

# Protocolo de Acesso do OPENBUS 2.0

Tecgraf

November 28, 2012

## 1 Introdução

O protocolo de acesso do OPENBUS 2.0 é o conjunto de regras para realização de chamadas CORBA através de um barramento OPENBUS que foi introduzido na versão 2.0 do OpenBus. O principal objetivo desse protocolo é garantir um nível adequado de segurança das chamadas que permita assegurar a identidade da entidade que iniciou cada chamada realizada através do barramento.

As chamadas através do barramento só podem ser realizadas através de um login de acesso autenticado em nome de uma entidade. Cada entidade que acessa o barramento é identificada através de um nome único, denominado identificador de entidade.

A identificação da entidade que inicia cada chamada permite que as aplicações que respondem às chamadas possam aceitar ou rejeitar essas chamadas de acordo com os privilégios de cada entidade, assim como permitir a implementação de mecanismos de auditoria de utilização das aplicações e serviços integrados ao barramento. As entidades que acessam o barramento podem ser diversas, tanto usuários humanos como sistemas computacionais.

O propósito deste documento é apresentar o protocolo de acesso do OPENBUS 2.0 em detalhes, de forma que o protocolo possa ser implementado em diferentes linguagens. Todo o protocolo de acesso do OPENBUS 2.0 é baseado na arquitetura CORBA, fazendo uso particular o recurso de interceptadores de chamada (*CORBA Portable Interceptors*) para introduzir informações adicionais nas mensagens GIOP (*General Inter-ORB Protocol*) de CORBA.

## 2 Chaves de Acesso

Toda validação dos acessos ao barramento é feita através da troca de dados encriptados usando o algoritmo de chave pública RSA. As chaves RSA utilizadas para esse acesso devem ter tamanho de 2048 bits <sup>1</sup>. Chamaremos essas chaves RSA de *chaves de acesso*.

Para acessar o barramento, é necessário criar um par dessas chaves, sendo uma pública e outra privada. A chave privada deve ser mantida secreta. A posse da chave privada de acesso permite assumir a identidade de qualquer entidade autenticada no barramento usando a chave pública correspondente.

As chaves públicas de acesso são transmitidas entre os diferentes processos que acessam o barramento como uma sequência de bytes (*CORBA::OctetSeq*). Para tanto as chaves públicas devem ser codificadas usando o formato *SubjectPublicKeyInfo* definido pelo padrão X.509, usando a codificação DER (*Distinguished Encoding Rules*).

Recomenda-se que cada processo que acesse um barramento OPENBUS utilize um único par exclusivo de chaves de acesso, evitando assim a geração excessiva de chaves. Contudo, é possível utilizar diferentes pares de chave de acesso, um para cada autenticação de uma entidade no barramento.

---

<sup>1</sup>O tamanho das chaves em bytes é dado pela constante `IDL::tecgraf::openbus::core::v2.00::EncryptedBlockSize` que também indica o tamanho dos blocos encriptados com uma chave desse tamanho.

## 3 Processo de Login

O processo de login consiste basicamente em três atividades: a autenticação de uma entidade através de algum método oferecido pelo OPENBUS; o registro da chave pública de acesso a ser utilizada na validação do acesso; e a geração de identificador de login que é usado para identificar aquela autenticação da entidade junto ao barramento.

O login é feito através da faceta de nome **AccessControl** fornecida pelo `::scs::core::IComponent` do barramento, que implementa a seguinte interface:

```
// File: access_control.idl
// Name: ::tecgraf::openbus::core::v2_0::services::access_control::AccessControl
interface AccessControl;
```

Há basicamente três tipos de autenticação disponíveis para o processo de login, que serão descritos nas seções seguintes.

### 3.1 Login por Senha

O login por senha é realizado através da operação `loginByPassword`, onde é fornecido o identificador da entidade a ser autenticada (**entity**), a chave pública de acesso (**pubkey**) e um bloco de dados encriptados (**encrypted**) com a chave pública do barramento, que é obtida através do atributo **buskey** desta mesma interface.

No caso do login por senha, os dados desse bloco encriptado é a seguinte estrutura codificada em CDR (encapsulado):

```
// File: access_control.idl
// Name: ::tecgraf::openbus::core::v2_0::services::access_control::LoginAuthenticationInfo
struct LoginAuthenticationInfo {
    HashValue hash; // Hash da chave publica a ser associada ao login.
    OctetSeq data; // Dado para autenticao.
};
```

**Campo hash** contém o hash SHA-256 do parâmetro **pubkey**.

**Campo data** contém a senha de autenticação.

Como resultado de um login por senha bem sucedido a operação `loginByPassword` devolve a estrutura **LoginInfo** contendo o identificador do login e o identificador da entidade autenticada. Adicionalmente, também é devolvido o tempo mínimo pelo qual o login permanecerá válido sem necessidade de renovação através do parâmetro de saída **lease**.

### 3.2 Login por Certificado

Uma segunda forma de autenticação junto ao barramento é através de certificados de autenticação registrados no barramento. Nesse caso, para efetuar a autenticação é necessário ter a chave privada correspondente ao certificado registrado.

O login por certificado é feito em duas etapas. Inicialmente, é necessário chamar a operação `startLoginByCertificate` onde é fornecido o identificador da entidade sendo autenticada (**entity**). Como resultado, é devolvido um objeto para a conclusão do processo de login e um desafio (**challenge**), que consiste de um valor secreto gerado pelo barramento e encriptado com a chave pública do certificado registrado.

Para concluir o processo de login por certificado é necessário chamar a operação `LoginProcess::login` do objeto devolvido na etapa anterior, fornecendo uma chave pública de acesso (**pubkey**) e um bloco de dados encriptados (**encrypted**) com a chave pública do barramento, que é obtida através do atributo **buskey** desta mesma interface.

No caso do login por certificado, os dados desse bloco encriptado é estrutura **LoginAuthenticationInfo** codificada em CDR (encapsulado), onde os campos devem ser preenchidos da seguinte forma:

**Campo hash** contém o hash SHA-256 do parâmetro **pubkey**.

**Campo data** contém o valor secreto, obtido pela descriptação usando a chave privada correspondente ao certificado registrado no barramento do desafio (**challenge**) fornecido pela chamada da operação **startLoginByCertificate** da etapa anterior.

Similarmente à autenticação por senha, como resultado de um login por certificado bem sucedido, a operação **LoginProcess::login** devolve a estrutura **LoginInfo** contendo o identificador do login e o identificador da entidade autenticada. Adicionalmente, também é devolvido o tempo mínimo pelo qual o login permanecerá válido sem necessidade de renovação através do parâmetro de saída **lease**.

### 3.3 Login por *Single Sign-On*

Uma terceira forma de autenticação junto ao barramento é através do processo denominado *Single Sign-On*, em que uma entidade já logada permite que outra possa se logar usando a mesma autenticação original. Nesse caso, para efetuar a autenticação é necessário primeiramente obter um desafio do barramento usando um login previamente estabelecido.

O login por *Single Sign-On* é feito em duas etapas. Inicialmente, é necessário chamar a operação **startLoginBySharedAuth** usando um login estabelecido. Como resultado, é devolvido um objeto para a conclusão do processo de login e um desafio (**challenge**), que consiste de um valor secreto gerado pelo barramento e encriptado com a chave pública de acesso do login estabelecido.

Para concluir o processo de login por *Single Sign-On* é necessário chamar a operação **LoginProcess::login** do objeto devolvido na etapa anterior, fornecendo uma chave pública de acesso (**pubkey**) e um bloco de dados encriptados (**encrypted**) com a chave pública do barramento, que é obtida através do atributo **buskey** desta mesma interface.

No caso do login por *Single Sign-On*, os dados desse bloco encriptado é estrutura **LoginAuthenticationInfo** codificada em CDR (encapsulado), onde os campos devem ser preenchidos da seguinte forma:

**Campo hash** contém o hash SHA-256 do parâmetro **pubkey**.

**Campo data** contém o valor secreto, obtido pela descriptação usando a chave privada de acesso do login que iniciou o processo de login por *Single Sign-On* do desafio (**challenge**) fornecido pela chamada da operação **startLoginBySharedAuth** da etapa anterior.

Similarmente à autenticação por certificado, como resultado de um login por *Single Sign-On* bem sucedido, a operação **LoginProcess::login** devolve a estrutura **LoginInfo** contendo o identificador do login e o identificador da entidade autenticada. Adicionalmente, também é devolvido o tempo mínimo pelo qual o login permanecerá válido sem necessidade de renovação através do parâmetro de saída **lease**.

## 4 Credenciais de Chamada

Uma credencial no OPENBUS é um dado associado às chamadas feitas através do barramento que assegura a identidade da entidade que iniciou a chamada. A credencial pode ser dividida em duas partes: a identificação da entidade de quem iniciou aquela chamada; e a identificação de todas as entidades que iniciaram cada chamada da cadeia de chamadas aninhadas onde esta chamada está inclusa. À primeira parte da credencial daremos o nome de *personalidade* e à segunda parte de *cadeia de chamada*.

As credenciais do OPENBUS são transmitidas como um *service context* das mensagens GIOP de *Request* que indicam uma chamada remota, usando o *context ID* definido pela constante abaixo:

```
// File: credential.idl
// Name: ::tecgraf::openbus::core::v2_0::credential::CredentialContextId
const unsigned long CredentialContextId;
```

## 4.1 Personalidade

O conteúdo do *service context* da credencial consiste da codificação CDR (encapsulado) da estrutura abaixo:

```
// File: credential.idl
// Name: tecgraf::openbus::core::v2_0::credential::CredentialData
struct CredentialData {
    Identifier bus;           // bus UUID
    Identifier login;        // caller UUID
    unsigned long session;    // credential session identifier
    unsigned long ticket;     // monotonically increasing counter
    HashValue hash;          // SHA-256 hash
    services::access_control::SignedCallChain chain;
};
```

**Campo bus** é o identificador do barramento onde o iniciador da chamada foi autenticado. O identificador do barramento é obtido através do atributo **busid** da *Faceta de Controle de Acesso* do barramento.

**Campo login** é o identificador de login do iniciador da chamada, que é obtido ao final do processo de login descrito na seção ??.

**Campo session** é o identificador de uma sessão de geração de credenciais. Basicamente, essa sessão indica o segredo emitido pelo receptor da chamada que foi usado para geração da credencial.

**Campo ticket** é um contador monotônico crescente. Cada credencial gerada com um mesmo segredo (ver campo **hash**) deve possuir um valor de **ticket** diferente, para impedir a reutilização de credenciais geradas. Idealmente, o valor do ticket deve ser incrementado de uma em uma unidade a cada geração de uma credencial, evitando que o lado que autentica as credenciais deva utilizar muita memória para lembrar de todos os tickets utilizados ou não utilizados. O lado da autenticação da credencial é livre para recusar credenciais com qualquer ticket, mesmo que estes nunca tenham sido utilizados em chamadas anteriores.

**Campo hash** é o hash SHA-256 de um conjunto de dados que contém um segredo que só é conhecido pelas duas partes envolvidas nessa comunicação (quem inicia a chamada e a quem ela está endereçada). A seção ?? descreve o processo de obtenção do segredo. Esse hash é recalculado pelo lado que autentica a credencial para verificar sua autenticidade. O conjunto de dados usado para o cálculo do hash consiste de uma sequência de bytes formada da seguinte maneira:

- Um byte indicando a versão maior do protocolo <sup>2</sup>;
- Um byte indicando a versão menor do protocolo <sup>3</sup>;
- Uma sequência bytes que representa o segredo (tipicamente o segredo tem 16 bytes);
- 4 bytes com o valor do campo **ticket** da credencial (em *little endian*);
- Uma sequência de bytes dos caracteres do nome da operação sendo chamada. Esse valor fornecido pelo interceptador de chamadas CORBA;

**Campo chain** é a identificação da cadeia de chamadas aninhadas ao qual a chamada correspondente a essa credencial pertence. A seção ?? descreve cadeias de chamada.

### 4.1.1 Validação da Personalidade

Ao receber uma chamada com uma credencial, é necessário validar a autenticidade dessa credencial. Caso a credencial indique um identificador de barramento (campo **bus**) desconhecido, a chamada deve ser recusada com a exceção de sistema **CORBA::NO\_PERMISSION** com **COMPLETED\_NO** e o *minor code* dado pela seguinte constante:

---

<sup>2</sup>Valor dado pela constante **tecgraf::openbus::core::v2\_00::MajorVersion**.

<sup>3</sup>Valor dado pela constante **tecgraf::openbus::core::v2\_00::MinorVersion**.

Em seguida, é necessário verificar se o login informado é válido no barramento indicado. Isso é feito através da operação `getLoginValidity` da faceta de nome *LoginRegistry*, que implementa a seguinte interface:

Caso o login informado não seja mais válido, a chamada deve ser recusada com a exceção de sistema `CORBA::NO_PERMISSION` com `COMPLETED_NO` e *minor code* dado pela seguinte constante:

Caso o login do iniciador da chamada for válido, é necessário recuperar o segredo compartilhado com aquele login correspondente à sessão indicada pelo campo `session` e utilizá-lo para recalcular o hash e comparar com o hash fornecido na credencial para averiguar a autenticidade da credencial.

Note que a ausência de um segredo conhecido referente à sessão indicada pelo campo `session` pode ser tanto porque é a primeira chamada proveniente do login informado na credencial ou porque o segredo compartilhado foi descartado por alguma política de gerência de memória.

Adicionalmente, essa resposta da chamada deve vir juntamente com um *service context* com o *context ID* definido pela mesma constante `CredentialContextId` usada pelo *service context* contendo a credencial das chamadas enviadas. Os dados do *service context* consistem da seguinte estrutura deve codificada em CDR (encapsulado):

**Campo session** contém um novo identificador da sessão de geração de credenciais correspondente ao segredo fornecido no campo **challenge**. O identificador de sessão gerado deve ser diferente de qualquer sessão de geração de credenciais ativa entre o receptor da chamada e o iniciador da chamada.

5

```
// File: access_control.idl
// Name: :tecgraf::openbus::core::v2_0::services::access_control::InvalidPublicKeyCode
const unsigned long InvalidPublicKeyCode;
```

O segredo gerado deve ser uma sequência de 16 bytes gerado aleatoriamente e potencialmente diferente de qualquer outro segredo emitido previamente.

#### 4.1.2 Obtenção do Segredo

Para obter o segredo para geração de uma credencial para uma chamada, é necessário fazer uma chamada com uma credencial inválida. Para tanto recomenda-se gerar uma credencial cujo valor dos campos **session** e **ticket** é zero, o campo **hash** é uma sequência de zeros e o campo **chain** é uma cadeia de chamada nula (veja seção ?? sobre como gerar uma cadeia nula). Neste caso, a credencial gerada será inválida, portanto o resultado deverá ser a exceção de sistema **CORBA::NO\_PERMISSION** juntamente com um *service context* informando o segredo a ser utilizado, conforme descrito acima.

### 4.2 Cadeia de Chamadas

A identificação da cadeia de chamadas é feita através de um dado assinado pelo barramento (com a chave privada do barramento). Esse dado deve estar presente em todas as credenciais através do campo **chain**, que contém uma estrutura do seguinte tipo:

```
// File: access_control.idl
// Name: tecgraf::openbus::core::v2_0::services::access_control::SignedCallChain
struct SignedCallChain {
    EncryptedBlock signature; // Hash de 'encoded' assinado pelo barramento.
    OctetSeq encoded; // estrutura 'CallChain' codificada usando CDR.
};
```

**Campo signature** contém uma assinatura com a chave privada do barramento do hash SHA-256 do campo **encoded**.

**Campo encoded** contém a seguinte estrutura codificada em CDR (encapsulado):

```
// File: access_control.idl
// Name: tecgraf::openbus::core::v2_0::services::access_control::CallChain
struct CallChain {
    Identifier target; // Identificador do login a quem a cadeia se destina.
    LoginInfoSeq originators; // Informações de login das entidades que realizaram as chamadas em cadeia que originam essa chamada.
    LoginInfo caller; // Informações de login da entidades que efetivamente fez chamada atual (última da cadeia).
};
```

**Campo target** contém o identificador do login a quem a cadeia está destinada. Ou seja, esse campo contém o mesmo identificador fornecido pelo campo **login** da estrutura **CredentialReset** descrito na seção ?? que informa o login do objeto ao qual a chamada se destina. Esse campo **target** da cadeia garante que a cadeia só possa ser utilizada (fazer novas chamadas dentro daquela cadeia) por quem detiver o login ao qual ela foi enviada.

**Campo originators** contém uma sequência de informações sobre os vários logins que realizaram as chamadas em cadeia que originam essa chamada.

**Campo caller** login da entidades que efetivamente fez chamada atual (última da cadeia).

#### 4.2.1 Validação da Cadeia

É responsabilidade de quem recebe uma credencial de verificar a integridade da cadeia de chamada fornecida. Para que a cadeia de chamada seja válida, as seguintes condições devem ser válidas:

- O campo **SignedCallChain::signature** deve conter uma assinatura válida do campo **SignedCallChain::encoded** a ser autenticada com a chave pública do barramento.

- O campo `CallChain::target` deve ser o login de quem recebe a chamada.
- O campo `CallChain::caller` deve indicar as informações do login de quem iniciou a chamada.

Se qualquer uma dessas condições não for válida, quem recebe a chamada deve recusar a chamada devolvendo a exceção de sistema CORBA::NO\_PERMISSION com COMPLETED\_NO e o *minor code* definido pela seguinte constante:

```
// File: access_control.idl
// Name: ::tecgraf::openbus::core::v2_0::services::access_control::InvalidCredentialCode
const unsigned long InvalidChainCode;
```

#### 4.2.2 Cadeias para o Barramento

Muitas chamadas para operações das facetas fornecidas pelo barramento são feitas usando credenciais de chamada como especificadas neste documento. Contudo, as cadeias de chamada enviadas nas credenciais em chamadas ao barramento são diferentes das cadeias enviadas em outras chamadas.

A identificação de que o objeto CORBA sendo chamado reside no barramento é feita comparando o campo `CredentialReset::login` com o identificador do barramento, que é obtido através do atributo `busid` da *Faceta de Controle de Acesso* do barramento.

Nas chamadas para o barramento, que são feitas fora de qualquer cadeia de chamada obtida previamente pelo iniciador da chamada, a cadeia a ser enviada na credencial é uma cadeia nula, que consiste da estrutura `SignedCallChain` em que o campo `signed` é uma sequência de zeros e o campo `encoded` é uma sequência vazia.

Nas chamadas para o barramento que são feitas dentro de uma cadeia de chamada obtida previamente pelo iniciador da chamada, a cadeia a ser enviada na credencial é a mesma cadeia originalmente obtida pelo iniciador, inalterada. Ou seja, não é necessário gerar uma nova cadeia para enviar ao barramento, como é necessário nas credenciais para outros destinos como será visto na seção ??.

#### 4.2.3 Geração de Cadeia

Ao gerar uma credencial para uma chamada remota, é necessário gerar uma cadeia assinada pelo barramento para cada login de destino. Isso é feito através da operação `signChainFor` da *Faceta de Controle de Acesso* do barramento. A operação `signChainFor` deve ser chamada com uma credencial gerada usando as regras descritas neste documento. Em particular, a credencial da chamada de `signChainFor` deve conter no campo `chain` a cadeia original a partir da qual a nova cadeia será gerada. Ou seja, é como se a chamada de `signChainFor` é feita como uma chamada aninhada da cadeia de chamadas original. Nesse sentido, para gerar uma cadeia nova que não seja uma extensão de outra cadeia previamente recebida, é necessário enviar uma cadeia nula na credencial da chamada de `signChainFor`, o que está em conformidade com a regra para geração de cadeias para o barramento, uma vez que a operação `signChainFor` é do barramento.

Como resultado da operação `signChainFor` é devolvida uma nova cadeia em que o campo `CallChain::caller` da cadeia original será adicionado à sequência do campo `CallChain::originators` e o campo `CallChain::caller` conterá informações do login de quem chamou a operação `signChainFor`. Adicionalmente, o campo `CallChain::target` conterá o identificador de login fornecido pelo parâmetro `target` da chamada de `signChainFor`.

Note que dessa forma é possível se adicionar a cadeias recebidas (processo denominado *join*), assim como gerar tais cadeias para quaisquer destinos para o qual seja necessário enviar credenciais.

## A *Minor Codes* do CORBA::NO\_PERMISSION

A seguir são descritos todos os valores de *minor code* usados nas exceções de CORBA::NO\_PERMISSION lançadas nas comunicações usando o protocolo do OPENBUS 2.0.

<b>Nome</b>	<b>Descrição</b>
InvalidCredentialCode	A credencial informada na chamada é inválida. Juntamente com essa exceção de <b>NO_PERMISSION</b> deve sempre vir um <b>CredentialReset</b> com informações a serem usadas na geração de uma credencial válida.
InvalidChainCode	A cadeia de chamadas informada na chamada é inválida.
InvalidLoginCode	O login informado não é válido. Isso ocorre quando o barramento informa a quem recebe a chamada que o login informado na chamada é inválido.
UnverifiedLoginCode	Não foi possível verificar o login informado por alguma razão. Isso pode ocorrer quando quem recebe a chamada perde o login com o barramento ou não consegue acessar o barramento por alguma razão.
UnknownBusCode	Identificador do barramento na credencial é desconhecido. Isso pode ocorrer quando quem recebe a chamada não está conectado no barramento informado.