# Tutorial do OpenDreams
# Versão 1.4

Tecgraf/PUC-Rio
*csgrid@tecgraf.puc-rio.br*

**Resumo**

O objetivo deste documento é descrever as orientações para utilizar o serviço OpenDreams disponível em um barramento OpenBus. Esse tutorial apresenta as informações necessárias para executar uma demonstração de uso desse serviço, usando a linguagem Java.

# Sumário

# 1 Introdução

O **OpenDreams** é um Serviço OpenBus[1] para submissão, monitoração e controle de execução remota de comandos. O objetivo do OpenDreams é fornecer uma interface programática para essas funcionalidades disponibilizadas em um servidor CSBase[2].

O serviço OpenDreams é disponibilizado no OpenBus como um componente SCS[3] que adota o padrão CORBA[4] como protocolo de comunicação entre os componentes. O OpenDreams é atualmente implementado e publicado no barramento por servidores CSBase. Um exemplo de servidor CSBase é o CSGrid[5], que oferece as funcionalidades básicas para gerenciamento de execução remota em uma grade computacional. Nesse documento, usaremos o CSGrid como exemplo para utilização do OpenDreams. No entanto, qualquer outro servidor CSBase está preparado para as mesmas funcionalidades apresentadas.

OpenDreams é um acrônimo para ***Open****Bus* ***D****istributed* ***Re****source and* ***A****lgorithms* ***M****anagement* ***S****ervice*. Sua interface de programação é baseada no padrão DRMAA[6]. Esse padrão, especificado pela OMG é atualmente implementado por outros sistemas DRMS (*Distributed Resource Management System*) como o Sun Grid Engine, o GridWay, o PBS/Torque e o Condor.

O objetivo desse documento é fornecer as informações necessárias para uso das funcionalidades disponíveis do OpenDreams. A Seção 2 descreve o ambiente necessário para uso do OpenDreams. A Seção 3 possui as orientações para configuração do CSGrid. A Seção 4 apresenta as classes para utilização do OpenDreams por um cliente Java. A Seção 5 descreve os passos necessários para compilação e execução de um programa de demonstração de um cliente OpenDreams em Java. O Apêndice A possui as IDLs CORBA que compõem a API do OpenDreams. O Apêndice B possui as principais mudanças disponibilizadas na release 1.4 do OpenDreams.

[1] http://www.tecgraf.puc-rio.br/openbus
[2] http://www.tecgraf.puc-rio.br/csbase
[3] http://www.tecgraf.puc-rio.br/scs
[4] http://www.corba.org
[5] http://www.tecgraf.puc-rio.br/publications/artigo_2005_csgrid_integracao_aplicacoes.pdf
[6] http://www.drmaa.org/

# 2 Requisitos para uso do OpenDreams

O OpenDreams não é uma aplicação. É um serviço disponibilizado por um servidor CSBase em um barramento OpenBus. Seu uso, portanto, exige um ambiente instalado que descrevemos a seguir.

A Figura 1 mostra um cenário com os diversos elementos que fazem parte da arquitetura desse ambiente. Um servidor CSBase (por exemplo, o CSGrid) gerencia uma ou mais máquinas para execução de programas (chamados algoritmos) utilizando um *daemon* SGA que executa em uma ou mais máquinas de uma grade computacional. Esse servidor CSBase gerencia uma área de dados e de algoritmos na qual os usuários organizam os arquivos de entrada e saída e os binários utilizados para execução nos nós da grade. Os sistemas CSBase fornecem uma interface cliente através da qual os usuários podem utilizar diversas funcionalidades do ambiente como, cadastrar algoritmos, gerenciar as máquinas de execução, monitorar a execução dos comandos e organizar em projetos os arquivos utilizados para a execução remota.

Além dessa interface Web (java webstart), um servidor CSBase também pode, opcionalmente, publicar em um barramento OpenBus dois serviços: o OpenDreams e o Serviço de Dados. Esses serviços oferecem uma API programática CORBA para que aplicações clientes possam utilizar a infra-estrutura de execução do CSGrid (ou de qualquer outro servidor CSBase). Usando esses serviços, a aplicação cliente se conecta ao barramento, transfere dados para o servidor, submete a execução remota dos algoritmos e recebe a informação do fim da execução.

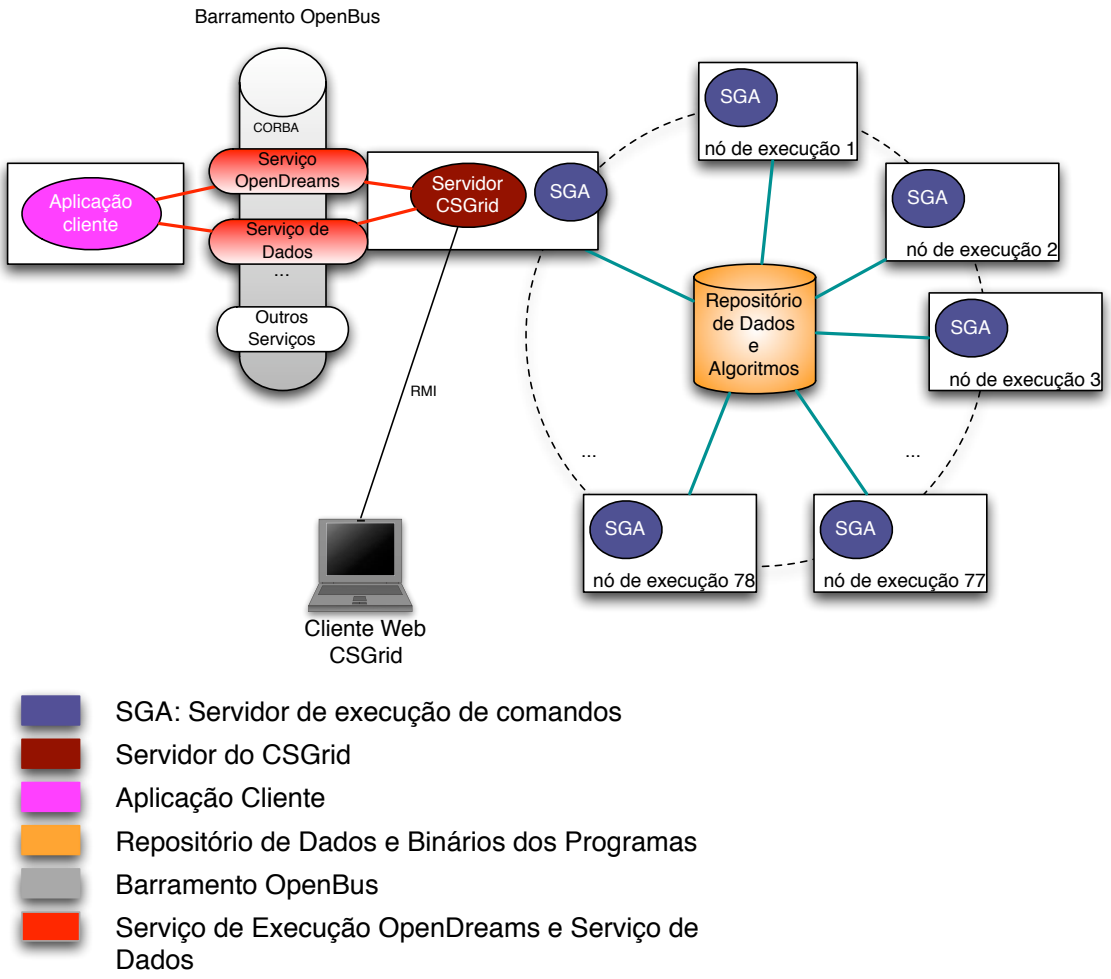


Figura 1: Arquitetura do OpenDreams

## 2.1 OpenBus

O **Openbus** é um middleware para integrar aplicações baseadas em componentes. Essas aplicações podem estar escritas em diferentes linguagens e ser específicas por plataforma (POSIX-compliant, Windows, MacOSX e outras). Dizemos que uma instalação do OpenBus instancia um **barramento** de componentes de serviço inter-operáveis através do padrão CORBA.

O Openbus é composto por:

- serviços básicos para controle de acesso, registro de serviços e gerência de sessões;
- bibliotecas (em diferentes linguagens) para o apoio ao desenvolvimento de novos serviços a serem publicados no Openbus;
- um conjunto de aplicações de exemplo;
- utilitários de apoio à implantação de ambos: o middleware e novos serviços desenvolvidos para esse.

Em `http://www.tecgraf.puc-rio.br/openbus` estão as informações necessárias para download, instalação e execução do OpenBus.

## 2.2 CSGrid

O **CSGrid** é um sistema para gerenciamento e integração de recursos de um ambiente de grade computacional. Entre esses recursos estão os dados dos usuários organizados em projetos, os programas (ou *algoritmos* na terminologia do CSBase) para execução remota, os servidores para execução remota de programas, e outros.

O objetivo do CSGrid é tornar transparente para os usuários o gerenciamento e uso desses recursos computacionais distribuídos e heterogêneos, facilitando a trabalho colaborativo e o melhor aproveitamento da grade computacional disponível. O CSGrid é uma instância do framework CSBase e, portanto, tem como característica a extensibilidade para implementação de novas funcionalidades integradas ao seu ambiente.

O CSGrid possui um desktop onde os usuários utilizam os recursos computacionais distribuídos como se estivessem trabalhando localmente. Utilizando um navegador web os usuários podem criar suas áreas de trabalho, monitorar as máquinas da grade, instalar algoritmos, executar aplicações instaladas no desktop, comandar a execução remota de algoritmos e acompanhar os processos que estão executando esses algoritmos nas máquinas do ambiente.

O servidor CSGrid, ao ser iniciado, faz a publicação do serviço OpenDreams em um barramento OpenBus. Na instalação do CSGrid, são fornecidas as informações da *máquina* e da *porta* onde o barramento está executando. Dessa forma, a configuração *default* do CSGrid é fornecer, através do OpenDreams, uma api programática para submissão e controle de execução de comandos remotos.

Consulte o **Manual de Administração do CSGrid** para obter as informações sobre como instalar e executar o CSGrid. Recomenda-se instalar o CSGrid depois de já ter um OpenBus

instalado e executando. O binário de instalação do CSGrid, bem como o manual, devem ser solicitados a equipe de desenvolvimento pelo email `csgrid@tecgraf.puc-rio.br`.

Antes de iniciar a execução do servidor CSGrid, é necessário instalar os certificados digitais usados no serviço de controle de acesso ao barramento.

- Colocar no diretório do OpenBus o certificado do CSGrid
  Copie o arquivo `CSGRID_HOME/src/csgrid/security/CSGrid.crt` para `OPENBUS_HOME/data/certificates`.
- Colocar no diretório do CSGrid o certificado de Controle de Acesso do OpenBus
  Copie o arquivo `OPENBUS_HOME/data/certificates/AccessControlService.crt` para o diretório `CSGRID_HOME/src/csgrid/security`

onde `OPENBUS_HOME` é o diretório de instalação do OpenBus e `CSGRID_HOME` é o diretório de instalação do CSGrid.

## 2.3 Controle de Acesso

Como o OpenDreams é disponibilizado através do OpenBus, seu uso está sujeito ao mecanismo de controle de acesso adotado no OpenBus e que se baseia em certificados digitais e chaves privadas.

Para acesso ao OpenBus, tanto o servidor CSGrid que publica o serviço no barramento como o cliente que acessa o serviço publicado, precisam ter disponíveis a chave pública (certificado digital) do **Serviço de Acesso** do OpenBus. A chave pública é o arquivo `AccessControlService.crt` que está disponível na instalação do OpenBus utilizada. Esse arquivo é passado como parâmetro no código que faz a conexão com o barramento.

Para que um cliente OpenDreams se conecte ao barramento OpenBus, é necessário gerar uma chave privada e um certificado digital para esse cliente. As chaves privadas e os certificados digitais utilizados no OpenBus seguem formatos bem definidos:

- A chave privada deve ser do tipo RSA e deve estar codificada no formato PKCS8, não criptografado.
- O certificado digital utilizado deve ser do tipo X.509 e deve estar codificado no formato DER.

Essas chaves e certificados podem ser geradas por quaisquer ferramentas, desde que esses formatos sejam obedecidos. Abaixo, seguem as instruções para geração utilizando o OpenSSL[7].

### 2.3.1 Chave Privada

A geração de uma chave privada é feita com o seguinte comando:

```
openssl genrsa -out teste_chave_openssl.key 2048
```

onde:

- **teste_chave_openssl.key**: nome do arquivo da chave privada (que será gerada)
- **2048**: tamanho da chave

Após gerar a chave privada, é necessário convertê-la para o formato PKCS8, usando o seguinte comando:

```
openssl pkcs8 -topk8 -in teste_chave_openssl.key -nocrypt > teste_chave.key
```

onde:

- **teste_chave_openssl.key**: nome do arquivo de entrada que é a chave gerada no comando anterior
- **teste_chave.key**: nome do arquivo da chave privada no formato PKCS8 (que será gerado)

O arquivo intermediário `teste_chave_openssl.key` pode ser apagado e o arquivo `teste_chave.key` deve ser renomeado para o nome do arquivo da chave privada que será usado para acesso ao OpenDreams pela aplicação cliente.

[7] `http://www.openssl.org`

### 2.3.2 Certificado Digital

A geração do certificado digital depende de uma chave privada e é feita com o seguinte comando:

```
openssl req -new -x509 -key teste_chave.key -out teste_crt.crt -outform DER
```

onde:

- **teste_chave.key**: nome do arquivo da chave privada no formato PKCS8
- **teste_crt.crt**: nome do arquivo do certificado digital (que será gerado)

Convém observar que os arquivos de chaves devem ser guardadas em locais seguros, os seja, diretórios com permissão de acesso apenas para a conta de usuário que executa o servidor CSGrid e o OpenBus.

## 2.4 Bibliotecas

### 2.4.1 Para compilação

Para usar o OpenDreams por uma aplicação Java, o código cliente precisa ser compilado com a seguinte biblioteca definida no *classpath* para o compilador `javac`:

- **opendreams-1.4.1.jar**: possui os *stubs* das IDLS do OpenDreams e classes de apoio ao uso do serviço

No caso de aplicações clientes desenvolvidas em outras linguagens, consulte o site do OpenBus para mais informações a respeito das bibliotecas necessárias.

### 2.4.2 Para execução

A execução de uma aplicação cliente OpenDreams desenvolvida em Java precisa de algumas bibliotecas definidas no *classpath* para a máquina virtual `java`. As bibliotecas listadas aqui são compatíveis com a versão atualmente disponibilizada para o servidor CSGrid e para o OpenBus.

- **OpenDreams**: possui os *stubs* das IDLS do OpenDreams e classes de apoio ao uso do serviço. Na versão 1.4 é a seguinte:
  - `opendreams-1.4.1.jar`
- **FTC**: são as bibliotecas para uso do servidor FTC para transferência de arquivos entre o cliente e a área do projeto do servidor CSGrid.
  - `ftc-1.1.0.jar`
- **SCS**: são as bibliotecas do framework de componentes usados no OpenBus.
  - `scs-idl-jacorb-1.0.2.jar`
  - `scs-core-1.0.2.jar`
- **OpenBus**: são as bibliotecas do OpenBus.
  - `openbus-api-1.4.6.jar`
  - `openbus-data_service-api-1.0.1.jar`
  - `openbus-data_service-idl-jacorb-1.0.1.jar`
  - `openbus-data_service-project-idl-jacorb-1.0.1.jar`
  - `openbus-data_service-project-valuetype-1.0.1.jar`
  - `openbus-data_service-valuetype-1.0.1.jar`
  - `openbus-idl-jacorb-1.4.6.jar`
  - `openbusproxy-1.4.1.jar`
- **Jacorb**: são as bibliotecas do ORB Jacorb[8].
  - `antlr-2.7.2.jar`
  - `avalon-framework-4.1.5.jar`
  - `backport-util-concurrent.jar`
  - `idl.jar`
  - `jacorb-2.3.0.jar`
  - `logkit-1.2.jar`
  - `picocontainer-1.2.jar`
  - `wrapper-3.1.0.jar`

[8] `http://www.jacorb.org/`

- **Commons Codec**: são as bibliotecas do projeto Commons Codec [9], para *encoding* e *decoding* comuns.
  - commons-codec-1.3.jar

[9] `http://commons.apache.org/codec/`

# 3 Configuração do CSGrid

## 3.1 Usuário e Permissões

O OpenBus utiliza o mecanismo de interceptação de CORBA no processo de controle de acesso aos serviços. Após a chamada de requisição de login no barramento, todas as chamadas seguintes possuem a credencial de acesso anexada.

No caso do OpenDreams, o serviço usa a informação da credencial de cada chamada para autorizar a execução dos serviços no CSGrid. No CSGrid, a autorização é baseada em permissões atribuídas aos usuários cadastrados. Portanto, um requisito importante é que o administrador do CSGrid cadastre os usuários que executam chamadas ao OpenDreams. O login desses usuários é obtido na credencial anexada às chamadas ao serviço.

Repare que, fica a cargo de cada instalação do ambiente CSGrid decidir qual a melhor política de cadastramento de usuários para acesso através do OpenDreams. Por exemplo, uma opção é ter apenas um usuário *sistemaX* que será responsável por atender todas as execuções do algoritmo de otimização requisitadas pelos diferentes usuários finais, clientes do *sistemaX*. A vantagem é que, dessa forma, o processo de administração do CSGrid se torna mais simples, já que basta ter um único usuário cadastrado por sistema. Por outro lado, essa política dificulta o controle mais "fino"de permissões particulares a cada usuário final do *sistemaX*.

Para execução de um algoritmo pelo OpenDreams, é necessário que o usuário possua dois tipos de permissão: uma permissão da classe **Utilização de servidores de execução de algoritmos** e outra permissão da classe **Execução de Algoritmos em um sistema específico**.

A primeira delas, **Utilização de servidores de execução de algoritmos**, define em quais servidores SGA os algoritmos podem ser executados. Por exemplo, um usuário pode estar autorizado a executar algoritmos apenas em um servidor SGA específico enquanto que outros usuários podem ter permissão a executar algoritmos em qualquer SGA. A Figura 2 mostra um exemplo de criação de uma permissão da classe **Utilização de servidores de execução de algoritmos** em qualquer servidor SGA.

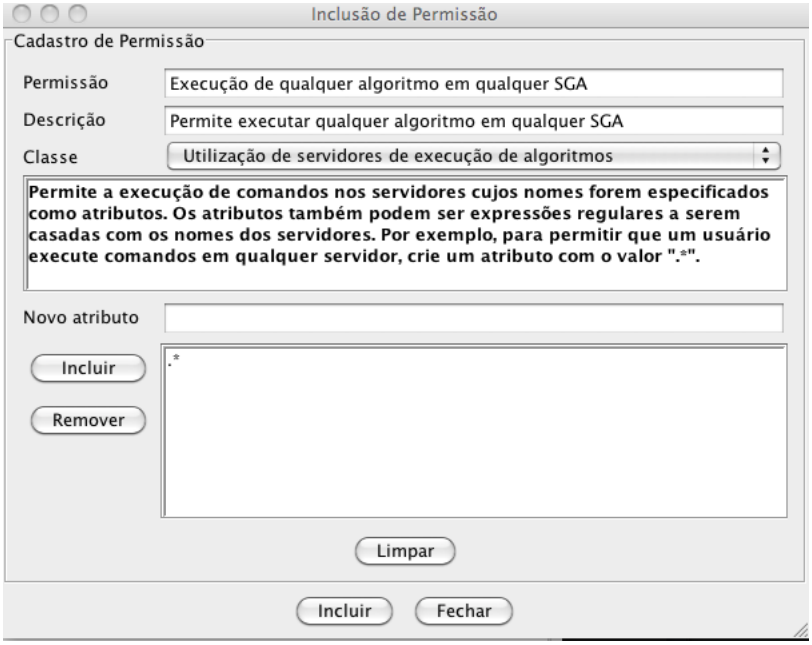


Figura 2: Permissão de execução em servidores SGA

O outro tipo de permissão necessária para execução de algoritmos, **Execução de Algoritmos em um sistema específico**, é a que define quais algoritmos podem ser executados através de qual sistema. A execução pelo OpenDreams usando um certificado denota que um sistema, no caso o detentor do certificado, está requisitando a execução de um determinado algoritmo. Nesse caso, esse tipo de permissão pode definir quais são os sistemas que estão autorizados a executar quais algoritmos. A figura 3 define que usuários que estejam executando através de um cliente dono da credencial com nome **TesteOpenDreams** podem executar apenas o algoritmo cujo nome é **Teste da SimpleDemoOpenDreams**.

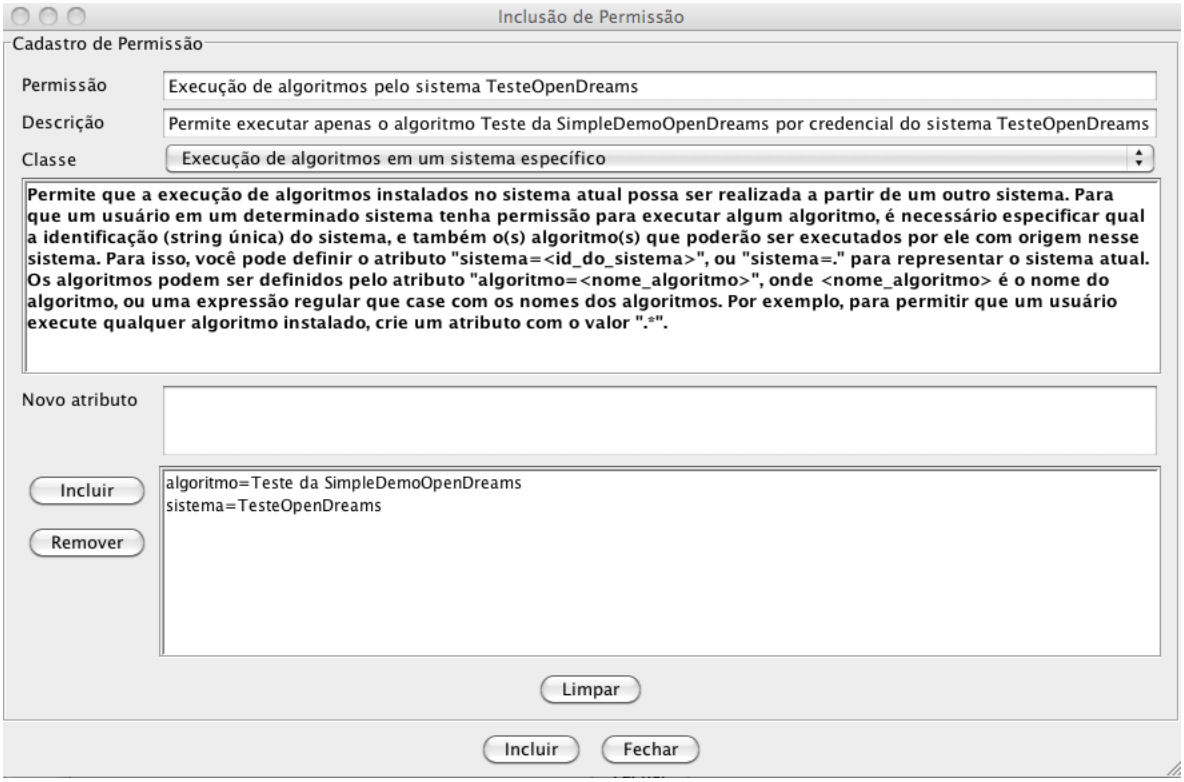


Figura 3: Permissão de execução de algoritmos por sistema

Note que a permissão **Utilização de servidores de execução de algoritmos** requer que o atributo sistema esteja definido. Do contrário, o usuário com essa permissão passa a não ter autorização de executar algoritmos. Como esse atributo permite apenas um único nome de sistema, pode ser útil atribuir ao usuário uma outra permissão desse mesmo tipo, com o atributo `sistema=.`. Isso faz com que o usuário possa executar os algoritmos especificados usando também a própria interface cliente do CSGrid.

É interessante que, além das permissões descritas acima, o usuário tenha também a permissão de **Gerenciamento de Algoritmos** para que consiga usar a interface cliente do CSGrid para cadastrar os executáveis e a configuração dos algoritmos que serão submetidos para execução pelo OpenDreams.

Consultem o **Manual de Administração do CSGrid**, seção 5, as orientações de como criar um usuário no CSGrid, criar permissões (se não existirem) e atribuí-las ao novo usuário.

## 3.2 Projeto

Além do usuário cadastrado no CSGrid, o OpenDreams requer que esse usuário tenha ao menos um projeto criado. Esse projeto é usado em uma sessão no OpenDreams para guardar todos os arquivos de entrada e saída usados nas submissões de comando.

Para criar um projeto, usem a interface web cliente do CSGrid, fazendo login com o usuário que o administrador criou na seção 3.1. No desktop principal do CSGrid, selecione a opção do menu para criar um novo projeto, conforme ilustrado na Figura 4.

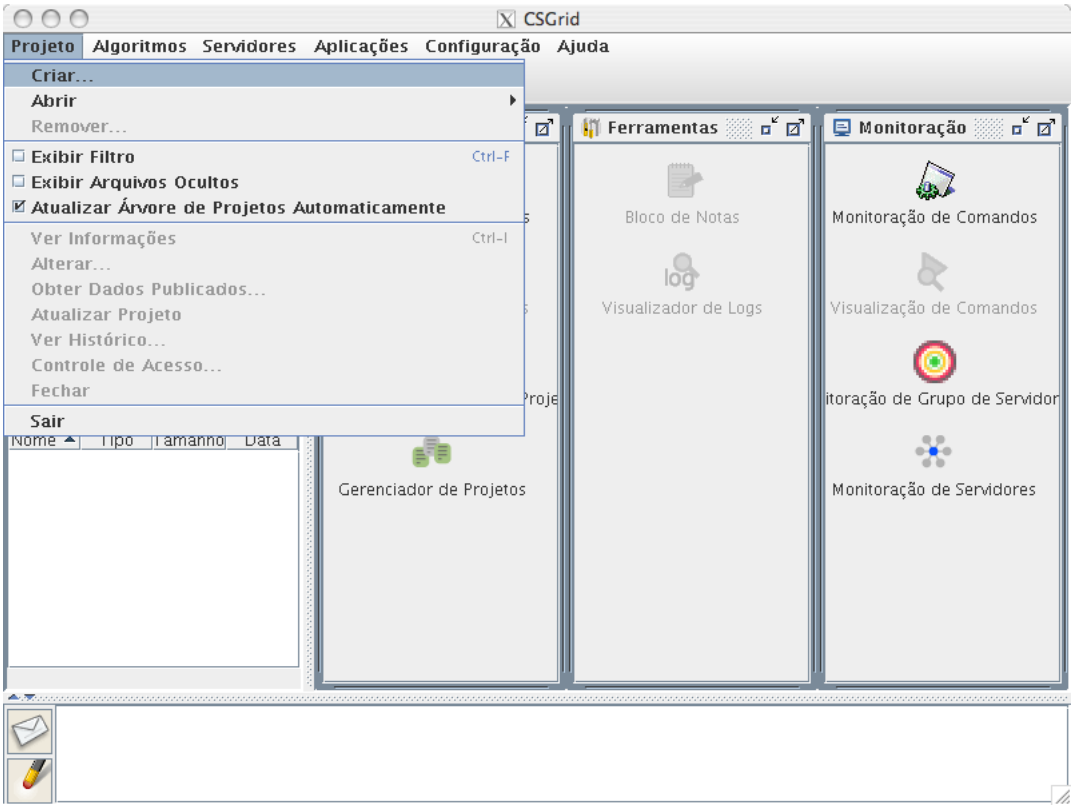


Figura 4: Criar projeto

Na janela que se abrirá, mostrada na Figura 5 entre com o nome do projeto (campo **Nome**) e, opcionalmente, uma descrição (campo **Descrição**).

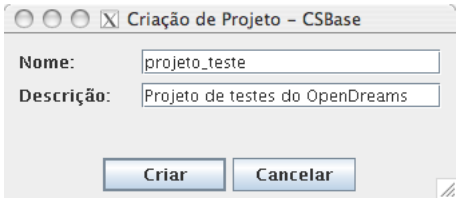


Figura 5: Nome do novo projeto

Repare que, no caso de um sistema que esteja usando o OpenDreams para execução de comandos provenientes de usuários finais diferentes, alguma política deve ser estabelecida no uso do OpenDreams para garantir que os arquivos de cada submissão estejam em diretórios diferentes ou em projetos diferentes. Por exemplo, uma opção é usar um único usuário *sistemaX* e um único projeto *sistemaX_prj*. Antes de submeter um comando para execução, o cliente do OpenDreams pode, programaticamente, criar sub-diretórios dentro do projeto desse usuário para guardar os arquivos de entrada e saída de cada submissão.

O CSGrid publica no barramento um serviço de acesso a dados, o *IHierchicalDataService*, que possibilita acesso a área de projetos do servidor CSGrid. Usando esse serviço, o cliente pode criar diretórios, ler e escrever dados em arquivos e requisitar outras ações de gerência nessa área de projetos.

## 3.3 Algoritmos para Execução

O OpenDreams aceita comandos para execução desde que o comando seja referente a um algoritmo cadastrado no repositório de algoritmos do CSGrid. O cadastramento de algoritmos no CGrid pode ser feito usando a interface Web do CSGrid.

Para cadastrar um algoritmo no CSGrid é necessário que o usuário tenha permissão de gerenciamento do repositório de algoritmos. Veja no **Manual de Administração**, na seção 5.5.2, como o administrador define a permissão para um usuário criar, alterar ou remover algoritmos no repositório do CSGrid. Se o usuário não possuir essa permissão, o cadastramento dos algoritmos deve ser feito pelo administrador do CSGrid.

Além disso, é necessário que o administrador tenha previamente cadastrado as plataformas de execução para a qual o executável do algoritmo está disponível. Veja no **Manual de Administração**, na seção 5.4, como o administrador registra as plataformas de execução de algoritmos.

Cada algoritmo é composto de um arquivo de descrição, um arquivo que contém a definição

de seus parâmetros e os arquivos binários para cada plataforma de execução. O **Manual de Configuradores de Algoritmos do CSGrid** descreve como criar os arquivos de definição de parâmetros, chamados configuradores e representados por um arquivo **config.xml**.

Para iniciar o cadastramento de um algoritmo, o usuário deve selecionar o item **Gerenciar Algoritmos** no menu **Algoritmos** do desktop principal do CSGrid, conforme ilustrado na Figura 6.

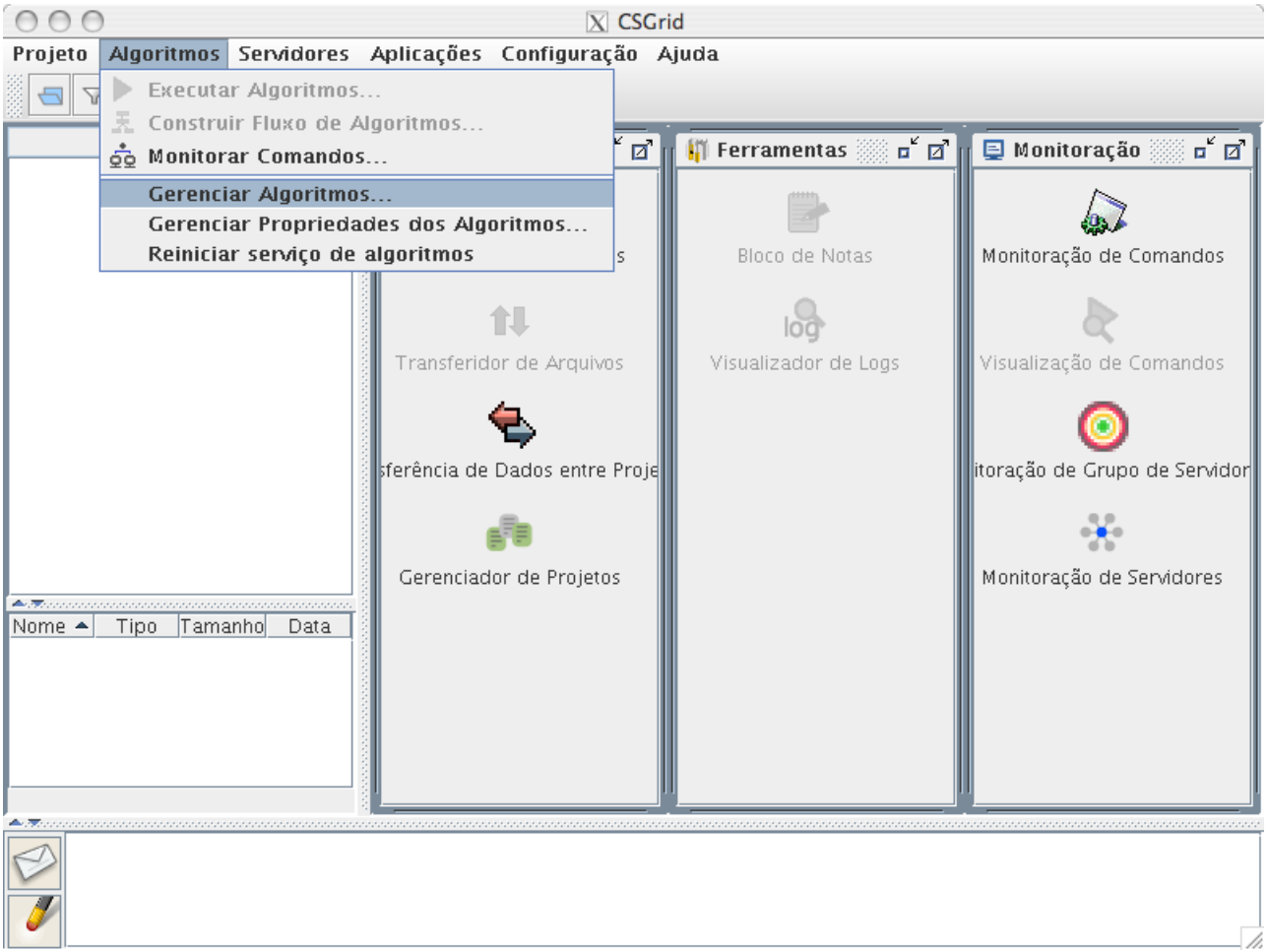


Figura 6: Gerenciar Algoritmos do Repositório

Na janela de **Gerência do repositório de Algoritmos**, o usuário deve pressionar o botão da direita sobre o item **Algoritmos** e escolher a opção **Incluir Algoritmos**, conforme ilustrado na Figura 7. A Figura 8 mostra o preenchimento do nome do algoritmo sendo criado.

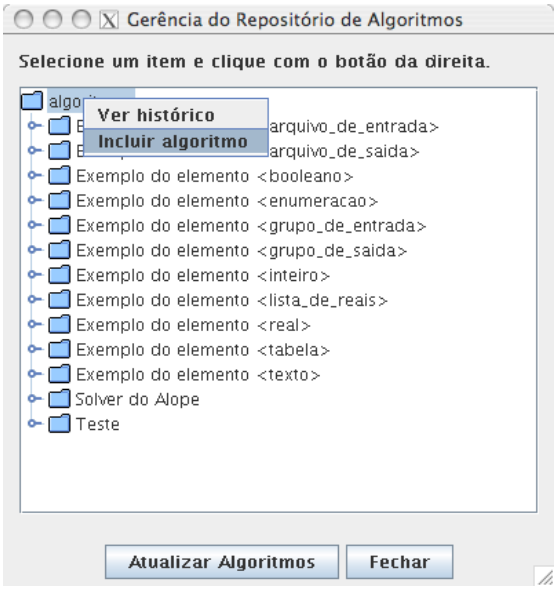


Figura 7: Incluir Algoritmo

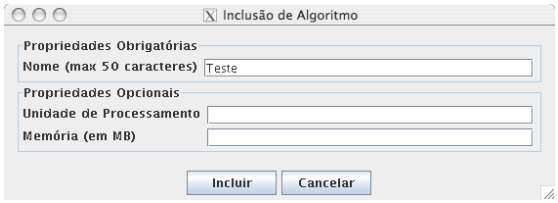


Figura 8: Nome do Algoritmo

Em seguida, é necessário incluir uma versão para o algoritmo. Um algoritmo pode ter várias versões. O usuário deve pressionar o botão direito sobre o item com o nome do algoritmo criado no passo anterior e escolher a opção **Incluir Versão**, conforme ilustrado na Figura 9. A Figura 10 mostra a janela para o usuário preencher com o número da versão do algoritmo sendo criado.

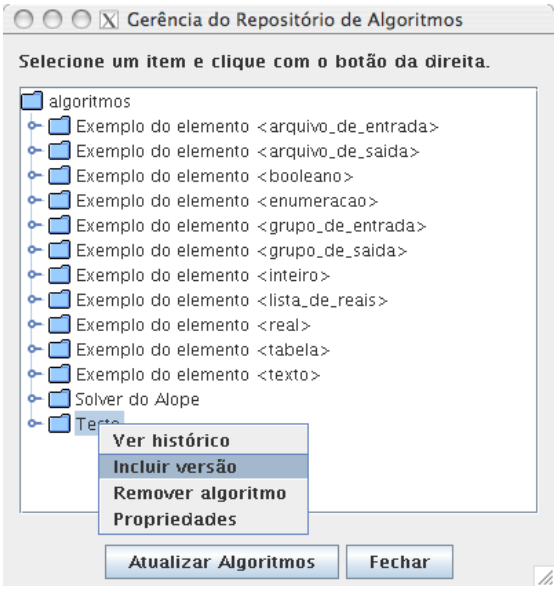


Figura 9: Incluir Versão do Algoritmo

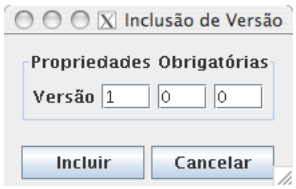


Figura 10: Versão do Algoritmo

Após criar a versão do algoritmo, deve-se incluir o arquivo que descreve o algoritmo (index.html) e o arquivo que contém a definição de seus parâmetros (config.xml). Para isso deve-se pressionar o botão direito sobre as pastas **html** e **configurator**, respectivamente, e selecionar o opção de inclusão. O usuário deverá selecionar um arquivo local que será copiado para o servidor.

O passo seguinte é incluir o executável propriamente dito. Para isso deve-se pressionar o botão direito sobre a pasta **bin** e incluir a plataforma para o qual aquele executável foi gerado.

# 4 API de Programação em Java

Atualmente o OpenDreams oferece uma API em Java com as classes já compiladas (usando Jacorb) das IDLs do OpenDreams e algumas classes de apoio que facilitam o uso do serviço.

## 4.1 Proxy de acesso ao OpenDreams

A classe **OpenDreamsProxy** tem como objetivo encapsular os mecanismos de acesso ao OpenDreams. Faz o login no barramento e recupera as interfaces dos serviços usados pelo cliente desse barramento.

O proxy do OpenDreams usa um conjunto de propriedades que configuram informações necessárias para o acesso ao OpenBus e o uso do serviço.

Propriedades para acesso ao OpenBus:

- **openbus.acs.host**: máquina (preferencialmente o IP) onde o barramento OpenBus está executando. Essa propriedade é obrigatória.
- **openbus.acs.port**: porta onde o serviço de acesso do OpenBus está disponível. Essa propriedade é obrigatória.
- **openbus.acs.entity.name**: nome do certificado público cadastrado no OpenDreams. Esse nome deve corresponder ao arquivo **.crt** fornecido ao administrador do barramento. Caso a propriedade **openbus.acs.delegate** não esteja definida, esse é o login do usuário passado na credencial para o serviço OpenDreams. Nesse caso, esse usuário deve existir no servidor CSGrid usado na publicação desse serviço. Essa propriedade é obrigatória.
- **openbus.acs.private.key**: caminho para o arquivo que possui a chave privada do cliente. Essa chave é usada para autenticação do cliente no barramento. A chave pública correspondente, com o nome definido em **openbus.acs.entity.name** deve estar instalada no barramento. Essa propriedade é opcional. Quando essa propriedade não está definida, a aplicação deve atribuir ao proxy do opendreams o *stream* correspondente a chave privada usado na conexão, antes do proxy ser iniciado.
- **openbus.acs.certificate**: caminho para a chave pública do serviço de acesso do barramento. Esse arquivo deve ter sido fornecido pelo administrador do barramento. Normalmente, chama-se **AccessControlService.crt**. Essa propriedade é opcional. Quando essa

propriedade não está definida, a aplicação deve atribuir ao proxy do opendreams o *stream* correspondente ao certificado público usado na conexão, antes do proxy ser iniciado.

- **openbus.acs.delegate**: login do usuário passado na credencial para o serviço OpenDreams. Essa propriedade é usada quando o certificado com o nome **openbus.acs.entity.name** não corresponder ao usuário do servidor CSGrid usado na execução do serviço. Essa propriedade é opcional.

Propriedades para o OpenDreams:

- **opendreams.component.name**: nome do componente IOpenDreams publicado pelo servidor CSGrid. Essa propriedade é necessária para encontrar o componente no barramento, já que mais de um servidor CSGrid pode estar conectado ao mesmo barramento. É opcional. Quando não definida, usa um valor default.
- **opendreams.component.version**: versão do componente IOpenDreams publicado pelo servidor CSGrid. Essa propriedade é necessária para encontrar o componente no barramento, já que mais de um servidor CSGrid pode estar conectado ao mesmo barramento. É opcional. Quando não definida, usa um valor default.
- **opendreams.project.name**: nome do projeto usado para execução no OpenDreams. Esse projeto deve estar criado e ser do usuário CSGrid usado na execução do serviço. Essa propriedade é obrigatória.

Propriedades para o Data Service:

- **ProjectDataService.component.name**: nome do componente IHierquicalDataService publicado pelo serviço de Projeto do servidor CSGrid. Essa propriedade é necessária para encontrar o componente no barramento, já que mais de um servidor CSGrid pode estar conectado ao mesmo barramento. Quando não definida, usa um valor default.
- **ProjectDataService.component.version**: versão do componente IHierquicalDataService publicado pelo serviço de Projeto do servidor CSGrid. Essa propriedade é necessária para encontrar o componente no barramento, já que mais de um servidor CSgrid pode estar conectado ao mesmo barramento. Quando não definida, usa um valor default.

Propriedades para o ORB:

- **org.omg.CORBA.ORBClass**: o nome da classe para uso na propriedade `org.omg.CORBA.ORBClass` do ORB. Quando não definida, usa um valor default.
- **org.omg.CORBA.ORBSingletonClass**: o nome da classe para uso na propriedade `org.omg.CORBA.ORBSingletonClass` do ORB. Quando não definida, usa um valor default.

A classe **OpenDreamsProxy** possui dois construtores: um construtor default e um outro que recebe as propriedades como parâmetro. O construtor default `OpenDreamsProxy()` cria um proxy do OpenDreams usando as propriedades definidas no arquivo **opendreams.properties** existente no diretório corrente (da execução). O construtor `OpenDreamsProxy(Properties properties)` cria um proxy usando as propriedades recebidas como parâmetro. Caso ocorra alguma falha na construção do proxy, é lançada uma exceção **OpenDreamsException**.

Após ser instanciado, o **OpenDreamsProxy** precisa ser iniciado. Na inicialização do proxy, é feita a conexão com o barramento para obtenção do serviço OpenDreams. Caso ocorra alguma falha na conexão com o serviço OpenDreams, será lançada uma exceção **OpenDreamsException**.

Na API, existem duas formas diferentes de fazer a inicialização, através dos métodos `init()` ou `init(String user, String password)`. No primeiro, a conexão com o barramento será feita através do uso de chave privada e certificado. No segundo caso, os parâmetros fornecidos serão utilizados para fazer o acesso ao barramento usando um login e senha.

Em particular, a chave privada e certificado público usados na conexão, podem ser definidos de duas maneiras diferentes. Uma forma de definir essas informações é utilizando as propriedades `openbus.acs.private.key` e `openbus.acs.certificate`. Nesse caso, as propriedades, carregadas na construção do proxy, definem os caminhos para os arquivos que possuem, respectivamente, a chave privada e o certificado público, usados na conexão. Os exemplos a seguir ilustram o uso da API onde a inicialização do proxy é feita exclusivamente com base nas propriedades carregadas na sua construção.

```
OpenDreamsProxy opendreamsProxy = new OpenDreamsProxy();
opendreamsProxy.init();
```

No exemplo acima, as propriedades são carregadas a partir do arquivo default `opendreams.properties` presente no diretório de execução da aplicação. As propriedades `openbus.acs.private.key` e `openbus.acs.certificate` estão definidas com os caminhos para os arquivos com a chave privada e o certificado público.

O exemplo a seguir é similar ao anterior. A única diferença é que as propriedades são carregadas a partir do arquivo `config.properties` ao invés de um arquivo default.

```
Properties properties = new Properties();
properties.load(new FileInputStream("config.properties"));
OpenDreamsProxy opendreamsProxy = new OpenDreamsProxy(properties);
opendreamsProxy.init();
```

Uma maneira alternativa da aplicação definir a chave privada e o certificado público usados na conexão, é chamando os métodos `setPrivatekey(InputStream privateKeyStream)` e `setACSCertificate(InputStream acsCertificateStream)` para atribuir ao proxy esses dados que já foram lidos previamente como *streams* de entrada. Se esses métodos forem usados, a chave privada e o certificado público são criados a partir dos streams especificados nos respectivos parâmetros. A definição da chave e do certificado por meio de *streams* serve para que a própria aplicação decida de onde recuperar esses dados. Um uso típico, por exemplo, é colocar os arquivos de chave e certificado dentro de um jar, junto com a classe da aplicação. Note que, quando esses métodos são usados, as propriedades `openbus.acs.private.key` e `openbus.acs.certificate` carregadas no contrutor, são ignoradas, caso estejam definidas. É importante observar que esses métodos somente estão disponíveis na api java do OpenDreams a partir da versão `1.3.2` e dependem do uso das bibliotecas do openbus a partir da versão `1.4.6`.

O exemplo a seguir mostra como atribuir a chave privada e o certificado ao proxy, como *streams* obtidos a partir do *class loader* da aplicação. Repare que ambos os métodos, `setPrivatekey` e `setACSCertificate`, são chamados antes iniciar o proxy.

```
opendreamsProxy.setPrivatekey(
  getClass().getResourceAsStream("security/chave.key"));
opendreamsProxy.setACSCertificate(
  getClass().getResourceAsStream("security/certicate.crt"));
opendreamsProxy.init();
```

Após finalizar o uso do OpenDreams, o proxy deve ser fechado com o método `close()`, liberando a conexão com o serviço do OpenBus.

```
opendreamsProxy.close();
```

## 4.2 Execução de um comando

Para executar um comando pelo OpenDreams, o primeiro passo é obter a referência para a interface **IOpenDreams** do serviço. Essa interface e as classes envolvidas na execução de um comando são provenientes da compilação das IDLs **drmaa.idl** e **opendreams.ild**.

```
/**
 * \brief Interface principal de acesso ao serviço OpenDreams.
 * A interface IOpenDreams cria uma sessão referente a um projeto
 * do usuário no OpenDreams.
 * Essa sessão implementa uma sessão DRMAA e oferece métodos para
 * submissão, controle e monitoração de jobs para execução remota.
 */
interface IOpenDreams{
   DRMAA::Session getSession(in string projectId)
   raises ( DRMAA::AuthorizationException,
         DRMAA::InternalException);
};
```

Uma sessão é usada para gerenciar os jobs submetidos para execução. A sessão é o principal ponto de acesso para os métodos do OpenDreams e atende à especificação do DRMAA. Para executar um comando, o cliente deve obter um **Job Template** que usará para especificar os parâmetros do comando. O Job Template é obtido a partir de uma sessão, usando o método `createJobTemplate`.

O Job Template possui métodos para parametrizar a execução de um comando. No OpenDreams, temos uma interface **OpenDreamsJobTemplate** com atributos adicionais para atender às funcionalidades específicas de execução de algoritmo do CSGrid.

Os principais atributos atualmente implementados no Job Template do OpenDreams são:

- **remoteCommand**: o comando a ser executado na máquina remota. O comando `execAlgo` é usado para indicar a execução de um algoritmo cadastrado em um servidor CSGrid. Atualmente, apenas esse valor está sendo aceito nesse atributo.
- **args**: a lista de argumentos para o comando. Atualmente, o comando `execAlgo` aceita como argumentos o nome do algoritmo e a versão do algoritmo. O nome do algoritmo é definido pelo argumento `-name`. A versão do algoritmo é definida pelo argumento `-version`.

- **jobDescription**: descrição do comando.
- **jobParameters**: conjunto de parâmetros como pares <nome, valor> que são passados para o algoritmo a ser executado. Esses parâmetros devem corresponder a configuração do algoritmo definida pelo seu configurador.
- **email**: uma lista de emails para os quais são enviados uma mensagem de término de execução do comando.
- **outputPath**: um caminho para um arquivo no projeto a ser criado com o *stdout* da execução do comando.

Para submeter um comando para execução, o cliente usa o método `runJob`, passando o Job Template previamente parametrizado. Esse método retorna uma identificação para o comando submetido.

O trecho de código a seguir mostra um exemplo de execução de um comando no OpenDreams.

```
OpenDreamsProxy opendreamsProxy = new OpenDreamsProxy();
opendreamsProxy.init();
IOpenDreams opendreams = opendreamsProxy.getIOpenDreams();

Session session = opendreams.getSession(
   opendreamsProxy.getProperties().getProjectName());
session.init("");

OpenDreamsJobTemplate jt =
   (OpenDreamsJobTemplate) session.createJobTemplate();
jt.remoteCommand = "execAlgo";
jt.args =
  new String[] { "-name", "Teste da SimpleDemoOpenDreams",
                 "-version", "1.0.0" };
jt.jobDescription = "Teste da Demo Simples do OpenDreams";
jt.jobParameters = new String[][] { { "qtde", "3" },
                                    { "duracao", "2" } };
jt.outputPath = "saida.out";
jt.email = new String[] { "opendreams@tecgraf.puc-rio.br" };

String jobName = session.runJob(jt);
session.deleteJobTemplate(jt);
```

O exemplo acima executa o algoritmo com o nome `teste` e a versão `1.0.0` cadastrado no repositório de algoritmos do servidor CSGrid. A configuração desse algoritmo é:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Algoritmo Teste -->
<algoritmo comando="teste" shell="" capturar_codigo_de_saida="sim">
  <formato_no_comando>$VALOR_DO_PARAMETRO</formato_no_comando>
  <grupo_de_entrada>
    <inteiro nome="qtde" rotulo="Sleeps"
     dica="Quantidade de sleeps que o teste vai fazer"
     minimo="1" padrao="10"/>
    <inteiro nome="duracao" rotulo="Sleep time (s)"
     dica="Tempo em segundos que o sleep vai durar."
     minimo="1" padrao="5"/>
  </grupo_de_entrada>
</algoritmo>
```

## 4.3 Sincronização de término de execução de comando

A sessão do OpenDreams oferece o método `_wait` para que o cliente obtenha a informação sobre o término da execução do comando. Essa chamada é síncrona e, portanto, o cliente fica bloqueado até que a execução finalize, com erro ou com sucesso. O método `_wait` aguarda um único job completar sua execução e retorna o status do job com a condição de término e as informações de uso de recursos pelo job.

Os parâmetros de entrada do método `_wait` são:

- o identificador do job ou `JOB_IDS_SESSION_ANY` que representa qualquer job da sessão ativa
- o tempo máximo (em segundos) a ser esperado pela finalização dos job. As constantes `TIMEOUT_WAIT_FOREVER` ou `TIMEOUT_NO_WAIT` também podem ser usadas.

O retorno é um **JobInfo**, que encapsula o estado de finalização do job e as informações sobre o uso de recursos durante sua execução.

O trecho a seguir ilustra a chamada ao método `_wait` para aguardar o término da execução de um job.

```
JobInfo jobInfo = session._wait(jobName,
  Session.TIMEOUT_WAIT_FOREVER);
```

O parâmetro `jobName` é o identificador retornado na submissão do comando por `runJob`. A constante `TIMEOUT_WAIT_FOREVER` indica que o cliente deve aguardar indefinidamente até o job completar sua execução.

## 4.4 Manipulação da área de projetos

Para que o cliente do OpenDreams tenha acesso a área de projetos usada na execução de comandos, usamos o serviço **IHierarquicalDataService** publicado no barramento pelo serviço de Projetos do CSGrid. A interface desse serviço pode ser obtida usando o método `getIDataService()` disponível no OpenDreamsProxy.

Como uma forma de facilitar o uso pelo cliente Java OpenDreams, o proxy também oferece uma classe auxiliar **Project** que possui métodos de criação de diretórios e leitura de arquivos no projeto. Essa classe ainda está em desenvolvimento e, por enquanto, oferece uma api simplificada de forma a atender a criação de diretórios no projeto, a navegação pelos diretórios do projeto e a recuperação dos dados de um arquivo.

O trecho abaixo ilustra como listar todos os projetos do usuário.

```
List<Project> allProjects = opendreamsProxy.getAllProjects();
for (Project project : allProjects) {
System.out.println("** Projeto: " + project.getName());
}
```

O trecho de código a seguir ilustra como listar todos os arquivos de um dado projeto e como recuperar o conteúdo do arquivo de saída *stdout* definido no atributo **outputPath** do exemplo anterior.

```
Project project = opendreamsProxy.getProject();

for (String fileName : project.list()) {
if (project.isDirectory(fileName))
       continue;
```

```
System.out.println("** Arquivo encontrado: " + fileName);
}
byte[] buffer = project.getDataFrom("saida.out");

System.out.println("** O log de saída do comando é:");

System.out.println(new String(buffer));
```

Neste segundo exemplo, o nome do projeto está definido na propriedade **opendreams.project.name**.

# 5 SimpleDemoOpenDreams

## 5.1 Passo 1: Descompactar o arquivo de demo

O arquivo `SimpleDemoOpenDreams.tgz` contém:

- **SimpleDemoOpenDreams.java**: o código fonte do programa
- **opendreams.properties**: a configuração para acesso ao OpenDreams
- **lib**: diretório contendo as bibliotecas para compilação e execução
- **security**: diretório contendo as chaves públicas e privadas para acesso ao OpenBus
- **algorithm**: diretório contendo o algoritmo a ser cadastrado no CSGrid
- **compile**: script de compilação do código fonte
- **run**: script de execução da demo

## 5.2 Passo 2: Configuração do CSGrid

Certifique-se que existe um usuário criado no CSGrid com permissão para execução de algoritmos. Esse usuário será usado para execução da demo.

Certifique-se também que esse usuário possui um projeto criado. Esse será o projeto no qual o algoritmo da demo é executado.

Instale no CSGrid o algoritmo que está no diretório `algorithm`.

## 5.3 Passo 3: Instale os certificados

Instale no OpenBus a chave pública `security/TesteOpenDreams.crt` e substitua o arquivo `security/AccessControlService.crt` com a chave pública do OpenBus.

## 5.4 Passo 4: Altere o arquivo de propriedades do OpenDreams

Edite o arquivo `opendreams.properties` de acordo com a instalação que você possui e a configuração a ser usada pela demo.

## 5.5 Passo 5: Compile o código fonte

Use o script `compile` para compilar a demo.

## 5.6 Passo 6: Execute a demo

Use o script `run` para executar a demo.

# A IDL

## A.1 drmaa.idl

```
/**
 * @mainpage
 * @section intro Introdução
 * <p>
 * O DRMAA (Distributed Resource Management Application API) é um padrão GGF
 * (Global Grid Forum) de api para sistemas responsáveis por gerenciar a
 * execução de tarefas (jobs) em uma grade de computadores.
 * <p>
 * O DRMAA define um conjunto de operações que permitem acesso programático a
 * funcionalidades comuns a diversos sistemas.
 * <ul>
 * <li>definição e configuração de jobs
 * <li>submissão, monitoramento e controle de jobs
 * <li>sincronização e informações no término de jobs
 * </ul>
 */

/**
 * @file drmaa.idl
 * Arquivo de especificação da IDL para o DRMAA.
 * @author Tecgraf PUC−Rio
 */

#ifndef _DRMAA_IDL
#define _DRMAA_IDL

/**
 * Módulo DRMAA.
 */
module tecgraf {

module openbus {

module DRMAA {

/**
```

```
 * \brief Uma sequência ordenada de string
 */
typedef sequence<string> OrderedStringList;

/**
 * \brief Uma sequência não ordenada de string.
 */
typedef sequence<string> StringList;

/**
 * \brief Dicionário para mapa de pares chave-valor
 */
typedef sequence< sequence<string,2> > Dictionary;

/**
 * \brief Quantidade de tempo com uma resolução de, no mínimo, segundos.
 */
typedef long long TimeAmount;

enum JobControlAction {
SUSPEND,
 RESUME,
 HOLD,
 RELEASE,
 TERMINATE
};

/**
 * \brief Possíveis estados de um job submetido.
 *
 * É usado como retorno da chamáda a <code>Session::jobStatus</code>.
 */
enum JobState {
 UNDETERMINED, /**< \brief O estado não pode ser determinado */
 QUEUED_ACTIVE, /**< \brief Na fila aguardando o escalonamento */
 SYSTEM_ON_HOLD, /**< \brief O job foi colocado "on hold" pelo sistema ou pelo administrador. */
 USER_ON_HOLD, /**< \brief O job foi colocado "on hold" por um usuário. */
 USER_SYSTEM_ON_HOLD, /**< \brief O job foi colocado "on hold" pelo sistema ou pelo administrador e ta
 RUNNING, /**< \brief O job foi O job foi removido da fila de escalonamento e está executando */
 SYSTEM_SUSPENDED, /**< \brief O job foi suspenso pelo sistema ou pelo administrador */
```

```
  USER_SUSPENDED, /**< \brief O job foi suspenso por um usuário */
  USER_SYSTEM_SUSPENDED, /**< \brief O job foi suspenso pelo sistema ou pelo administrador e por um us
  DONE, /**< \brief O job terminou de forma normal */
  FAILED /**< \brief O job terminou de forma anormal */
};

/**
 * \brief Possíveis estados iniciais de um job ao ser submetido para execução.
 *
 * É usado como tipo do atributo <code>JobTemplate::jobSubmissionState</code>.
 */
enum JobSubmissionState {
   HOLD_STATE, /**< \brief Pode ser enfileirado, mas não está elegível para execução */
   ACTIVE_STATE /**< \brief Está elegível para execução */
};

/**
 * \brief Usado no JobTemplate para indicar o valor do atributo <code>transferFiles</code>.
 * Possui três atributos, que indicam o stage in ou out dos arquivos de
 * entrada padrão, saída padrão e erro padrão, respectivamente.
 */
struct FileTransferMode {
   boolean transferInputStream; /**< \brief Se o arquivo de entrada padrão será transferido */
   boolean transferOutputStream; /**< \brief Se o arquivo de saída padrão será transferido */
   boolean transferErrorStream; /**< \brief Se o arquivo de erros padrão será transferido */
};

/**
 * \brief Usado para representar a versão da implementação do DRMAA.
 * A versão é recuperada como um atributo de <code>Session</code>.
 * Equivale a uma representação de versão no formato <major>.<minor>.
 */
struct Version {
   long major; /**< \brief O componente primário do número versão */
   long minor; /**< \brief O componente secundário do número da versão */
};

/**
 * \brief Falha na inicialização devido a uma sessão DRMAA existente.
 */
```

```
exception AlreadyActiveSessionException {string message;};
/**
 * \brief O usuário não está autorizado a executar a operação requisitada.
 */
exception AuthorizationException {string message;};
/**
 * \brief O valor do atributo tem um conflito com uma ou mais propriedades
 * previamente atribuídas.
 */
exception ConflictingAttributeValuesException {string message;};
/**
 * \brief A implementação DRMAA não pode usar o valor default de contato para
 * conectar com o sistema DRM.
 */
exception DefaultContactStringException {string message;};
/**
 * \brief O sistema DRM rejeitou o job devido a configuração do sistema DRM ou
 * das propriedades atribuídas no JobTemplate.
 */
exception DeniedByDrmException {string message;};
/**
 * \brief Não foi possível contactar o sistema DRM.
 */
exception DrmCommunicationException {string message;};
/**
 * \brief Um problema foi encontrado na tentativa de finalização da sessão.
 */
exception DrmsExitException {string message;};
/**
 * \brief Um problema foi encontrado na tentativa de iniciar uma sessão.
 */
exception DrmsInitException {string message;};
/**
 * \brief As chamadas aos métodos wait() ou synchronize() da interface da Sessão
 * retornaram antes de todos os jobs selecionados ficarem no estado DONE ou FAILED.
 */
exception ExitTimeoutException {string message;};
/**
 * \brief O job não pode mudar para o estado HOLD.
 */
exception HoldInconsistentStateException {string message;};
```

```
/**
 * \brief A instância do JonInfo não está no estato apropriado para o tipo
 * de operação requisitada.
 */
exception IllegalStateException {string message;};
/**
 * \brief Exceção lançada quando ocorre um erro inesperado ou interno do DRMAA.
 */
exception InternalException {string message;};
/**
 * \brief Um valor de parâmetro é inválido, como por exemplo, no caso de ser de
 * um tipo errado.
 */
exception InvalidArgumentException {string message;};
/**
 * \brief Exceção lançada quando o formato de uma propriedade do JobTemplate é inválido,
 * por exemplo, um time stamp com formato inválido.
 */
exception InvalidAttributeFormatException {string message;};
/**
 * \brief Exceção lançada quando o valor de uma propriedade do JobTemplate é inválida.
 */
exception InvalidAttributeValueException {string message;};
/**
 * \brief Exceção lançada quando o parâmetro contato é inválido.
 */
exception InvalidContactStringException {string message;};
/**
 * \brief O job correspondente a um determinado id não existe ou já foi liberado
 * por uma chamada ao método Session::synchronize() com dispose==TRUE.
 */
exception InvalidJobException {string message;};
/**
 * \brief O JobTemplate não está válido quando, por exemplo, foi criado incorretamente
 * sem utilizar o método Session::createJobTemplate(), ou quando já foi removido
 * por uma chamada ao método Session::deleteJobTemplate().
 */
exception InvalidJobTemplateException {string message;};
/**
 * \brief A chamada falhou porque não há uma sessão ativa.
 */
```

```
exception NoActiveSessionException {string message;};
/**
 * \brief Não foi fornecido ou selecionado um contato default para o sistema DRM.
 * O DRMAA requer que uma string default de contato seja selecionada quando há
 * mais de uma possível implementação DRMAA disponível.
 */
exception NoDefaultContactStringSelectedException {string message;};
/**
 * \brief Pode ser lançada a qualquer momento quando a implementação DRMAA executar
 * sem memória livre suficiente.
 */
exception OutOfMemoryException {string message;};
/**
 * \brief O job não está no estado HOLD e, portanto, não pode ser liberado.
 */
exception ReleaseInconsistentStateException {string message;};
/**
 * \brief O job não está em um estado suspenso (ex, *_SUSPENDED) e, portanto,
 * não pode ser ativado.
 */
exception ResumeInconsistentStateException {string message;};
/**
 * \brief O job não está em um estado que possa ser suspenso.
 */
exception SuspendInconsistentStateException {string message;};
/**
 * \brief o sistema DRM rejeitou a operação, possivelmente devido a uma sobrecarga
 * de requisições. Uma nova tentativa pode ter sucesso.
 */
exception TryLaterException {string message;};
/**
 * \brief O atributo fornecido ao JobTemplate não é suportado pela implementação
 * corrente do DRMAA.
 */
exception UnsupportedAttributeException {string message;};

/**
 * \brief Usado para representar atributos <code>Time Stamp</code> do job template.
 * Possui uma implementação nativa.
 */
native PartialTimestamp;
```

```
/**
 * \brief Informações sumarizadas a respeito do resultado da execução de um job.
 * Pelo <code>Job Info</code> o cliente pode descobrir informações sobre o consumo
 * de resursos e o código de término da execução do job.
 */
valuetype JobInfo {
  /**
   * \brief O identificador do job finalizado
   */
  public string jobId;
  /**
   * \brief Dados sobre o uso de recursos pelo job finalizado.
   * Os dados retornados pelo OpenDreams são:
   * cpu-time:
   * elapsed-time:
   * user-time:
   */
  public Dictionary resourceUsage;
  /**
   * \brief Se verdadeiro, significa que o job terminou normalmente e existem
   * informações mais detalhadas sobre seu término no atributo exitStatus.
   */
  public boolean hasExited;
  /**
   * \brief Se <code>hasExit</code> é verdadeiro, possui o código de finalização
   * informado pelo sistema operacional onde o job executou.
   * O valor zero indica que o job terminou com sucesso.
   * Valores diferentes de zero são indicativos de término com falha.
   * No OpenDreams indica se o job terminou com sucesso (EventType.SUCCESS) ou
   * não.
   */
  public long exitStatus;
  /**
   * \brief Se verdadeiro, significa que o job terminou devido ao recebimento
   * de um signal do sistema operacional e existe informação sobre esse signal
   * no atributo terminatingSignal.
   * Esse atributo não é usado no OpenDreams e possui o valor sempre falso.
   */
  public boolean hasSignaled;
  /**
```

```
    * \brief Se <code>hasSignaled</code> é verdadeiro, possui o código do sinal
    * que causou o término do job.
    * Esse atributo não é usado no OpenDreams;
    */
   public string terminatingSignal;
   /**
    * \brief Se <code>hasSignaled</code> é verdadeiro, esse atributo é verdadeiro
    * se a <i>core image</i> do job foi criada.
    * Esse atributo não é usado no OpenDreams e possui o valor sempre falso.
    */
   public boolean hasCoreDump;
   /**
    * \brief Se verdadeiro, significa que o job foi abortado antes de entrar em
    * execução.
    * É usado no OpenDreams para indicar que o job foi abortado (EventType.KILLED).
    */
   public boolean wasAborted;
};

/**
 * \brief Um <code>Job Template</code> define os atributos necessários a
 * submissão de um job para execução.
 *
 * As instâncias de um job template são criadas pela sessão ativa, através do
 * método <code>Session::createJobTemplate()</code>.
 * Uma aplicação DRMAA obtém um job template, altera os valores dos atributos
 * desse job template e o retorna para o serviço na requisição de execução de
 * jobs. Após usar um job template, a aplicação deve chamar o método
 * <code>Session::deleteJobTemplate().
 */
valuetype JobTemplate{
   /**
    * \brief Representa o <i>home directory</i> do usuário.
    * Serve para usar na definição dos <i>paths</i> nos atributos
    * <code>workingDirectory</code>, <code>inputPath</code>,
    * <code>outputPath</code>, e <code>errorPath</code>.
    */
   const string HOME_DIRECTORY = "$drmaa_hd_ph$";
   /**
    * \brief Representa o diretório corrente de trabalho.
```

```
  * Serve para usar na definição dos <i>paths</i> nos atributos
  * <code>inputPath</code>, <code>outputPath</code> e <code>errorPath</code>.
  */
const string WORKING_DIRECTORY = "$drmaa_wd_ph$";
/**
  * \brief Representa o índice paramétrico do job.
  * Serve para usar na definição dos <i>paths</i> nos atributos
  * <code>workingDirectory</code>, <code>inputPath</code>,
  * <code>outputPath</code>, e <code>errorPath</code>.
  */
const string PARAMETRIC_INDEX = "$drmaa_incr_ph$";
/**
  * \brief O comando a ser executado
  */
public string remoteCommand;
/**
  * \brief Os argumentos command-line para execução do comando.
  */
public OrderedStringList args;
/**
  * \brief O estado do job no momento da submissão.
  */
public JobSubmissionState jobSubmissionState;
/**
  * \brief Valores de variáveis de ambiente para serem usados na máquina de
  * execução.
  * Os valores devem sobreescrever os valores das variáveis no ambiente remoto
  * no caso de haver colisão de nomes. Se não for possível, o comportamento
  * é dependente da implementação do DRMS.
  * <b>Esse atributo não é implementado no OpenDreams.</b>
  */
public Dictionary jobEnvironment;
/**
  * \brief Especifica o diretório onde o job é executado.
  * Se esse atributo não estiver configurado, o comportamento é dependente da
  * implementação do DRMS.
  * O valor desse atributo é relativo ao sistema de arquivo na máquina de execução.
  * Esse valor pode usar as constantes HOME_DIRECTORY ou PARAMETRIC_INDEX na sua
  * definição. No caso de iniciar com HOME_DIRECTORY significa que a parte
  * restante é relativa ao diretório <i>home</i> do usuário. A constante
  * PARAMETRIC_INDEX pode ser usada em qualquer posição da definição do diretório
```

```
  * e é substituída pelo índice relativo ao job nas execuções paramétricas.
  * <b>Esse atributo não é implementado no OpenDreams.</b>
  */
public string workingDirectory;
/**
  * \brief Define o uso de recursos ou políticas de acordo com categorias de
  * jobs criadas pelo administrador do DRMS.
  * Depende da implementação do DRMS. Os administradores podem criar uma categoria de job
  * adequada para uma aplicação que usa o DRMS. A implementação DRMAA pode usar
  * a categoria especificada no job template para gerenciar recursos e requisitos
  * dos jobs nessa categoria.
  * No OpenDreams, os seguintes valores serão válidos: CSBase ou System.
  * A categoria CSBase identifica a execução de algoritmos do repositório.
  * A categoria System serviria para atender a execução de comandos do sistema,
  * de acordo com o uso comum do DRMAA.
  * Por enquanto, o OpenDreams somente reconhece a categoria CSBase.
  */
public string jobCategory;
/**
  * \brief Define o uso de recursos ou políticas de acordo com o valor passado
  * pelos usuário final do sistema.
  * Depende da implementação do DRMS.
  * <b>Esse atributo não é implementado no OpenDreams.</b>
  */
public string nativeSpecification;
/**
  * \brief Lista de emails usados para notificar o status do job e sua
  * finalização.
  */
public StringList email;
/**
  * \brief Indica se o envio de emails deve estar bloqueado, por default.
  * Se o valor for true, o envio de email é bloqueado. Se o valor for false,
  * o envio de email depende da configuração do DRMS.
  */
public boolean blockEmail;
/**
  * \brief Especifica uma data/hora a partir da qual o job pode ser elegível
  * para execução.
  * TODO AINDA NÃO SEI COMO DEFINIR O TIPO.
  * <b>Esse atributo não é implementado no OpenDreams.</b>
```

```
  */
 /* public PartialTimestamp startTime;*/
 /**
  * \brief O nome de um job, fornecido pelo cliente, deve ser formado de
  * caracteres alfanuméricos e '_'.
  * <b>Esse atributo não é implementado no OpenDreams.</b>
  */
 public string jobName;
 /**
  * \brief Especifica como direcionar a entrada padrão para um arquivo.
  * Possui o formato[hostname]:file_path.
  * Se o atributo transferFiles for implementado e possui um valor verdadeiro
  * em FileTransferMode::inputStream, o arquivo de entrada deve ser recuperado
  * a partir do hostname ou do host de submissão se o hostname não for definido.
  * Se o atributo transferFiles não for implementado ou possui um valor falso
  * em FileTransferMode::inputStream, assume-se que o arquivo de entrada sempre
  * estará na máquina de execução do job, independente de um hostname ter sido
  * especificado.
  * O valor do atributo pode usar o <i>placeholder</i> PARAMETRIC_INDEX em qualquer
  * posição para ser substituído pelo índice paramétrico de execução de um job.
  * O valor do atributo pode usar o <i>placeholder</i> HOME_DIRECTORY no início,
  * indicando que o restante é um caminho relativo ao diretório do home do usuário.
  * O valor do atributo pode usar o <i>placeholder</i> WORKING_DIRECTORY no início,
  * indicando que o restante é um caminho relativo ao diretório de trabalho.
  * <b>Esse atributo não é implementado no OpenDreams.</b>
  */
 public string inputPath;
 /**
  * \brief Especifica como direcionar a saída padrão para um arquivo.
  * Possui o formato[hostname]:file_path.
  * Se o atributo transferFiles for implementado e possui um valor verdadeiro
  * em FileTransferMode::outputStream, o arquivo de saída deve ser transferido
  * para o hostname especificado ou para o host de submissão se o hostname não
  * for definido.
  * Se o atributo transferFiles não for implementado ou possui um valor falso
  * em FileTransferMode::outputStream, assume-se que o arquivo de saída sempre
  * estará na máquina de execução do job, independente de um hostname ter sido
  * especificado.
  * Toda saída enviada para a saída padrão do job deve ser "appended".
  * Se o arquivo não existir no momento da execução do job, ele é criado.
  * O valor do atributo pode usar o <i>placeholder</i> PARAMETRIC_INDEX em qualquer
```

```
  * posição para ser substituído pelo índice paramétrico de execução de um job.
  * O valor do atributo pode usar o <i>placeholder</i> HOME_DIRECTORY no início,
  * indicando que o restante é um caminho relativo ao diretório do home do usuário.
  * O valor do atributo pode usar o <i>placeholder</i> WORKING_DIRECTORY no início,
  * indicando que o restante é um caminho relativo ao diretório de trabalho.
  */
public string outputPath;
/**
  * \brief Especifica como direcionar a saída padrão de erros para um arquivo.
  * Possui o formato[hostname]:file_path.
  * Se o atributo transferFiles for implementado e possui um valor verdadeiro
  * em FileTransferMode::errorStream, o arquivo de erros deve ser transferido
  * para o hostname especificado ou para o host de submissão se o hostname não
  * for definido.
  * Se o atributo transferFiles não for implementado ou possui um valor falso
  * em FileTransferMode::errorStream, assume-se que o arquivo de erros sempre
  * estará na máquina de execução do job, independente de um hostname ter sido
  * especificado.
  * Toda saída enviada para a saída padrão de erros do job deve ser "appended".
  * Se o arquivo não existir no momento da execução do job, ele é criado.
  * O valor do atributo pode usar o <i>placeholder</i> PARAMETRIC_INDEX em qualquer
  * posição para ser substituído pelo índice paramétrico de execução de um job.
  * O valor do atributo pode usar o <i>placeholder</i> HOME_DIRECTORY no início,
  * indicando que o restante é um caminho relativo ao diretório do home do usuário.
  * O valor do atributo pode usar o <i>placeholder</i> WORKING_DIRECTORY no início,
  * indicando que o restante é um caminho relativo ao diretório de trabalho.
  */
public string errorPath;
/**
  * \brief Indica se a saída de erros padrão deve ser junta com a saída padrão,
  * independente do atributo errorPath ter sido especificado.
  * O default é false.
  */
public boolean joinFiles;
/**
  * \brief Especifica se os arquivos de entrada padrão, saída padrão e erro
  * padrão, definidos respectivamente em inputPath, outputPath e errorPath devem
  * ser transferidos tendo como referência o hostname especificado.
  * Se o atributo não for explicitamente atribuído o efeito é o mesmo que
  * usar false em todos os membros do FileTransferMode.
  */
```

```
public FileTransferMode transferFiles;
/**
 * \brief Especifica o tempo limite após o qual o DRMS deve abortar ou terminar o job
 * TODO AINDA NÃO SEI COMO DEFINIR O TIPO.
 * <b>Esse atributo não é implementado no OpenDreams.</b>
 */
/* public PartialTimestamp deadlineTime; */
/**
 * \brief Especifica o tempo de parede limite após o qual o DRMS deve abortar
 * ou terminar o job.
 * <b>Esse atributo não é implementado no OpenDreams.</b>
 */
public TimeAmount hardWallClockTimeLimit;
/**
 * \brief Especifica uma estimativa de qual seria o tempo de parede limite
 * para o job terminar. É usado para ajudar o escalonamento do job.
 * A implementação pode, por exemplo, aplicar uma penalidade aos jobs que não
 * atenderem esse limite.
 * <b>Esse atributo não é implementado no OpenDreams.</b>
 */
public TimeAmount softWallClockTimeLimit;
/**
 * \brief Especifica o tempo de execução limite após o qual o DRMS deve
 * abortar ou terminar o job.
 * <b>Esse atributo não é implementado no OpenDreams.</b>
 */
public TimeAmount hardRunDurationLimit;
/**
 * \brief Especifica uma estimativa de qual seria o tempo de execução limite
 * para o job terminar. É usado para ajudar o escalonamento no job. A
 * implementação pode, por exemplo, aplicar uma penalidade aos jobs que não
 * atenderem esse limite.
 * <b>Esse atributo não é implementado no OpenDreams.</b>
 */
public TimeAmount softRunDurationLimit;
/**
 * Especifica a lista dos nomes de atributos definidos no JobTemplate,
 * incluindo os atributos mandatórios e opcionais bem como os atributos
 * específicos da implementação DRMAA.
 */
```

```
    readonly attribute StringList attributeNames;
};

/**
 * Representa uma sessão DRMAA.
 */
interface Session{
        const long long TIMEOUT_WAIT_FOREVER = -1;
        const long long TIMEOUT_NO_WAIT = 0;
        const string JOB_IDS_SESSION_ANY = "DRMAA_JOB_IDS_SESSION_ANY";
        const string JOB_IDS_SESSION_ALL = "DRMAA_JOB_IDS_SESSION_ALL";
    /**
     * \brief Faz a inicialização de uma sessão, tornando-a ativa.
     *
     * Esse método tem que ser chamado antes de qualquer outra chamada na
     * sessão, com exceção dos métodos <code>contact</code>,
     * <code>drmsInfo</code> e <code>drmaaImplementation</code> definidos na
     * interface <code>Session</code>.
     *
     * \param [in] contactString Especifica qual sistema DRMS deve ser usado, no
     * caso de haver mais de um. Quando vazio, o DRMS default é usado.
     *
     * \exception DrmsInitException Falha durante a inicialização da sessão.
     * \exception InvalidContactStringException Caso o parâmetro contactString
     * seja inválido.
     * \exception AlreadyActiveSessionException Caso esse método seja chamado
     * mais de uma vez.
     * \exception DefaultContactStringException Caso o
     * <code>contactString</code> vazio e o DRMS default não
     * consiga ser conectado.
     * \exception NoDefaultContactStringSelectedException Caso o
     * <code>contactString</code> seja vazio e exista mais de um DRMS
     * disponível.
     * \excepion OutOfMemoryException Caso o sistema DRMS não possua memória
     * suficiente para executar essa operação.
     * \excepion DrmCommunicationException Falha na conexão com o sistema DRMS.
     * \exception AuthorizationException Caso o usuário não possua permissão
     * para executar essa operação.
     * \exception InvalidArgumentException Valor de argumento inválido.
     * \exception InternalException Falha na execução da operação no DRMS.
     */
```

```
        void init(in string contactString)
                raises ( DrmsInitException,
        InvalidContactStringException,
        AlreadyActiveSessionException,
        DefaultContactStringException,
        NoDefaultContactStringSelectedException,
        OutOfMemoryException,
        DrmCommunicationException,
        AuthorizationException,
        InvalidArgumentException,
        InternalException);
    /**
     * \brief Faz a finalização de uma sessão ativa.
     *
     * Esse método termina a conexão com o DRMS e finaliza a sessão ativa.
     * A chamada a esse método não afeta qualquer job já submetido (na fila ou
     * em execução). Qualquer instância de job template que ainda não tenha sido
     * removida, se torna inválida após esse método ser executado, mesmo depois
     * de uma subsequente chamada ao método <code>init()</code>.
     *
     * \exception DrmsExitException Falha durante a finalização da sessão.
     * \exception NoActiveSessionException Se a sessão ainda não foi inicializada ou
     * se a sessão já foi finalizada antes.
     * \excepion DrmCommunicationException Falha na conexão com o sistema DRMS.
     * \excepion OutOfMemoryException Caso o sistema DRMS não possua memória
     * suficiente para executar essa operação.
     * \exception AuthorizationException Caso o usuário não possua permissão
     * para executar essa operação.
     * \exception InternalException Falha na execução da operação no DRMS.
     */
        void exit()
                raises ( DrmsExitException,
        NoActiveSessionException,
        DrmCommunicationException,
        AuthorizationException,
        OutOfMemoryException,
        InternalException);
    /**
     * \brief Faz a criação de um job template.
     *
     * O job template é usado para
```

```
    * definir as características para submissão de jobs. Uma vez que o job
    * template tenha sido criado, ele deve ser posteriormente removido
    * (usando o método <code>deleteJobTemplate()</code>) quando não for mais
    * necessário.
    * \return Um novo job template.
    * \excepion DrmCommunicationException Falha na conexão com o sistema DRMS.
    * \exception NoActiveSessionException Se a sessão ainda não foi inicializada
    * ou se a sessão já foi finalizada.
    * \excepion OutOfMemoryException Caso o sistema DRMS não possua memória
    * suficiente para executar essa operação.
    * \exception AuthorizationException Caso o usuário não possua permissão
    * para executar essa operação.
    * \exception InternalException Falha na execução da operação no DRMS.
    */
        JobTemplate createJobTemplate()
                raises ( DrmCommunicationException,
        NoActiveSessionException,
        OutOfMemoryException,
        AuthorizationException,
        InternalException);
    /**
    * \brief Libera um job template.
    *
    * Esse método não possui nenhum efeito em jobs em execução.
    * Somente pode ser usado em job templates criados pelo método
    * <code>createJobTemplate()</code> e que ainda não foram liberados.
    *
    * \param [in] jobTemplate_ O job template a ser liberado.
    *
    * \excepion DrmCommunicationException Falha na conexão com o sistema DRMS.
    * \exception NoActiveSessionException Se a sessão ainda não foi inicializada
    * ou se a sessão já foi finalizada.
    * \excepion OutOfMemoryException Caso o sistema DRMS não possua memória
    * suficiente para executar essa operação.
    * \exception AuthorizationException Caso o usuário não possua permissão
    * para executar essa operação.
    * \exception InvalidJobTemplateException se o job não foi criado adequadamente
    * ou se o job template já foi liberado.
    * \exception InternalException Falha na execução da operação no DRMS.
    */
        void deleteJobTemplate(in JobTemplate jobTemplate_)
```

```
                raises ( DrmCommunicationException,
        NoActiveSessionException,
        OutOfMemoryException,
        AuthorizationException,
        InvalidArgumentException,
        InvalidJobTemplateException,
        InternalException);
    /**
     * \brief Submete um job para execução.
     *
     * O job submetido está definido pelo job template, passado como parâmetro.
     * O identificador retornado deve ser o mesmo fornecido pelo DRMS responsável
     * pelo agendamento do job.
     *
     * \param [in] jobTemplate_ O job template que define o job submetido para
     * execução.
     *
     * \return Um identificador do job proveniente do DRMS.
     *
     * \excepion TryLaterException a requisição não pode ser processada devido
     * a uma carga excessiva do sistema.
     * \excepion DeniedByDrmException o DRMS rejeitou o job devido a configuração
     * do job template fornecida ou da configuração do próprio DRMS.
     * \excepion DrmCommunicationException Falha na conexão com o sistema DRMS.
     * \exception AuthorizationException Caso o usuário não possua permissão
     * para executar essa operação.
     * \exception InvalidJobTemplateException se o job não foi criado adequadamente
     * ou se o job template já foi liberado.
     * \exception NoActiveSessionException Se a sessão ainda não foi inicializada
     * ou se a sessão já foi finalizada.
     * \excepion OutOfMemoryException Caso o sistema DRMS não possua memória
     * suficiente para executar essa operação.
     * \exception InvalidArgumentException Valor de argumento inválido.
     * \exception InternalException Falha na execução da operação no DRMS.
     */
        string runJob(in JobTemplate jobTemplate_)
                raises ( TryLaterException,
        DeniedByDrmException,
        DrmCommunicationException,
        AuthorizationException,
        InvalidJobTemplateException,
```

```
        NoActiveSessionException,
        OutOfMemoryException,
        InvalidArgumentException,
        InternalException);
    /**
     * \brief Operação ainda não implementada.
     */
        StringList runBulkJobs( in JobTemplate jobTemplate_,
                in long beginIndex,
                in long endIndex,
                in long step)
                raises ( TryLaterException,
        DeniedByDrmException,
        DrmCommunicationException,
        AuthorizationException,
        InvalidJobTemplateException,
        NoActiveSessionException,
        OutOfMemoryException,
        InvalidArgumentException,
        InternalException);
    /**
     * \brief Operação ainda não implementada.
     */
        void control( in string jobName,
                in JobControlAction operation)
                raises ( DrmCommunicationException,
        AuthorizationException,
        ResumeInconsistentStateException,
        SuspendInconsistentStateException,
        HoldInconsistentStateException,
        ReleaseInconsistentStateException,
        InvalidJobException,
        NoActiveSessionException,
        OutOfMemoryException,
        InvalidArgumentException,
        InternalException);
    /**
     * \brief Operação ainda não implementada.
     */
        void synchronize( in StringList jobList,
                in long long timeout,
```

```
                in boolean dispose)
                raises ( DrmCommunicationException,
        AuthorizationException,
        ExitTimeoutException,
        InvalidJobException,
        NoActiveSessionException,
        OutOfMemoryException,
        InvalidArgumentException,
        InternalException);
    /**
     * \brief Aguarda um único job completar sua execução e retorna o status do
     * com a condição de término e as informações de uso de recursos pelo job.
     *
     * Se for chamado com o JOB_IDS_SESSION_ANY como parâmetro no
     * job name e não houver jobs ativos na sessão, a exceção InvalidJobException
     * é lançada.
     *
     * Para um mesmo job name, apenas uma chamada ao método wait ocorre com
     * sucesso. Para qualquer outra que esteja aguardando uma chamada wait ao
     * mesmo job, é lançada uma exceção InvalidJobException.
     *
     * Em um ambiente multi-threaded com chamadas ao método wait usando
     * JOB_IDS_SESSION_ANY, apenas a thread ativa recebe o resultado
     * do wait enquanto as outras continuam esperando por outro job. Se não
     * houver mais jobs submetidos, as outras threads recebem a exceção
     * InvalidJobException.
     *
     * Se uma thread A está aguardando por um determinado job e uma outra thread,
     * B, aguardando pelo mesmo job ou por JOB_IDS_SESSION_ANY, recebe a informação
     * que esse job terminou, a thread A deve receber a exceção InvalidJobException.
     *
     * \param [in] jobName O identificador de um job ou a constante
     * Session.JOB_IDS_SESSION_ANY que representa qualquer job da sessão ativa
     * \param [in] timeout O tempo máximo (em segundos) a ser esperado pela
     * finalização dos job ou as constantes TIMEOUT_WAIT_FOREVER e TIMEOUT_NO_WAIT.
     *
     * \return Um objeto JobInfo, que encapsula o estado de finalização do job
     * e as informações sobre o uso de recursos durante sua execução.
     */
        JobInfo wait( in string jobName,
                in long long timeout)
```

```
                raises ( DrmCommunicationException,
        AuthorizationException,
        ExitTimeoutException,
        InvalidJobException,
        NoActiveSessionException,
        OutOfMemoryException,
        InvalidArgumentException,
        InternalException);
    /**
     * \brief Retorna o estado do job identificado pelo parâmetro jobName.
*
   * O OpenDreams verifica se o job já finalizou usando um cache próprio que
   * mantém as informações sobre jobs que finalizaram. Se não encontrar a
   * informação nesse cache, pede ao serviço {@link CommandPersistenceService}
   * que possui persistido em arquivo as informações sobre todos os jobs
   * submetidos.
   *
   * É importante observar que nem sempre o estado de retorno DONE (finalizado
   * normalmente) significa que o comando terminou com código de retorno igual a
   * zero. Se o configurador do algoritmo não tiver definido a opção que captura
   * a saída do comando, o estado retornado pode ser DONE também. Se o
   * configurador tiver definido a opção que captura a saída do comando e se o
   * comando terminar com um valor diferente de zero, o estado é retornado como
   * FAILED.
   *
   * Os seguintes estados podem ser retornados:
   *
   * JobState.QUEUED_ACTIVE: se o comando estiver na fila.
   *
   * JobState.RUNNING: se o comando já tiver sido removido da fila para ser
   * executado.
   *
   * JobState.DONE: se o comando tiver finalizado e o código de saída do job não
   * tiver sido capturado ou se o código de saída do job tiver sido capturado
   * com valor zero.

   * JobState.FAILED: se o comando tiver finalizado e o código de saída do job
   * tiver sido capturado com valor diferente de zero ou se houve falha na
   * inicialização do comando ou se ele foi cancelado ou se houve falha na
   * obtenção do código de saída.
```

```
*
* JobState.UNDETERMINED: em qualquer condição que não se enquadra nas
* descritas acima.
*
* Na implementação do OpenDreams, os seguintes estados não são tratados:
* SYSTEM_ON_HOLD, USER_ON_HOLD, USER_SYSTEM_ON_HOLD, SYSTEM_SUSPENDED,
* USER_SUSPENDED e USER_SYSTEM_SUSPENDED.
*
*/
        JobState jobStatus(in string jobName)
                raises ( DrmCommunicationException,
        AuthorizationException,
        InvalidJobException,
        NoActiveSessionException,
        OutOfMemoryException,
        InvalidArgumentException,
        InternalException);
 /**
  * \brief Se esse atributo for lido antes de uma chamada ao método init(),
  * retorna uma string contendo a lista (separada por vírgula) com os nomes
  * das instalações DRMAA disponíveis. Cada um desses nomes representa
  * uma instalação específica de um sistema DRM específico (por exemplo, a versão
  * do CSGRid executando em um determinado IP e porta).
  * Se esse atributo for lido depois de uma chamada ao método init(),
  * retorna uma string contendo a instalação DRMAA associada a sessão ativa.
  */
        readonly attribute string contact;
        /**
  * \brief Contém uma instância de <code>Version</code> que representa o número
  * da versão da implementação do DRMAA.
  */
        readonly attribute DRMAA::Version version;
        /**
  * \brief Se esse atributo for lido antes de uma chamada ao método init(),
  * retorna uma string contendo a lista (separada por vírgula) com identificadores
  * dos DRMS disponíveis. Um identificador de DRMS denota um tipo específico
  * de DRMS (por exemplo, Sun Grid Engine, CSGrid).
  * Se esse atributo for lido depois de uma chamada ao método init(),
  * retorna uma string contendo o identificador do DRMS associado a sessão ativa.
  */
        readonly attribute string drmsInfo;
```

```
    /**
     * \brief Se esse atributo for lido antes de uma chamada ao método init(),
     * retorna uma string contendo a lista (separada por vírgula) com implementações
     * DRMAA disponíveis. A string de uma implementação DRMAA denota uma versão
     * específica do DRMS (por exemplo, Condor v6.6).
     * Se esse atributo for lido depois de uma chamada ao método init(),
     * retorna uma string contendo a implementação DRMAA associada a sessão ativa.
     */
        readonly attribute string drmaaImplementation;
};

}; // DRMAA

}; // openbus

}; // tecgraf

#endif
```

## A.2 opendreams.idl

```
/**
 * @mainpage
 * @section intro Introdução
 * <p>
 * Descrever em linhas gerais o opendreams
 * O OpenDreams é a interface de um serviço para submissão, monitoração e controle de execução remota de co
 * submetidos para execução em sistemas CSBase. Comandos para execução pelo OpenDreams são, portanto,
 * algoritmos ou fluxos instalados em um servidor CSBase.
 * A API do OpenDreams é baseada no padrão DRMAA.
 */

/**
 * @file opendreams.idl
 * Arquivo de especificação da IDL para o OpenDreams.
 * @author Tecgraf PUC−Rio
 */

#ifndef _OPENDREAMS_IDL
```

```
#define _OPENDREAMS_IDL

#include "drmaa.idl"

/**
 * Módulo OpenDreams.
 */
module tecgraf {

module openbus {

module opendreams {

/**
 * \brief O JobTemplate estendido para o OpenDreams.
 * O job template do OpenDreams possui alguns atributos adicionais que são
 * específicos para a execução de algoritmos e de fluxo de algoritmos
 * cadastrados.
 */
valuetype OpenDreamsJobTemplate : DRMAA::JobTemplate {
    /**
     * \brief O identificador unívoco de um job template
     */
    private long long id;
    /**
     * \brief Os parâmetros para execução do job
     */
    public DRMAA::Dictionary jobParameters;
    /**
     * \brief A descrição de um job submetido
     */
    public string jobDescription;
    /**
     * \brief A prioridade do job para escalonamento
     */
    public short jobPriority;
    /**
     * \brief Número de processos para execução paralela ou distribuída
     */
    public long numberOfProcesses;
};
```

```
/**
 * \brief O tipo de finalização do Job, de acordo com o que pode acontecer
 * durante a execução de um algoritmo no CSBase.
 */
enum FinalizationType {
  UNDEFINED, /**< \brief Quando não há um mapeamento para o tipo de
                          finalização do comando no CSBase com um tipo
                          previsto na FinalizationType do OpenDreams */
  NO_FINALIZATION_INFO, /**< \brief Não há informação sobre a finalização do
                          comando. */
  NOT_FINISHED, /**< \brief O comando não está no estado finalizado. */
  UNKNOWN, /**< \brief Não se sabe em que condições o comando terminou,
                          pois o comando é antigo. É utilizado para comandos
                          persistidos antes da criação desta classificação. */
  END, /**< \brief Não se sabe em que condições o comando terminou,
                          pois o comando não informa. É utilizado para comandos
                          que não informam o código de saída durante a execução. */
  SUCCESS, /**< \brief O comando terminou com sucesso (retornou código
                          de saída de execução igual a zero) */
  EXECUTION_ERROR, /**< \brief O comando terminou com erro (retornou código de
                          saída de execução diferente de zero) */
  FAILED, /**< \brief Houve uma falha na inicialização do comando. Por
                          exemplo, nao existe um servidor disponivel para execucao.*/
  KILLED, /**< \brief O comando foi cancelado. */
  LOST, /**< \brief O comando foi perdido. */
  NO_EXIT_CODE /**< \brief O comando terminou, mas não foi possível ler o
                          código de retorno, para determinar se foi bem sucedido. */
};

/**
 * \brief O JobInfo estendido para fornecer informações sobre a execução
 * no OpenDreams.
 */
valuetype OpenDreamsJobInfo : DRMAA::JobInfo {
  /**
   * \brief O identificador do algoritmo do fluxo que causou o término
   * da execução.
   */
  public string guiltyNodeId;
```

```
    /**
     * \brief O tipo de finalização do job, de acordo com a enumeração
     * definida por FinalizationType. O tipo de finalização indica se o
     * o código de retorno (exitStatus definido no JobInfo) possui um valor
     * válido obtido na finalização do job.
     */
     public FinalizationType finalizationType;
};


/**
 * \brief Interface principal de acesso ao serviço OpenDreams.
 * A interface IOpenDreams cria uma sessão para o usuário dono de um projeto.
 * Essa sessão implementa uma sessão DRMAA e oferece métodos para submissão,
 * controle e monitoração de jobs para execução remota.
 */
interface IOpenDreams{
     DRMAA::Session getSession(in string projectId)
     raises ( DRMAA::AuthorizationException,
                    DRMAA::InternalException);
};

}; // opendreams

}; // openbus

}; // tecgraf

#endif
```

# B Release Notes

## B.1 Versão 1.4.0

As principais mudanças com relação a release anterior são as seguintes:

- Alteração na API do OpenDreamsJobTemplate, definida na IDL opendreams.idl. Os métodos *getter* e *setter* foram removidos e os atributos se tornaram públicos. Dessa forma, a aplicação cliente passar a atribuir os valores diretamente aos atributos do JobTemplate, ao invés de usar métodos para isso. Essa mudança foi efetuada para facilitar a portabilidade da IDL para diferentes linguagens de programação.
- Implementação de uma demo para execução de fluxo de algoritmos.
- Definição do OpenDreamsJobInfo como uma extensão de JobInfo. O OpenDreamsJobInfo tem informações adicionais sobre o término do Job.
- Implementação do método jobStatus da api do DRMAA.