putAll(orbProps);
      //Inicializa o ORB
      orb = ORB.init(args, System.getProperties());

      //Conecta-se com o serviço de nomes do CORBA
      try {
        rootContext = NamingContextExtHelper.narrow(
                                        orb.resolve_initial_references("NameService"));
      }
      catch (org.omg.CORBA.SystemException exc) {
        System.out.println("Serviço de nomes não localizado. Verifique as "+
                           "propriedades do ORB:");
        System.out.println("     org.omg.CORBA.ORBInitialPort");
        System.out.println("     org.omg.CORBA.ORBInitialHost");
      }
 ...
      //Finaliza o ORB no cliente
      orb.shutdown(false);
    }
    catch (Exception exc) {
      exc.printStackTrace();
    }
  }
}
```

Com o ORB inicializado, é preciso se conectar com o serviço de autenticação no MACA CS através de um objeto cliente do tipo `PrincipalAuthenticator`. A chamada do método `conectaPrincipalAuthenticator`

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
//Conecta-se com o objeto CORBA servidor de autenticão "pa"
PrincipalAuthenticator pa;
pa = conectaPrincipalAuthenticator(rootContext);
```

logo após a inicialização do ORB faz esta conexão. Caso o serviço de nomes tenha sido localizado (`rootContext != null`), então recupera-se a referência ao objeto através do método `resolve_str`, passando-se como parâmetro o nome o objeto no serviço de nomes que implementa a interface `PrincipalAuthenticator`. Para o MACA CS, o nome completo deste objeto é "*maca_cs.ctx/authenticators.ctx/pa.server*". Caso o serviço de nomes não tenha sido localizado (`rootContext == null`), então basta usar o IOR do objeto servidor recuperado anteriormente para se conectar.

Para finalizar, é preciso converter, via "*type casting*", o tipo deste objeto cliente para o tipo `PrincipalAuthenticator`, implementado pelo servidor de autenticação. O código do método `conectaPrincipalAuthenticator` é mostrado a seguir com os respectivos destaques:

```
static PrincipalAuthenticator conectaPrincipalAuthenticator(NamingContextExt
rootContext) throws Exception {
  org.omg.CORBA.Object corbaObj;
  if (rootContext == null) { //Serviço de nomes indisponível. Usa o IOR obtido via HTTP
    //Conecta-se com o objeto CORBA servidor do tipo PrincipalAuthenticator
    corbaObj = orb.string_to_object(paIOR);
  }
  else { //Conecta-se com o objeto CORBA servidor do tipo PrincipalAuthenticator via
         //serviço de nomes. O nome do objeto CORBA servidor do tipo
         //PrincipalAuthenticator é "maca_cs.ctx/authenticators.ctx/pa.server"
    corbaObj = rootContext.resolve_str("maca_cs.ctx/authenticators.ctx/pa.server");
  }
  //Faz o "type casting" do objeto do tipo org.omg.CORBA.Object para o tipo
  //PrincipalAuthenticator
  PrincipalAuthenticator pa = PrincipalAuthenticatorHelper.narrow(corbaObj);
  return pa;
}
```

Uma vez disponível, o objeto `pa` do tipo `PrincipalAuthenticator` pode ser usado para realizar a autenticação de usuário através da chamada do método `authenticate`, conforme indicado no trecho código abaixo:

```
//Definição dos parâmetros do método de autenticação do PrincipalAuthenticator
//A lista de atributos de segurança apenas para autenticação pode ser vazia
SecAttribute[]         attrs = {};
//Holder que retornará a credencial criada para o usuário
CredentialsHolder credHolder = new CredentialsHolder();
//Parametros de retorno para continuação da autenticação - não usados
OpaqueHolder        contData = new OpaqueHolder();
OpaqueHolder       authEspec = new OpaqueHolder();

AuthenticationStatus authStatus =
      pa.authenticate(CLEAR_TEXT_PWD,    //Médoto de autenticação
                      "nome.sobrenome",  //Nome de login do usuário
                      "maca".getBytes(), //Senha
                      attrs,             //Atributos de segurança fornecidos
                      credHolder,        //Retorna a credencial criada para o
                                         //usuário em caso de sucesso
                      contData,          //Valor de retorno não usado
                      authEspec);        //Valor de retorno não usado
```


www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

No MACA CS, o método de autenticação atualmente disponível é baseado em senha simples, sendo indicado pelo valor da constante `CLEAR_TEXT_PWD` igual a 0. O argumento `attrs` pode ser usado para fornecer, ainda durante a autenticação, o nome de um papel inicial para ativar, por exemplo. Neste caso, nenhum atributo é fornecido. A ativação de papéis no MACA é totalmente automática. A chamada a `authenticate` retorna um objeto do tipo `AuthenticationStatus` que indica se a autenticação foi bem sucedida ou não. O código abaixo examina os valores possíveis deste retorno.

```
//Verifica o status da autenticação
      switch (authStatus.value()) {
        case AuthenticationStatus._SecAuthContinue : {
          //Este caso ocorre quando o usuário é obrigado a mudar a senha ou quando
          //a senha expira. Em ambos os casos, a troca imediata da senha é obrigatória
          //para completar o processo de autenticação. Neste caso, é preciso
          //recuperar credencial criada para esta sessão do usuário
          Credentials creds = credHolder.value;
          //Recupera mensagem sobre a autenticação
          String authMessage = new String(authEspec.value);
          System.out.println(authMessage);
          //Nesta situação a credencial fica com sua funcionalidade restrita
          //enquanto a senha não é trocada. Após a troca, a credencial fica
          //completamente operacional.
          //Muda a senha do usuário
          String msg = mudaSenha(creds, "caca", "bola");
          if (!msg.equalsIgnoreCase("")) { //A senha não foi trocada
            System.out.println("Senha não modificada: "+msg);
            //sessão é encerrada
            creds.destroy();
            break;
          }
          System.out.println("Senha modificada");
          //Continua o processo de autenticação
          if (pa.continue_authentication(new byte[0], creds,
                        contData, authEspec) != AuthenticationStatus.SecAuthSuccess) {
            System.out.println("Falha na continuaçao da autenticação!");
            //sessão é encerrada
            creds.destroy();
            break;
          }
        }
        case AuthenticationStatus._SecAuthSuccess : {
          System.out.println("Autenticação realizada com sucesso!");
          //Recupera mensagem sobre a autenticação, por exemplo,
          //a que avisa que a senha irá expirar numa determinada data.
          //Recomenda-se sempre mostrar esta mensagem para o usuário
          String authMessage = new String(authEspec.value);
          System.out.println(authMessage);
          //Recupera credencial criada para esta sessão do usuário
          Credentials creds = credHolder.value;


          //Encerra sessão do usuário destruindo a credencial
          creds.destroy();
          System.out.println("Sessão encerrada!");
          break;
        }
        case AuthenticationStatus._SecAuthFailure : {
          System.out.println("Falha na autenticação!");
          break;
        }
      }
```

Caso a autenticação tenha sido bem sucedida, o argumento de saída `credHolder` armazena a referência de uma credencial válida do usuário. Enquanto existir, esta credencial corresponderá à sessão

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

do usuário. A sessão é encerrada quando é destruída pela chamada do método `destroy`, ou com *timeout* por inatividade, ou através de um mecanismo opcional de *callback* do servidor para o cliente (ver subseção 2.5). Caso a autenticação não tenha sido bem sucedida, os motivos possíveis são os seguintes:

- Falha na autenticação (`authStatus = SecAuthFailure`): o login ou a senha fornecidos são inválidos;
- A autenticação deve ser continuada (`authStatus = SecAuthContinue`): este caso ocorre quando o usuário é obrigado a mudar a senha para completar a autenticação. Acontece quando o usuário realiza o primeiro login e a propriedade (`maca_ca.passwordPolicy.passwordMustChange`) está ativada, ou então, quando a senha do usuário expira (propriedade `maca_ca.passwordPolicy.passwordExp=true`) e aí, em ambas as situações, ele deve, necessariamente, trocar a senha para continuar a usar o sistema normalmente. Nesta situação, a credencial retornada tem sua funcionalidade restringida, somente permitindo a operação de troca de senha (ver detalhes na seção 2.3.6) e a operação para destruí-la. Qualquer solicitação de autorização será negada enquanto a senha não for trocada. Após a mudança bem sucedida da senha, o processo de autenticação continua com a chamada do método `continue_authentication`. A credencial deve ser destruída, com o encerramento da sessão do usuário caso a autenticação não seja concluída com sucesso.

Recomenda-se tratar todos estes casos no processo de autenticação. **Note que é importante recuperar a mensagem com dados sobre a expiração da senha para mostrá-la ao usuário, avisando a data da expiração da senha, orientando a trocá-la imediatamente**. Esta informação é retornada no parâmetro `authEspec` do método `authenticate`, conforme ilustrado no código anterior.

2.3 Utilizando a Interface “Credentials”

Através da interface “Credentials” é possível realizar as seguintes ações, dentre outras possíveis:

- Obter o nome do usuário associado à conexão;
- Obter um conjunto de atributos selecionados da conta do usuário
- Obter a relação de papéis ativos e disponíveis para o usuário;
- Ativar papéis para o usuário;
- Modificar a senha do usuário;
- Verificar se a credencial está válida – método is_valid();
- Refrescar a credencial para ela não expirar por inatividade – método refresh();

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

- Destruir a credencial – método `destroy()`.

2.3.1 Obtendo o *Login* do Usuário

Recuperar o nome de *login* do usuário é bastante simples. O método abaixo, disponível no programa MacaCliente.java, faz isto.

```
static String pegaLogin (Credentials creds) {
    //Verifica se a credencial foi criada
    if (creds != null) {
      //Define a família dos atributos de segurança
      //O primeiro argmento indica quem é o definidor da família
      //Valor 0 significa que o OMG é  o definidor
      //O segundo argmento indica a família de atributos
      //Valor 1 significa que são atributos de privilégio
      ExtensibleFamily extFamily = new ExtensibleFamily((short) 0, (short) 1);
      //Define o tipo de atributo a recuperar, o login (AccessId) do usuário, neste caso
      AttributeType[]   attrType = {new AttributeType(extFamily, AccessId.value)};
      //Recupera o valor do tipo de atributo a partir da credencial
      SecAttribute[]       attrs = creds.get_attributes(attrType);
      //Examina a lista de valores em busca do tipo de atributo requisitado
      for (int n = 0; n < attrs.length; n++) {
        if ((attrs[n].attribute_type.attribute_family.family_definer == 0) &&
            (attrs[n].attribute_type.attribute_family.family == 1) &&
            (attrs[n].attribute_type.attribute_type == AccessId.value)) {
          return new String(attrs[n].value);
        }
      }
    }
    return "";
}
```

Primeiro, é preciso definir o tipo de atributo a recuperar, no caso a identificação de acesso do usuário, especificada pela constante `AccessId`. Os outros tipos possíveis são: `AccountingId`, `AuditId`, `GroupId`, `PrimaryGroupId`, `Role`, `Capability` e `Clearance`. Destes, o que vamos mais utilizar é o `Role`. Para testar a função acima, faça o seguinte:

1. Com o editor de texto de sua preferência, abra o arquivo **MacaCliente.java**, localize o trecho de código abaixo

```
//Recupera credencial criada para esta sessão do usuário
Credentials creds = credHolder.value;
```

   e adicione na linha seguinte o código

```
System.out.println("Login: "+pegaLogin(creds));
```

2. Salve o arquivo;
3. Compile o programa executando o seguinte comando:

   `javac -classpath ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar -d ./ MacaCliente.java`

4. Execute o programa alterado com o comando abaixo:

   `java -cp ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar;./ tutorial.MacaCliente`

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

5. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Sessão encerrada!
```

2.3.2 Obtendo um Conjunto de Atributos da Conta do Usuário

Com as credenciais é possível recuperar um conjunto de atributos selecionado da conta do usuário. Para saber os possíveis tipos de atributos retornados, entre em contato com o administrador do MACA CS e solicite a lista de nomes de atributos presentes na propriedade "maca_cs.ldap.userAccountAttrList" da configuração do MACA CS. O método abaixo, disponível no programa MacaCliente.java, mostra como recuperar os atributos a partir das credenciais do usuário.

```
static String[] pegaAtributosUsuario (Credentials creds) {
    String[] atributos = {};
    //Verifica se a credencial foi criada
    if (creds != null) {
      //Define a família dos atributos de segurança
      ExtensibleFamily extFamily = new ExtensibleFamily((short) 0, (short) 1);
      //Define o tipo de atributo a recuperar, o conjunto de atributos da conta do
      //usuário, neste caso
      AttributeType[]   attrType = {new AttributeType(extFamily, AttributeSet.value)};
      //Recupera o valor do tipo de atributo a partir da credencial
      //Cada elemento da lista retorna um atributo, do tipo string, com o seguinte
      //formato:
      //<nome do atributo>: <valor_1 do atributo>, <valor_2 do atributo>, ..., <valor_n
      //do atributo>
      SecAttribute[]       attrs = creds.get_attributes(attrType);
      //Define a quantidade de atributos da conta retornados
      atributos = new String[attrs.length];
      //Examina a lista de valores em busca do tipo de atributo requisitado
      for (int n = 0; n < attrs.length; n++) {
        if ((attrs[n].attribute_type.attribute_family.family_definer == 0) &&
            (attrs[n].attribute_type.attribute_family.family == 1) &&
            (attrs[n].attribute_type.attribute_type == AttributeSet.value)) {
          atributos[n] = new String(attrs[n].value);
        }
      }
    }
    return atributos;
  }
```

Note que este método é semelhante ao que recupera o *login*. A diferença é que o tipo do atributo a recuperar é `AttributeSet`, isto é, um conjunto de atributos selecionados para conta de usuários. Caso um atributo esperado (um daqueles listados na propriedade "maca_cs.ldap.userAccountAttrList" do MACA CS) não retorne, estão é porque ele não está armazenado na conta do usuário. A lista de valores de atributos terá uma entrada por tipo de atributo. O formato de cada valor retornado é o seguinte:

- <nome do atributo>: <valor_1 do atributo>, <valor_2 do atributo>, ..., <valor_n do atributo>

Por exemplo, o valor retornando para o atributo `title`, com dois valores armazenados na conta de um usuário no servidor LDAP, poderia ser o seguinte:

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

- title: Analista, Gerente

Finalmente, para testar a função acima, faça o seguinte:

1. Com o editor de texto de sua preferência, abra o arquivo **MacaCliente.java**, localize o trecho de código abaixo

```
System.out.println("Login: "+pegaLogin(creds));
```

e adicione na linha seguinte o código

```
//Recupera o conjunto de atributos do usuário
String[] atributos = pegaAtributosUsuario(creds);
System.out.println("Atributos: ");
for (int i = 0; i < atributos.length; i++) {
  System.out.println(atributos[i]);
}
```

2. Salve o arquivo;
3. Compile o programa executando o seguinte comando:

   javac -classpath ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar -d ./ MacaCliente.java

4. Execute o programa alterado com o comando abaixo:

   java -cp ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar;./ tutorial.MacaCliente

5. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
employeeNumber: 1
Sessão encerrada!
```

2.3.3 Obtendo os Papéis Associados ao Usuário

Recuperar os papéis do usuário é bastante simples. O método abaixo, disponível no programa MacaCliente.java, faz isto.

```
static String[] pegaPapeis (Credentials creds) {
    String[] roles = {};
    //Verifica se a credencial foi criada
    if (creds != null) {
      //Define a família dos atributos de segurança
      ExtensibleFamily extFamily = new ExtensibleFamily((short) 0, (short) 1);
      //Define o tipo de atributo a recuperar, os papéis (Role) do usuário, neste caso
      AttributeType[]   attrType = {new AttributeType(extFamily, Role.value)};
      //Recupera o valor do tipo de atributo a partir da credencial
      //O primeiro elemento de attrs tem a lista dos papéis ativos separados por ";".
      //É um string vazio quando não há papéis ativados
      //Os demais elementos do tipo Role correspondem aos papéis associados ao usuário
      SecAttribute[]       attrs = creds.get_attributes(attrType);
```


www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
      //Define a quantidade de papéis associados retornados
      roles = new String[attrs.length-1];
      //Examina a lista de valores em busca do tipo de atributo requisitado
      //Começa pelo segundo elemento do vetor porque se procura por papéis associados e
      //não os ativos
      for (int n = 1; n < attrs.length; n++) {
        if ((attrs[n].attribute_type.attribute_family.family_definer == 0) &&
            (attrs[n].attribute_type.attribute_family.family == 1) &&
            (attrs[n].attribute_type.attribute_type == Role.value)) {
          roles[n-1] = new String(attrs[n].value);
        }
      }
    }
    return roles;
}
```

Note que este método é semelhante ao que recupera o *login*. A diferença é que o tipo do atributo a recuperar é `Role`, isto é, os papéis do usuário. Na lista de valores de atributos retornados, o primeiro elemento é um *string* com a lista de papéis ativados para o usuário separados por ";". Como nenhum papel foi ativado, o valor atual é um *string* vazio. Os demais elementos da lista (posições >= 1) correspondem aos papéis associados que estamos interessados. Para testar a função acima, faça o seguinte:

1. Com o editor de texto de sua preferência, abra o arquivo **MacaCliente.java**, localize o trecho de código abaixo

```
for (int i = 0; i < atributos.length; i++) {
  System.out.println(atributos[i]);
}
```

   e adicione na linha seguinte o código

```
String[] roles = pegaPapeis(creds);
System.out.print("Papéis: ");
for (int i = 0; i < roles.length; i++) {
  System.out.print(roles[i]+(i==(roles.length-1)?"\n":", "));
}
```

2. Salve o arquivo;
3. Compile o programa executando o seguinte comando:

   `javac -classpath ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar -d ./ MacaCliente.java`

4. Execute o programa alterado com o comando abaixo:

   `java -cp ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar;./ tutorial.MacaCliente`

5. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
```


www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
employeeNumber: 1
Papéis: Vendedor Sênior, Gerente de Produto
Sessão encerrada!
```

2.3.4 Ativação de Papéis

Agora que sabemos como obter os papéis associados a um usuário, vamos ver como proceder para ativá-los. O método abaixo, disponível no programa MacaCliente.java, nos diz como.

```
static boolean ativaPapel(Credentials creds, String umPapel) {
    //Verifica se a credencial foi criada
    if (creds != null) {
      //Define a família dos atributos de segurança
      ExtensibleFamily extFamily = new ExtensibleFamily((short) 0, (short) 1);
      //Define o tipo de atributo a ativar, Role neste caso
      AttributeType    attrType  = new AttributeType(extFamily, Role.value);
      //Define o valor do atributo, neste caso o papel passado como parâmetro
      SecAttribute[] reqstedPriv = {new SecAttribute(attrType, new byte [0],
                                                     umPapel.getBytes())};
      //Define um Holder que retornará a lista dos atributos efetivamnete ativados
      AttributeListHolder actualPriv = new AttributeListHolder();
      //Tenta ativar o papel chamando o método set_privileges
      //O parâmetro true indica que a mudança tem de ser imediata - FORCE_COMMIT = TRUE
      //Retorna um booleano que indica se o papel foi ativado ou não
      return creds.set_privileges(true, reqstedPriv, actualPriv);
    }
    return false;
  }
```

Para ativar um papel, é preciso definir o valor do atributo do tipo `Role` com o nome do papel a ativar e incluí-lo na lista atributos requisitados (`reqstedPriv`) no método acima. Depois o método `set_privileges` deve ser chamado. Retornando `true` indica que o papel foi ativado com sucesso.

Após a ativação do primeiro papel, a ativação dos demais é automática e ocorre de acordo com a necessidade de acesso aos recursos protegidos. Para testar a função acima, faça o seguinte:

1. Com o editor de texto de sua preferência, abra o arquivo **MacaCliente.java**, localize o trecho de código abaixo

```
for (int i = 0; i < roles.length; i++) {
            System.out.print(roles[i]+(i==(roles.length-1)?"\n":", "));
}
```

e adicione na linha seguinte o código

```
//Ativa o papel default
if (roles.length > 0) {
  String defaultRole = roles[0];
  System.out.println("Papel '"+defaultRole+
                     (ativaPapel(creds, defaultRole)?"' ":"' não ")+
                     "ativado!");
}
```

2. Salve o arquivo;
3. Compile o programa executando o seguinte comando:

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
javac -classpath ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar -d ./ MacaClien-
te.java
```

4. Execute o programa alterado com o comando abaixo:

```
java -cp ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar;./ tutorial.MacaCliente
```

5. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
employeeNumber: 1
Papéis: Vendedor Sênior, Gerente de Produto
Papel 'Vendedor Sênior ' ativado!
Sessão encerrada!
```

2.3.5 Obtendo os Papéis Ativos de um Usuário

Recuperar os papéis ativados para um usuário é bastante simples. O método abaixo, disponível no programa MacaCliente.java, faz isto.

```
static String[] pegaPapeisAtivos (Credentials creds) {
    String[] roles = {};
    //Verifica se a credencial foi criada
    if (creds != null) {
      //Define a família dos atributos de segurança
      ExtensibleFamily extFamily = new ExtensibleFamily((short) 0, (short) 1);
      //Define o tipo de atributo a recuperar, os papéis (Role) do usuário, neste caso
      AttributeType[]   attrType = {new AttributeType(extFamily, Role.value)};
      //Recupera o valor do tipo de atributo a partir da credencial
      //O primeiro elemento de 'attrs' tem a lista dos papéis ativos separados por ";".
      // É um string vazio quando não há papéis ativados
      //Os demais elementos são desprezados neste caso
      SecAttribute[]       attrs = creds.get_attributes(attrType);
      String activeRoles = "";
      //Examina a lista de valores em busca do tipo de atributo requisitado
      //Ler apenas o primeiro elemento do vetor porque se procura pelos papéis ativos
      for (int n = 0; n < attrs.length; n++) {
        if ((attrs[n].attribute_type.attribute_family.family_definer == 0) &&
            (attrs[n].attribute_type.attribute_family.family == 1) &&
            (attrs[n].attribute_type.attribute_type == Role.value)) {
          activeRoles = new String(attrs[n].value);
          break;
        }
      }
      //Separa a lista de papéis ativados e coloca o array 'roles'
      StringTokenizer strTk = new StringTokenizer(activeRoles, ";");
      roles = new String[strTk.countTokens()];
      for (int n = 0; strTk.hasMoreTokens(); n++) {
        roles[n] = strTk.nextToken();
      }
    }
    return roles;
  }
```

Pega o primeiro elemento da lista de atributos retornado, que armazena num *string* a lista de papéis ativos separados por ";". Depois, separa os papéis ativos e os coloca num array de *strings* que é retornado. Para testar a função acima, faça o seguinte:

1. Com o editor de texto de sua preferência, abra o arquivo **MacaCliente.java**, localize o trecho de código abaixo

```
System.out.println("Papel '"+defaultRole+
                           (ativaPapel(creds, defaultRole)?"' ":"' não ")+
                           "ativado!");
}
```

   e adicione na linha seguinte o código

```
//Recupera a lista de papéis ativados para o usuário
roles = pegaPapeisAtivos(creds);
System.out.print("Papéis ativos: ");
for (int i = 0; i < roles.length; i++) {
  System.out.print(roles[i]+(i==(roles.length-1)?"\n":", "));
}
```

2. Salve o arquivo;
3. Compile o programa executando o seguinte comando:

   javac -classpath ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar -d ./ MacaCliente.java

4. Execute o programa alterado com o comando abaixo:

   java -cp ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar;./ tutorial.MacaCliente

5. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
employeeNumber: 1
Papéis: Vendedor Sênior, Gerente de Produto
Papel 'Vendedor Sênior' ativado!
Papéis ativos: Vendedor Sênior
Sessão encerrada!
```

2.3.6 Alterando a Senha

Usuários podem alterar a própria senha através de sua credencial. Para isto, deverá fornecer a sua senha atual e o valor da nova senha. O método abaixo, disponível no programa MacaCliente.java, mostra o que é preciso.

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
static String mudaSenha(Credentials creds, String senhaAtual, String novaSenha) {
    //Indica que o tipo de atributo é uma senha
    final int PRIVILEGE_ATTRIBUTE_CREDENTIALS = 10;
    //Verifica se a credencial foi criada
    if (creds != null) {
      //Define a família dos atributos de segurança
      ExtensibleFamily extFamily = new ExtensibleFamily((short) 0, (short) 1);
      //Define o tipo de atributo a modificar, a senha neste caso
      AttributeType    attrType  =
                   new AttributeType(extFamily, PRIVILEGE_ATTRIBUTE_CREDENTIALS);
      //Define o valor do atributo, a senha atual e a nova senha separados pelo
      //caracter de 'nova linha' - LINE FEED
      SecAttribute[] reqstedPriv = {new SecAttribute(attrType, new byte [0],
                                          (senhaAtual+"\n"+novaSenha).getBytes())};
      //Define um Holder que retornará a lista com um atributo cujo valor é
      //uma mensagem com o motivo do insucesso da modificação
      AttributeListHolder actualPriv = new AttributeListHolder();
      //Tenta modificar a senha chamando o método set_privileges
      //O parâmetro true indica que a mudança tem de ser imediata - FORCE_COMMIT = TRUE
      //Retorna um booleano que indica se a senha foi modificada ou não
      boolean changed = creds.set_privileges(true, reqstedPriv, actualPriv);
      String  message = "";
      if (!changed && (actualPriv.value.length > 0)) {
        //Motivo para não trocar a senha
        message = new String(actualPriv.value[0].value);
      }
      return message;
    }
    return "credenciais inválidas";
  }
```

Primeiro, define que o tipo de atributo a modificar é a senha. O valor deste atributo é a senha atual e a nova senha, separados por pelo caracter de *line feed*. Para a modificação ser efetivada, é preciso que a senha atual seja válida e que a nova senha não seja vazia. O booleano retornado indicará se a operação foi bem sucedida ou não. O parâmetro `actualPriv` retorna um atributo cujo valor é uma mensagem indicando motivo do insucesso da mudança de senha, quando houver. Para testar a função acima, faça o seguinte:

1. Com o editor de texto de sua preferência, abra o arquivo **MacaCliente.java**, localize o trecho de código abaixo

```
System.out.print("Papéis ativos: ");
for (int i = 0; i < roles.length; i++) {
  System.out.print(roles[i]+(i==(roles.length-1)?"\n":", "));
}
```

   e adicione na linha seguinte o código

```
//Muda a senha do usuário
String msg = mudaSenha(creds, "caca", "bola");
System.out.println("Senha"+
                   (msg.equalsIgnoreCase("")?" ":" não ")+
                   "modificada: "+msg);
```

2. Salve o arquivo;
3. Compile o programa executando o seguinte comando:

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
javac -classpath ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar -d ./ MacaClien-
te.java
```

4. Execute o programa alterado com o comando abaixo:

```
java -cp ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar;./ tutorial.MacaCliente
```

5. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
employeeNumber: 1
Papéis: Vendedor Sênior, Gerente de Produto
Papel 'Vendedor Sênior' ativado!
Papéis ativos: Vendedor Sênior
Senha modificada!
Sessão encerrada!
```

Note que agora, se você executar novamente o programa, ele não autenticará, pois a senha foi modificada. Você terá de modificar o código para incluir a nova senha.

2.4 Utilizando o Serviço de Autorização de Acesso

Para obter autorizações de acesso, é preciso se conectar com o serviço correspondente no MACA CS através de um objeto cliente do tipo `AccessDecision`. A chamada do método `conectaAccessDecisionObject` que está definido no programa MacaCliente.java faz isto. Entretanto, é preciso modificá-lo para alterar no nome do servidor onde o MACA CS está executando. Então, faça o seguinte:

1. Com o editor de texto de sua preferência, abra o arquivo **MacaCliente.java**
2. Agora, localize o trecho de código abaixo

```
System.out.println("Senha"+
                   (msg.equalsIgnoreCase("")?" ":" não ")+
                   "modificada: "+msg);
```

e adicione na linha seguinte o código para se conectar com ado – *Access Decision Object*

```
//Conecta-se com o objeto CORBA servidor de autorização de acesso "ado"
AccessDecision ado;
ado = conectaAccessDecisionObject(rootConext);
```

3. Salve o arquivo;

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

2.4.1 Obtendo uma Autorização de Acesso Simples

Uma autorização de acesso simples é obtida com a chamada do método `access_allowed`, passando-se como parâmetros o nome do recurso que se pretende acessar, o nome da operação e os atributos de segurança do usuário. O método abaixo mostra como fazer isto.

```
static boolean acessoAutorizado (Credentials creds, AccessDecision ado,
                                 String nomeRecurso, String operacao) {
    //Verifica se a credencial e 'ado' foram criados
    if ((creds != null) && (ado != null)) {
      //Define a família dos atributos de segurança
      ExtensibleFamily extFamily = new ExtensibleFamily((short) 0, (short) 1);
      //Define os tipos de atributos a recuperar, Role e AccessId neste caso
      AttributeType[] attrTypes = {new AttributeType(extFamily, Role.value),
                                   new AttributeType(extFamily, AccessId.value)};
      //Recupera os atributos de segurança das credenciais do usuário
      SecAttribute[] attrs = creds.get_attributes(attrTypes);
      //Define o componente com o nome do recurso e a sua categoria - 'Aplicacoes'
      ResourceNameComponent[] resNameCompts =
                             {new ResourceNameComponent("Aplicacoes", nomeRecurso)};
      //Define o nome do recurso para o 'ado', composto pelo nome do servidor LDAP e
      //pela lista dos componentes do nome
      ResourceName resName = new ResourceName("127.0.0.1", resNameCompts);
      try {
        //Invoca o método de solicitaçao de autorizaçao de acesso
        //Passa como parâmetros o nome do recurso, a operação a realizar e os atributos
        //de segurança do usuário. Retorna um booleano indicando se o acesso está
        //autorizado ou não
        return ado.access_allowed(resName, operacao, attrs);
      }
      //O método 'access_allowed' pode levantar esta exceçao quando da
      //ocorrência de erros
      catch (org.omg.DfResourceAccessDecision.InternalError exc) {
        System.out.println(exc);
      }
    }
    return false;
  }
```

Para testar a função acima, faça o seguinte:

1. Com o editor de texto de sua preferência, abra o arquivo **MacaCliente.java**, localize o trecho de código abaixo

```
//Conecta-se com o objeto CORBA servidor de autorização de acesso "ado"
AccessDecision ado;
ado = conectaAccessDecisionObject(rootConext);
```

   e adicione na linha seguinte o código

```
//Verifica se o usuário tem acesso ao recurso 'SGP'
if (acessoAutorizado(creds, ado, "SGP", "execução")) {
  System.out.println("Acesso autorizado: SGP - [execução]");
  //Verifica se o usuário tem acesso ao recurso 'ConcederDesconto;Itens;Pedidos;SGP'
  //Mostra como passar parâmetros para uma autorização parametrizada
  if (acessoAutorizado(creds, ado,
                       "ConcederDesconto;Itens;Pedidos;SGP(35)", "execução")) {
    System.out.println("Acesso autorizado: ConcederDesconto;Itens;Pedidos;SGP – "+
                                                                    "[execução]");
  }
  else {
    System.out.println("Acesso NÃO autorizado: ConcederDesconto;Itens;Pedidos;SGP - "+
                                                                    "[execução]");
  }
}
else {
  System.out.println("Acesso NÃO autorizado: SGP - [execução]");
}
```

2. Salve o arquivo;
3. Compile o programa executando o seguinte comando:

   javac -classpath ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar -d ./ MacaClien-te.java

4. Execute o programa alterado com o comando abaixo:

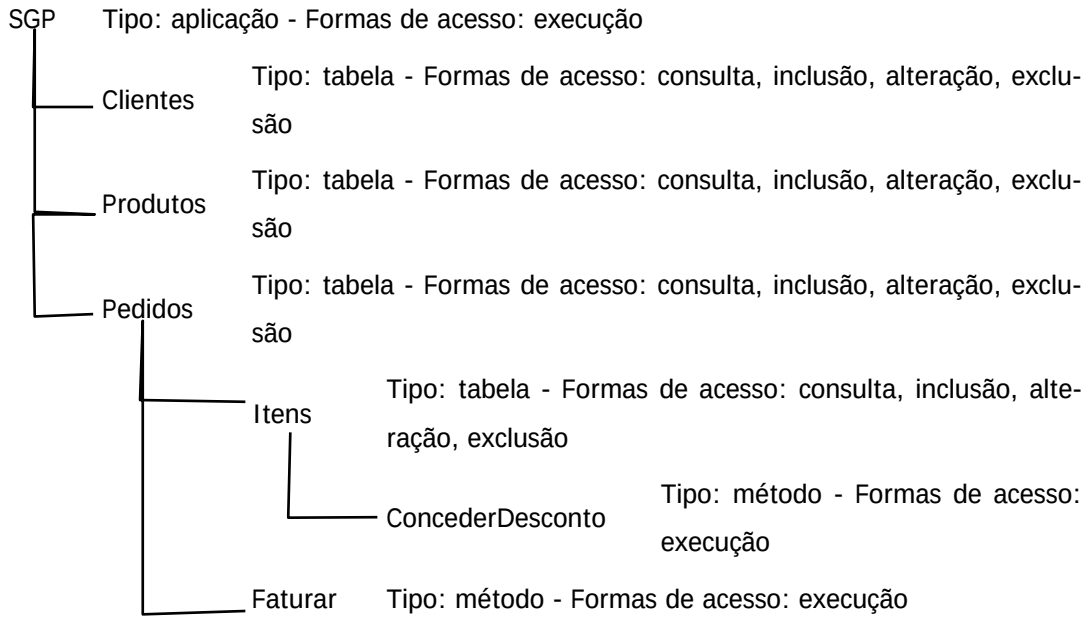
   java -cp ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar;./ tutorial.MacaCliente

5. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
employeeNumber: 1
Papéis: Vendedor Sênior, Gerente de Produto
Papel 'Vendedor Sênior' ativado!
Papéis ativos: Vendedor Sênior
Senha não modificada!
Acesso autorizado: SGP - [execução]
Acesso autorizado: ConcederDesconto;Itens;Pedidos;SGP - [exec...
Sessão encerrada!
```

Dois pontos merecem ser destacados neste exemplo: o nome dos recursos e a forma de passagem de parâmetros. Sabemos que no servidor LDAP, os nomes dos recursos são armazenados na forma de árvore, conforme ilustrado a seguir, na representação dos recursos da aplicação SGP.

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP



Numa solicitação de autorização de acesso, o nome completo do recurso é formado pela concatenação do nome do nó de mais baixo nível, com os nomes dos nós ancestrais até o nó raiz. Estes nomes são separados por ";". Por exemplo, para obter uma autorização de acesso ao recurso "Faturar", deve-se usar o seu nome completo, que o identifica unicamente no servidor LDAP:

- Faturar;Pedidos;SGP

O MACA CS e o protocolo LDAP não diferencia letras maiúsculas ou minúsculas. Caso haja autorizações parametrizadas associadas ao recurso, os parâmetros devem, necessariamente, ser passados juntamente com o nome do recurso. O recurso "Faturar" deve receber três parâmetros: o código do pedido a faturar – inteiro –, o código do cliente – inteiro – e o nome do vendedor - *string*. Logo, o nome do recurso e respectivos parâmetros tomam a seguinte forma:

- Faturar;Pedidos;SGP(<código pedido>, <código cliente>, "<nome vendedor>")

Estes parâmetros poderão ser usados em regras de autorização associadas ao recurso, por exemplo:

```
exp-abs(umPedido, umCliente, umVendedor)  {
  //Teste se quem vendeu não é quem está faturando o pedido
   umVendedor != userCtx.login &
  // Testa se o dia de hoje é um dia de semana e
  dtCtx.dia_semana in dtCtx.dia_de_semana &
  // se hora atual está no intervalo das 8 às 18 horas.
  (dtCtx.hora >= 8 & dtCtx.hora <= 18)
}
```

2.4.2 Obtendo Múltiplas Autorizações de Acesso

Autorizações de acesso múltiplas são obtidas com chamando do método `multiple_access_allowed`, passando-se como parâmetros uma lista de nomes de recursos com respectivas operações e os atributos de segurança do usuário. O método abaixo mostra como fazer isto.

```
static boolean[] autorizaMultiplosAcessos (Credentials creds, AccessDecision ado,
                                           String[] nomeRecursos, String[] operacoes) {
  //Verifica se a credencial e 'ado' foram criados
  if ((creds != null) && (ado != null) && (nomeRecursos.length == operacoes.length)) {
    try {
      //Define a família dos atributos de segurança
      ExtensibleFamily extFamily = new ExtensibleFamily((short) 0, (short) 1);
      //Define os tipos de atributos a recuperar, Role e AccessId neste caso
      AttributeType[] attrTypes = {new AttributeType(extFamily, Role.value),
                                   new AttributeType(extFamily, AccessId.value)};
      //Recupera os atributos de segurança das credenciais do usuário
      SecAttribute[] attrs      = creds.get_attributes(attrTypes);
      //Define a lista de requisições de autorização de acesso
      AccessDefinition[] access_requests = new AccessDefinition[nomeRecursos.length];
      //Para cada recurso, monta o componente com o nome do recurso e operação, e o
      //inclui na lista 'access_requests'
      for (int i = 0; i < access_requests.length; i++) {
        ResourceNameComponent[] resNameCompts =
                              {new ResourceNameComponent("Aplicacoes", nomeRecursos[i])};
        ResourceName resName = new ResourceName("127.0.0.1", resNameCompts);
        access_requests[i]   = new AccessDefinition(resName, operacoes[i]);
      };
      //Invoca o método que retorna um array de booleanos, cada booleano correspondendo
      // ao resultado da solicitaçao de acesso ao recurso correspondente a mesma posição
      //do array
      return ado.multiple_access_allowed(access_requests, attrs);
    }
    catch (org.omg.DfResourceAccessDecision.InternalError exc) {
      System.out.println(exc);
    }
  }
  return new boolean[0];
}
```

Neste caso, em vez de passar um único nome de recurso e operação, passamos uma lista de nomes de recursos e operações – `access_requests` – e os atributos do usuário. O retorno é uma lista de booleanos, indicando o resultado de cada solicitação de acesso nas posições correspondentes a cada nome de recurso. Para testar a função acima, faça o seguinte:

6. Com o editor de texto de sua preferência, abra o arquivo **MacaCliente.java**, localize o trecho de código abaixo

```
else {
  System.out.println("Acesso NÃO autorizado: SGP - [execução]");
}
```

e adicione na linha seguinte o código

```
//Usa método que faz múltiplas solicitaçoes de acesso
//Define array com dois nomes de recursos
String[]  recursos = {"SGP", "Pedidos;SGP(1, 1, \"gustavo.motta\")"};
//Define array com duas formas de acesso ou operações, uma para cada recurso anterior
String[] operacoes = {"execução", "alteração"};
boolean[]   result = autorizaMultiplosAcessos(creds, ado, recursos, operacoes);
for (int n = 0; n < recursos.length; n++) {
  System.out.println("Acesso"+(result[n]?"":" NÃO")+" autorizado: "+recursos[n]+
                                                          " - ["+operacoes[n]+"]");
}
```

7. Salve o arquivo;

8. Compile o programa executando o seguinte comando:

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
javac -classpath ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar -d ./ MacaClien-
te.java
```

9. Execute o programa alterado com o comando abaixo:

```
java -cp ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar;./ tutorial.MacaCliente
```

10. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
employeeNumber: 1
Papéis: Vendedor Sênior, Gerente de Produto
Papel 'Vendedor Sênior' ativado!
Papéis ativos: Vendedor Sênior
Senha não modificada!
Acesso autorizado: SGP - [execução]
Acesso autorizado: ConcederDesconto;Itens;Pedidos;SGP(35) – [exe
Acesso autorizado: SGP - [execução]
Acesso autorizado: Pedidos;SGP(1, 1, "gustavo.motta") - [alteraç
Sessão encerrada!
```

2.5 Verificando a Inatividade ou Queda do Cliente com *Callback*

Normalmente, a sessão do usuário do MACA CS é encerrada com a chamada, pelo cliente, do método `destroy` existente nas credenciais. Entretanto, em casos anormais (e. g., a aplicação cliente aborta, a comunicação cai, o usuário abandona a aplicação), corre-se o risco da sessão do usuário ocupar recursos e dar chances a vulnerabilidades no servidor por um tempo indefinido. O MACA CS ataca este problema de duas formas: através do *timeout* por inatividade do cliente e por meio de um mecanismo de *callback*. No caso do *timeout*, o MACA CS verifica, passivamente, para cada sessão aberta, a intervalos regulares, se houve atividade por parte do cliente. Não havendo atividade, a sessão é encerrada pelo MACA CS, a despeito do cliente. Deste modo, fica assegurado, num tempo finito, o encerramento de todas as sessões de clientes inativos ou incomunicáveis. Entretanto, apenas esta solução é indesejável em casos de servidores intensivamente usados, onde o tempo para *timeout* pode ser grande o suficiente para que se acumule um número excessivo de sessões que não vão mas ser usadas porque os clientes estão inativos ou incomunicáveis.

Com a solução de *callback*, o MACA CS verifica cada sessão aberta, ativamente, tentando se comunicar com o cliente a intervalos regulares para saber se o mesmo se encontra em atividade e comunicável. Ou seja, cada cliente atua como um servidor para o MACA CS. Havendo impossibilidade

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

de comunicação entre o MACA CS e o cliente, conclui-se que o cliente está inacessível e a sessão correspondente é destruída no MACA CS antes mesmo da ocorrência do *timeout*. Diferente do *timeout*, a solução de *callback* é opcional e requer a participação do cliente. Para sua utilização, o cliente deve atender os seguintes requisitos:

- Implementar a interface `Callback` que contém um único método `isClienteAlive`, que é invocado pelo servidor. Abaixo a ILD CORBA da interface é apresentada:

```
interface Callback {
    oneway void isClientAlive (in string message);
};
```

- O método é definido como *oneway* para evitar que o cliente bloqueie a *thread* do servidor MACA CS que faz os *callbacks* nos clientes. A classe `CallbackImpl` no programa **MacaCliente.java** é ilustrada a seguir e mostra como a interface pode ser implementada. A interface `Callback` está disponível no pacote `maca_cs` importado pelo programa.

```
//Classe que implementa o método de callback "isClientAlive"
//no cliente. Este é o único método existente na interface
//"Callback" e deve ser implementado com a extensão da
//classe "CallbackPOA"
static class CallbackImpl extends CallbackPOA {
  CallbackImpl() {
  }

  //Método de callback implementado pelo cliente e
  //invocado pelo servidor MACA CS. Embora seja chamado
  //com modo "oneway", sua execução deve ser a mais breve
  //possível
  public void isClientAlive (String message) {
    System.out.println(message);
  }
}
```

- Ativar uma instância da classe que implementa a interface (`CallbackImpl`) como um objeto servidor CORBA;
- Configurar no servidor MACA CS, para a sessão corrente (credenciais), o objeto para *callback* no cliente com uma chamada ao método `setClientCallback` da interface `CallbackCredentials`, subtipo da interface `Credentials`.

Para testar um exemplo de *callback*, faça o seguinte:

11. Com o editor de texto de sua preferência, abra o arquivo **MacaCliente.java**, localize o trecho de código abaixo

```
for (int n = 0; n < recursos.length; n++) {
  System.out.println("Acesso"+(result[n]?"":" NÃO")+" autorizado: "+recursos[n]+
                                                          " - ["+operacoes[n]+"]");
}
```

e adicione na linha seguinte o código

```
//Exemplo de uso do callback

//Primeiro, é preciso recuperar o acesso ao "RootPOA" do CORBA,
//que vai habilitar a ativação do objeto de callback como um
//servidor no cliente
POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
//Depois, cria-se uma instância da interface "Callback", ativando-a
//através do POA. Assim, esta instância passa a atuar como um
//objeto servidor no cliente apto a receber requisições
Callback callback = CallbackHelper.narrow(rootPOA.servant_to_reference(new
                                                                CallbackImpl()));
rootPOA.the_POAManager().activate();

//Como a interface "Credentials" do CORBAsec não oferece facilidades
//para definição no servidor do MACA CS de um objeto de callback, foi
//definida a interface "CallbackCredentials", que estende a interface
//"Credentials" com a inclusão do método "setClientCallback". Neste
//caso, é preciso fazer o "type casting" do objeto creds com tipo
//"Credentials" para o subtipo "CallbackCredentials" para se ter
//acesso ao método "setClientCallback".
CallbackCredentials cbkCreds = CallbackCredentialsHelper.narrow(creds);
//Define o objeto de callback do cliente
//no servidor do MACA CS
cbkCreds.setClientCallback(callback);

System.out.println("Espera pelo callback do servidor MACA CS ...");
System.out.println("    Tecle <enter> para encerrar sem testar ou ...");
System.out.println("    aguarde pela mensagem de callback ou ...");
System.out.println("    aborte o programa para verificar se o callback funciona");
//Espera pelo callback
System.in.read();
```

12. Salve o arquivo;

13. Compile o programa executando o seguinte comando:

    javac -classpath ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar -d ./ MacaClien-te.java

14. Execute o programa alterado com o comando abaixo:

    java -cp ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar;./ tutorial.MacaCliente

15. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
employeeNumber: 1
Papéis: Vendedor Sênior, Gerente de Produto
Papel 'Vendedor Sênior' ativado!
Papéis ativos: Vendedor Sênior
Senha não modificada!
Acesso autorizado: SGP - [execução]
Acesso autorizado: ConcederDesconto;Itens;Pedidos;SGP(35) – [exe
Acesso autorizado: SGP - [execução]
```


www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
Acesso autorizado: Pedidos;SGP(1, 1, "gustavo.motta") - [alteraç
Espera pelo callback do servidor MACA CS ...
    Tecle <enter> para encerrar sem testar ou ...
    aguarde pela mensagem de callback ou ...
    aborte o programa para verificar se o callback funciona
```

16. Caso você aguarde, após o primeiro intervalo de verificação de *callback* a seguinte linha será impressa:

```
Cliente CAIO está vivo?
```

17. Caso você tecle <enter>, a sessão é encerrada normalmente e a chamada periódica de *callbacks* correspondente a sessão é removida no servidor. A linha abaixo será impressa:

```
Sessão encerrada!
```

18. Se o programa for abortado, pode-se verificar no arquivo de log do servidor MACA CS o efeito do *callback*. Neste caso, ele falha e a sessão (credenciais) correspondente é encerrada.

2.6 Configurando IIOP sobre TLS/SSL

Clientes podem ser configurados para permitir ou obrigar que a conexão com o servidor MACA CS seja através do protocolo IIOP sobre TLS/SSL. Adicionalmente, o MACA CS pode ser configurando para permitir a autenticação do cliente, através de um certificado digital, junto ao servidor MACA CS.

Para configurar o cliente, siga os seguintes passos:

1. Mude-se para o diretório onde o cliente está instalado;
2. Caso o certificado do servidor MACA CS seja auto assinado ou emitido por uma AC não confiável, é necessário criar um *keystore* para armazenar tais certificados. Caso as instruções da seção *2.2.3 Configurando o MACA CS para Comunicação com os Clientes via IIOP sobre TLS/SSL* (ver MACA – Guia de Instalação e Configuração) tenham sido seguidas, obtenha com o responsável o certificado auto assinado criado disponível em `/usr/local/maca_cs/bin/certificados/maca_cs.cer` e o copie para o diretório corrente;
3. Agora execute o comando a seguir e entre com os dados solicitados conforme o modelo para importar o certificado:

```
$JAVA_HOME/bin/keytool -import -trustcacerts -alias maca_cs -file ./maca_cs.cer -keystore ./ca_keystore↵
Enter keystore password: <entre com uma senha para o keystore>↵
Owner: EMAILADDRESS=gustavo.motta@incor.usp.br, CN=pc440.incor.usp.br, OU=SPD, O=InCor, L=Sao Paulo, ST=SP, C=BR
Issuer: EMAILADDRESS=gustavo.motta@incor.usp.br, CN=pc440.incor.usp.br, OU=SPD,O=InCor, L=Sao Paulo, ST=SP, C=BR
Serial number: 0
Valid from: Wed May 28 10:52:33 BRT 2003 until: Thu May 27 10:52:33 BRT 2004
```


www.**incor**.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
Certificate fingerprints:
         MD5:  D6:FC:AF:CC:E0:F7:50:89:BA:A4:C4:85:6A:0C:C9:6A
         SHA1: 40:67:07:E1:72:4B:60:78:A5:37:04:0F:57:10:4E:72:66:EB:45:36
Trust this certificate? [no]:  yes↵
Certificate was added to keystore
```

No Windows NT/2000, entre

```
"%JAVA_HOME%/bin/keytool" -import -trustcacerts -alias maca_cs -file ./maca_cs.cer -keystore ./ca_keystore↵
Enter keystore password: <entre com uma senha para o keystore>↵
Owner: EMAILADDRESS=gustavo.motta@incor.usp.br, CN=pc440.incor.usp.br, OU=SPD, O=InCor, L=Sao Paulo, ST=SP, C=BR
Issuer: EMAILADDRESS=gustavo.motta@incor.usp.br, CN=pc440.incor.usp.br, OU=SPD,O=InCor, L=Sao Paulo, ST=SP, C=BR
Serial number: 0
Valid from: Wed May 28 10:52:33 BRT 2003 until: Thu May 27 10:52:33 BRT 2004
Certificate fingerprints:
         MD5:  D6:FC:AF:CC:E0:F7:50:89:BA:A4:C4:85:6A:0C:C9:6A
         SHA1: 40:67:07:E1:72:4B:60:78:A5:37:04:0F:57:10:4E:72:66:EB:45:36
Trust this certificate? [no]:  yes↵
Certificate was added to keystore
```

Note que o arquivo `ca_keystore` foi criado ou atualizado e armazena o certificado importado `maca_cs.cer`.

4. Edite o arquivo `orb.properties` e acrescente as seguintes propriedades:

```
#####################################
# Configuracoes para IIOP sobre SSL #
#####################################

jacorb.security.support_ssl=on

jacorb.security.ssl.client.supported_options=20
jacorb.security.ssl.client.required_options=0

jacorb.ssl.socket_factory=org.jacorb.security.ssl.sun_jsse.SSLSocketFactory
jacorb.ssl.server_socket_factory=org.jacorb.security.ssl.sun_jsse.SSLServerSocketFactory

javax.net.ssl.trustStore=./ca_keystore
javax.net.ssl.trustStorePassword=<senha do keystore>
```

Estas propriedades configuram o cliente para aceitar conexões IIOP sobre TLS/SSL, mas não o obrigam a realizar. Isto é, se o servidor MACA CS estiver configurado para obrigar conexões IIOP sobre TLS/SSL, então o cliente as realizará. Por outro lado, caso o MACA CS não obrigue tais conexões, o cliente fará uma conexão IIOP convencional. Para obrigar o cliente a realizar somente conexões IIOP sobre TLS/SSL, então altere o valor da propriedade `jacorb.security.ssl.client.required_options` para

```
jacorb.security.ssl.client.required_options=20
```

Esta configuração, entretanto, não habilita a autenticação do cliente junto ao MACA CS através de certificados digitais (ver item 8 a seguir para saber como configurar);

5. Salve o arquivo;
6. Compile o programa executando o seguinte comando:

   `javac -classpath ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar -d ./ MacaCliente.java`

7. Execute o programa alterado com o comando abaixo:

www.**incor**.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
java -cp ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar;./ tutorial.MacaCliente
```

8. Para habilitar a autenticação do cliente, é preciso criar para ele um certificado digital, que poderá ser auto assinado, ou emitido por uma AC confiável para o MACA CS. Para criar um certificado auto assinado, execute o comando abaixo entrando os valores solicitados de acordo com o modelo apresentado:

```
$JAVA_HOME/bin/keytool -genkey -alias cliente -keyalg RSA -validity 365 -keystore ./cliente_keystore↵
Enter keystore password: <entre com a senha da chave privada> ↵
Enter keystore password: <repita a senha anterior> ↵
What is your first and last name?
   [Unknown]: <entre com o DNS completo da estação do cliente> ↵
What is the name of your organizational unit?
   [Unknown]: <entre com o nome do setor onde você trabalha> ↵
What is the name of your organization?
   [Unknown]: <entre com o nome da empresa onde você trabalha> ↵
What is the name of your City or Locality?
   [Unknown]: <entre com o nome da cidade onde você trabalha> ↵
What is the name of your State or Province?
   [Unknown]: <entre com a sigla do estado onde você trabalha> ↵
What is the two-letter country code for this unit?
   [Unknown]: BR ↵
Is CN=organizacao.com.br, OU=setor, O=organizacao, L=cidade, ST=estado, C=BR correct?
   [no]: yes ↵
Enter key password for <cliente>
(RETURN if same as keystore password): ↵
```

No Windows NT/2000, faça assim:

```
"%JAVA_HOME%/bin/keytool" -genkey -alias cliente -keyalg RSA -validity 365 -keystore ./cliente_keystore↵
Enter keystore password: <entre com a senha da chave privada> ↵
Enter keystore password: <repita a senha anterior> ↵
What is your first and last name?
   [Unknown]: <entre com o DNS completo da estação do cliente> ↵
What is the name of your organizational unit?
   [Unknown]: <entre com o nome do setor onde você trabalha> ↵
What is the name of your organization?
   [Unknown]: <entre com o nome da empresa onde você trabalha> ↵
What is the name of your City or Locality?
   [Unknown]: <entre com o nome da cidade onde você trabalha> ↵
What is the name of your State or Province?
   [Unknown]: <entre com a sigla do estado onde você trabalha> ↵
What is the two-letter country code for this unit?
   [Unknown]: BR ↵
Is CN=organizacao.com.br, OU=setor, O=organizacao, L=cidade, ST=estado, C=BR correct?
   [no]: yes ↵
Enter key password for <cliente>
(RETURN if same as keystore password): ↵
```

Isto cria um certificado auto assinado e sua respectiva chave privada no arquivo `cliente_keystore`. Para requisição de um certificado para emissão por uma AC confiável, veja os detalhes em http://java.sun.com/j2se/1.4.1/docs/tooldocs/solaris/keytool.html.

O arquivo `cliente_keystore` é um tipo de banco de dados que armazena a chave privada e o certificado, que é público. É necessário exportar este certificado para fornecê-los a todos os servidores MACA CS que desejam aceitar conexões deste cliente através do IIOP sobre TLS/SSL. Para exportá-lo, execute o seguinte comando:

```
$JAVA_HOME/bin/keytool -export -alias cliente -file cliente.cer
-keystore ./cliente_keystore
```


www.**incor**.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

No Windows NT/2000, faça assim:

```
"%JAVA_HOME%/bin/keytool" -export -alias cliente -file cliente.cer
-keystore ./cliente_keystore
```

Após a execução deste comando, o arquivo `cliente.cer` armazenará o certificado a ser fornecido para os servidores do MACA CS com os quais o cliente deseja autenticar;

9. Com o certificado do cliente criado, agora edite o arquivo `orb.properties` e acrescente as seguintes propriedades:

```
jacorb.security.keystore=./cliente_keystore
jacorb.security.keystore_password=<senha da chave privada>
```

Depois altere o valor da propriedade jacorb.security.ssl.client.supported_options para

```
jacorb.security.ssl.client.supported_options=40
```

Grave o arquivo.

Agora o cliente está apto a se autenticar junto ao serviço MACA CS. ATENÇÃO, não esquecer que o MACA CS deve ter sido configurado para exigir a autenticação de clientes e que o certificado deste cliente deve ser um certificado confiável para o MACA CS.

10. Compile o programa executando o seguinte comando:

```
javac -classpath ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar -d ./ MacaClien-
te.java
```

11. Execute o programa alterado com o comando abaixo:

```
java -cp ./maca_jacorb_stubs-3.2.2.jar;./jacorb.jar;./ tutorial.MacaCliente
```


www.**incor**.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

3 Tutorial – Usando o MACA CS em DELPHI

Este tutorial mostra como utilizar os serviços de autenticação e autorização de acesso do MACA CS em programas cliente DELPHI.

3.1 Requisitos

Antes de iniciar este tutorial, baixe o pacote de APIs para clientes da versão escolhida do MACA no *site* http://maca.sourceforge.net/index.html#Downloads e o descompacte com o seguinte comando no LINUX:

```
gunzip -c maca_clientes_VERSÃO.tgz | tar xf -
```

Caso esteja no MS-Windows, utilize o utilitário WinRar, disponível no *site* http://www.rarlab.com/, para descompactá-lo. Em seguida, verifique a lista abaixo:

- Ambiente "Borland DELPHI 6" instalado para desenvolvimento do cliente (com opção do Visibroker 4.1);
- Servidor MACA CS instalado e iniciado (remota ou localmente);
- Acesso aos seguintes arquivos fornecidos com o diretório de instalação do MACA
  - \maca_cliente_VERSÃO\idls\delphi_stubs\dcus*.dcu
  - \ maca_cliente_VERSÃO\maca_cliente_delphi\MacaCliente.drp
- A distribuição de clientes CORBA em Delphi necessita das seguintes DLLs, disponíveis no diretório **\maca_cliente_VERSÃO\maca_cliente_delphi\corba_dlls**, do diretório de instalação do MACA:
  - orb_br.dll
  - OrbPas41.dll
  - orbpas60.dll
  - cp3245mt.dll
  - borlndmm.dll
  - vport_br.dll
  - cc3250mt.dll

3.2 Utilizando o Serviço de Autenticação

O programa "MacaCliente.dpr" apresenta um exemplo bem simples de como efetuar uma autenticação. Para ver como funciona, faça o seguinte:

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

1. Na sua estação de trabalho, crie um subdiretório denominado **maca_cliente_delphi** e copie o arquivo **MacaCliente.dpr** nele;
2. Abaixo do diretório **maca_cliente_delphi** crie um subdiretório denominado **lib** e copie todos os arquivos do subdiretório \maca_cliente_VERSÃO\idls\delphi_stubs\dcus\ do diretório de instalação do MACA;
3. No ambiente do DELPHI, abra o projeto **MacaCliente.dpr**. Selecione a opção *Project* | *Options*. Selecione a pasta *Directories/Conditionals* e adicione o caminho "...**maca_cliente_delphi\lib**" nos campos *Unit output directory* e *Search path*. Pressione o botão "OK";
4. Agora, no arquivo **MacaCliente.dpr**, localize o trecho de código abaixo

```
readIORs('http://127.0.0.1', paIOR, adoIOR);
```

no início do programa principal e troque o *string* `"127.0.0.1"` pelo *string* contendo o nome do servidor onde o servidor MACA CS está ativado, por exemplo, `"ldap.datasus.gov.br"`;

5. Agora, localize o trecho de código abaixo

```
//Define senha como um array de bytes
  senha := 'maca';
  SetLength(pwd, Length(senha));
  for i := 1 to Length(senha) do begin
    pwd[i-1] := Byte(senha[i]);
  end;

  //Faz a autenticação
  authStatus := pa.authenticate(CLEAR_TEXT_PWD, //Médoto de autenticação
                                'nome.sobrenome', //Nome de login do usuário
                                pwd,              //Senha
                                attrs,            //Atributos de segurança fornecidos
                                creds,            //Retorna a credencial criada para o
                                                  //usuário em caso de sucesso
                                contData,         //Valor de retorno não usado
                                authEspec);       //Valor de retorno não usado
```

e modifique o nome do *login* do usuário "`nome.sobrenome`" para o seu *login* e a senha "`maca`" para a sua senha;

6. Salve o arquivo e selecione a opção *Project* | *Build MacaCliente* do menu do DELPHI;
7. Para testá-lo, basta selecionar a opção *Run* | *Run* do menu do DELPHI ou pressionar a tecla **F9**;
8. Uma autenticação bem sucedida apresentará a saída abaixo:

```
Autenticação realizada com sucesso!
Sessão encerrada!
```


www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

3.2.1 Entendendo o programa "MacaCliente.dpr"

Para utilizar o serviço de autenticação em CORBA, a primeira tarefa a realizar é ler os IORs (*Interoperable Object Reference*) que referenciam os objetos no servidor MACA CS. Neste caso, o IOR do objeto servidor de autenticação é publicado via HTTP num arquivo denominado "maca_cs.iors", juntamente os IORs de outros objetos servidores. Após sua recuperação, o IOR é transformado num objeto CORBA cliente correspondente ao objeto CORBA servidor. Atenção, é imperativo que os IORs usados sejam obtidos da mesma chamada à função `readIORs`. Recomenda-se ler os IORs imediatamente antes de usá-los, para que não fiquem desatualizados quando o servidor MACA CS for reinicializado. A segunda tarefa é a inicialização do ORB do DELPHI, presente na *unit* `CORBA`, que deve ser incluído na cláusula *uses* no programa cliente. A variável correspondente ao ORB – `orb` – é única, isto é, é um *singleton*. Ao final, antes de sair do programa, o ORB deve ser finalizado. Caso se deseje utilizar o serviço de nomes do CORBA, basta passar como parâmetro do programa a propriedade `-ORBInitRef` para incluir a porta e o nome do servidor onde se localiza o serviço de nomes CORBA. O esta propriedade deve ter o seguinte padrão: `-ORBInitRef NameService=iioploc://host:port/NameService`. Para configurar esta propriedade no ambiente do Delphi, selecione a opção *Run* | *Parameters* do menu principal e entre com a propriedade na caixa de edição denominada *Parameters*. Note que esta propriedade deve corresponder às propriedades usadas pelo MACA CS, de modo que o mesmo serviço de nomes seja usado, tanto pelo servidor MACA CS, quanto pelo cliente. Os trechos de código correspondentes a esta etapa são mostrados em destaque abaixo:

```
program MacaCliente;
{$APPTYPE CONSOLE}

uses SysUtils, Classes, Corba, COSNaming, NMHttp, Psock, Dialogs, StrUtils,
     Security_c, Security_i,
     SecurityLevel2_c, SecurityLevel2_i,
     DfResourceAccessDecision_c, DfResourceAccessDecision_i
     maca_cs_c, maca_cs_i, maca_cs_s;

...

const
  CLEAR_TEXT_PWD = 0;

var
  extFamily    : ExtensibleFamily;
  pa           : PrincipalAuthenticator;
  ado          : AccessDecision;
  creds        : Credentials;
  authStatus   : AuthenticationStatus;
  senha        : String;
  pwd          : Opaque;
  attrs        : AttributeList;
  contData     : Opaque;
  authEspec    : Opaque;
  response_data: Opaque;
  i            : Integer;
  roles        : TStrings;
  defaultRole  : String;
```

```
  recursos,
  operacoes    : TStrings;
  results      : BooleanList;
  cbkSk        : TCallbackSkeleton;
  cbkCreds     : CallbackCredentials;
  msg          : String;
  paIOR,
  adoIOR       : String;
begin
  //Ler via HTTP os IORs do serviços de autenticação e de
  //controle de acesso junto ao MACA CS, isto é, no
  //servidor onde o MACA CS foi instalado.
  readIORs('http://127.0.0.1', paIOR, adoIOR);

  //Inicializa o ORB
  CorbaInitialize;
//Conecta-se com o serviço de nomes do CORBA - rootContext
  try
    rootContext := nil;
    rootContext :=
      TNamingContextExtHelper.Narrow(Orb.ResolveInitialReferences('NameService'), True);
  except
    on Exception do
      Writeln('Serviço de nomes não localizado. Verifique a propriedade de ' +
              'inicialização do ORB: -ORBInitRef');
  end;

...

  //Finaliza o ORB no cliente
  orb.shutdown;
  readln;
end.
```

Com o ORB inicializado, é preciso se conectar com o serviço de autenticação no MACA CS através de um objeto cliente do tipo `PrincipalAuthenticator`. A chamada da função `conectaPrincipalAuthenticator`

```
//Conecta-se com o objeto CORBA servidor de autenticão "pa"
  pa := conectaPrincipalAuthenticator(rootConext, paIOR);
```

logo após a inicialização do ORB faz esta conexão. Caso o serviço de nomes tenha sido localizado (`rootContext <> nil`), então recupera-se a referência ao objeto através do método `resolve_str`, passando-se como parâmetro o nome o objeto no serviço de nomes que implementa a interface `PrincipalAuthenticator`. Para o MACA CS, o nome completo deste objeto é "*maca_cs.ctx/authenticators.ctx/pa.server*". Caso o serviço de nomes não tenha sido localizado (`rootContext = nil`), então usa-se o IOR (*Interoperable Object Reference*) do objeto servidor passado como parâmetro (`paIOR`). No caso do MACA CS, o IOR do objeto servidor de autenticação é publicado via HTTP num arquivo denominado "pa.ior", disponível no mesmo servidor onde o MACA CS está executando. Após sua recuperação, o IOR é transformado num objeto CORBA cliente correspondente ao objeto CORBA servidor.

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

Para finalizar, é preciso converter, via "*type casting*", o tipo deste objeto cliente para o tipo `PrincipalAuthenticator`, implementado pelo servidor de autenticação. O código do método `conectaPrincipalAuthenticator` é mostrado abaixo com os respectivos destaques:

```
Function conectaPrincipalAuthenticator : PrincipalAuthenticator;
var
  corbaObj : CORBAObject;
  iorURL   : TNMHTTP;
begin
  if rootContext = nil then begin //Serviço de nomes indisponível. Usa o IOR obtido via
                                  //HTTP
    //Conecta-se com o objeto CORBA servidor do tipo PrincipalAuthenticator
    corbaObj := Orb.StringToObject(paIOR);
  end
  else begin //Conecta-se com o objeto CORBA servidor do tipo PrincipalAuthenticator via
             //serviço de nomes. O nome do objeto CORBA servidor do tipo
             //PrincipalAuthenticator é "maca_cs.ctx/authenticators.ctx/pa.server"
    corbaObj := rootContext.resolve_str('maca_cs.ctx/authenticators.ctx/pa.server');
  end;
  //Faz o "type casting" do objeto do tipo CORBAObject para o tipo
  //PrincipalAuthenticator
  result := TPrincipalAuthenticatorHelper.Narrow(corbaObj, true);
end;
```

Uma vez disponível, o objeto `pa` do tipo `PrincipalAuthenticator` pode ser usado para realizar a autenticação de usuário através da chamada do método `authenticate`, conforme indicado no trecho código abaixo:

```
//Definição dos parâmetros do método de autenticação do PrincipalAuthenticator
//A lista de atributos de segurança apenas para autenticação pode ser vazia
SetLength(attrs, 0);
//Parametros de retorno para continuação da autenticação - não usados
SetLength(contData, 0);
SetLength(authEspec, 0);
//Define senha como um array de bytes
senha := 'maca';
SetLength(pwd, Length(senha));
for i := 1 to Length(senha) do begin
  pwd[i-1] := Byte(senha[i]);
end;

//Faz a autenticação
authStatus := pa.authenticate(CLEAR_TEXT_PWD, //Médoto de autenticação
                              'nome.sobrenome', //Nome de login do usuário
                              pwd,              //Senha
                              attrs,            //Atributos de segurança fornecidos
                              creds,            //Retorna a credencial criada para o
                                                //usuário em caso de sucesso
                              contData,         //Valor de retorno não usado
                              authEspec);       //Valor de retorno não usado
```

No MACA CS, o método de autenticação atualmente disponível é baseado em senha simples, sendo indicado pelo valor da constante `CLEAR_TEXT_PWD` igual a 0. O argumento `attrs` pode ser usado para fornecer, ainda durante a autenticação, o nome de um papel inicial para ativar, por exemplo. Neste caso, nenhum atributo é fornecido. A ativação de papéis no MACA é totalmente automática. A chamada a `authenticate` retorna um objeto do tipo `AuthenticationStatus` que indica se a autenticação foi bem sucedida ou não. O código abaixo examina os valores possíveis deste retorno.

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
//Verifica o status da autenticação
case authStatus of
  SecAuthSuccess, SecAuthContinue : begin
    if authStatus = SecAuthContinue then begin
       //Este caso ocorre quando o usuário é obrigado a mudar a senha ou quando
       //a senha expira. Em ambos os casos, a troca imediata da senha é obrigatória
       //para completar o processo de autenticação. Neste caso, é preciso
       //recuperar credencial criada para esta sessão do usuário
      //Recupera mensagem sobre a autenticação
      SetString(msg, PChar(authEspec), Length(authEspec));
      Writeln(msg);
      //Nesta situação a credencial fica com sua funcionalidade restrita
      //enquanto a senha não é trocada. Após a troca, a credencial fica
      //completamente operacional.
      //Muda a senha do usuário
      msg := mudaSenha(creds, 'caca', 'bola');
      if msg <> '' then begin //A senha não foi trocada
        Writeln('Senha não modificada: '+msg);
        //sessão é encerrada
        creds._destroy;
        Readln;
        Exit;
      end;
      Writeln('Senha modificada');
      //Continua o processo de autenticação
      SetLength(response_data, 0);
      if pa.continue_authentication(response_data, creds,
                                    contData, authEspec) <> SecAuthSuccess then begin
        Writeln('Falha na continuaçao da autenticação!');
        //sessão é encerrada
        creds._destroy;
        Readln;
        Exit;
      end;
    end;
    Writeln('Autenticação realizada com sucesso!');

   //Encerra sessão do usuário destruindo a credencial
   creds._destroy;
   Writeln('Sessão encerrada!');
  end;
  SecAuthFailure : begin
    Writeln('Falha na autenticação!');
  end;
  SecAuthExpired : begin
    Writeln('Conta expirada!');
    //Recupera mensagem com informações sobre a expiração
    //da conta do usuário
    SetString(msg, PChar(authEspec), Length(authEspec));
    Writeln(msg);
 end;
end;
```

Caso a autenticação tenha sido bem sucedida, o argumento de saída `creds` armazena a referência de uma credencial válida do usuário. Enquanto existir, esta credencial corresponderá à sessão do usuário. A sessão é encerrada quando é destruída pela chamada do método `destroy`, ou com *timeout* por inatividade, ou através de um mecanismo opcional de *callback* do servidor para o cliente (ver subseção 3.5). Caso a autenticação não tenha sido bem sucedida, os motivos possíveis são os seguintes:

- Falha na autenticação (`authStatus = SecAuthFailure`): o login ou a senha fornecidos são inválidos;

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

- A autenticação deve ser continuada (`authStatus = SecAuthContinue`): este caso ocorre quando o usuário é obrigado a mudar a senha para completar a autenticação. Acontece quando o usuário realiza o primeiro login e a propriedade (`maca_ca.passwordPolicy.passwordMustChange`) está ativada, ou então, quando a senha do usuário expira (propriedade `maca_ca.passwordPolicy.passwordExp=true`) e aí, em ambas as situações, ele deve, necessariamente, trocar a senha para continuar a usar o sistema normalmente. Nesta situação, a credencial retornada tem sua funcionalidade restringida, somente permitindo a operação de troca de senha (ver detalhes na seção 2.3.6) e a operação para destruí-la. Qualquer solicitação de autorização será negada enquanto a senha não for trocada. Após a mudança bem sucedida da senha, o processo de autenticação continua com a chamada do método `continue_authentication`. A credencial deve ser destruída, com o encerramento da sessão do usuário caso a autenticação não seja concluída com sucesso.

Recomenda-se tratar todos estes casos no processo de autenticação. **Note que é importante recuperar a mensagem com dados sobre a expiração da senha para mostrá-la ao usuário, avisando a data da expiração da senha, orientando a trocá-la imediatamente**. Esta informação é retornada no parâmetro `authEspec` do método `authenticate`, conforme ilustrado no código anterior.

3.3 Utilizando a Interface "Credentials"

Através da interface "Credentials" é possível realizar as seguintes ações, dentre outras possíveis:

- Obter o nome do usuário associado à conexão;
- Obter a relação de papéis ativos e disponíveis para o usuário;
- Ativar papéis para o usuário;
- Modificar a senha do usuário;
- Verificar se a credencial está válida – método is_valid();
- Refrescar a credencial para ela não expirar por inatividade – método refresh();
- Destruir a credencial – método destroy().

3.3.1 Obtendo o *Login* do Usuário

Recuperar o nome de *login* do usuário é bastante simples. A função abaixo, disponível no programa MacaCliente.dpr, faz isto.

```
Function pegaLogin (creds : Credentials) : String;
var
  extFamily    : ExtensibleFamily;
  attrType     : AttributeType;
  attrTypeList : AttributeTypeList;
```


www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
  attrs        : AttributeList;
  n            : Integer;
  login        : String;
begin
  result := '';
  //Verifica se a credencial foi criada
  if (creds <> nil) then begin
    //Define a família dos atributos de segurança
    //O primeiro argmento indica quem é o definidor da família
    //Valor 0 significa que o OMG é  o definidor
    //O segundo argmento indica a família de atributos
    //Valor 1 significa que são atributos de privilégio
    extFamily := TExtensibleFamily.Create(0, 1);
    //Define o tipo de atributo a recuperar, o login (AccessId) do usuário, neste caso
    attrType  := TAttributeType.Create(extFamily, AccessId);
    SetLength(attrTypeList, 1);
    attrTypeList[0] := attrType;
    //Recupera o valor do tipo de atributo a partir da credencial
    attrs := creds.get_attributes(attrTypeList);
    //Examina a lista de valores em busca do tipo de atributo requisitado
    for n := 0 to Length(attrs) - 1 do begin
      if ((attrs[n].attribute_type.attribute_family.family_definer = 0) and
          (attrs[n].attribute_type.attribute_family.family = 1) and
          (attrs[n].attribute_type.attribute_type = AccessId)) then begin
        SetString(login, PChar(attrs[n].value), Length(attrs[n].value));
        result := login;
        break;
      end;
    end;
  end;
  //Libera memória dos arrays dinâmicos
  attrs        := nil;
  attrTypeList := nil;
end;
```

Primeiro, é preciso definir o tipo de atributo a recuperar, no caso a identificação de acesso do usuário, especificada pela constante `AccessId`. Os outros tipos possíveis são: `AccountingId`, `AuditId`, `GroupId`, `PrimaryGroupId`, `Role`, `Capability` e `Clearance`. Destes, o que vamos mais utilizar é o `Role`. Para testar a função acima, faça o seguinte:

1. No arquivo **MacaCliente.dpr**, localize o trecho de código abaixo

```
SecAuthSuccess : begin
    Writeln('Autenticação realizada com sucesso!');
```

e adicione na linha seguinte o código

```
//Imprime o login do usuário
Writeln('Login: '+pegaLogin(creds));
```

2. Salve o arquivo e selecione a opção *Project* | *Build MacaCliente* do menu do DELPHI;
3. Para testá-lo, selecione a opção *Run* | *Run* do menu do DELPHI ou pressione a tecla **F9**;
4. A seguinte saída será apresentada:

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
Autenticação realizada com sucesso!
Login: caio
Sessão encerrada!
```

3.3.2 Obtendo um Conjunto de Atributos da Conta do Usuário

Com as credenciais é possível recuperar um conjunto de atributos selecionado da conta do usuário. Para saber os possíveis tipos de atributos retornados, entre em contato com o administrador do MACA CS e solicite a lista de nomes de atributos presentes na propriedade "maca_cs.ldap.userAccountAttrList" da configuração do MACA CS. O método abaixo, disponível no programa MacaCliente.java, mostra como recuperar os atributos a partir das credenciais do usuário.

```
Function pegaAtributosUsuario (creds : Credentials) : TStrings;
var
  extFamily    : ExtensibleFamily;
  attrType     : AttributeType;
  attrTypeList : AttributeTypeList;
  attrs        : Security_i.AttributeList;
  n            : Integer;
  atributos    : TStrings;
  auxStr       : String;
begin
  atributos := TStringList.Create;
  //Verifica se a credencial foi criada
  if (creds <> nil) then begin
    //Define a família dos atributos de segurança
    extFamily := TExtensibleFamily.Create(0, 1);
    //Define o tipo de atributo a recuperar, o conjunto de atributos da conta do
    //usuário, neste caso
    attrType  := TAttributeType.Create(extFamily, AttributeSet);
    SetLength(attrTypeList, 1);
    attrTypeList[0] := attrType;
    //Recupera o valor do tipo de atributo a partir da credencial
    //Cada elemento da lista retorna um atributo, do tipo string, com o seguinte
    //formato:
    //<nome do atributo>: <valor_1 do atributo>, <valor_2 do atributo>, ..., <valor_n do
    //atributo>
    attrs := creds.get_attributes(attrTypeList);
    //Examina a lista de valores em busca do tipo de atributo requisitado
    for n := 0 to Length(attrs)-1 do begin
      if (attrs[n].attribute_type.attribute_family.family_definer = 0) and
         (attrs[n].attribute_type.attribute_family.family = 1) and
         (attrs[n].attribute_type.attribute_type = AttributeSet) then begin
        SetString(auxStr, PChar(attrs[n].value), Length(attrs[n].value));
        atributos.Add(auxStr);
      end;
    end;
  end;
  result       := atributos;
  //Libera memória dos arrays dinâmicos
  attrTypeList := nil;
  attrs        := nil;
end;
```

Note que este método é semelhante ao que recupera o *login*. A diferença é que o tipo do atributo a recuperar é `AttributeSet`, isto é, um conjunto de atributos selecionados para conta de usuários. Caso um atributo esperado (um daqueles listados na propriedade "maca_cs.ldap.userAccountAttrList" do MACA CS) não retorne, estão é porque ele não está armazenado na conta do usuário. A lista de

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

valores de atributos terá uma entrada por tipo de atributo. O formato de cada valor retornado é o seguinte:

- <nome do atributo>: <valor_1 do atributo>, <valor_2 do atributo>, ..., <valor_n do atributo>

Por exemplo, o valor retornando para o atributo `title`, com dois valores armazenados na conta de um usuário no servidor LDAP, poderia ser o seguinte:

- title: Analista, Gerente

Finalmente, para testar a função acima, faça o seguinte:

1. No arquivo **MacaCliente.dpr**, localize o trecho de código abaixo

```
//Imprime o login do usuário
Writeln('Login: '+pegaLogin(creds));
```

e adicione na linha seguinte o código

```
//Recupera o conjunto de atributos do usuário
atributos := pegaAtributosUsuario(creds);
Writeln('Atributos: ');
for i := 0 to (atributos.Count - 1) do begin
  Writeln(atributos[i]);
end;
```

2. Salve o arquivo e selecione a opção *Project* | *Build MacaCliente* do menu do DELPHI;
3. Para testá-lo, selecione a opção *Run* | *Run* do menu do DELPHI ou pressione a tecla **F9**;
4. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
employeeNumber: 1
Sessão encerrada!
```

3.3.3 Obtendo os Papéis Associados ao Usuário

Recuperar os papéis do usuário é bastante simples. A função abaixo, disponível no programa MacaCliente.dpr, faz isto.

```
Function pegaPapeis (creds : Credentials) : TStrings;
var
  extFamily    : ExtensibleFamily;
  attrType     : AttributeType;
  attrTypeList : AttributeTypeList;
  attrs        : Security_i.AttributeList;
  n            : Integer;
```


www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
  userRoles    : TStrings;
  auxStr       : String;
begin
  userRoles := TStringList.Create;
  //Verifica se a credencial foi criada
  if (creds <> nil) then begin
    //Define a família dos atributos de segurança
    extFamily := TExtensibleFamily.Create(0, 1);
    //Define o tipo de atributo a recuperar, os papéis (Role) do usuário, neste caso
    attrType  := TAttributeType.Create(extFamily, Role);
    SetLength(attrTypeList, 1);
    attrTypeList[0] := attrType;
    //Recupera o valor do tipo de atributo a partir da credencial
    //O primeiro elemento de attrs tem a lista dos papéis ativos separados por ";".
    //É um string vazio quando não há papéis ativados
    //Os demais elementos do tipo Role correspondem aos papéis associados ao usuário
    attrs := creds.get_attributes(attrTypeList);
    for n := 1 to Length(attrs)-1 do begin
      if (attrs[n].attribute_type.attribute_family.family_definer = 0) and
         (attrs[n].attribute_type.attribute_family.family = 1) and
         (attrs[n].attribute_type.attribute_type = Role) then begin
        SetString(auxStr, PChar(attrs[n].value), Length(attrs[n].value));
        userRoles.Add(auxStr);
      end;
    end;
  end;
  result       := userRoles;
  //Libera memória dos arrays dinâmicos
  attrTypeList := nil;
  attrs        := nil;
end;
```

Note que esta função é semelhante à que recupera o *login*. A diferença é que o tipo do atributo a recuperar é `Role`, isto é, os papéis do usuário. Na lista de valores de atributos retornados, o primeiro elemento é um *string* com a lista de papéis ativados para o usuário separados por ";". Como nenhum papel foi ativado, o valor atual é um *string* vazio. Os demais elementos da lista (posições >= 1) correspondem aos papéis associados que estamos interessados. Para testar a função acima, faça o seguinte:

1. No arquivo **MacaCliente.dpr**, localize o trecho de código abaixo

```
for i := 0 to (atributos.Count - 1) do begin
  Writeln(atributos[i]);
end;
```

e adicione na linha seguinte o código

```
//Recupera a lista de papéis do usuário
roles := pegaPapeis(creds);
Write('Papéis: ');
for i := 0 to (roles.Count - 1) do begin
  if ((roles.Count - 1) = i) then begin
    Writeln(roles[i]);
  end
  else begin
    Write(roles[i]+', ');
  end;
end;
```

2. Salve o arquivo e selecione a opção *Project* / *Build MacaCliente* do menu do DELPHI;

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

3. Para testá-lo, selecione a opção *Run* | *Run* do menu do DELPHI ou pressione a tecla **F9**;

4. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
employeeNumber: 1
Papéis: Vendedor Sênior, Gerente de Produto
Sessão encerrada!
```

3.3.4 Ativação de Papéis

Agora que sabemos como obter os papéis associados a um usuário, vamos ver como proceder para ativá-los. A função abaixo, disponível no programa MacaCliente.dpr, nos diz como.

```
Function ativaPapel(creds : Credentials; umPapel : String) : Boolean;
var
  extFamily    : ExtensibleFamily;
  attrType     : AttributeType;
  reqstedPriv  : SecAttribute;
  reqstedPrivList,
  attrsList    : Security_i.AttributeList;
  defAutho     : Opaque;
  newActiveRole: Opaque;
  i            : Integer;
begin
  result := false;
  //Verifica se a credencial foi criada
  if (creds <> nil) then begin
    //Define a família dos atributos de segurança
    extFamily   := TExtensibleFamily.Create(0, 1);
    //Define o tipo de atributo a ativar, Role neste caso
    attrType    := TAttributeType.Create(extFamily, Role);
    //Define o valor do atributo, neste caso o papel passado como parâmetro
    SetLength(newActiveRole, Length(umPapel));
    for i := 0 to Length(umPapel) - 1 do begin
      newActiveRole[i] := Byte(umPapel[i+1]);
    end;
    SetLength(defAutho, 0);
    //Cria a lista com os papéis (atributos) para ativar (solicitados)
    reqstedPriv := TSecAttribute.Create(attrType, defAutho, newActiveRole);
    SetLength(reqstedPrivList, 1);
    reqstedPrivList[0] := reqstedPriv;
    //Tenta ativar o papel chamando o método set_privileges
    //O parâmetro true indica que a mudança tem de ser imediata - FORCE_COMMIT = TRUE
    //'attrsList' retorna a lista dos papéis efetivamente ativados
    //Retorna um booleano que indica se o papel foi ativado ou não
    result :=  creds.set_privileges(true, reqstedPrivList, attrsList)
  end;
  //Libera memória dos arrays dinâmicos
  reqstedPrivList := nil;
  newActiveRole   := nil;
  attrsList       := nil;
  defAutho        := nil;
end;
```


www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

Para ativar um papel, é preciso definir o valor do atributo do tipo `Role` com o nome do papel a ativar e incluí-lo na lista atributos requisitados (`reqstedPriv`) na função acima. Depois o método `set_privileges` deve ser chamado. Retornando `true` indica que o papel foi ativado com sucesso.

Após a ativação do primeiro papel, a ativação dos demais é automática e ocorre de acordo com a necessidade de acesso aos recursos protegidos. Para testar a função acima, faça o seguinte:

1. No arquivo **MacaCliente.dpr**, localize o trecho de código abaixo

```
else begin
    Write(roles[i]+', ');
  end;
end;
```

e adicione na linha seguinte o código

```
//Ativa o papel default
if (roles.Count > 0) then begin
  defaultRole := roles[0];
  Write('Papel "'+defaultRole);
  if ativaPapel(creds, defaultRole) then
    Writeln('" ativado!')
  else
    Writeln('" NÃO ativado!')
end;
```

2. Salve o arquivo e selecione a opção *Project* | *Build MacaCliente* do menu do DELPHI;
3. Para testá-lo, selecione a opção *Run* | *Run* do menu do DELPHI ou pressione a tecla **F9**;
4. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
employeeNumber: 1
Papéis: Vendedor Sênior, Gerente de Produto
Papel 'Vendedor Sênior ' ativado!
Sessão encerrada!
```

3.3.5 Obtendo os Papéis Ativos de um Usuário

Recuperar os papéis ativados para um usuário é bastante simples. A função abaixo, disponível no programa MacaCliente.dpr, faz isto.

```
Function pegaPapeisAtivos(creds : Credentials) : TStrings;
var
  extFamily       : ExtensibleFamily;
  attrType        : AttributeType;
  attrTypeList    : AttributeTypeList;
```


www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
  attrs           : Security_i.AttributeList;
  userActiveRoles : TStrings;
  auxStr          : String;
begin
  userActiveRoles := TStringList.Create;
  //Verifica se a credencial foi criada
  if (creds <> nil) then begin
    //Verifica se a credencial foi criada
    extFamily := TExtensibleFamily.Create(0, 1);
    //Define o tipo de atributo a recuperar, os papéis (Role) do usuário, neste caso
    attrType  := TAttributeType.Create(extFamily, Role);
    SetLength(attrTypeList, 1);
    attrTypeList[0] := attrType;
    //Recupera o valor do tipo de atributo a partir da credencial
    //O primeiro elemento de attrs tem a lista dos papéis ativos separados por ";".
    //É um string vazio quando não há papéis ativados
    //Os demais elementos são desprezados neste caso
    attrs := creds.get_attributes(attrTypeList);
    //Examina a lista de valores em busca do tipo de atributo requisitado
    //Ler apenas o primeiro elemento do vetor porque se procura pelos papéis ativos
    if Length(attrs) > 0 then begin
      if (attrs[0].attribute_type.attribute_family.family_definer = 0) and
         (attrs[0].attribute_type.attribute_family.family = 1) and
         (attrs[0].attribute_type.attribute_type = Role) then begin
        SetString(auxStr, PChar(attrs[0].value), Length(attrs[0].value));
      end;
      //Separa a lista de papéis ativados e coloca na lista 'userActiveRoles'
      pos := AnsiPos(';', auxStr);
      while (pos > 0) do begin
        userActiveRoles.Add(Copy(auxStr, 1, pos-1));
        Delete(auxStr, 1, pos);
        pos := AnsiPos(';', auxStr);
      end;
      if Length(auxStr) > 0 then
        userActiveRoles.Add(auxStr);
    end;
  end;
  result       := userActiveRoles;
  attrs        := nil;
  attrTypeList := nil;
end;
```

Pega o primeiro elemento da lista de atributos retornado, que armazena num *string* a lista de papéis ativos separados por ";". Depois, separa os papéis ativos e os coloca numa lista de *strings* que é retornada. Para testar a função acima, faça o seguinte:

1. No arquivo **MacaCliente.dpr**, localize o trecho de código abaixo

```
if ativaPapel(creds, defaultRole) then
    Writeln('" ativado!')
  else
    Writeln('" NÃO ativado!')
end;
```

e adicione na linha seguinte o código

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
//Libera memória da lista de papeis do usuário
roles.Free;
//Recupera a lista de papéis ativados para o usuário
roles := pegaPapeisAtivos(creds);
Write('Papéis ativos: ');
for i := 0 to (roles.Count - 1) do begin
  if ((roles.Count - 1) = i) then begin
    Writeln(roles[i]);
  end
  else begin
    Write(roles[i]+', ');
  end;
end;
//Libera memória da lista de papéis ativados para o usuário
roles.Free;
```

2. Salve o arquivo e selecione a opção *Project* | *Build MacaCliente* do menu do DELPHI;
3. Para testá-lo, selecione a opção *Run* | *Run* do menu do DELPHI ou pressione a tecla **F9**;
4. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
employeeNumber: 1
Papéis: Vendedor Sênior, Gerente de Produto
Papel 'Vendedor Sênior' ativado!
Papéis ativos: Vendedor Sênior
Sessão encerrada!
```

3.3.6 Alterando a Senha

Usuários podem alterar a própria senha através de sua credencial. Para isto, deverá fornecer a sua senha atual e o valor da nova senha. A função abaixo, disponível no programa MacaCliente.dpr, mostra o que é preciso.

```
Function mudaSenha(creds :Credentials; senhaAtual, novaSenha : String) : String;
const
  //Indica que o tipo de atributo é uma senha
  PRIVILEGE_ATTRIBUTE_CREDENTIALS = 10;
var
  extFamily    : ExtensibleFamily;
  attrType     : AttributeType;
  reqstedPriv  : SecAttribute;
  reqstedPrivList,
  attrsList    : Security_i.AttributeList;
  defAutho     : Opaque;
  pwdInfo      : Opaque;
  i            : Integer;
  n            : Integer;
  msg          : String;
begin
  result := 'credencial inválida';
  //Verifica se a credencial foi criada
```


www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
  if (creds <> nil) then begin
    //Define a família dos atributos de segurança
    extFamily   := TExtensibleFamily.Create(0, 1);
    //Define o tipo de atributo a modificar, a senha neste caso
    attrType    := TAttributeType.Create(extFamily, PRIVILEGE_ATTRIBUTE_CREDENTIALS);
    //Define o valor do atributo, a senha atual e a nova senha separados pelo caracter
    // de 'nova linha' - LINE FEED
    SetLength(pwdInfo, Length(senhaAtual) + Length(novaSenha) + 1);
    for i := 0 to Length(senhaAtual) - 1 do begin
      pwdInfo[i] := Byte(senhaAtual[i+1]);
    end;
    pwdInfo[Length(senhaAtual)] := Byte(#10);   //Line feed
    n := 1;
    for i := Length(senhaAtual) + 1 to Length(pwdInfo) - 1 do begin
      pwdInfo[i] := Byte(novaSenha[n]);
      Inc(n);
    end;
    reqstedPriv := TSecAttribute.Create(attrType, defAutho, pwdInfo);
    SetLength(reqstedPrivList, 1);
    reqstedPrivList[0] := reqstedPriv;
    SetLength(defAutho, 0);
    //Tenta modificar a senha chamando o método set_privileges
    //O parâmetro true indica que a mudança tem de ser imediata - FORCE_COMMIT = TRUE
    //'attrsList' retornará uma lista com um atributo cujo valor é
    //uma mensagem com o motivo do insucesso da modificação, caso ocorra
    //Retorna um booleano que indica se a senha foi modificada ou não
    if creds.set_privileges(true, reqstedPrivList, attrsList) then begin
      result :=  ''; //indica que a troca de senha foi bem sucedida
    end
    else begin //Motivo para não trocar a senha
      if Length(attrsList) > 0 then begin
        SetString(msg, PChar(attrsList[0].value), Length(attrsList[0].value));
        result := msg;
      end;
    end;
  end;
  //Libera memória dos arrays dinâmicos
  reqstedPrivList := nil;
  pwdInfo         := nil;
  attrsList       := nil;
  defAutho        := nil;
end;
```

Primeiro, define que o tipo de atributo a modificar é a senha. O valor deste atributo é a senha atual e a nova senha, separados por pelo caracter de *line feed*. Para a modificação ser efetivada, é preciso que a senha atual seja válida e que a nova senha não seja vazia. O booleano retornado indicará se a operação foi bem sucedida ou não. O parâmetro `actualPriv` retorna um atributo cujo valor é uma mensagem indicando motivo do insucesso da mudança de senha, quando houver. Para testar a função acima, faça o seguinte:

1. No arquivo **MacaCliente.dpr**, localize o trecho de código abaixo

```
//Libera memória da lista de papéis ativados para o usuário
roles.Free;
```

e adicione na linha seguinte o código

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
//Muda a senha do usuário
msg := mudaSenha(creds, 'caca', 'bola');
if  msg = '' then
  Writeln('Senha modificada!')
else
  Writeln('Senha NÃO modificada: '+msg);
```

2. Salve o arquivo e selecione a opção *Project* | *Build MacaCliente* do menu do DELPHI;
3. Para testá-lo, selecione a opção *Run* | *Run* do menu do DELPHI ou pressione a tecla **F9**;
4. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
employeeNumber: 1
Papéis: Vendedor Sênior, Gerente de Produto
Papel 'Vendedor Sênior' ativado!
Papéis ativos: Vendedor Sênior
Senha modificada!
Sessão encerrada!
```

3.4 Utilizando o Serviço de Autorização de Acesso

Para obter autorizações de acesso, é preciso se conectar com o serviço correspondente no MACA CS através de um objeto cliente do tipo `AccessDecision`. A chamada do método `conectaAccessDecisionObject` que está definido no programa MacaCliente.java faz isto. Entretanto, é preciso modificá-lo para alterar no nome do servidor onde o MACA CS está executando. Então, faça o seguinte:

1. No arquivo **MacaCliente.dpr**, localize o trecho de código abaixo

```
else
  Writeln('Senha NÃO modificada: '+msg);
```

e adicione na linha seguinte o código para se conectar com ado – *Access Decision Object*

```
//Conecta-se com o objeto CORBA servidor de autorização de acesso "ado"
ado := conectaAccessDecisionObject(rootConext, adoIOR);
```

2. Salve o arquivo;

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

3.4.1 Obtendo uma Autorização de Acesso Simples

Uma autorização de acesso simples é obtida com a chamada da função `access_allowed`, passando-se como parâmetros o nome do recurso que se pretende acessar, o nome da operação e os atributos de segurança do usuário. O código abaixo mostra como fazer isto.

```
Function acessoAutorizado (creds : Credentials; ado : AccessDecision;
                           nomeRecurso, operacao : String) : boolean;
var
  extFamily     : ExtensibleFamily;
  attrTypeRole,
  attrTypeID    : AttributeType;
  attrs         : Security_i.AttributeList;
  attrTypeList  : AttributeTypeList;
  resNameCompts : ResourceNameComponentList;
  resNameCompt  : ResourceName;
begin
  SetLength(attrs, 0);
  result := false;
  //Verifica se a credencial e 'ado' foram criados
  if ((creds <> nil) and (ado <> nil)) then begin
    //Define a família dos atributos de segurança
    extFamily    := TExtensibleFamily.Create(0, 1);
    //Define os tipos de atributos a recuperar, Role e AccessId neste caso
    attrTypeRole := TAttributeType.Create(extFamily, Role);
    SetLength(attrTypeList, 2);
    attrTypeList[0] := attrTypeRole;
    attrTypeID      := TAttributeType.Create(extFamily, AccessId);
    attrTypeList[1] := attrTypeID;
    //Recupera os atributos de segurança das credenciais do usuário
    attrs := creds.get_attributes(attrTypeList);
    //Define o componente com o nome do recurso e a sua categoria - 'Aplicacoes'
    SetLength(resNameCompts, 1);
    resNameCompts[0] := TResourceNameComponent.Create('Aplicacoes', nomeRecurso);
    //Define o nome do recurso para o 'ado', composto pelo nome do servidor LDAP e
    //pela lista dos componentes do nome - 'resNameCompts'
    resNameCompt := TResourceName.Create('127.0.0.1', resNameCompts);
    try
      //Invoca o método de solicitaçao de autorizaçao de acesso
      //Passa como parâmetros o nome do recurso, a operação a realizar e os atributos
      //de segurança do usuário. Retorna um booleano indicando se o acesso está
      //autorizado ou não
      result :=  ado.access_allowed(resNameCompt, operacao, attrs);
    except
      //O método 'access_allowed' pode levantar esta exceçao quando da ocorrência de
      //erros
      on exc : EInternalError do
        Writeln(exc.Message);
    end;
  end;
//Libera memória dos arrays dinâmicos
  resNameCompts   := nil;
  attrTypeList    := nil;
  attrs           := nil;
end;
```

Para testar a função acima, faça o seguinte:

1. No arquivo **MacaCliente.dpr**, localize o trecho de código abaixo

```
//Conecta-se com o objeto CORBA servidor de autorização de acesso "ado"
ado := conectaAccessDecisionObject(rootConext, adoIOR);
```

e adicione na linha seguinte o código

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

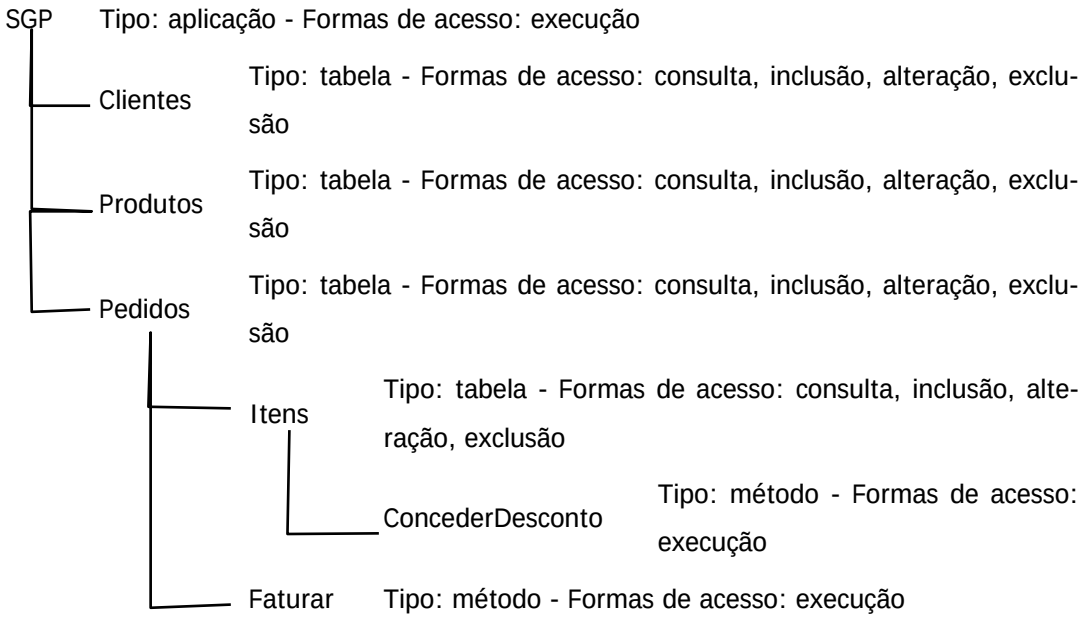
```
//Verifica se o usuário tem acesso ao recurso 'SGP'
if (acessoAutorizado(creds, ado, 'SGP', 'execução')) then begin
  Writeln('Acesso autorizado: SGP - [execução]');
  //Verifica se o usuário tem acesso ao recurso 'ConcederDesconto;Itens;Pedidos;SGP'
 //Mostra como passar parâmetros para uma autorização parametrizada
  if (acessoAutorizado(creds, ado,
                       'ConcederDesconto;Itens;Pedidos;SGP(35)', 'execução')) then begin
    Writeln('Acesso autorizado: ConcederDesconto;Itens;Pedidos;SGP(35) - [execução]');
  end
  else begin
    Writeln('Acesso NÃO autorizado: ConcederDesconto;Itens;Pedidos;SGP(35) – ' +
                                                                    ' [execução]');
  end;
end
else begin
  Writeln('Acesso NÃO autorizado: SGP - [execução]');
end;
```

2. Salve o arquivo e selecione a opção *Project* | *Build MacaCliente* do menu do DELPHI;
3. Para testá-lo, selecione a opção *Run* | *Run* do menu do DELPHI ou pressione a tecla **F9**;
4. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
employeeNumber: 1
Papéis: Vendedor Sênior, Gerente de Produto
Papel 'Vendedor Sênior' ativado!
Papéis ativos: Vendedor Sênior
Senha não modificada!
Acesso autorizado: SGP - [execução]
Acesso autorizado: ConcederDesconto;Itens;Pedidos;SGP - [exec...
Sessão encerrada!
```

Dois pontos merecem ser destacados neste exemplo: o nome dos recursos e a forma de passagem de parâmetros. Sabemos que no servidor LDAP, os nomes dos recursos são armazenados na forma de árvore, conforme ilustrado a seguir, na representação dos recursos da aplicação SGP.

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP



Numa solicitação de autorização de acesso, o nome completo do recurso é formado pela concatenação do nome do nó de mais baixo nível, com os nomes dos nós ancestrais até o nó raiz. Estes nomes são separados por ";". Por exemplo, para obter uma autorização de acesso ao recurso "Faturar", deve-se usar o seu nome completo, que o identifica unicamente no servidor LDAP:

- Faturar;Pedidos;SGP

O MACA CS e o protocolo LDAP não diferencia letras maiúsculas ou minúsculas. Caso haja autorizações parametrizadas associadas ao recurso, os parâmetros devem, necessariamente, ser passados juntamente com o nome do recurso. O recurso "Faturar" deve receber três parâmetros: o código do pedido a faturar – inteiro –, o código do cliente – inteiro – e o nome do vendedor - *string*. Logo, o nome do recurso e respectivos parâmetros tomam a seguinte forma:

- Faturar;Pedidos;SGP(<código pedido>, <código cliente>, "<nome vendedor>")

Estes parâmetros poderão ser usados em regras de autorização associadas ao recurso, por exemplo:

```
exp-abs(umPedido, umCliente, umVendedor)  {
  //Teste se quem vendeu não é quem está faturando o pedido
   umVendedor != userCtx.login &
  // Testa se o dia de hoje é um dia de semana e
  dtCtx.dia_semana in dtCtx.dia_de_semana &
  // se hora atual está no intervalo das 8 às 18 horas.
  (dtCtx.hora >= 8 & dtCtx.hora <= 18)
}
```

3.4.2 Obtendo Múltiplas Autorizações de Acesso

Autorizações de acesso múltiplas são obtidas com chamando do método `multiple_access_allowed`, passando-se como parâmetros uma lista de nomes de recursos com respectivas operações e os atributos de segurança do usuário. A função abaixo mostra como fazer isto.

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
Function autorizaMultiplosAcessos (creds : Credentials; ado : AccessDecision;
                                   nomeRecursos, operacoes : TStrings) : BooleanList;
var
  extFamily       : ExtensibleFamily;
  attrTypeRole,
  attrTypeID      : AttributeType;
  attrs           : Security_i.AttributeList;
  attrTypeList    : AttributeTypeList;
  access_requests : DfResourceAccessDecision_i.AccessDefinitionList;
  resNameCompts   : ResourceNameComponentList;
  resName         : ResourceName;
  res             : BooleanList;
  n               : Integer;
begin
  SetLength(attrs, 0);
  SetLength(res, 0);
  result := res;
  //Verifica se a credencial e 'ado' foram criados
  if ((creds <> nil) and (ado <> nil) and
      (nomeRecursos.Count = operacoes.Count)) then begin
    try
      //Define a família dos atributos de segurança
      extFamily    := TExtensibleFamily.Create(0, 1);
      //Define os tipos de atributos a recuperar, Role e AccessId neste caso
      attrTypeRole := TAttributeType.Create(extFamily, Role);
      SetLength(attrTypeList, 2);
      attrTypeList[0] := attrTypeRole;
      attrTypeID      := TAttributeType.Create(extFamily, AccessId);
      attrTypeList[1] := attrTypeID;
      //Recupera os atributos de segurança das credenciais do usuário
      attrs := creds.get_attributes(attrTypeList);
      //Define a lista de requisições de autorização de acesso
      SetLength(access_requests, nomeRecursos.Count);
      //Para cada recurso, monta o componente com o nome do recurso e operação, e o
      //inclui na lista 'access_requests'
      for n := 0 to nomeRecursos.Count - 1 do begin
        SetLength(resNameCompts, 1);
        resNameCompts[0] := TResourceNameComponent.Create('Aplicacoes',
                                                          nomeRecursos[n]);
        resName := TResourceName.Create('127.0.0.1', resNameCompts);
        access_requests[n] := TAccessDefinition.Create(resName, operacoes[n]);
      end;
      //Invoca o método que retorna um array de booleanos, cada booleano correspondendo
      //ao resultado da solicitaçao de acesso ao recurso correspondente a mesma posição
      //do array
      result := ado.multiple_access_allowed(access_requests, attrs);
    except
      //O método 'multiple_access_allowed' pode levantar esta exceção quando da
      //ocorrência de erros
      on exc : EInternalError do
        Writeln(exc.Message);
    end;
  end;
  //Libera memória dos arrays dinâmicos
  access_requests := nil;
  res             := nil;
  attrTypeList    := nil;
  attrs           := nil;
end;
```

Neste caso, em vez de passar um único nome de recurso e operação, passamos uma lista de nomes de recursos e operações – `access_requests` – e os atributos do usuário. O retorno é uma lista de booleanos, indicando o resultado de cada solicitação de acesso nas posições correspondentes a cada nome de recurso. Para testar a função acima, faça o seguinte:

1. No arquivo **MacaCliente.dpr**, localize o trecho de código abaixo

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
else begin
  Writeln('Acesso NÃO autorizado: SGP - [execução]');
end;
```

e adicione na linha seguinte o código

```
//Usa método que faz múltiplas solicitaçoes de acesso
//Define array com dois nomes de recursos
recursos := TStringList.Create;
recursos.Add('SGP');
recursos.Add('Pedidos;SGP(1, 1, "gustavo.motta")');
//Define array com duas formas de acesso ou operações, uma para cada recurso anterior
operacoes := TStringList.Create;
operacoes.Add('execução');
operacoes.Add('alteração');
results := autorizaMultiplosAcessos(creds, ado, recursos, operacoes);
for i := 0 to Length(results) - 1 do begin
  if results[i] then begin
    Writeln('Acesso autorizado: '+recursos[i]+' - ['+operacoes[i]+']');
  end
  else begin
    Writeln('Acesso NÃO autorizado: '+recursos[i]+' - ['+operacoes[i]+']');
  end;
end;
operacoes.Free;
recursos.Free;
```

2. Salve o arquivo e selecione a opção *Project* | *Build MacaCliente* do menu do DELPHI;
3. Para testá-lo, selecione a opção *Run* | *Run* do menu do DELPHI ou pressione a tecla **F9**;
4. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
employeeNumber: 1
Papéis: Vendedor Sênior, Gerente de Produto
Papel 'Vendedor Sênior' ativado!
Papéis ativos: Vendedor Sênior
Senha não modificada!
Acesso autorizado: SGP - [execução]
Acesso autorizado: ConcederDesconto;Itens;Pedidos;SGP(35) – [exe
Acesso autorizado: SGP - [execução]
Acesso autorizado: Pedidos;SGP(1, 1, "gustavo.motta") - [alteraç
Sessão encerrada!
```


www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

3.5 Verificando a Inatividade ou Queda do Cliente com *Callback*

Normalmente, a sessão do usuário do MACA CS é encerrada com a chamada, pelo cliente, do método `destroy` existente nas credenciais. Entretanto, em casos anormais (e. g., a aplicação cliente aborta, a comunicação cai, o usuário abandona a aplicação), corre-se o risco da sessão do usuário ocupar recursos e dar chances a vulnerabilidades no servidor por um tempo indefinido. O MACA CS ataca este problema de duas formas: através do *timeout* por inatividade do cliente e por meio de um mecanismo de *callback*. No caso do *timeout*, o MACA CS verifica, passivamente, para cada sessão aberta, a intervalos regulares, se houve atividade por parte do cliente. Não havendo atividade, a sessão é encerrada pelo MACA CS, a despeito do cliente. Deste modo, fica assegurado, num tempo finito, o encerramento de todas as sessões de clientes inativos ou incomunicáveis. Entretanto, apenas esta solução é indesejável em casos de servidores intensivamente usados, onde o tempo para *timeout* pode ser grande o suficiente para que se acumule um número excessivo de sessões que não vão mas ser usadas porque os clientes estão inativos ou incomunicáveis.

Com a solução de *callback*, o MACA CS verifica cada sessão aberta, ativamente, tentando se comunicar com o cliente a intervalos regulares para saber se o mesmo se encontra em atividade e comunicável. Ou seja, cada cliente atua como um servidor para o MACA CS. Havendo impossibilidade de comunicação entre o MACA CS e o cliente, conclui-se que o cliente está inacessível e a sessão correspondente é destruída no MACA CS antes mesmo da ocorrência do *timeout*. Diferente do *timeout*, a solução de *callback* é opcional e requer a participação do cliente. Para sua utilização, o cliente deve atender os seguintes requisitos:

- Implementar a interface `Callback` que contém um único método `isClienteAlive`, que é invocado pelo servidor. Abaixo a ILD CORBA da interface é apresentada:

```
interface Callback {
    oneway void isClientAlive (in string message);
};
```

- O método é definido como *oneway* para evitar que o cliente bloqueie a *thread* do servidor MACA CS que faz os *callbacks* nos clientes. A classe `TCallback` no programa **MacaCliente.dpr** é ilustrada a seguir e mostra como a interface pode ser implementada. A interface `Callback` está disponível na *unit* `maca_cs_i` importada pelo programa.

```
//Classe que implementa o método de callback "isClientAlive"
//no cliente. Este é o único método existente na interface
//"Callback" e deve ser implementado com a extensão da
//classe "TInterfacedObject" do Delphi
TCallback = class(TInterfacedObject, Callback)
public
  constructor Create;
  //Método de callback implementado pelo cliente e
  //invocado pelo servidor MACA CS. Embora seja chamado
  //com modo "oneway", sua execução deve ser a mais breve
  //possível
```


www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

```
  procedure isClientAlive (const message : AnsiString);
end;

constructor TCallback.Create;
begin
  inherited;
end;

procedure TCallback.isClientAlive (const message : AnsiString);
begin
  Writeln(message);
end;
```

- Ativar uma instância da classe que implementa a interface (`TCallback`) como um objeto servidor CORBA;
- Configurar no servidor MACA CS, para a sessão corrente (credenciais), o objeto para *callback* no cliente com uma chamada ao método `setClientCallback` da interface `CallbackCredentials`, subtipo da interface `Credentials`.

Para testar um exemplo de *callback*, faça o seguinte:

1. No arquivo **MacaCliente.dpr**, localize o trecho de código abaixo

```
operacoes.Free;
recursos.Free;
```

e adicione na linha seguinte o código

```
//Exemplo de callback

//Cria-se uma instância da interface "Callback", ativando-a
//através do BOA. Assim, esta instância passa a atuar como um
//objeto servidor no cliente apto a receber requisições
cbkSk := TCallbackSkeleton.Create('ClientCallbackObj', TCallback.Create);
BOA.ObjIsReady(cbkSk as _Object);

//Como a interface "Credentials" do CORBAsec não oferece facilidades
//para definição no servidor do MACA CS de um objeto de callback, foi
//definida a interface "CallbackCredentials", que estende a interface
//"Credentials" com a inclusão do método "setClientCallback". Neste
//caso, é preciso fazer o "type casting" do objeto creds com tipo
//"Credentials" para o subtipo "CallbackCredentials" para se ter
//acesso ao método "setClientCallback".
cbkCreds := TCallbackCredentialsHelper.Narrow(creds as CORBAObject, true);
//Define o objeto de callback do cliente
//no servidor do MACA CS
cbkCreds.setClientCallback(cbkSk);

Writeln('Espera pelo callback do servidor MACA CS ...');
Writeln('    Tecle <enter> para encerrar sem testar ou ...');
Writeln('    aguarde pela mensagem de callback ou ...');
Writeln('    aborte o programa para verificar se o callback funciona');
//Espera pelo callback
Readln;
```

2. Salve o arquivo e selecione a opção *Project* | *Build MacaCliente* do menu do DELPHI;
3. Para testá-lo, selecione a opção *Run* | *Run* do menu do DELPHI ou pressione a tecla **F9**;

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

4. A seguinte saída será apresentada:

```
Autenticação realizada com sucesso!
Login: caio
Atributos:
givenName: Caio Prado
employeeNumber: 1
Papéis: Vendedor Sênior, Gerente de Produto
Papel 'Vendedor Sênior' ativado!
Papéis ativos: Vendedor Sênior
Senha não modificada!
Acesso autorizado: SGP - [execução]
Acesso autorizado: ConcederDesconto;Itens;Pedidos;SGP(35) – [exe
Acesso autorizado: SGP - [execução]
Acesso autorizado: Pedidos;SGP(1, 1, "gustavo.motta") - [alteraç
Espera pelo callback do servidor MACA CS ...
    Tecle <enter> para encerrar sem testar ou ...
    aguarde pela mensagem de callback ou ...
    aborte o programa para verificar se o callback funciona
```

5. Caso você aguarde, após o primeiro intervalo de verificação de *callback* a seguinte linha será impressa:

```
Cliente CAIO está vivo?
```

6. Caso você tecle <enter>, a sessão é encerrada normalmente e a chamada periódica de *callbacks* correspondente a sessão é removida no servidor. A linha abaixo será impressa:

```
Sessão encerrada!
```

7. Se o programa for abortado, pode-se verificar no arquivo de log do servidor MACA CS o efeito do *callback*. Neste caso, ele falha e a sessão (credenciais) correspondente é encerrada.

www.incor.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP

4 Definições e Acrônimos

- **ADO**: Access Decision Object;
- **API**: Application Programming Interface;
- **BIGS**: Base de Informações de Gerenciamento de Segurança;
- **CABP**: Controle de Acesso Baseado em Papéis;
- **CORBA**: Common Object Request Broker Architecture;
- **CSS**: CORBA Security Service;
- **HC.FMUSP**: Hospital das Clínicas da Faculdade de Medicina da Universidade de São Paulo;
- **IDL**: Interface Definition Language;
- **InCor**: Instituto do Coração;
- **IOR**: Interoperable Object Reference;
- **LDAP**: Lightweight Directory Access Protocol;
- **MACA AD**: módulo para administração das políticas de autorização e controle de acesso e gerência de contas de usuários;
- **MACA CS**: servidor CORBA do MACA para autenticação de usuários e controle e acesso;
- **MACA**: *Middleware* de Autenticação e Controle de Acesso;
- ***Middleware***: serviço propósito geral que se situa entre plataformas e aplicações;
- **NIST**: National Institute of Standards and Technology;
- **OMG**: Object Management Group;
- **ORB**: Object Request Broker;
- **RAD – Facility**: Resource Access Decision Facility;
- **SASL**: Simple Authentication and Security Layer;
- **SGBD**: Sistema Gerenciador de Bancos de Dados;
- **TLS:** Transport Layer Security;

www.**incor**.usp.br
Av. Dr. Enéas de Carvalho Aguiar, 44 – 05403-000 São Paulo – SP