Documentation of some parts of the nuptse compiler

Maxime Louvel

July 2, 2007

1 introduction

This document aims to present some points which I think are not enough documented. It is based on my scholarship, and expect to complete the existing Think documentation. The Think version I work with is **Nuptse**.

2 Cast and Dtd mapping

There is a mapping between the ast nodes ant the xml nodes (representing the ADL). There are some dtd, (for now 2) which explicit those mapping [1]. For example, in the file `think.dtd` the following line says that an ast Component can be map with an InterfaceContainer ADL component.

```
<?add ast="component" itf="org.objectweb.fractal.adl.interfaces.InterfaceContainer"
   ?>
```

This mean you can cast an ast node Component in an InterfaceContainer, for add an interface to the component for example.
Note that with the method java method `getClass().getName()` you can see the class of an ast node. If this name is something like `ComponentImpl` that means you can cast the node into a `Component`. `Impl` is used because the class has been generated.

3 How to add a new Aspect Loader

3.1 tutorial

1. Create a new sub-class of `AbstractAspect`, which overrides the `transform` method and in the constructor, call `super` with the new Aspect name as parameter.
2. Define the new aspect : this is done in the fractal language, see example 3.2. In deed, an aspect in a fractal component so you need to define it in fractal.

[1] file `thinkadl/src/org/objectweb/adl/parser/xml/*.dtd`

3. you need to add the following statement in the `build.properties` file in order to load your new plug-in:

   ```
   aspect MyNewAspect
   ```

4. If you want to apply the aspect **myAspect** on a component **compo** you need to add the following line (using the language **Flexprop**) in the `src/kernel.properties` file :

   ```
   */compo myAspect(prop1=val1,prop2=val2)
   ```

   - `*` is the root component
   - `compo` is the sub-component on which the aspect `myAspect` will be applied.
   - Between the parenthesis you can provide properties useful to your aspect as a list of keys/values (`val1` is the value for `prop1`)
   - If you want to add properties to a deeper sub-component (sub-component of `compo`, sub-component of the root) you have to use:

     ```
     */compo/sub-compo myAspect(prop1=val1,prop2=val2)
     ```

   - Finally, if you want apply properties to the root component you simply write :

     ```
     * myAspect(prop1=val1,prop2=val2)
     ```

   Note that parenthesis are mandatory even if there is no property.

3.2 example

If you want to add the new Aspect **active** in the file `thinkadl/src/org/objectweb/think/adl/aspects/Active.java` you must write :

```
package org.objectweb.think.adl.aspects;

import org.objectweb.fractal.adl.ADLException;
import org.objectweb.fractal.adl.Definition;
import org.objectweb.fractal.adl.Node;
import org.objectweb.fractal.adl.bindings.Binding;
import org.objectweb.fractal.adl.bindings.BindingContainer;
import org.objectweb.fractal.adl.components.Component;
import org.objectweb.fractal.adl.components.ComponentContainer;
import org.objectweb.fractal.adl.interfaces.Interface;
import org.objectweb.fractal.adl.interfaces.InterfaceContainer;
import org.objectweb.fractal.adl.types.TypeInterface;

public class Active extends AbstractAspect {

    /* constructor */
```

```
    public Active() throws ADLException {
  super("active");

    }

/**
 * @param nodes
 * @throws ADLException
 */
    public void transform(Set<ASTTransformationHelper.Node> nodes, Definition
        definition) throws ADLException {
        //this code will be executed when the plug-in loader will have select nodes which map the
            plug-in
  if (nodes == null)
      return;

  for(ASTTransformationHelper.Node node : nodes) {
      // get the noe param :
      // the map's keys are the properties name
      // and the map's value are the corresponding properties values
      Map<String, String> params = node.getParams();
      // Here we deal with the "prop1" property only
      String prop1 = params.get("prop1");

      if(prop1 == null)
      {
    System.err.println("prop1 property not found !!!! ");
      }
      else
      {
    if(prop1.equals("yes"))
    {
        //try to add a new cons on a new interface
        System.out.println("prop1 is provided");
    }

    if(prop1.equals("no"))
    {
        //try to add a new cons on a new interface
        System.out.println("prop1 is not provided");
    }
      }
  }
    }
```

And the fractal definition of the new aspect :

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE definition PUBLIC "-//objectweb.org//DTD Fractal ADL 2.0//EN"
  "classpath://org/objectweb/fractal/adl/xml/standard.dtd">

<definition name="org.objectweb.think.adl.aspects.Active"
           extends="org.objectweb.think.adl.aspects.AbstractAspect">
  <interface name="loader" role="client"
    signature="org.objectweb.fractal.adl.Loader" />
  <interface name="partial-loader" role="client"
    signature="org.objectweb.fractal.adl.Loader" />
  <content class="org.objectweb.think.adl.aspects.Active" />
</definition>
```

Then you add this line in `build.properties` :

`aspect Active`.

Finally to apply an aspect on the component **compo1** in the file `kernel.properties` you add the following line :

`*/compo1 active(prop1=val1,prop2=val2)`.

4 AST manipulation

Here are presented some features allowing to change the AST when applying an Aspect. The function/method calls present here are executed in the transform method of the applied Aspect.

4.1 tools

In the file `adl/aspects/ASTTransformationHelper.java` there are some methods use for browse the AST. There are also methods for modify the AST e.g. to add a sub component, add an interface, a binding, etc. Indeed, this file is supposed to be extended. In the file `adl/ASTHelper.java` there is some static method (class method) allowing to access to various information of ASTNode (like getSubComp(), getBoundToItf()).

4.2 browsing

The transform method is called with the AST in parameter, presented in a set of nodes :

`Set<ASTTransformationHelper.Node> nodes`

So, there is two way of browsing the AST :

- node by node, without parent / child relationships
- go back in the AST : from a child to its parent, using the method `getParent()` on the child.

be careful : the root's node can't be accessed with the method `getParent()` on its children. In fact the root is not a component node but definition one. Thus it is handle in a little different manner.

4.3 Classes presentation

When you browse the AST you have to cast nodes into real class. For instance, if you want add a component to a node, this last one must be an instance of ComponentContainer.

```
if(node.getNode() instanceof ComponentContainer)
   ((Component) node.getNode()).addComponent(compo);
```

Here we explain the meaning of some classes.

4.3.1 Definition

A Definition object represent a class of component, not an instance. It is possible to have several instance of one Definition.

4.4 add an interface

4.5 add a component

4.6 load a component

When you load a component if you want change its content you will have to update the component If you want load a component without content (this last one will be added later) you should (this is a little hack it could be a better way) said your component has a content (in the `.adl`) and overload the link to the source file when it is created. `myCompo.adl` file :

```
component myCompo{
    provides api.Scheduler as sched

    content emptyContent
}
```

`emptyContent.c` :

```
// empty content
```

Then, when you load the component, you change the implementation `className` : replace "emptyContent" with "goodFoler/realFile".

4.7 bind two interfaces

4.8 example

The following example add to the parent node of each node having the property "side=server" two sub components : one providing a `consoleMax` interface, the other requiring the same interface, and bind them together.

```
public void transform(Set<ASTTransformationHelper.Node> nodes,
         Definition definition)
    throws ADLException {

    if (nodes == null)
  return;

    ASTTransformationHelper.Node nodeContainBinding = null, nodeClient = null;

    // look at the AST nodes propeties
    for(ASTTransformationHelper.Node node : nodes){
  Map<String, String> params = node.getParams();

  String cote = params.get("side");
  // we try to add a thread to this component
  if(side!=null && cote.equals("server"))
  {
      // create and add (to the parent AST node) a provider component
      Component CompoConsole = loadComponent("lib.consoleMax","CompoCons");
      ((Component) node.getParent()).addComponent(CompoConsole);
      // creat and add (to the parent AST node) a requirer component
      Component CompoConsCli = loadComponent("lib.consoleMaxCli","CompoConsCli");
      ((Component) node.getParent()).addComponent(CompoConsCli);
      // bind both interfaces
      Binding b1 = createBinding("CompoConsCli.consCli","CompoCons.consSrv");
      ((BindingContainer) node.getParent()).addBinding(b1);
  }
    }
}
```