A factory for GCM

Luc Hogie

October 3, 2010

Contents

The aim of the document is to provide a documentation for the new GCM factory.

1 Introduction

1.1 Context

1.1.1 Fractal

According to its authors, Fractal is a modular, extensible and programming language agnostic component model that can be used to design, implement, deploy and reconfigure systems and applications, from operating systems to middleware platforms and to graphical user interfaces.

1.1.2 ADL

The ADL used by Fractal allows the description of a hierarchy of components.

A basic tutorial for ADL syntax can be found at the following web URL: `http://fractal.ow2.org/tutorials/adl/index.html`

This description is written in a dialect of XML. Even though the authors of Fractal claim the independance of Fractal from XML, no parser for other syntax is available.

Fractal incorrectly refers to ADL descriptions as ADL files. In fact, the ADL description is not stored in a file: it is stored as a Java resource. A Java resource can be either a plain file or an entry within a ZIP or JAR container.

1.1.3 ProActive

1.1.4 GCM

1.1.5 The component factory

The component factory of Fractal is a piece of code within Fractal whose the role is to build a component out of its XML description. This description is called an ADL (Architecture Description Language).

GCM comes with its own factory which is an extension of Fractal's one.

Althought the objectives of the factory are to merely perform a number of verifications on the description and instantiate a set of components out of it, its implementation is cumbersome: it consists of nothing less than about 150 Java classes and configuration files. In order to implement is extensions, GCM adds about 40 classes and description files.

1.2 Description of the problem

The complex design of Fractal factory turned out to be

The objective of the new factory is then to make a factory that is *much shorter, cleaner, easier to understand, faster than the original Fractal/GCM factory.*

2 Design of the factory

The new factory is object-oriented.

2.1 Definitions

2.2 Input format

2.2.1 ADL

The architecture of the component system must be written in a dialect of XML.

2.2.2 Argument values

The arguments values are described in a file as a set of key = value pairs. The format of the argument file is then:

$$\begin{aligned} k_1 &= v_1 \\ k_2 &= v_2 \\ &\dots \\ k_n &= v_n \end{aligned}$$

Java programmers will notice that the syntax is the one used for properties files.

The name of the argument files is passed to the factory `newComponent()` *method.*

2.2.3 Deployment descriptor

2.3 The new factory

2.3.1 Lexical analysis: from XML to DOM

2.3.2 Semantic analysis: from DOM to AD

Each XML element type into the ADL description is represented by a `description` *class. Then at runtime, each XML element is represented at an instance of this class: a* `description` *object. The description models which attributes and sub-elements (sub-descriptions) the element allows.*

In pratise, Fractal ADL defines the following elements:

definition *describes the root component of the component tree;*

component *describes a sub-component of a given component;*

interface *describes an interface of a given component;*

binding *describes an interface binding involving two interfaces of two parent/child components;*

attributes *describes attributes in a given components.*

content *describes the content class for the implementation code of a given component.*

The Factory defines the following classes, all sub-classes of the `Description` *class:*

componentDescription *represents both definition and component elements;*

interface *describes the interface elements;*

binding *the binding elements;*

attributes *describes the attribute elements;*

Note that the content *XML element is not represented as a specific sub-class of class* `description`*. Instead it is represented as a field into the*
The description is build

2.3.3 Component creation: from AD to components

3 Enhancement of the ADL

3.1 The DTD is no longer required

All the verifications are done in the semantic analyser.
This simplifies the writing of the XML file.
The DTD can still be useful for documentation purposes.

3.2 Storing the ADL in files rather than in Java resources

3.3 the *definition* XML element

A component is described by the use of the component *element, except at the root-level where the component is to be referred as a* definition*. This choice probably comes from the fact, reinforced by the use of a DTD, that the* definition *element allows extra attributes that the* component *element does not. The* arguments *attribute might be of these.*

3.4 the *arguments* attribute

Suppressing the arguments *attribute into component description element.*

3.5 the *content* XML element

In the description of a component, only one content *element is allowed, and this element accepts only one attribute, the* class *attribute.*

```
<component name="boh">
        ...
        <content class="java.util.ArrayList" />
        ...
</component>
```

Instead, the content information for a component should be expressed as an attribute in the `component` *description.*

```
<component name="boh" content="java.util.ArrayList">
        ...
</component>
```

4 How to

4.1 Perform a new verification

Verifications are all performed in the `check()` *method of the* `Description` *class. Depending on the description you want to perform the verification (component, interface, etc), you will have to look into the corresponding description class (respectively* `ComponentDescription`*,* `IntefaceDescription`*, etc).*

Adding a new verification consists in adding a line to the `check()` *method. Typically, such line is in the form:*

```
if (!condition)
        throw new ADLException("condition failed");
```

For convenience purpose, the use of the following construct is encouraged:

```
Assertions.ensure(condition, "condition failed");
```

4.2 Add a new attribute to an existing XML element

As described in Section **??***, each element type is represented by a* `description` *class. In turn, each attribute of the element is represented as a field into its corresponding* `description` *class. Adding a new attribute consists then in adding a new field in this class.*

4.3 Add a new XML element

4.4 Modify the way attributes values are processed

5 Conclusion