Cortex

Anatomia de um Framewok

Versão 1.1

Cortex: Anatomia de um Framework
Versão 1.0 - 31/12/2010
Versão 1.1 - 03/05/2011

Copyright 2010-2011 - Departamento de Ciência e Tecnologia / Exército Brasileiro
Todos os direitos reservados.

Autores:

Alisson Sandes Palmeira
Thiago Mael de Castro

Índice

- Memória Genética
    - Origens
    - O que o Cortex é
    - O que o Cortex não é
- Uma Cirurgia Exploratória
    - Passo 1 - Reunindo os instrumentos
        - Qt SDK
        - Cortex
    - Passo 2 - Costurando o tecido
    - Passo 3 - Ativando uma sinapse de exemplo
- Primeiro Estímulo Cerebral: Alô, Mundo!
    - Fazendo a incisão
    - Ligando os neurônios
- Visão Geral do Tecido
    - Camadas
    - Estrutura padrão de diretórios
- Dissecando umaSinapse
    - Estrutura padrão de diretórios
    - Arquivo de projeto (.pro)
    - Manifesto
    - Implementação
    - De volta ao exemplo
- Eletrodos de Teste
    - Manifesto
    - Implementação
    - Argumentos
    - Exemplo
- Nas Entranhas do Cortex
    - Carregando as Sinapses
    - Resolução de Dependências
    - Log da Execução

Memória Genética

Origens

O Cortex surgiu em 2009, da necessidade de se reutilizar componentes de um software de Comando e Controle (C2) desenvolvido pelo Exército Brasileiro.

O Programa C2 em Combate, como um software de C2, destina-se a permitir que um comandante militar, em qualquer nível, possa emitir ordens a seus subordinados e acompanhar a sua execução durante uma operação militar. O desenvolvimento teve início em 2003, por determinação do Comandante do Exército, e em 2008 contava com uma série de funcionalidades com grande potencial de reutilização por parte de outros projetos em andamento no próprio Exército.

Uma primeira abordagem foi tornar o C2 em Combate um programa “plugável”, de forma que um novo projeto pudesse agregar novas funcionalidades de acordo com suas necessidades. O problema é que nem todos os projetos precisavem do arcabouço de C2: para alguns, o módulo de informações geográficas era suficiente; para outros, comunicações e criptografia, por exemplo.

Assim, optou-se pelo desenvolvimento de um framework de propósito geral, inspirado na filosofia da então emergente arquitetura orientada a serviços (SOA).

O que o Cortex é

O Cortex é um framework orientado a serviços para desenvolvimento de aplicativos multiplataforma em C++, para desktop. No Cortex, as tradicionais bibliotecas C++ são substituídas por componentes reutilizáveis, com alta coesão e baixo acoplamento, disponiblizados sobre o framework como serviços, segundo a filosofia de arquitetura orientada a serviços (SOA).

O que o Cortex *não* é

O Cortex não é um um framework de Comando e Controle; muito menos destina-se a aplicações exclusivamente militares; decididamente não é um conjunto de bibliotecas utilitárias; tampouco é multilinguagem; e definitivamente não é um framework para web wervices (embora seja possível implementá-los com o Cortex).

Uma Cirurgia Exploratória

Antes de se aprofundar nos conceitos do Cortex, talvez seja interessante dar uma rápida olhada no framework em ação. Cada serviço disponibilizado sobre o framework recebe o nome de sinapse. Esta seção irá guiá-lo na execução de um aplicativo de exemplo chamado “Calculator View”, que é composto de três sinapses.

Passo 1 - Reunindo os instrumentos

Qt SDK

O pré-requisito para o desenvolvimento sobre o Cortex é o Qt SDK, mantido pela Nokia, na versão 4.5.2 (2009.03) ou superior.

O instalador da versão LGPL do Qt SDK pode ser obtida no seguinte endereço: http://qt.nokia.com/downloads. À época da escrita deste guia, a versão mais recente era a 4.7.0, estando a versão Windows disponível no endereço http://get.qt.nokia.com/qtsdk/qt-sdk-win-opensource-2010.05.exe e a versão linux (32 bits) em http://get.qt.nokia.com/qtsdk/qt-sdk-linux-x86-opensource-2010.05.1.bin.

Após o donwload, basta executar e seguir os passos do instalador.

Cortex

A última versão estável do Cortex pode ser obtida no Portal do Software Público Brasileiro, no seguinte endereço: http://www.softwarepublico.gov.br/dotlrn/clubs/cortex/one-community?page_num=2. Após o download, basta descompactar o arquivo.

Se preferir, é possível baixar a versão mais recente a partir do repositório svn: http://svn.softwarepublico.gov.br/svn/cortex/trunk/.

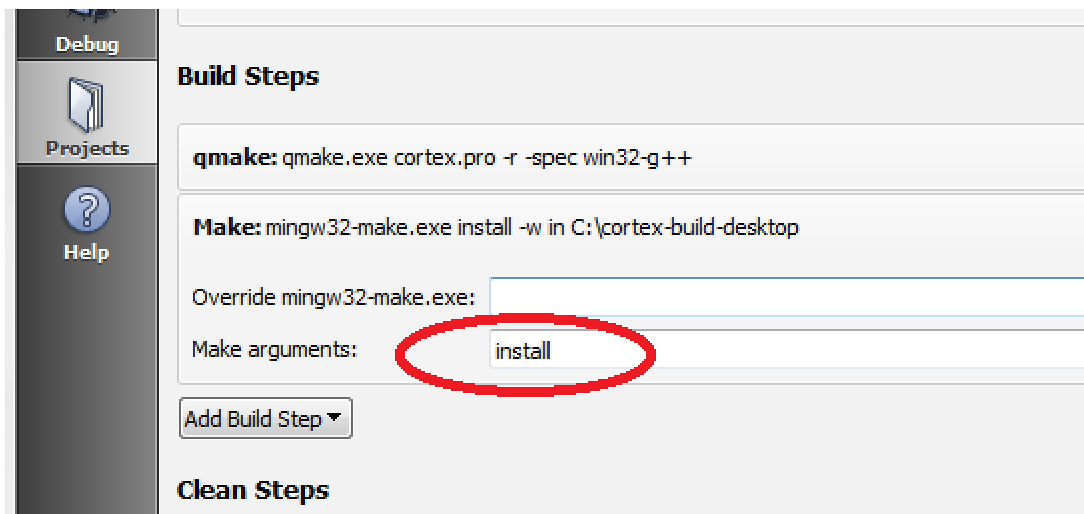
Passo 2 - Costurando o tecido

Execute o **Qt Creator**.

Clique em **“Open Project...”** (Ctrl+O), navegue até o diretório do Cortex e abra o arquivo de projeto **“cortex.pro”**.

Na barra da esquerda, clique no botão **“Projects”**.

Na seção **“Build Steps”**, localize o comando **“Make”** e clique em **“Details”**. Na propriedade **“Make arguments”**, digite a opção **“install”**.

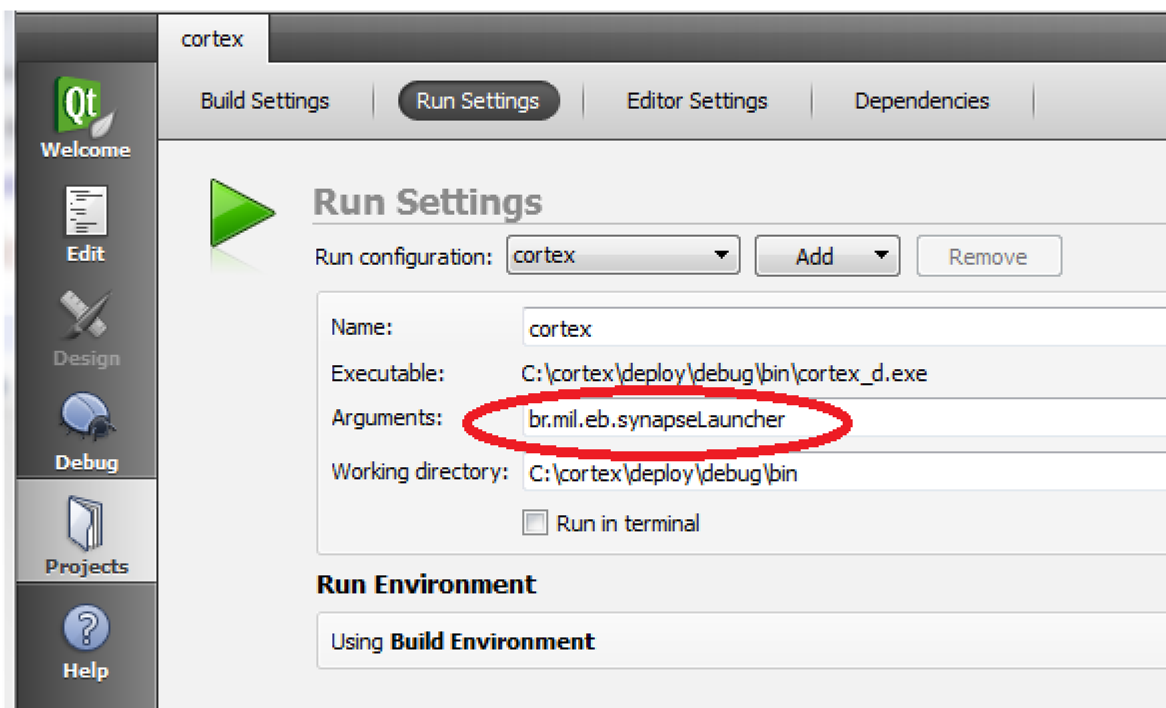


No menu **“Build”**, clique em **“Build All”** (Ctrl+Shift+B).

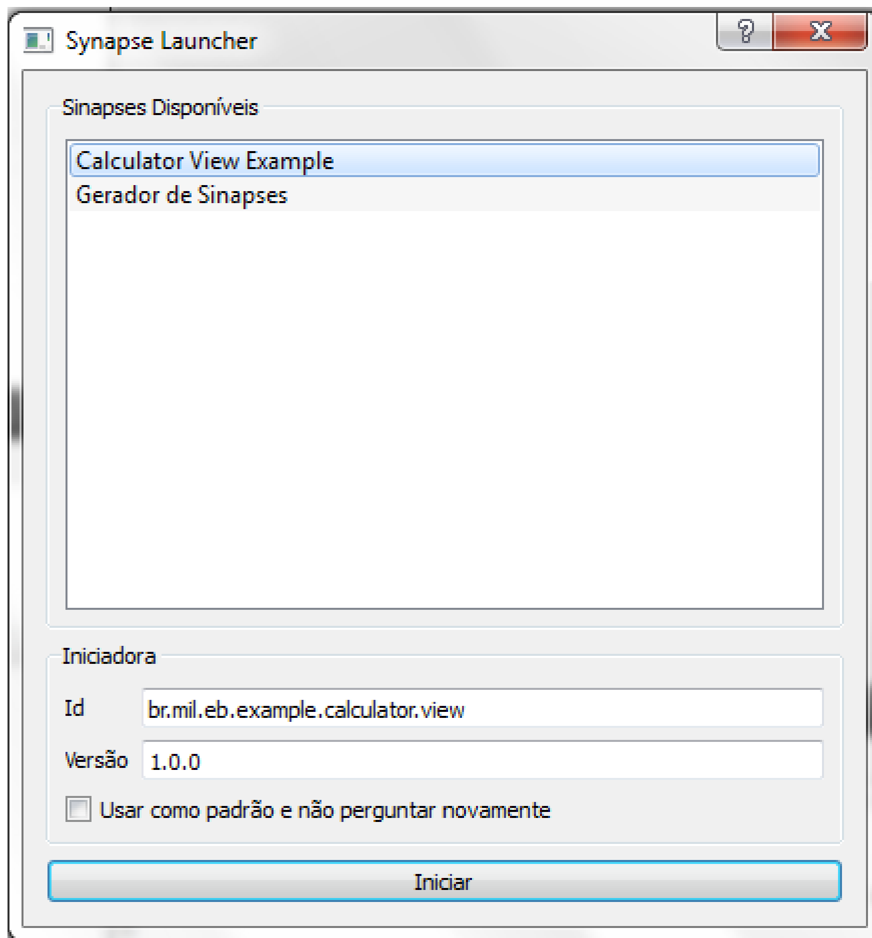
Passo 3 - Ativando uma sinapse de exemplo

Ainda em **“Projects”,** selecione a aba **“Run Settings”**.

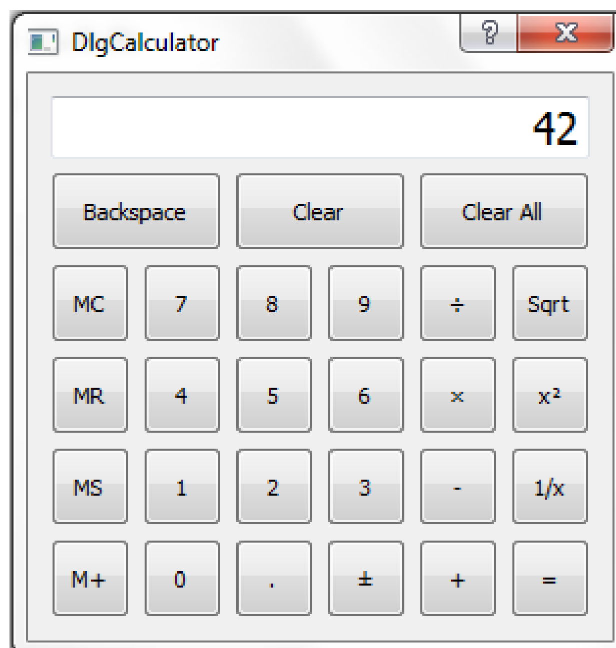
Na propriedade **“Arguments”**, digite **“br.mil.eb.synapseLauncher”** (ATENÇÃO: o Cortex faz distinção entre caracteres maiúsculos e minúsculos).



No menu **“Build”**, clique em **“Run”** (Ctrl+R). Nesse momento, o Cortex iniciará o aplicativo **“Synapse Launcher”**.



Selecione **"Calculator View Example"** e clique em **"Iniciar"**.



Primeiro Estímulo Cerebral: Alô, Mundo!

O que seria deste guia sem o clássico “Alô, Mundo!” revisitado? Esta seção irá conduzi-lo na construção de uma primeira sinapse!

Fazendo a incisão

Há um aplicativo distribuído junto com o Cortex chamado **“Synapse Generator”** que facilita a criação de toda a infraestrutura necessária para o desenvolvimento de uma sinapse.

Execute o **Qt Creator**.

Clique em **“Open Project...”** (Ctrl+O), navegue até o diretório do Cortex e abra o arquivo **“cortex.pro”**.

Na barra da esquerda, clique no botão **“Projects”** e selecione a aba **“Run Settings”**.

Na propriedade **“Arguments”**, digite **“br.mil.eb.synapseLauncher”**, conforme o passo 3 da seção anterior.

No menu **“Build”**, clique em **“Run”** (Ctrl+R). Nesse momento, o Cortex iniciará o aplicativo **“Synapse Launcher”**.

Selecione **“Gerador de Sinapses”** e clique em **“Iniciar”**.

Na tela **“Criação de uma nova sinapse para o Cortex”**, preencha os campos com os seguintes dados e clique em **“Next”**:

- ID: **br.gov.softwarepublico.aloMundo**
- Versão: **1.0.0**
- Título: **Alô, Mundo!**
- Descrição: **Minha primeira sinapse.**

Novo Projeto de Sinapse

Criação de uma nova sinapse para o Cortex

ID: br.gov.softwarepublico.aloMundo

Versão: 1.0.0

Diretório: C:/Users/c2cmb/Desktop/Alisson/cortex/synapses ...

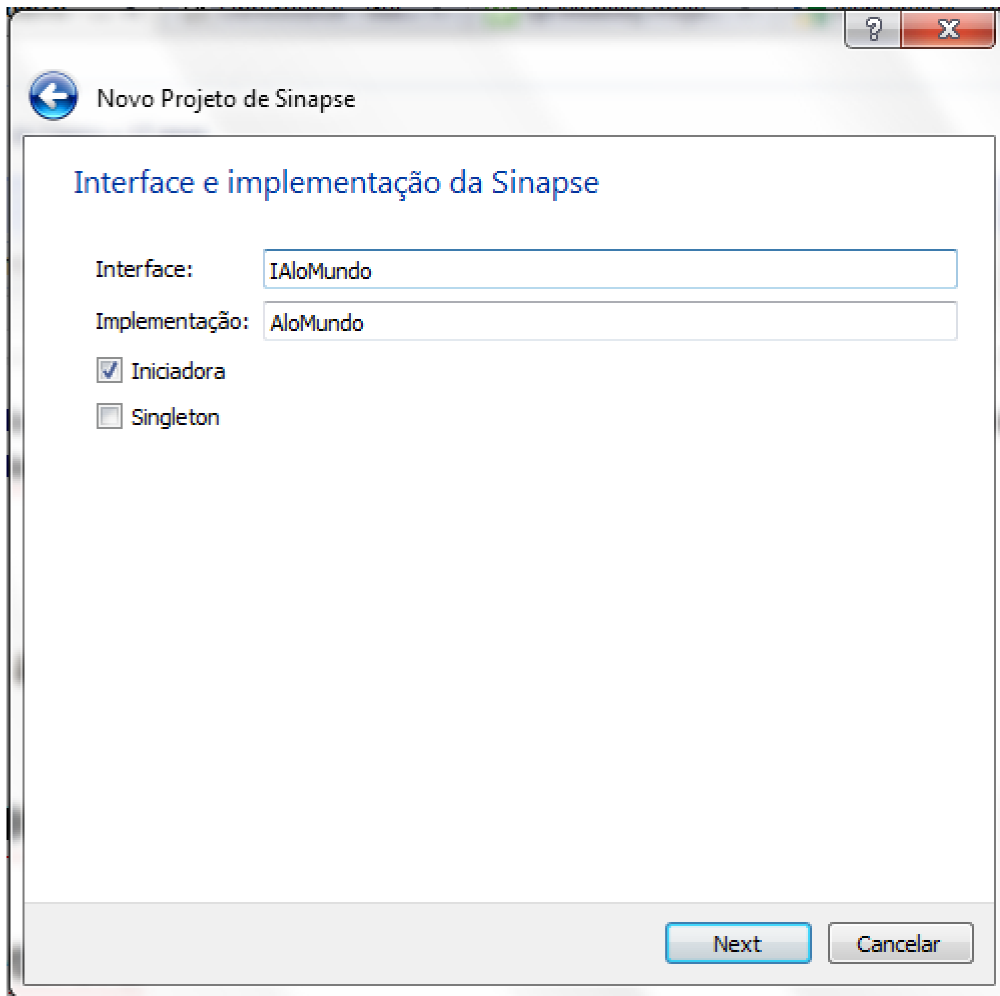
Use o diretório padrão

Título: Alô, Mundo!

Descrição: Minha primeira sinapse.

Next Cancelar

Na tela **“Interface e implementação da Sinapse”**, preencha o campo interface com **“IAloMundo”,** marque a opção **“Iniciadora”** e clique em **“Next”**.



Na tela **“Sinapses a serem consumidas”**, clique em **“Terminar”**.

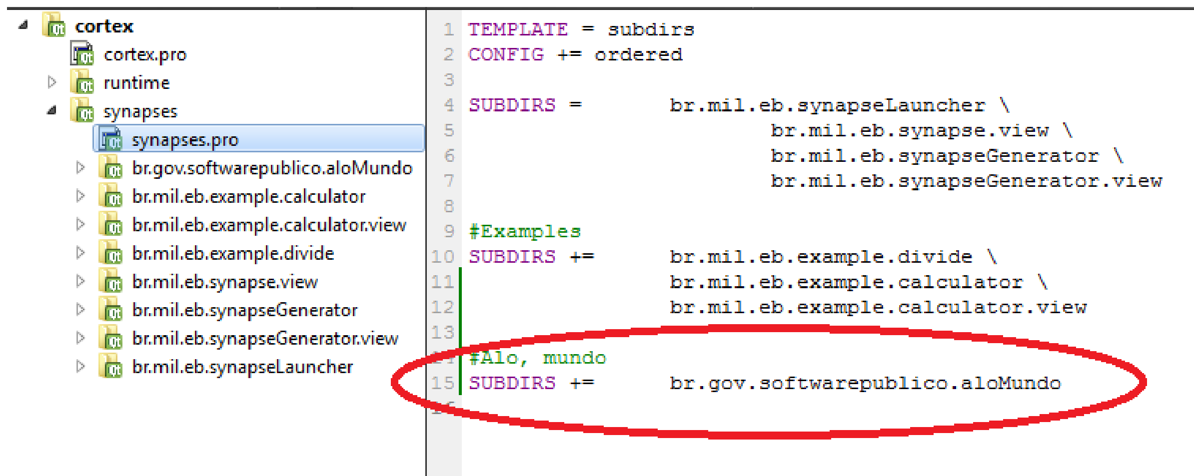
Nesse ponto, será criado o diretório “**br.gov.softwarepublico.aloMundo**” dentro de do diretório **“synapses”** do Cortex.

Ligando os neurônios

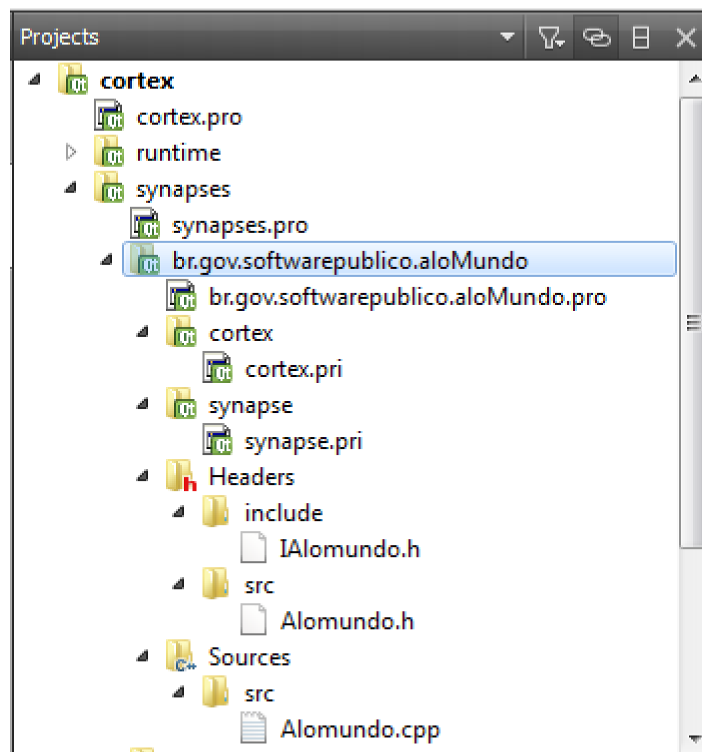
De volta ao **Qt Creator**, na barra da esquerda, clique no botão **“Edit”**.

Na esturuta do projeto **“cortex”**, abra a pasta **“synapses”** e selecione o arquivo de projeto **“synapses.pro”**.

Ao final do arquivo, adicione uma linha com o conteúdo abaixo e salve o arquivo (Ctrl+S):



Note que, ao salvar o arquivo, aparecerá, dentro de **"synapses"**, uma pasta chamada "**br.gov.softwarepublico.aloMundo**", contendo a seguinte estrutura:



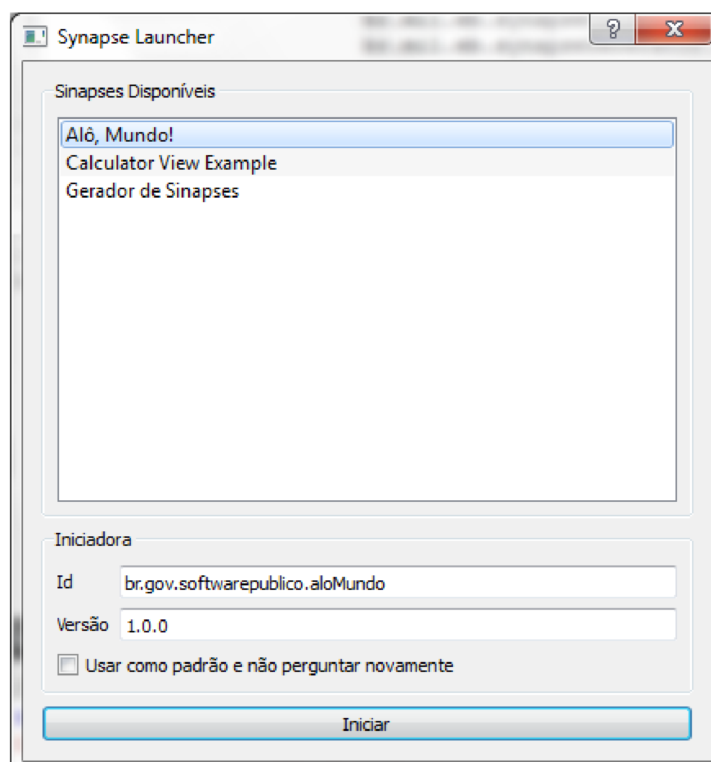
Na seção **"Dissecando uma Sinapse"** serão discutidos todos os detalhes de um projeto de sinapse. Por ora, basta saber que a classe **"IAloMundo"** é a interface da sinapse recém-criada (que estende a interface ISynapse do Cortex), e a classe **"AloMundo"** é a sua correspondente implementação..

Abra o arquivo **"AloMundo.cpp"**, faça as alterações abaixo e salve o arquivo (Ctrl+S).

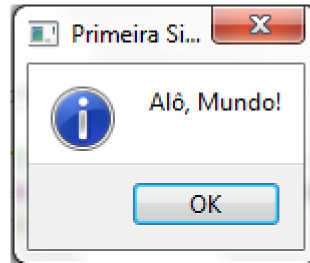
```
1    #include "AloMundo.h"
2    #include "QMessageBox"
3
4  ◢ namespace br {
5  ◢ namespace gov {
6  ◢ namespace softwarepublico {
7  ◢ namespace aloMundo {
8
9
10 ▷ AloMundo::AloMundo() {...}
13
14 ▷ AloMundo::~AloMundo() {...}
17
18 ◢ bool AloMundo::start()
19   {
20           //TODO Automatically generated implementation
21           QMessageBox::information(0, "Minha Primeira Sinapse", "Alô, Mundo!");
22           return true;
23   }
24
25 ▷ bool AloMundo::stop() {...}
30
31   // IAloMundo interface:
32
33   Q_EXPORT_PLUGIN2(br.gov.softwarepublico.aloMundo, AloMundo)
34
35   }
36   }
37   }
38   }        //namespace
39
```

No menu **"Build"**, clique em **"Build All"** (Ctrl+Shift+B).

No menu **"Build"**, clique em **"Run"** (Ctrl+R). Nesse momento, o Cortex iniciará o aplicativo **"Synapse Launcher"**.



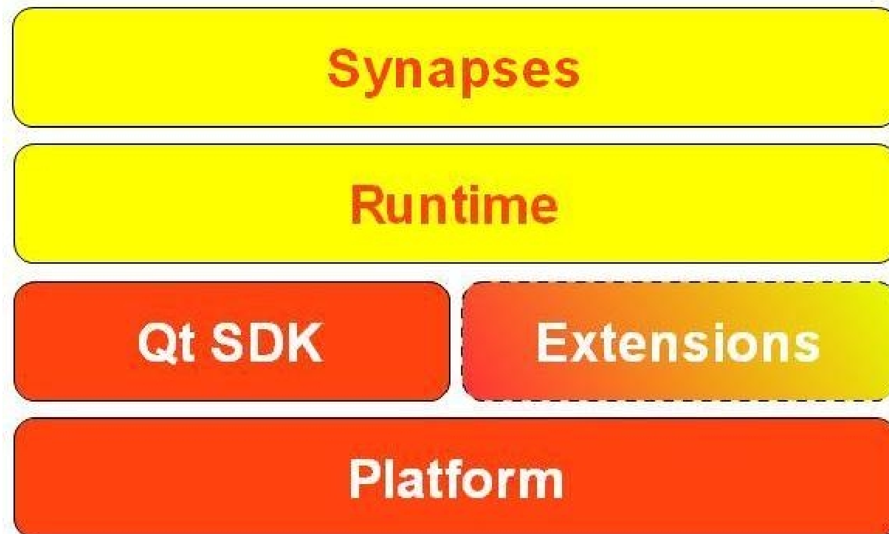
Repare que agora há uma nova sinapse iniciadora (um novo aplicativo), chamada **“Alô, Mundo!”**. Selecione-a e clique em **“Iniciar”**. Eis nosso primeiro estímulo cerebral!



Visão Geral do Tecido

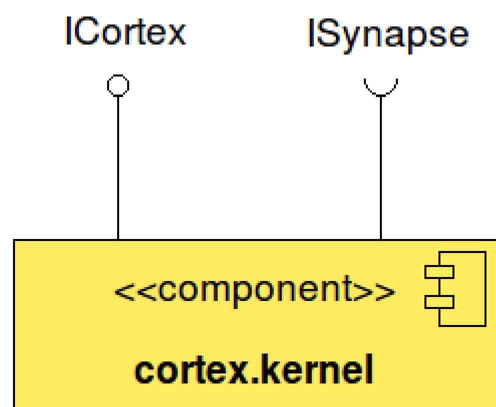
Camadas

O cortex é dividido em quatro camadas:sinapses, runtime, multiplataforma e plataforma.



No topo do framework, encontra-se a camada de **sinapses**, na qual são disponibilizados os serviços desenvolvidos como componentes reutilizáveis. Cada sinapse deve expor suas operações por meio de uma única interface, caracterizando o padrão de projeto “Façade”.

A camada de **runtime** inclui o componente principal da framework, chamado **“cortex.kernel”**, bem como o ambiente de execução em si. Aqui, implementa-se o padrão arquitetural microkernel, com apenas duas interfaces de interesse: ICortex, que atua como um localizador de serviços (padrão Service Locator) e ISynapse, que é a classe base para qualquer interface de sinapse.



Uma camada **multiplataforma** inclui alguns componentes que permitem a compilação e execução da estrutura em diferentes sistemas operacionais. É fortemente baseada no Qt SDK, um framework C + + de alto desempenho para desenvolvimento de aplicações portáveis. Por exemplo, cada sinapse é efetivamente implementada como um plugin do Qt e está disponível à aplicação em tempo de execução através do mecanismo de plugins do prórpio Qt (Qt Plugin System).

Ainda na camada **multiplataforma**, existe o conceito de **extensões**, que nada mais são do que bibliotecas de terceiros que podem ser instaladas no framework para aumentar o poder de portabilidade de uma aplicação. O Cortex dispõe de um processo para resolver as dependências e carregar as extensões que é análogo ao processo de carregamento de sinapses.

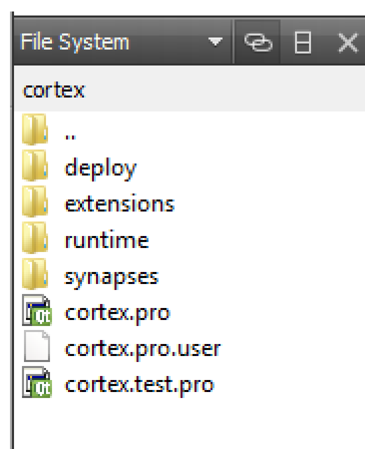
Por último, na base de tudo, está a camada chamada **plataforma**, que constiui-se do ambiente nativo do sistema operacional, como o Windows, Mac OS X, GNU/Linux, Maemo ou MeeGo.

Estrutura padrão de diretórios

Por convenção, o framework possui uma estrutura padrão de diretórios para desenvolvimento:

- **deploy:** arquivos para implantação e execução, destino final de toda compilação.
- **extensions:** projetos desenvolvimento/incorporação de extensões
- **runtime:** projetos dos componentes de runtime
- **synapses:** projetos de desenvolvimento de synapses

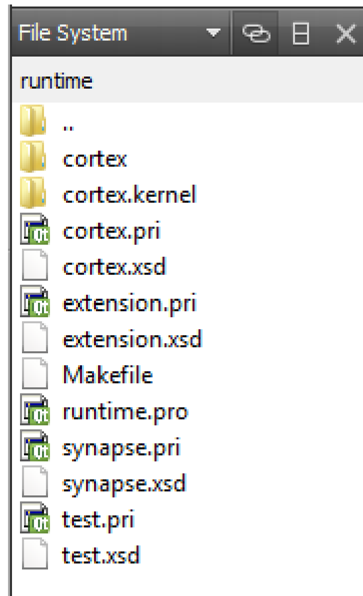
Por tratar-se de convenção, nem todos esses diretórios estarão necessariamente presentes. O diretório **"deploy"**, por exemplo, só é criado após a primeira compilação; **"extensions"**, por outro lado, pode ser criado pelo desenvolvedor se houver alguma demanda por bibliotecas de terceiros.



Dentro do diretório **"runtime"**, encontram-se os arquivos de inclusão de projeto Qt (.pri), que contém as instruções para a correta geração de Makefiles e consequente compilação dentro do framework. os arquivos são os seguintes:

- **cortex.pri** - definição de variáveis e funções gerais.
- **extension.pri** - configurações para a compilação de extensões. Deve ser incluído pelos arquivos de projeto (.pro) de cada extensão.

- **synapse.pri** - configurações para a compilação de synapses. Deve ser incluído pelos arquivos de projeto (.pro) de cada synapse.
- **test.pri** - configurações para a compilação de testes automatizados. Deve ser incluído pelos arquivos de projeto (.pro) de cada teste.



Ainda em **“runtime”**, são encontrados os esquemas XML para permitir a validação dos arquivos de manifesto. Esses arquivos são copiados para o diretório de implantação do framework, por ocasião da instalação. Os arquivos são os seguintes:

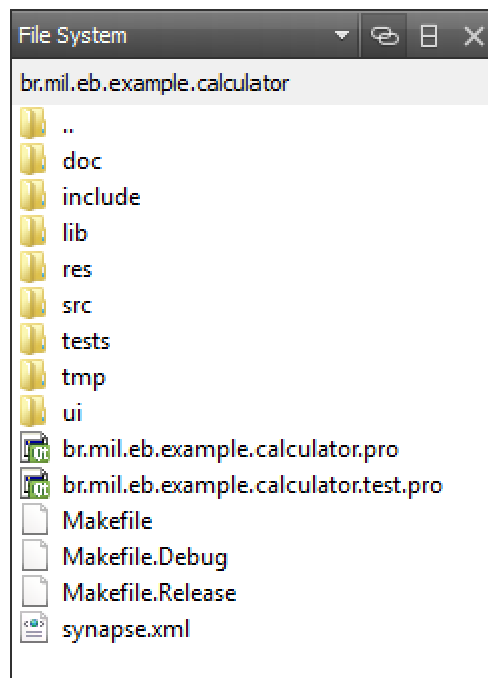
- **cortex.xsd** - contém os elementos comuns a todos os esquemas
- **extension.xsd** - contém os elementos específicos para uma extensão
- **synapse.xsd** - contém os elementos específicos para uma synapse
- **test.xsd** - contém os elementos específicos para um teste

Dissecando uma Sinapse

Estrutura padrão de diretórios

Assim como o Cortex, uma sinapse também possui sua própria convenção de diretórios padrão:

- **doc:** arquivos de documentação
- **include:** arquivos de interface (.h)
- **lib:** destino para as bibliotecas compiladas.
- **res:** arquivos de recursos, a serem copiados posteriormente para o diretório de instalação da sinapse
- **src:** arquivos de implementação (.h e .cpp)
- **tests:** testes automatizados da synapse
- **tmp:** arquivos temporários de pré-processamento do Qt.
- **ui:** arquivos de interface com o usuário (.ui)



Arquivo de projeto (.pro)

O arquivo de projeto de Qt (.pro) situa-se na raiz do diretório da sinapse e contém as instruções para a correta geração de Makefiles e consequente compilação da synapse. Ele recebe um nome no seguinte formato: ”<synapse_id>.pro”.

- **General** - seção que contém as configurações gerais do projeto. Permite as seguintes configurações:
    - SYNAPSE_ID - id da synapse
    - SYNAPSE_VERSION - versão da synapse
    - DEFINES - definição da macro de guarda da synapse

        - QT - adição/remoção de módulos do Qt
        - FORMS - especificação dos arquivos de interface com o usuário (quando for o caso, sugere-se ui/*.ui)
        - RESOURCES - especificação de arquivo de recursos (quando for o caso, sugere-se *.qrc)
        - include - inclusão do arquivo '**synapse.pri**', fornecido com o Cortex
- **Synapses** - seção destinada às declarações das synapses que serão consumidas. Para cada synapse, é feita a chamada às seguintes funções:
    - INCLUDEPATH += $$synapseIncludePath(<synapse_id>)
    - LIBS += $$pseudoSynapseLibs(<synapse_id>) - função **OBSOLETA**, porém necessária devido às "pseudo synapses"*
- **Extensions** - seção destinada às declarações das extensões requeridas. Para cada extensão, é feita a chamada às seguintes funções:
    - INCLUDEPATH += $$extensionIncludePath(<extension_id>)
    - LIBS += $$extensionLibs(<extension_id>)

Manifesto

O manifesto também fica localizado na raiz do diretório da sinapse e possui as informações necessárias para sua correta carga e inicialização pelo Cortex. Ele recebe o seguinte nome: ”synapse.xml”

O manifesto deve ser validado em relação esquema '**synapse.xsd**', fornecido com o Cortex. Um arquivo de manifesto possui as seguintes tags:

- <**info**> - contém as informações básicas, como id, versão, título, descrição, etc.
- <**initiator**> - define se a synpase é iniciadora ou não
- <**singleton**> - define se a synapse é singleton ou não
- <**synapses**> - declara as synapses a serem consumidas
- <**extensions**> - declara as extensões requeridas pela synapse

As seção <synapses> do manifesto é apenas uma dica de dependência, mas que é nromalmente respeitada a: 1) a dependência não seja encontrada no ambiente de runtime ou; 2) o manifesto da iniciadora apresente uma dependência que a suplante, ou seja, implemente a mesma interface de sinapse.

Implementação

Em termo de implementação, uma sinapse deve possuir:

- uma interface de serviços, estendendo a interface cortex::ISynapse;
- a implementação dessa insterface, que deve ser uma classe concreta e ser ao mesmo tempo um plugin do Qt válido.

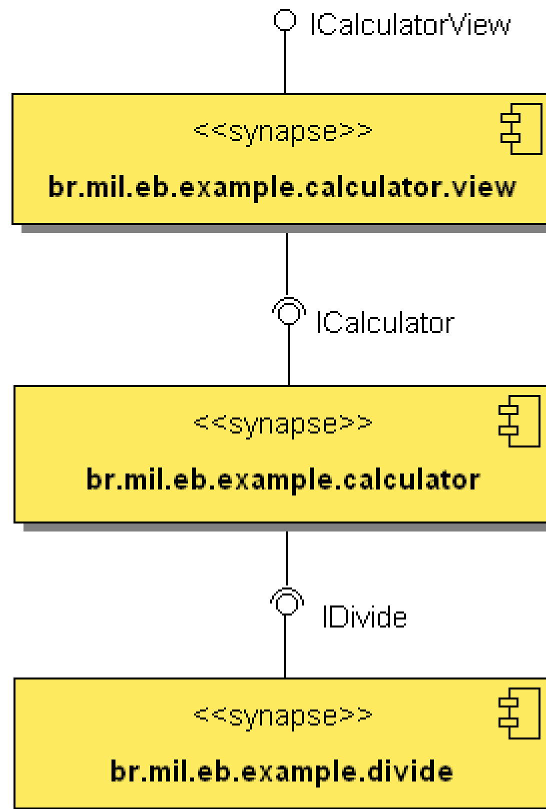
```
1  ▷ /*********************************************************************
22   #ifndef CORTEX_I_SYNAPSE_H
23   #define CORTEX_I_SYNAPSE_H
24
25   #include <QtPlugin>
26   #include <QSharedPointer>
27   #include <QStringList>
28   #include <QWeakPointer>
29
30 ◢ namespace cortex
31   {
32           class ICortex;
33
34 ▷         /** ...*/
40 ◢         class ISynapse
41           {
42                   friend class ICortex;
43           public:
44 ▷                 /** ...*/
47                   virtual ~ISynapse() {}
48 ▷                 /** ...*/
57                   virtual bool start() = 0;
58 ▷                 /** ...*/
66                   virtual bool stop() = 0;
67           };
68
69           typedef QSharedPointer<ISynapse> ISynapsePtr;
70           typedef QWeakPointer<ISynapse> ISynapseWeakPtr;
71   }
72
73   Q_DECLARE_INTERFACE(cortex::ISynapse, "br.eb.mil.cortex.ISynapse/1.0")
74
75   #endif // CORTEX_I_SYNAPSE_H
```

TODO: explicar a interface ISynapse.
TODO: explicar os detalhes necessárias para implementação do plugin.

De volta ao exemplo

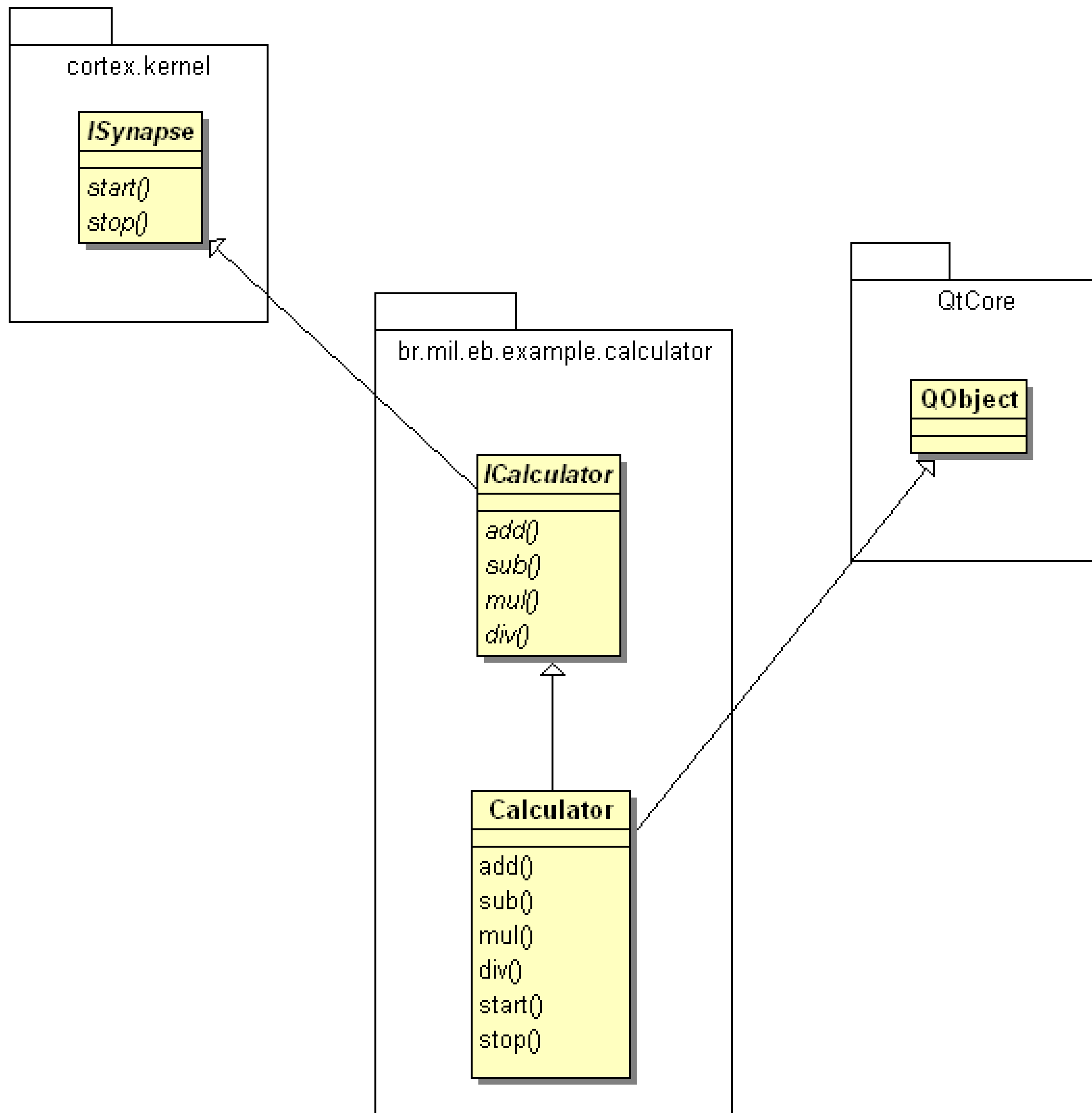
O exemplo da calculadora apresentado na seção “Uma Primeira Cirurgia Exploratória” foi originalmente desenvolvido como uma prova de conceito do framework, e é aqui usado para ilustrar os principais conceitos na implementação de uma sinapse. Trata-se de uma adaptação do “Calculator Example fornecido com a documentção do Qt, porém consumindo serviços para a execução das operações aritméticas básicas, ao invés de usar os operadores próprios da linguagem.

O aplicativo é uma composição de três sinapses: “br.mil.eb.example.calculator”, “br.mil.eb.example.calculator.view” e “br.mil.eb.example.divide”. Cada uma delas expõe sua própria interface, respectivamente ”ICalculatorView”, “ICalculator” e “IDivide”. É importante lembrar que essas são dependências fracas, graças ao mecanismo de composição fornecido por Cortex, cujos beneficios são alta coesão e baixo acoplamento.



O exemplo é executado a partir da sinapse iniciadora, “br.mil.eb.example.calculator.view”. O ponto de partida para o aplicativo, que tradicionamente seria a função “main()”, é o método “start()” da implementação da interface da sinapse, “ICalculatorView”.

À medida que o usuário clica nos botões para executar as operações, a GUI consome os serviços prestados pela sinapse “br.mil.eb.example.calculator”. Como pode ser observado na figura abaixo, a sinapse (representada como um pacote) expõe a sua própria interface (“ICalculator”), que estende “ISynapse”, a interface do microkernel.



A interface ICalculator expõe as operações necessárias para fazer os cálculos e retornar os resultados para a exibição para mostrar.

```
1 #ifndef BR_MIL_EB_EXAMPLE_CALCULATOR_I_CALCULATOR_H
2 #define BR_MIL_EB_EXAMPLE_CALCULATOR_I_CALCULATOR_H
3
4 #include "ISynapse.h"
5
6 class ICalculator : public cortex::ISynapse {
7 public:
8     virtual double add(double arg1, double arg2) = 0;
9     virtual double sub(double arg1, double arg2) = 0;
10    virtual double mul(double arg1, double arg2) = 0;
11    virtual double div(double arg1, double arg2) = 0;
12 };
13
14 Q_DECLARE_INTERFACE(ICalculator, "br.mil.eb.example.calculator.ICalculator/1.0.0")
15
16 typedef QSharedPointer<ICalculator> ICalculatorPtr;
17
18 #endif //BR_MIL_EB_EXAMPLE_CALCULATOR_I_CALCULATOR_H
```

A classe Calculator, por sua vez, implementa implementa as interface ICalculator e ainda herda da classe QObject, o que é mandatório para plugins do Qt

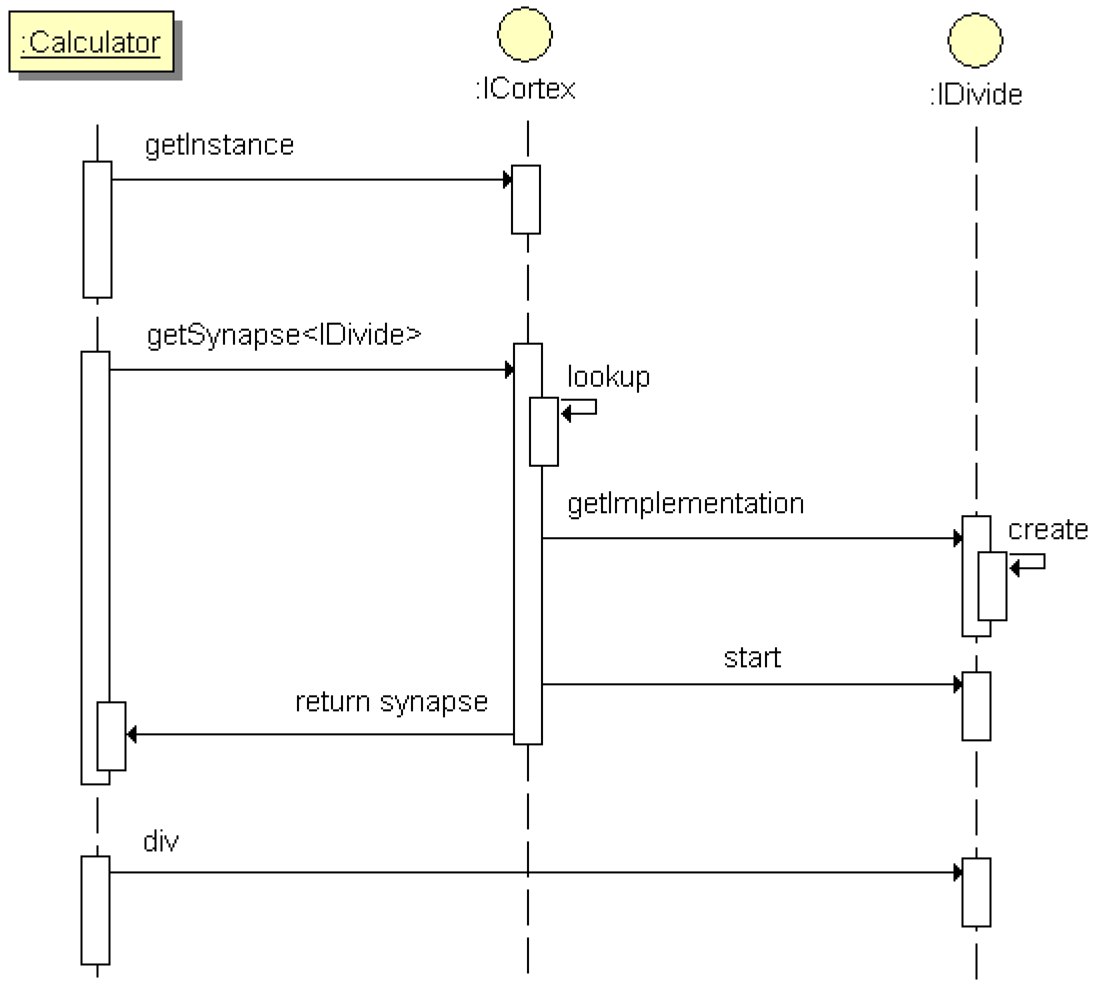
```
1 #ifndef BR_MIL_EB_EXAMPLE_CALCULATOR_CALCULATOR_H
2 #define BR_MIL_EB_EXAMPLE_CALCULATOR_CALCULATOR_H
3
4 #include "ICalculator.h"
5 #include <QObject>
6
7 class Q_DECL_EXPORT Calculator : public QObject, public ICalculator
8 {
9     Q_OBJECT
10     Q_INTERFACES(cortex::ISynapse ICalculator)
11 public:
12     Calculator();
13     virtual ~Calculator();
14
15     // ISynapse interface
16     virtual bool start();
17     virtual bool stop();
18
19     // ICalculator interface
20     virtual double add(double arg1, double arg2);
21     virtual double sub(double arg1, double arg2);
22     virtual double mul(double arg1, double arg2);
23     virtual double div(double arg1, double arg2);
24
25 protected:
26     // ISynapse interface
27     virtual cortex::ISynapse* getImplementation();
28 };
29
30 #endif //BR_MIL_EB_EXAMPLE_CALCULATOR_CALCULATOR_H
```

A implementação de ICalculator, a classe Calculadora, realiza as quatro operações básicas: adição, subtração, multiplicação e divisão. Para esta última, porém, consome a sinapse “br.mil.eb.example.divide”.

```
54
55 double Calculator::div(double arg1, double arg2)
56 {
57     double result = 0;
58
59     IDividePtr divideSynapse = cortex::ICortex::getInstance()->getSynapse<IDivide>();
60
61     result = divideSynapse->divide(arg1, arg2);
62
63     return result;
64 }
65
66 Q_EXPORT_PLUGIN2(br.mil.eb.example.calculator, Calculator)
```

A seqüência de chamadas de métodos para o consumo dessa sinapse é mostrado na próxima figura, onde pode-se ver o padrão Service Locator em ação. Após obter a instância de “ICortex”, “Calculator” solicita a sinapse que implementa a interface “IDivide”.

caso naõ seja encontrada, “ICortex” chama a implementação do serviço, garantindo que ele seja corretamente iniciado antes de devolvê-lo à sinapse cliente.

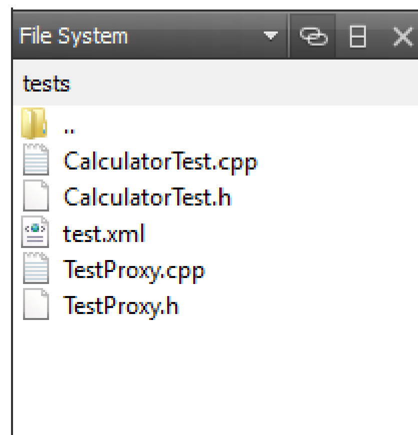


Neste ponto, é interessante notar que o requisito para a correta execução da divisão é uma sinapse que implemente a interface “IDivide”. O framework seleciona “br.mil.eb.example.divide” com base na linha 21 no manifesto de “br.mil.eb.example.calculator”.

```xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <synapse xmlns="urn:br:mil:eb:cortex"
3        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="urn:br:mil:eb:cortex ../synapse.xsd">
5
6     <info id="br.mil.eb.example.calculator" version="1.0.0">
7         <title>
8             Calculator Example
9         </title>
10
11        <description>
12            This is the calculator example
13        </description>
14    </info>
15
16    <initiator>false</initiator>
17
18    <singleton>true</singleton>
19
20    <synapses>
21        <consumes synapse="br.mil.eb.example.divide" version="1.0.0"/>
22    </synapses>
23
24 </synapse>
25
```

Eletrodos de Teste

Cortex implementa uma infraestrutura básica para auxiliar a implementação de testes automatizados de sinapses, tanto de interface quanto das classes internas. Esse mecanismo permite executar os testes ao mesmo tempo em que se dispõe da funcionalidade de localização de serviços do Cortex. Com isso, é possível testar comportamentos que são dependentes da presença de determinada sinapse e utilizar implementações falsas de serviços consumidos pela sinapse em teste



Manifesto

TODO: ressaltar as peculiaridades de um manifesto de teste.

Implementação

TODO: explicar a Interface ISynapseTest, abstraindo o framework de testes utilizado.

TODO: ressaltar que é vantajoso o projeto como um todo usar o mesmo framework.

Argumentos

TODO: citar os argumentos do Cortex para teste.

TODO: explicar o tunelamento de argumentos para uso pelo framework de testes escolhido

Exemplo

TODO: explorar o teste automatizado da calculadora como exemplo.

Nas Entranhas do Cortex

Nesta seção serão abordados com maior profundidade alguns detalhes de implementação do Cortex.

Carregando as Sinapses

TODO: detalhar como o Qt Plugin Framework é usado para implementar as sinapses
TODO: mostrar os loaders (SynapseLoader, TestRunner e ExtensionLoader)
TODO: explicar os SynapseContainers (default e singleton)

Resolução de Dependências

TODO: resumir a implementação da montagem do grafo de dependências e do DFS antes da carga. ManifestParsers, ComponentManager e DependencySolver

Log da Execução

TODO: explicar o X9, o logger básico interno do Cortex