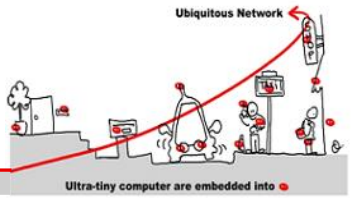
Tutorial 5: Middleware for Ubiquitous Computing, WComp 3.x and Designers



1 Creating a WComp Designer

The aim of this tutorial will be to create a simple adaptation mechanism for the WComp dynamic composition platform. You will first learn how to interact with containers in order to modify their current composition at runtime, using their Web Service for Device interface. Finally, you will learn how to create composite Web services for device with WComp's functional interface.

Start by uninstalling previous versions of WComp you have, including removing generated UPnP proxy beans you had in the repository. Install the latest version (3.2 series) of WComp from the website.

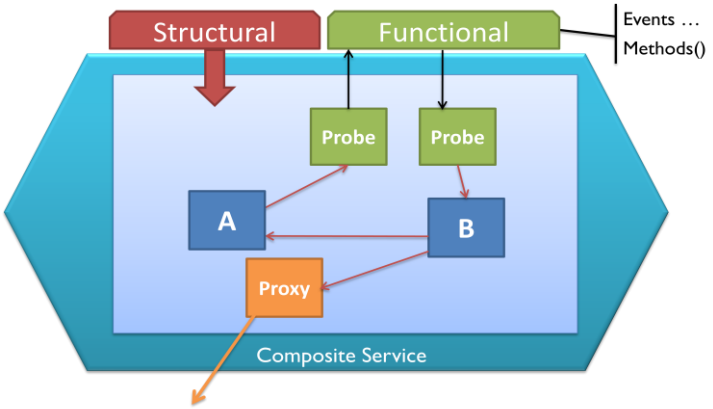
Reminder: you need to have a Windows® OS. Installing SharpDevelop 3.2 and the WComp AddIn is mandatory. In SharpDevelop, you can choose your language in the menu "Outils / Options / Options de SharpDelevop/ Langue de l'utilisateur" in french, or "Tools / Options/ General / UI Language" in english.

1.1 Using the Control Interface

To familiarize with the control interface of the container, you will use the "Tools for UPnP Technologies", that you have to install from this website:

http://opentools.homeip.net/dev-tools-for-upnp

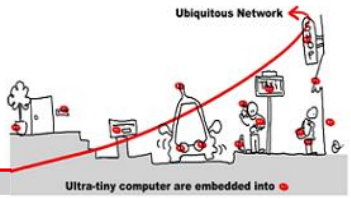
Launch the UPnP "*Device Spy*" tool, a WComp container, and activate the UPnP interfaces of your container using the "WComp.NET / Bind to UPnP Device" menu entry. You will see appear two new devices in the "*Device Spy*", the control interface containing one service (`Container1_Structural_0` for example) and another one (`Container1_Functional_0`) containing no service. It is easier to locate your UPnP devices among others if you disconnect from the school's network.



The control interface allows you to remotely modify the structure of the component assembly of the container. You can instantiate new beans, create new links, or destroy both. The control interface also allows the assembly of components to be displayed and manipulated by other tools than the built-in graphical designer of the SharpDevelop AddIn.

Exercice 1 : Create some beans (like a UPnP proxy on the Light UPnP service) in the container (the application), and invoke the "GetBeans" method on the control interface with the "*Device Spy*". That's a simple way of getting the list of the beans instantiated in the container and their type. You can also find

Tutorial 5: Middleware for Ubiquitous Computing, WComp 3.x and Designers



the type of a bean when you select it, at the top of SharpDevelop's property tab ("GetBeansNames" only allow to get the bean names without their type).

Exercice 2 : Subscribe to container structural events (right click on the service in the "*Device Spy*") and try to create some components and links in the assembly using the control interface.

Exercice 3 : Try to create a button bean with the structural service of your container. Create a link to connect the light and the checkbox to allow switch on the light.

1.2 The UPnP Wizard Designer

You will now see an example of designer that automatically instantiates UPnP proxy components when UPnP *devices* are discovered on the network: the UPnP Wizard Designer. It should already been installed and available with on shortcut on your desktop for more convenience (it is installed `in C:\Program Files (x86)\SharpDevelop\3.0\UPnPWizardDesigner`).

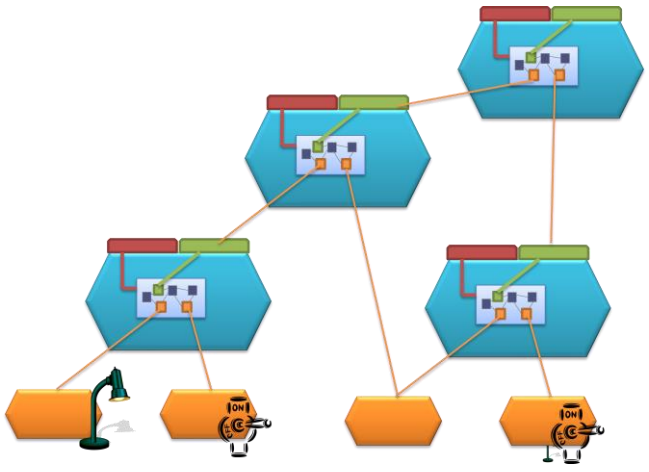
Launch the designer, and start the UPnP interfaces of a SharpWComp container. From this moment, if you launch a UPnP device like the "*Network Light*", the proxy component will be automatically generated, compiled, loaded in the container, and instantiated with the correct URI property. The proxy component will also be removed when the device disappears.

You have to define with which container you want to connect this tool by selecting the right one in the Connect menu. You can also filter some device type to avoid to automatically create beans for such kind of devices.

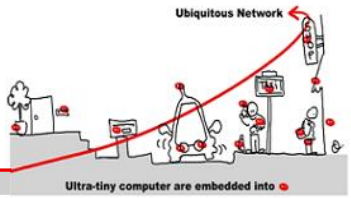
Exercice 4 : Destroy all the beans in your container. Start the UPnPWizardDesigner, filter the unwanted type of UPnP devices and start your UPnP Light. You should obtain a UPnP proxy bean corresponding to this UPnP device instantiated in your container.

2 More WComp Interfaces: Creating a Composite Service

You have now seen how to use the control interface of WComp containers to remotely manage their internal assembly of components, resulting in dynamic application adaptation. A second Web service for device interface is provided by containers, the *functional interface*. It allows to export the functionalities created by the component assembly as a new (composite) Web service for device. Functionalities can then be distributed as a net of containers, as you can see below.



Tutorial 5: Middleware for Ubiquitous Computing, WComp 3.x and Designers



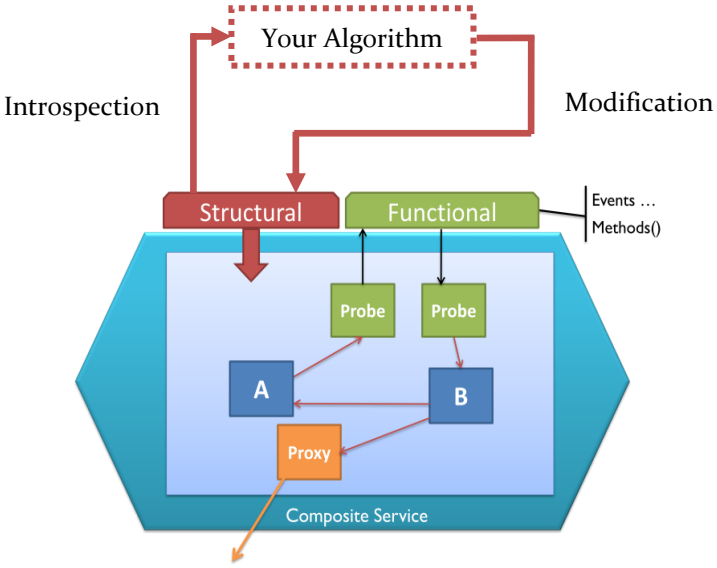
Two specific types of components are used to manipulate the functional interface, called probe components. The "EmitProbe" component adds an action (method) to the UPnP functional interface of the container, and when invoked, an event with the same argument is emitted in the assembly of components. Conversely, the "SourceProbe" component adds an evented variable to the UPnP functional interface of the container, and this event is emitted when the method in the component assembly is invoked. These two components can be found in the "UPnP Probes" category in the tools tab.

The most basic example using two probes is to create an EmitProbe and a SourceProbe and connecting the event of the first to the method of the second. The resulting application will be a simple UPnP relay: by invoking the "FireEvent(string)" action on the functional interface, an event will be immediately sent on the sourceProbe1 event of the other service of the functional interface. Note that it can also be done with only one EmitProbe, since this probe provides the capability of sending an event corresponding to a return value of the invoked action.

The goal of this interface is to export functionalities created as an assembly of components into a new service, which can be later reused as any service provided by a device. To illustrate that, take the designer and the assembly you created in exercise 1.3, and replace the checkbox by an EmitProbe to provide a UPnP action that will switch on all lights. You should now be able to control the light with the Device Spy and in fact with any other WComp container instantiating a proxy on your first container service.

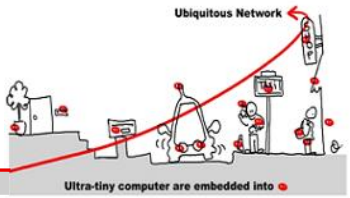
3 Reacting to infrastructure modifications

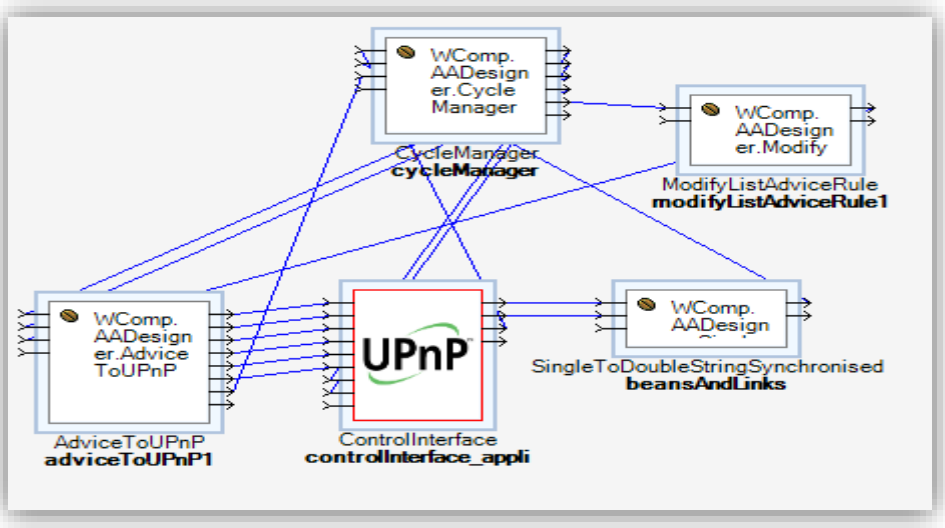
When you want to react to modifications of you application, you have to monitor your application and introspect what components are defining the assembly.



Such a functionality can be implemented with the following assembly (this assembly is provided to you with the `introspect-modify.wcc` file):

Tutorial 5: Middleware for Ubiquitous Computing, WComp 3.x and Designers





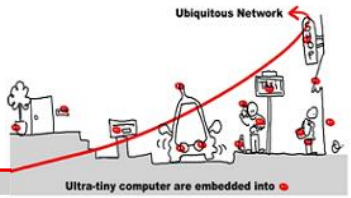
The `ControlInterface` bean is a proxy bean to the application container. It can send to the `CycleManager` bean the list of components and links that composes the assembly running in the application container. The `CyleManager` bean can send these data (list of components and links) as an event (CurrentAssemblyEvent). These data are send to the `AdviceToUPnP` component that convert all the actions (CreateBean, CreateLink, RemoveBean, RemoveLink,…) and send them as events to the application container `ControlInterface`.

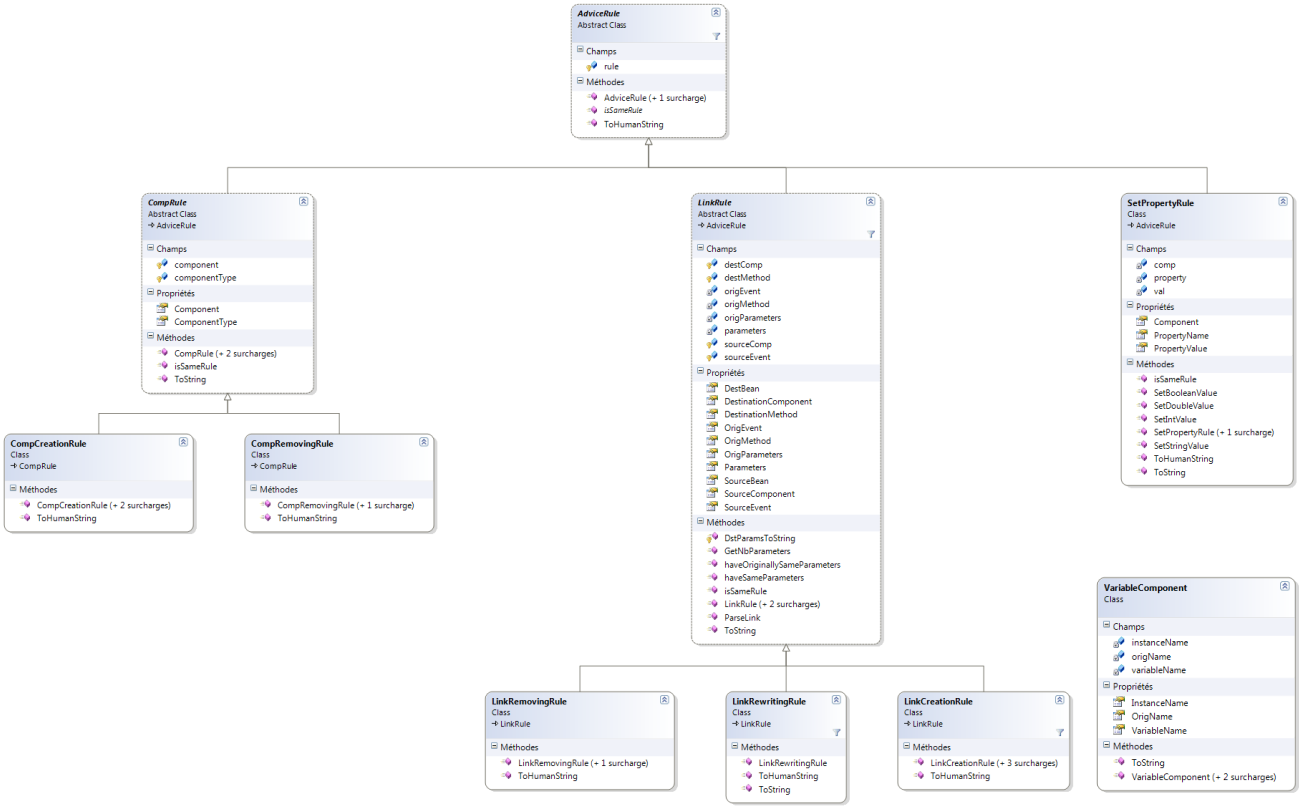
In such an architecture, you can add a bean that deals with the application assembly and try to add some modifications to it depending on rules you define. This is the aim of ModifyListAdviceRule bean.

The assembly is defined as a List of "AdviceRule". A rule can be a Component or a Link rule. A component rule can be to create or to remove a component from the assembly. The hierarchy of classes defining the kind of existing rules are defined as a set of classes. The UML diagram of the useful classes is the following one (a more readable format is in the file `ClassDiagram.png`).

The `ModifyListAdviceRule` bean received a List of "AdviceRule" (CurrentAssembly) and can generate an event to add rules to this basic assembly (AddRules). The AdviceToUPnP component will take the basic assembly (a List of AdviceRule corresponding to the assembly in the application assembly) and will mix all these rules with the Added rules to produce the diff between the initial assembly and the aimed assembly and send all the actions to the container to add or remove bean or links. If you want to modify add some rules, you have to modify `AutoAdaptation` method in the source code of `ModifyListAdviceRule` to implement the kind of algorithm you would like.

Tutorial 5: Middleware for Ubiquitous Computing, WComp 3.x and Designers





Exercice 5 : Add an algorithm to modify the application assembly in the following way. Launch Light and Switch UPnP service and connect them manually to automatically switch on the light when you activate switch (you have to interconnect the switch and the link with a link between (Switch.^StatusEvent to Light.SetTarget). Your self-adaptive algorithm must remember all the created links between beans and if beans are destroyed and recreated (UPnP service stops and starts again after a moment), your algorithm must recreate the previously existing links between beans. So you have to save the links list of the initial assembly in a variable and when you introspect the assembly you have to add rules to recreated links if they are not in the current assembly.