TP Fractal

Environnement Logiciel

Prérequis:

- une machine Java installée (avec variable d'environnement JAVA_HOME)
- Ant installé (avec variable d'environnement ANT_HOME)

Ce TP est structuré comme suit :

- Il y a un répertoire par étape du TP : de exo1 à exo5.
- Le répertoire externals contient tous les jar de l'implantation Julia de Fractal
- Le répertoire doc contient la javadoc de Fractal
- Le répertoire initial contient l'application Comanche dans son état initial. Vous pouvez aller jeter un coup d'oeil aux source dans ce répertoire lorsque vous le jugez utile.

Dans chaque étape, on trouve (presque) toujours la même structure :

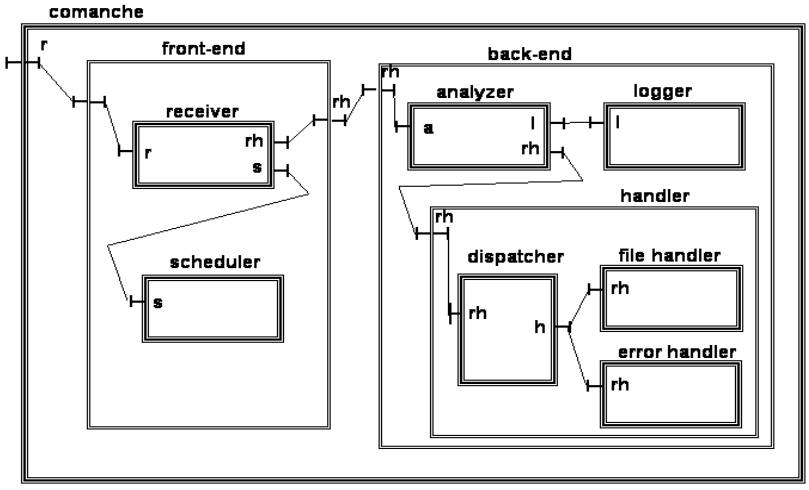
- build contient les fichiers générés lors des étapes de compilation
- etc contient des fichiers de configuration
- src contient les fichiers source

Un fichier build.xml permet de gérer les aspects compilation et exécution des programmes. On retrouve les cibles suivantes (lorsque cela à un sens en fonction de l'étape du TP) :

- clean : pour effacer les fichiers générés
- compile : pour compiler les sources
- execute : pour lancer l'exécution

Etape exo0

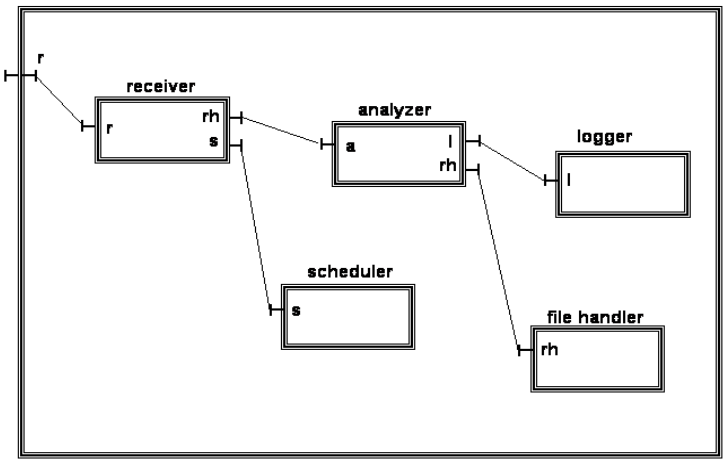
Vous devez simplement regarder les sources de l'application dans initial, en particulier les fichiers ADL et vérifier qu'elle est conforme à l'architecture vue en cours :



Faites marcher l'application (serveur web) et testez la avec un client web.

Etape exo1

Vous devez implanter en la programmant en Java (voir le chassis de code dans src) cette autre architecture :



Liens entre les composants et l'implantation Java (classes et interfaces) :

composant receiver : comanche.RequestReceiver

composant analyzer : comanche.RequestAnalyzer

composant scheduler : comanche.MultiThreadScheduler

composant logger : comanche.BasicLogger

composant filehandler : comanche.FileRequestHandler

interface r : java.lang.Runnable

interface rh : comanche.RequestHandler

interface a : comanche.RequestHandler

interface s : comanche.Scheduler

interface l : comanche.Logger

Etape exo2

Vous devez implanter la même architecture avec l'ADL Fractal.

Etape exo3

Observez l'application avec fractalexplorer (le fichier build.xml lance l'observation de la version initiale). A noter que pour démarrer l'application, il faut appeler l'opération "invoke" sur l'interface "r" du composite "comanche" (click souris droit dans l'observateur).

Etape exo4

Vous devez implanter un nouvel Handler (HttpHandler) qui implante un point de montage vers un autre site web (vous partez de la version initiale du serveur http comanche). Pour ce faire vous

devez copier les fichiers ADL Fractal que vous voulez modifier (de la version initiale), ceux que vous ne modifiez pas peuvent être réutilisé tels quels depuis le package comanche. Tester que le nouvel handler permet de charge une page web d'un autre site web.

Etape exo5

Vous devez implanter un nouvel Analyzer qui, lorsqu'il voit passer une requête "http://localhost:8080/reconf", ajoute le handler précédent (HttpHandler) qui n'est initialement pas présent. On implante ainsi une reconfiguration dynamique de l'application Comanche.

org.objectweb.fractal.api

Interface Component

`public interface` **`Component`**

A component interface to introspect the external interfaces of the component to which it belongs.

Method Summary	
Object	**getFcInterface**(String interfaceName) Returns an external interface of the component to which this interface belongs.
Object[]	**getFcInterfaces**() Returns the external interfaces of the component to which this interface belongs.
Type	**getFcType**() Returns the type of the component to which this interface belongs.

---

org.objectweb.fractal.api

Interface Interface

`public interface` **`Interface`**

An interface to introspect component interfaces. If a Fractal component supports interface introspection, then the interfaces returned by the `getFcInterfaces` and `getFcInterfaces` methods can be cast into this Java interface, in order to get their name or their type.

Method Summary	
String	**getFcItfName**() Returns the name of this interface inside its component.
Component	**getFcItfOwner**() Returns the component to which this interface belongs.
Type	**getFcItfType**() Returns the type of this interface.
boolean	**isFcInternalItf**() Returns `true` if this interface is an internal interface.

org.objectweb.fractal.api

Interface Type

All Known Subinterfaces:

ComponentType, InterfaceType

`public interface` **`Type`**

Specifies the minimal interface that all type systems must implement. This interface defines only one method to test if a type is a sub-type of another one.

Method Summary

`boolean`	**`isFcSubTypeOf`**`(Type type)` Returns `true` if the given type is a sub-type of this type.

---

org.objectweb.fractal.api.control

Interface AttributeController

`public interface` **`AttributeController`**

A component interface to control the attributes of the component to which it belongs. More precisely this interface denotes the component interfaces that can control component attributes: a component interface whose Java type is a Java interface that extends this Java interface is indeed considered as an interface to control the attributes of the component to which it belongs. Such interfaces *must* only contain getter and setter methods, such as `int getX (); void setX (int x)`, `double getSize (); void setSize (double x)`, and so on. These methods should only be used to configure "primitive" values such as integers or strings: they must not be used to configure bindings (this is the role of the `BindingController` interface). For example, they can be used to configure the size of a cache component, the load factor of a hashtable component, the label or color of a button component...

---

org.objectweb.fractal.api.control

Interface BindingController

`public interface` **`BindingController`**

A component interface to control the bindings of the component to which it belongs. It is implicitely assumed here that the component's type system makes a distinction between "client" and "server" interfaces.

Method Summary	
void	**bindFc**(String clientItfName, Object serverItf) Binds the client interface whose name is given to a server interface.
String[]	**listFc**() Returns the names of the client interfaces of the component to which this interface belongs.
Object	**lookupFc**(String clientItfName) Returns the interface to which the given client interface is bound.
void	**unbindFc**(String clientItfName) Unbinds the given client interface.

---

org.objectweb.fractal.api.control

Interface ContentController

---

public interface **ContentController**

A component interface to control the content of the component to which it belongs. This content is supposed to be made of an unordered, unstructured set of components.

Method Summary	
void	**addFcSubComponent**(Component subComponent) Adds a sub-component to this component.
Object	**getFcInternalInterface**(String interfaceName) Returns an internal interface of the component to which this interface belongs.
Object[]	**getFcInternalInterfaces**() Returns the internal interfaces of the component to which this interface belongs.
Component[]	**getFcSubComponents**() Returns the sub-components of this component.
void	**removeFcSubComponent**(Component subComponent) Removes a sub-component from this component.

---

org.objectweb.fractal.api.control

Interface LifeCycleController

`public interface` **`LifeCycleController`**

A component interface to control the lifecycle of the component to which it belongs. The lifecycle of a component is supposed to be an automaton, whose states represent execution states of the component. This interface corresponds to an automaton with two states called `STARTED` and `STOPPED`, where all the 4 four possible transitions are allowed. It is however possible to define completely different lifecycle controller Java interfaces to use completely different automatons, or to define sub interfaces of this interface to define automatons based on this one, but with more states and more transitions.

Note: the sub-interfaces of this interface should use the conventions used in this interface, which are the following. The interface contains one method per state in the lifecycle automaton. Each of these methods changes the current state to the state corresponding to its name, if there is a transition from the current state to this state. The interface also contains one field per state. The names and values of these fields correspond to the names of the methods.

Field Summary

`static String`	**`STARTED`** The state of a component just after `startFc` has been executed.
`static String`	**`STOPPED`** The state of a component just after `stopFc` has been executed.

Method Summary

`String`	**`getFcState`**`()` Returns the execution state of the component to which this interface belongs.
`void`	**`startFc`**`()` Starts the component to which this interface belongs.
`void`	**`stopFc`**`()` Stops the component to which this interface belongs.

---

org.objectweb.fractal.api.control

Interface NameController

`public interface` **`NameController`**

A component interface to control the name of the component to which it belongs.

Method Summary	
String	**getFcName**() Returns the name of the component to which this interface belongs.
void	**setFcName**(String name) Sets the name of the component to which this interface belongs.

---

org.objectweb.fractal.api.control

Interface SuperController

---

public interface **SuperController**

A component interface to control the super components of the component to which it belongs.

Method Summary	
Component[]	**getFcSuperComponents**() Returns the components that contain the component to which this interface belongs.

---

org.objectweb.fractal.api.factory

Interface Factory

---

public interface **Factory**

A component interface to create components of the same type.

Method Summary	
Object	**getFcContentDesc**() Returns a description of the content part of the components instantiated by this factory.
Object	**getFcControllerDesc**() Returns a description of the controller part of the components instantiated by this factory.
Type	**getFcInstanceType**() Returns the functional type of the components instantiated by this factory.

Component	**newFcInstance**() Instantiates a component from this factory.

---

org.objectweb.fractal.api.factory

Interface GenericFactory

---

public interface **GenericFactory**

A component interface to create arbitrary components.

Method Summary	
Component	**newFcInstance**(Type type, Object controllerDesc, Object contentDesc) Creates a component.

---

org.objectweb.fractal.api.type

Interface ComponentType

All Superinterfaces:
Type

---

public interface **ComponentType**extends Type

A component type. A component type is just a collection of component interface types, which describes the interfaces that components of this type must or may have at runtime.

Method Summary	
InterfaceType	**getFcInterfaceType**(String name) Returns an interface type of this component type from its name.
InterfaceType[]	**getFcInterfaceTypes**() Returns the types of the interfaces of components of this type.

Methods inherited from interface org.objectweb.fractal.api.Type
isFcSubTypeOf

org.objectweb.fractal.api.type

Interface InterfaceType

All Superinterfaces:

Type

```
public interface InterfaceTypeextends Type
```

A component interface type. Such a type is made of a name, which is the name of the interface described by this type inside its component (see `getFcItfName`), a list of method signatures, which describes the methods provided or required by this interface, and various flags that indicates if this interface is provided or required, mandatory or not...

Method Summary

String	**getFcItfName**() Returns the name of component interfaces of this type.
String	**getFcItfSignature**() Returns the signatures of the methods of interfaces of this type.
boolean	**isFcClientItf**() Returns `true` if component interfaces of this type are client interfaces.
boolean	**isFcCollectionItf**() Indicates how many interfaces of this type a component may have.
boolean	**isFcOptionalItf**() Returns `true` if component interfaces of this type are optional.

Methods inherited from interface org.objectweb.fractal.api.Type
isFcSubTypeOf

---

org.objectweb.fractal.api.type

Interface TypeFactory

```
public interface TypeFactory
```

A component interface to create component and interface type objects.

Field Summary

static boolean	**CLIENT** The `isClient` value to be used in `createFcItfType` to create a client interface type.
static boolean	**COLLECTION** The `isCollection` value to be used in `createFcItfType` to create a collection interface type.
static boolean	**MANDATORY** The `isOptional` value to be used in `createFcItfType` to create a mandatory interface type.
static boolean	**OPTIONAL** The `isOptional` value to be used in `createFcItfType` to create an optional interface type.
static boolean	**SERVER** The `isClient` value to be used in `createFcItfType` to create a server interface type.
static boolean	**SINGLE** The `isCollection` value to be used in `createFcItfType` to create a singleton interface type.

Method Summary

InterfaceType	**createFcItfType**(String name, String signature, boolean isClient, boolean isOptional, boolean isCollection) Creates an interface type.
ComponentType	**createFcType**(InterfaceType[] interfaceTypes) Creates a component type.

org.objectweb.fractal.util

Class Fractal

org.objectweb.fractal.util.Fractal

public class **Fractal** extends Object

Provides static methods to access standard interfaces of Fractal components.

Method Summary

static AttributeController	**getAttributeController**(Component component) Returns the AttributeController interface of the given component.
static BindingController	**getBindingController**(Component component) Returns the BindingController interface of the given component.
static Component	**getBootstrapComponent**() Returns a bootstrap component to create other components.
static ContentController	**getContentController**(Component component) Returns the ContentController interface of the given component.
static Factory	**getFactory**(Component component) Returns the Factory interface of the given component.
static GenericFactory	**getGenericFactory**(Component component) Returns the GenericFactory interface of the given component.
static LifeCycleController	**getLifeCycleController**(Component component) Returns the LifeCycleController interface of the given component.
static NameController	**getNameController**(Component component) Returns the NameController interface of the given component.
static SuperController	**getSuperController**(Component component) Returns the SuperController interface of the given component.
static TypeFactory	**getTypeFactory**(Component component) Returns the TypeFactory interface of the given component.