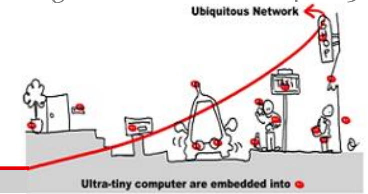
Tutorial 7: Self Adaptation Middleware and Aspect of Assembly



1 Aspects of Assembly

1.1 Introduction to Aspects of Assembly (AA)

The concept of Aspect of Assembly (AA) was introduced in the Daniel Cheung's Ph.D. thesis, defended in March 2009. You can find details of the concepts presented here in this document:

> **Daniel Cheung-Foo-Wo** *"Dynamic Adaptation by weaving aspects of assembly"*, Ph.D. Thesis, University of Nice - Sophia Antipolis, 223 pages, March 2009.

or the following journal articles:

> **Jean-Yves Tigli, Stéphane Lavirotte, Gaëtan Rey, Vincent Hourdin, Daniel Cheung-Foo-Wo, Eric Callegari, Michel Riveill.** "*Wcomp Middleware for Ubiquitous Computing: Aspects and Composite Event-based Web Services*" in Annals of Telecommunications (AoT), 64 (3-4), pages 197-214, Springer, 2009 AD
>
> **Jean-Yves Tigli, Stéphane Lavirotte, Gaëtan Rey, Nicolas Ferry, Vincent Hourdin, Sana Fathallah, Christophe Vergoni et Michel Riveill**. "*Aspects of Assembly: from Theory to Performance*". LNCS Transactions on Aspect-Oriented Software Development (TAOSD), volume 7271, 2012. ISSN 1864-3027 (Print) 1864-3035 (Online).

Designer of the aspects of assembly (AADesigner) is a compositional adaptive mechanism based on aspect-oriented programming, changing the structure of component assemblies. Adaptation rules are called aspects of assembly (AA) and the mechanism performing the adaptation is called the weaver. The tool maintains a list of AA and the weaver is called the AA designer.

The AA consists of two parts, according to the traditional aspects:

1. The pointcut, corresponds to rules in the first part of the AA. These rules allow finding the sets of pre-required components (called joint points) to apply an AA and modify the application.
2. The advice, corresponds to rules in the second part of the AA. These rules describe the modifications applied on the joint points, previously found in the application.

1.2 Definition of the AA Language

A specific language has been created to define the AA. In the version that you use in this tutorial, the pointcuts and the advices use a simplified version of the original language described in the thesis of Daniel Cheung.

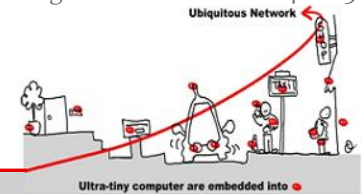
AA are defined as a couple of a pointcuts list and an advice.

1.2.1 Pointcut definition

Pointcuts specifies the component you are looking for in the base assembly. Pointcuts are defined as assigning a variable to a string that may contain a wildcard at the beginning or the end of a string, or both. In the current implementation of AA, an expression is a simple regular expression to match components' name

```
identifier = <expression>
```

Tutorial 7: Self Adaptation Middleware and Aspect of Assembly



1.2.2 Advice definition

An advice specify the rules you want to add, in fact the kink of assembly you want to create based on the existing components you identified with the help of pointcuts. Two kind of rules exist for an advice: you can create components or links between components.

1.2.2.1 Creating component

The creation of a component, called *local* component, is defined with the following syntax:

```
identifier : type
```

or optionally by specifying one or more initialization values for the component properties :

```
identifier : type (property_name = [, ...] )
```

These *local* components can be used by the advice that creates them. So, each AA should include the definitions of *local* components used to achieve the desired adaptation.

1.2.2.2 Creating link between components

Defining the rules to create links between components is based on the following syntax:

```
component.^output_port -> (component.input_port)
```

WComp components propose two types of ports. The input ports are the methods of the components; the output ports are emitted events that can be linked to methods of other component to design assemblies. The token '^' prefix the events to differentiate them from methods.

The rules can create links between existing or just created components in the AA. In both cases, the components will be represented in these rules by variable identifiers:

1. If the component already exists in the assembly, we must find it in the application to apply adaptation. For this, it's necessary to define a pointcut rule, which defines a variable `(identifier)`. During the weaving process, the variable will contain the actual name of the found components in the assembly.
2. If the component is created in the advice of the AA, the corresponding defined variable can be used. During the weaving process, it will contain the actual name of the created component.

As you see during previous labs, you can also create links that uses some parameters from the component emitting event to create the right signature to call the method of the destination component. To specify it in an AA, you can use the following syntax (two underscore with space(s) before and after them):

```
component.^output port __ get_Value -> (component.input_port)
```

1.2.3 Complete example of AA

The AA below defines an adaptation with the creation of a link between two components identified in the pointcuts, whose names begin with "switch" and "light" (followed by any number and kind character). In addition, the advice definition begins with a special line starting with "`advice`" followed by the name of the AA.

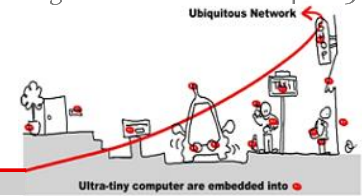
```
emitter = switch*
receiver = light*

advice switch_and_light (emitter, receiver):
    emitter.^Status_Event -> ( receiver.SetTarget )
```

2 Advanced Programming in the weaving process

To illustrate the adaptation capability of WComp applications, you will create a simple designer linking checkboxes to "*Lights*" when they appear. Two policies are possible:

- creating a checkbox for each discovered light,
- control all lights with the same checkbox that you would have to previously create manually.

Tutorial 7: Self Adaptation Middleware and Aspect of Assembly



2.1 Weaver as a new WComp assembly

This designer will be created as a SharpWComp component assembly, that will act as a UPnP control point of the application's container. Creating a UPnP control point in WComp has some advantages, mainly that discovery of devices is already handled by the UPnP Wizard Designer, and that its internal logic will be dynamic.

In order to interact with the application targeted for adaptation, we will need to use a proxy component to the control interface of SharpWComp (the control interface you used with Device Spy to create and suppress beans or links in the container).

First of all, we will change the name of the container that will contain you application. You must use the WComp.NET/Custom Binding Attributes menu entry in order to change the name of the container (and the ports of structural and functional UPnP services). You must choose:

- Name: Appli
- Control Interface: 3000
- Functional Interface: 3001

Don't forget to rebind UPnPWizardDesigner with this "new" application container.

Then we will create a new WComp container to instantiate a new assembly that corresponds to the weaving logic. Import this assembly for `C:\Program Files (x86)\SharpDevelop\3.0\AADesigner\weaver.wcc`. In the middle of this assembly, you should notice a UPnP proxy component which name is ControlInterface. It's a component connected to the Control Interface of your application container (that will automatically connect to `localhost:3000`). The other beans implement the logic of the weaving algorithm describe during the Lecture.

But interacting using this weaver assembly is not easy (there isn't any controls but you can identify a lot of UPnP probes components). So we will reify this assembly as a service with the WComp/Custom Binding Attributes menu. Set the following attributes to this container:

- Name: Weaver
- Control Interface: 3100
- Functional Interface: 3101

You should see this new UPnP service in the UPnPWizardDesigner.

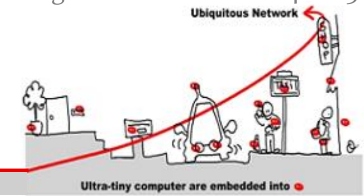
2.2 Aspects of Assembly Designer

In order to interact with the weaver (that will modify the application assembly), you will have to use the AADesigner program. You already have the Aspect of Assembly designer, written for SharpWComp-3.2. You can run the program using the desktop shortcut AA Designer UI.

If you used the right names for application and weaver containers, this tool should discover them and bind itself to the services interfaces. If it's not the case, you should do it manually.
Then you just have to specify the directory where to find the adaptation schema you want to apply to your application (use the commit button at the bottom right of the application). This will update the list of available adaptations (the one that are in the sample directory).

Exercice 1 : Select an adaptation in the list. What is the consequence in the weaver container? What is the consequence in the appli container? Try to add all the elements specified in the pointcut of the AA in the appli container. What is the consequence? Try to suppress one of these elements. What is the consequence?

Tutorial 7: Self Adaptation Middleware and Aspect of Assembly



3 The Aspects of Assembly Designer

Aspects of Assembly provide more generic ways of describing adaptations than writing them in the code of a component (as we made it during last Lab). They are based on a specific language for adaptation rules, and component identification can be made with regular expression matching. Operators of the language also enable complex adaptations to be done and are defined in a way providing the symmetry property of the adaptation process. This means that the resulting application will always be the same and consistent with defined adaptations, no matter what aspect of assembly is weaved first or the conflicts they have in common.

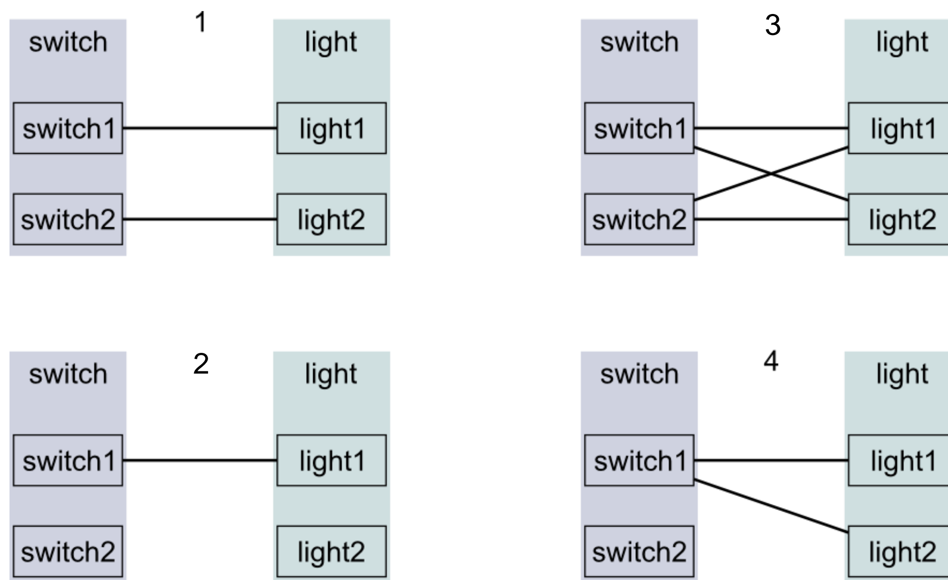
Exercice 2 : Modify the existing AA light-switch to just interconnect switch to light (and not light to switch anymore).

Exercice 3 : Write a new AA that will connect the checkbox to each discovered lights, by inspiring from sample AAs that you can find in the exampleAA directory.

Exercice 4 : Look at the new process chain of bean that appears when you select a new AA. Identify added components in the weaver container when you select one AA rule in AADesigner. What is the role of the Bean components in the weaving process chain? Describe the aim of each added component. Identify the each component corresponding to functionalities in the global schema describing the weaving process (schema presented in lecture).

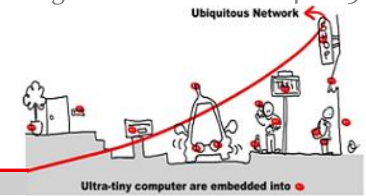
3.1 Multiple identification pointcuts and joint point combinaison

Simple regular expressions can be used to define the pointcuts to identify several components in an assembly. For example, the expression `"*light*"` will identify the components named `light1, light22, room_light1,` etc. When multiple components are identified, a number of policies exist to define how the AA will be woven. To illustrate this, let use the AA discussed above and a component assembly containing `switch1, switch2, light1 and light2`.

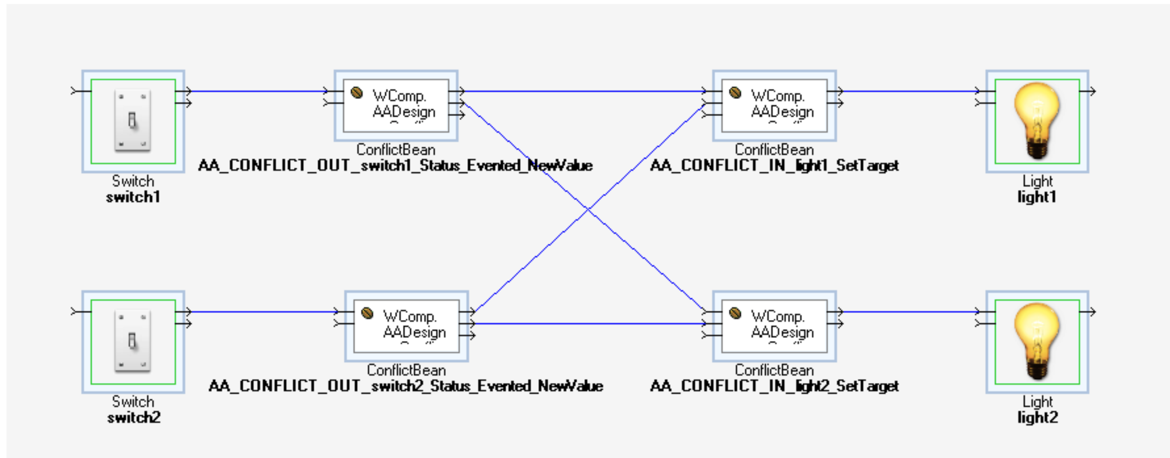


In the Weaver used, the combination policy between join points is number 3 of the figure above. All possible combinations are performed and the advice is duplicated as many times as there are combinations. In each instance of advices, the weaver replaces variables representing the components by the value of the combination.

Tutorial 7: Self Adaptation Middleware and Aspect of Assembly



If you run several times the virtual UPnP light or switch, you will see this process. However, the resulting assembly is somewhat more complex due to the new components introduced by the weaver to solve conflicts between AAs, as can see in the figure below.



These components symbolize the detection of a possible conflict between two links of the assembly, after adaptation of the application. They currently have no effect on the progress of the application, but will eventually replace to merge advices in a specific way.

Exercice 5 : Identify the bean component, in the weaver, that produces the combinations between joint points. We would like to replace it by another one that doesn't produce all the possible combinations but only when joint points are matching like that: lightXXX with SwithXXX. Identify where the JointPoint_Type combination is specified. Try to change it (let's have a look to AA-designer-dev bean category). Test it on simple cases like LightN and SwitchN, where N is the number of instantiation. After changing it, try to deactivate and reactivate the adaptation rules. What can you conclude on how it's currently implemented in the Weaver.

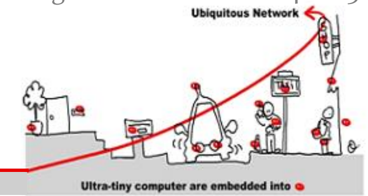
Exercice 6 : If we want to introduce some semantic in the pointcut matching and joint point combination processes, we have to add some meta-data to bean proxy components. What can you propose for that? Explain in both processes and the way to implement that.

The last kind of work that you would have to solve is to introduce dome new components to sole conflicts, instead of the "ConflictBean" currently instantiated that just identify the place where a conflict can occur (potentially some conflict bean can be replaced by nothing.

Exercice 7 : Identify the two king of conflict than can occur after the weaving process. Can you propose some methods to implement a default behavior for each kind of conflict? What is the main difference between these two kinds of conflict?

4 Use case for a complex adaptation with the TrafficLight

Here, this is a complex problem that you must solve using AA. Try to decompose step by step your objectives and the modifications you have to do in the weaving process (even for conflicts and the corresponding generated component to manage these).



Tutorial 7: Self Adaptation Middleware and Aspect of Assembly

Problem: "*To manage one road on an intersection, we need two traffic lights with the same suffix like TrafficLightX_1 and TrafficLightX_2 where X is the name of the road. When I want to add several roads on a same intersection, I need to model conflict that may appear and to define a mechanism to manage them.*"

The objective of such problem is to automatically build an intersection manager from a set of TrafficLight Device for how many roads you want.