First steps with the SCA Composite Designer

Discover SCA and the SCA Composite Designer

Stéphane Drapeau – Obeo

STP/SCA Composite Designer leader

March 5, 2008 (V0.1.0)

Abstract: The objective of this tutorial is to discover the STP/SCA Composite Designer through the development of a simple SCA application. The tutorial illustrates:

- How to install STP/SCA plugins and Apache Tuscany,
- How to define an SCA assembly with the SCA Composite Designer,
- How to refine properties in an SCA assembly,
- How to define an SCA assembly with different implementation technologies,
- How to define an RMI binding,
- How to run and test SCA assemblies with Tuscany.

Summary

Installation

The first thing you must to do is to install the STP/SCA plugins and the Tuscany distribution.

Install the STP/SCA plugins

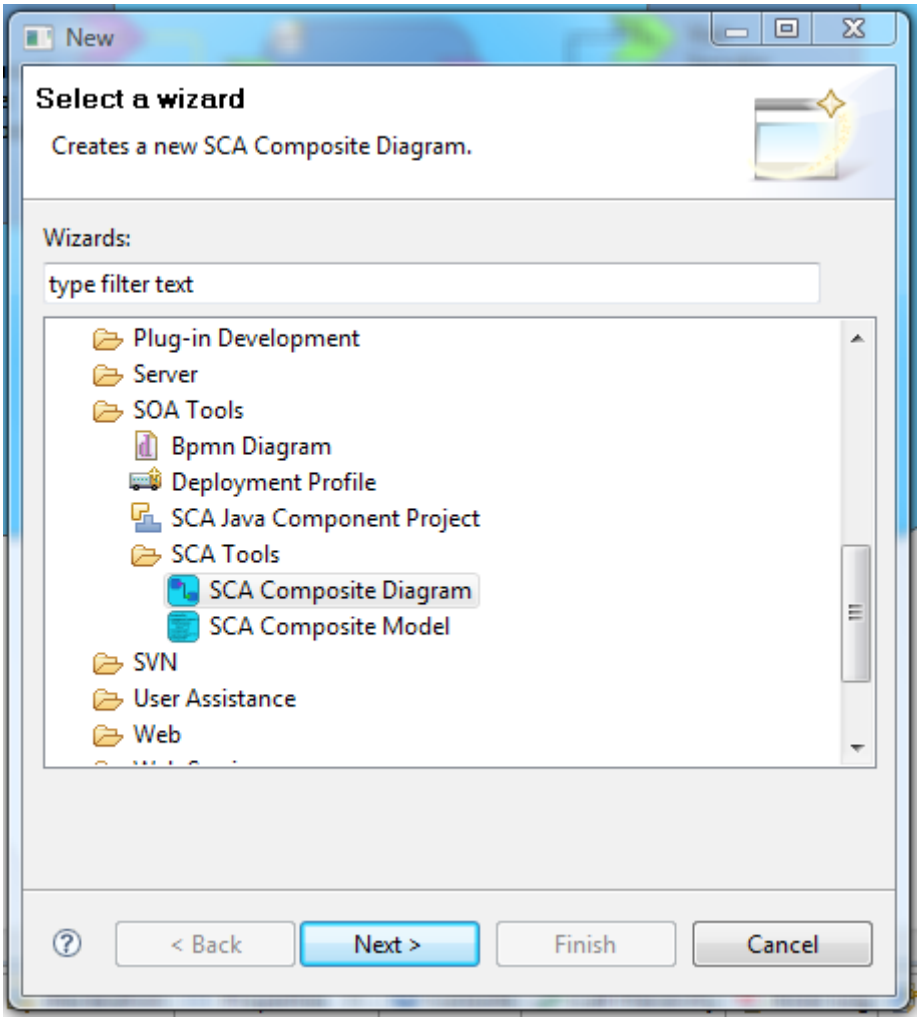
The STP/SCA tools need EMF 2.3.0 and GMF 2.0.1. You can download all the GMF requirements plus the Eclipse platform here and GMF 2.0.1 here.

Once you have installed Eclipse with all requirements, download the STP/SCA plugins here (check for the last build in Download section).

Unzip the file, and install the STP/SCA plugins in the Eclipse plugins directory.

Start Eclipse. Select **File>New>Other**. Two new creation wizards appear:

- « SCA Composite Diagram » and
- « SCA Composite Model ».

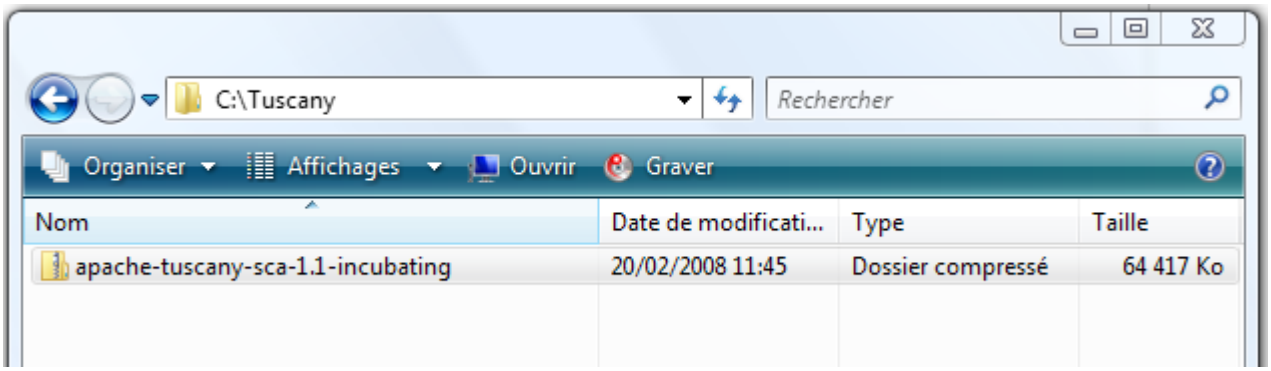


Install the Tuscany distribution

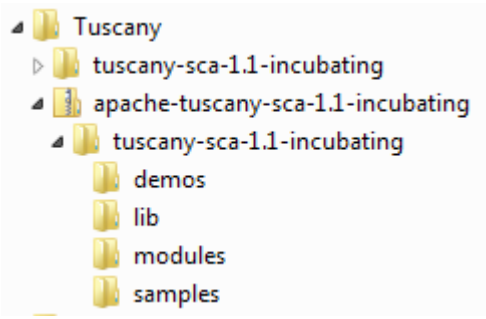
Now that Eclipse and STP/SCA are installed, it is necessary to install Tuscany.

First, create a folder on your disk where you will save the Tuscany distribution. Then, open your browser and enter the following URL to download the latest Tuscany release distribution: http://incubator.apache.org/tuscany/sca-java-releases.html

Download the bin zip to the folder that you created on your disk.



Next, unzip the file. You should see the following folder structure.

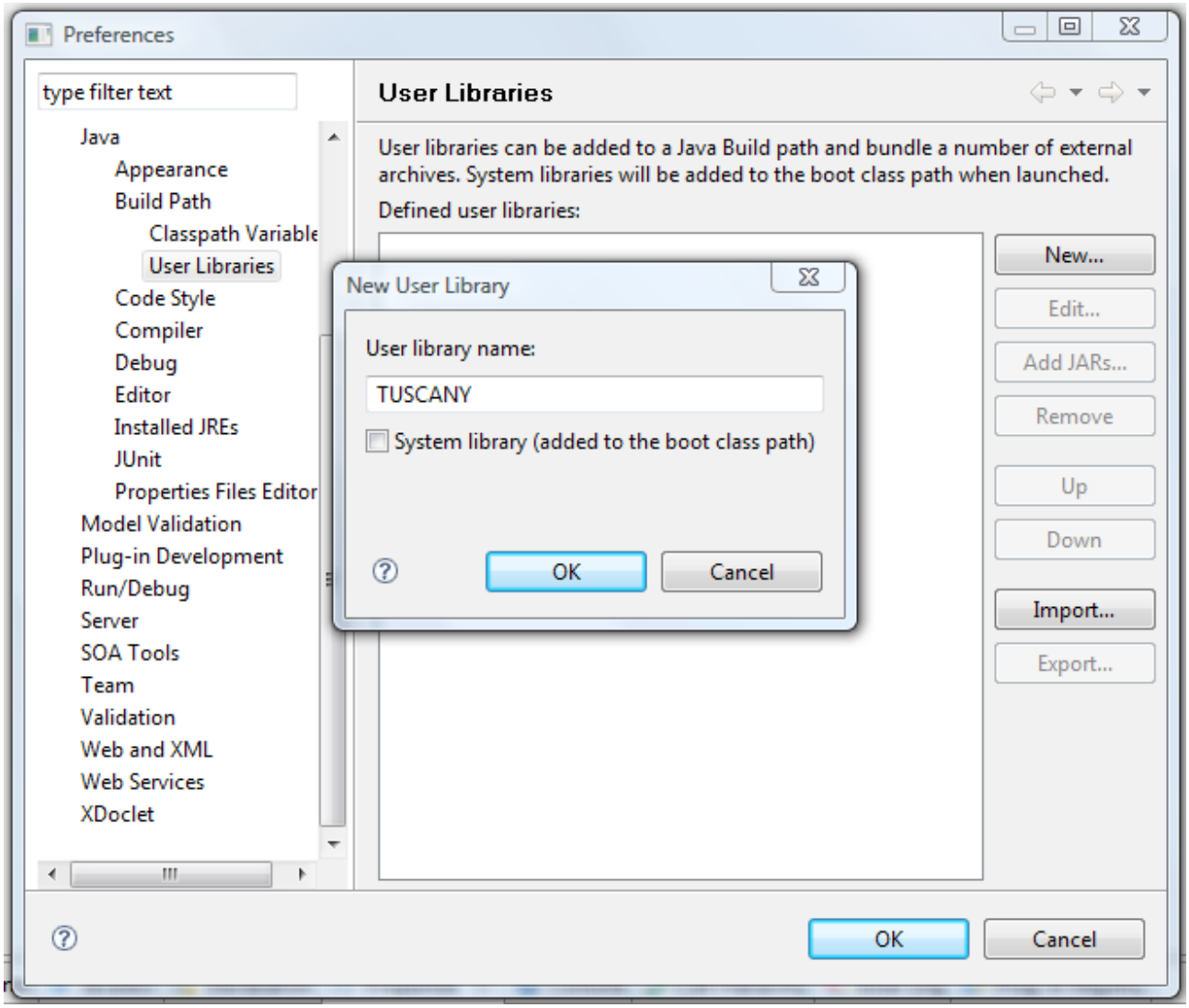


Setup Eclipse for Tuscany

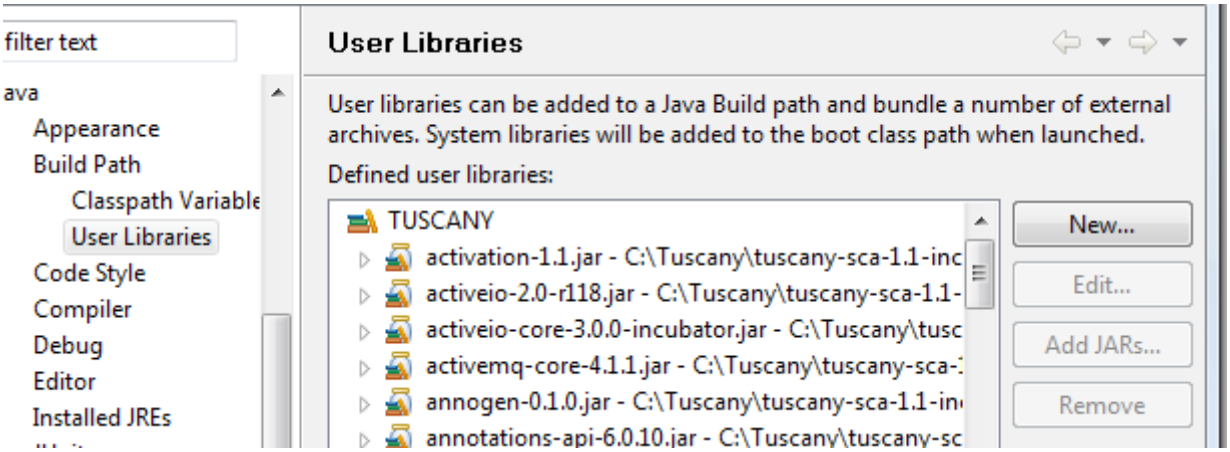
Start Eclipse.

Create a **User Library** that will contain the Tuscany runtime jars and depending jars:

- From the menu bar select **Window>Preference...**.
- The preferences dialog will appear. Select **Java>Build Path>User Libraries**.
- Click **New** button to create a new user library.
- Enter TUSCANY as the name of the user library.

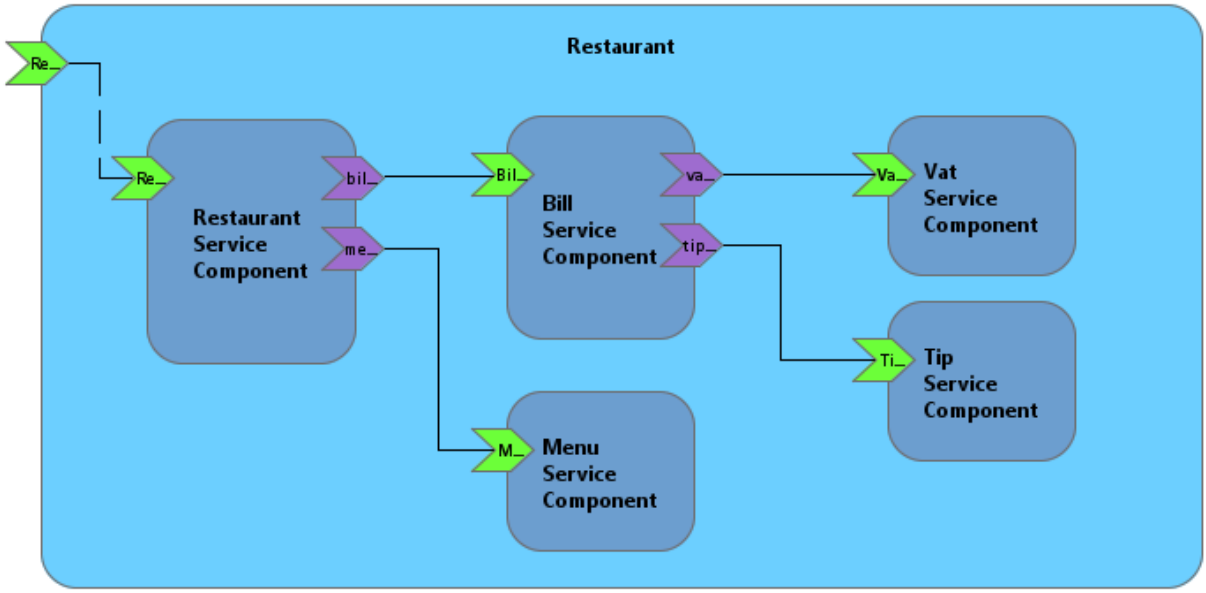


The user library created is empty. Push **Add Jars**... button. Add all the jars from the Tuscany folder that you have installed before.



The SCA Restaurant application

The following figure shows the SCA assembly of the application that you will create.



This composite, named restaurant, is a composition of five components:

- **RestaurantServiceComponent**: This component allows to see the Menus proposed by the restaurant. It allows also to compute the bill for a particular menu.
- **MenuServiceComponent:** This component provides different menus. A Menu is defined by a description and the price without taxes.
- **BillServiceComponent**: This component computes the price of a menu with the different taxes.
- **VATServiceComponent**: This component computes the VAT (Value Added Tax).
- **TipServiceComponent**: This component computes the tip.

In the following, we will see:

- How to implement this application as an SCA application,
- How to refine a component property
- How to define an SCA assembly with different implementation technologies,
- How to access the service proposed by this application throught RMI.

First SCA application

Create a new Java Project

First, create a **Java Project** to hold the Restaurant application:

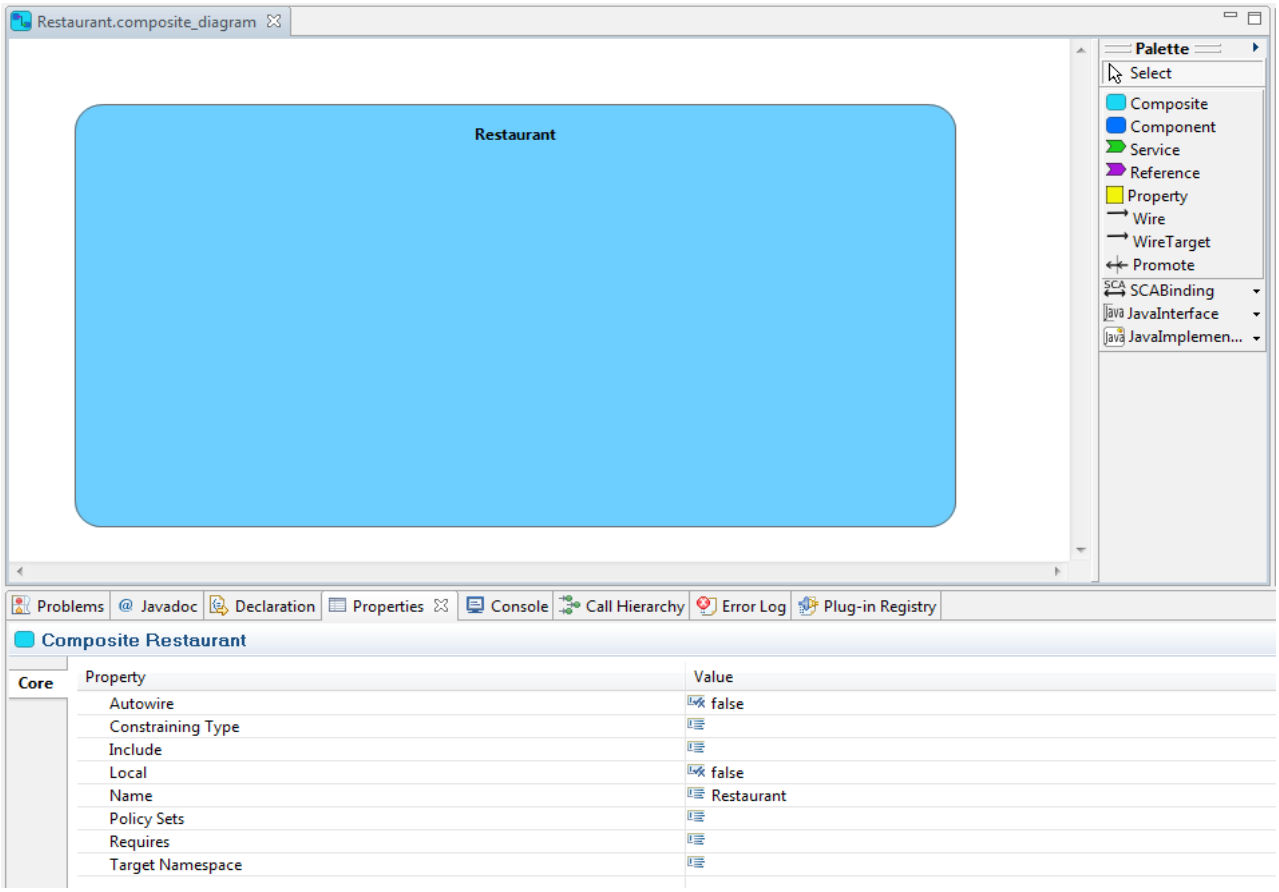
- Select **File>New>Java Project**.
- Set « Restaurant » as the **Project name**. For **Project layout**, select **Create separate folders for sources and class files**. Click on the **Next** button.
- Go to **Librairies tab** and add TUSCANY library (**Add Library...** button -> Select **User Library** -> **Next** button -> Select TUSCANY -> **Finish** button).
- Click on the **Finish** button.

Create a new SCA Composite diagram

To create a new SCA Composite Diagram:

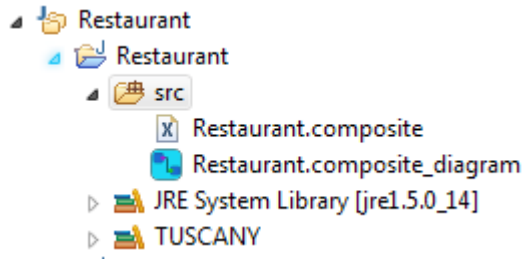
- Do a right click on the src folder of the Restaurant project, then select **New>Other...**.
- The New wizard will appear. In the **SCA Tools** folder, select **SCA Composite Diagram** then click on the **Next** button.
- Set Restaurant.composite_diagram as **file name**.
- Click on the **Finish** button.

The new created file is open automatically with the SCA Composite Designer. To open the Properties view, do a right click on the diagram and then select **Show Properties View**.



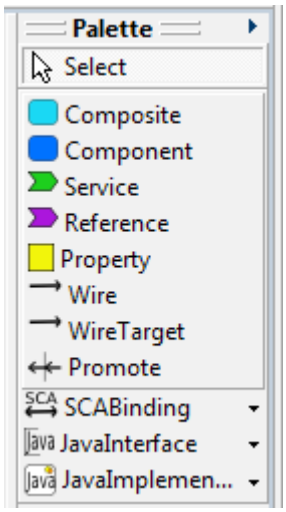
In accordance with the SCA specifications, the name of the composite is the same of the composite file. The name of the composite is set automatically.

Now, you should see the following project structure:

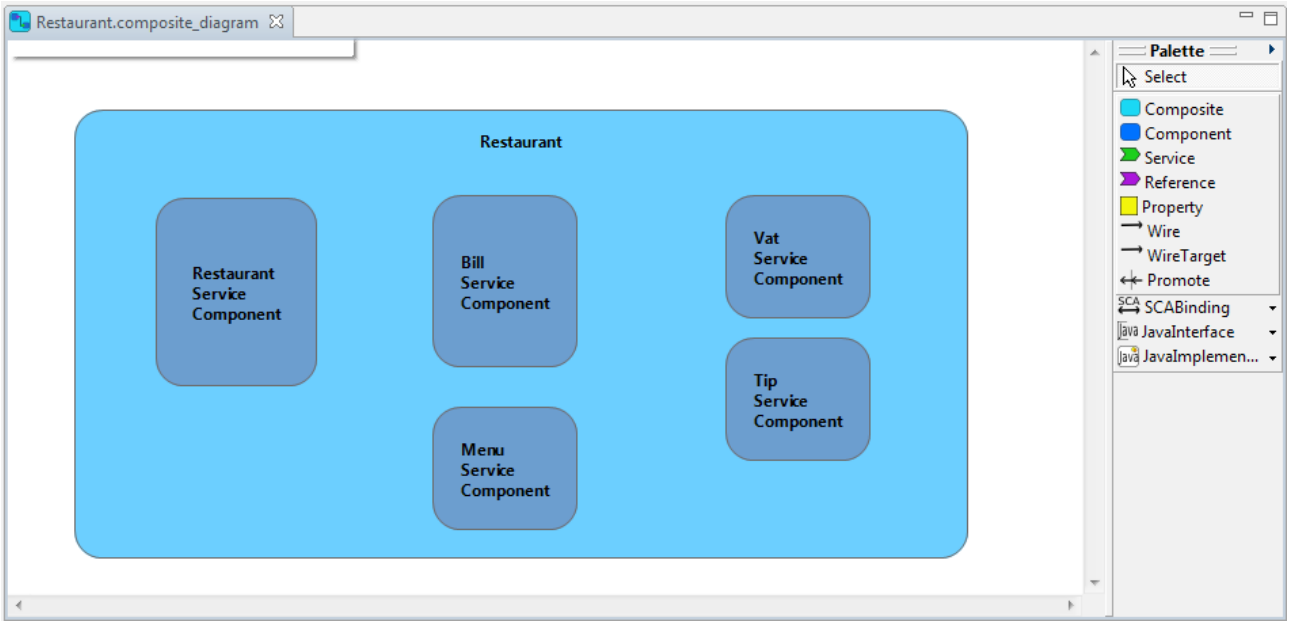


Create a Restaurant Composite

Add a first component named RestaurantServiceComponent. You can do it with the Component creation tool which is in the palette or using the contextual menu.

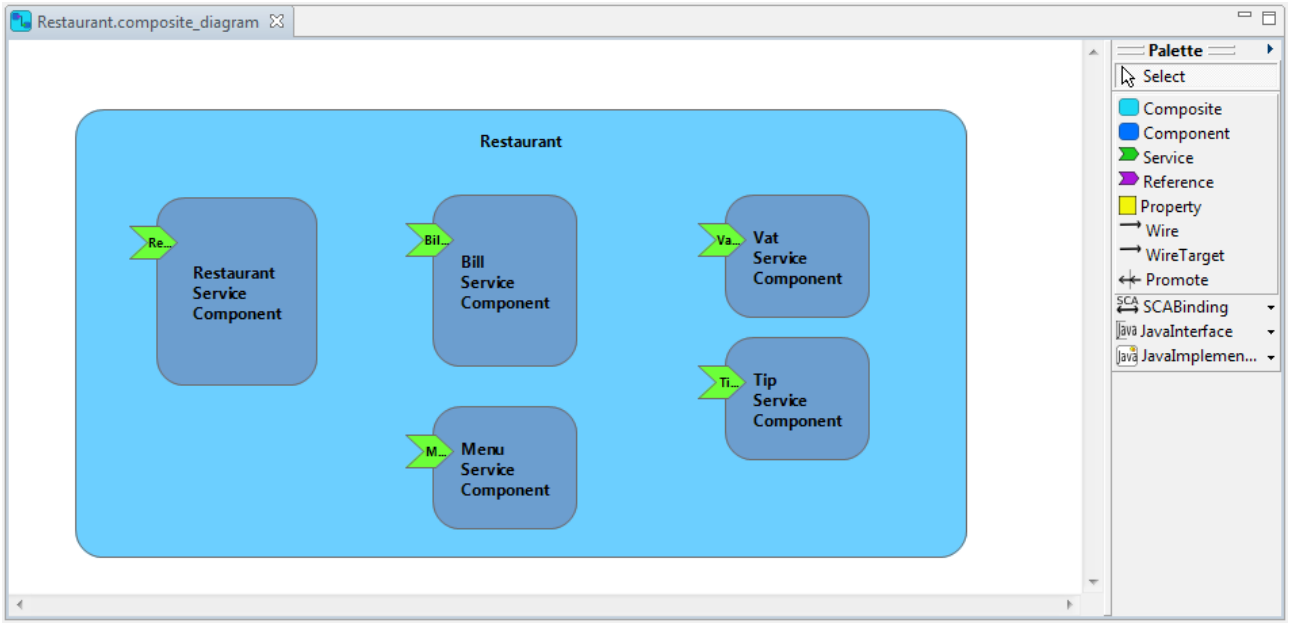


Then, add MenuServiceComponent, BillServiceComponent, VatServiceComponent and TipServiceComponent.



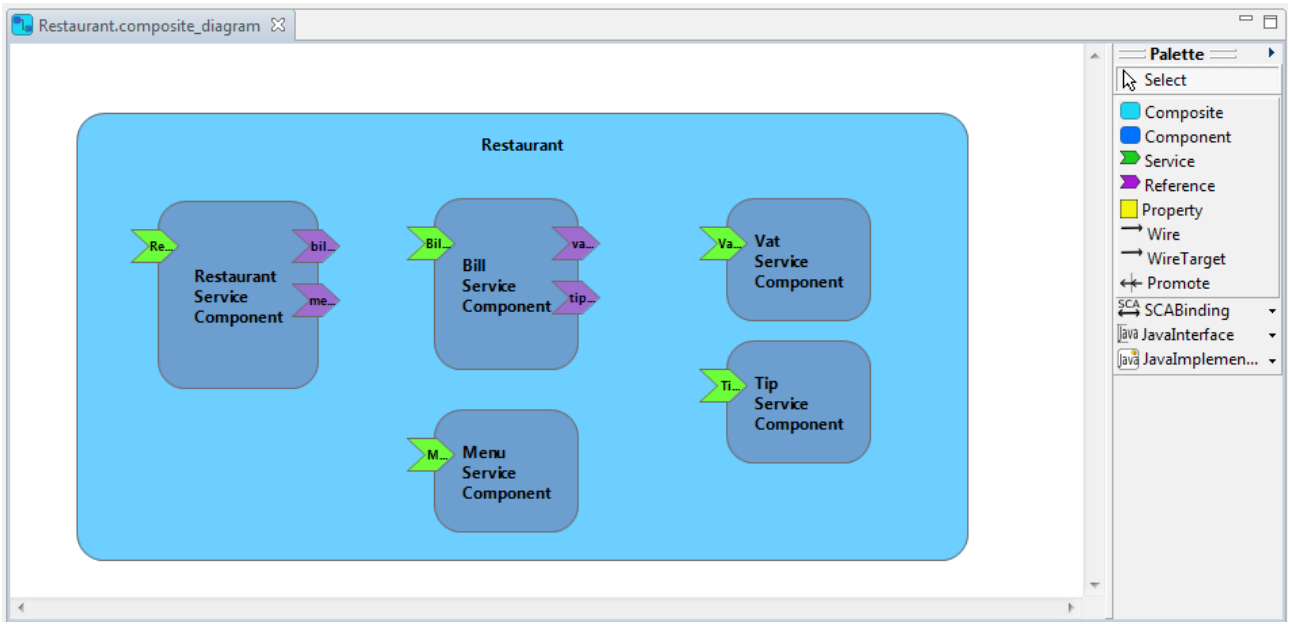
Next, add the following services:

- A service named RestaurantService on the restaurant service component
- A service named MenuService on the menu service component
- A service named BillService on the bill service component
- A service named VatService on the vat servicecomponent
- A service named TipService on the tip service component



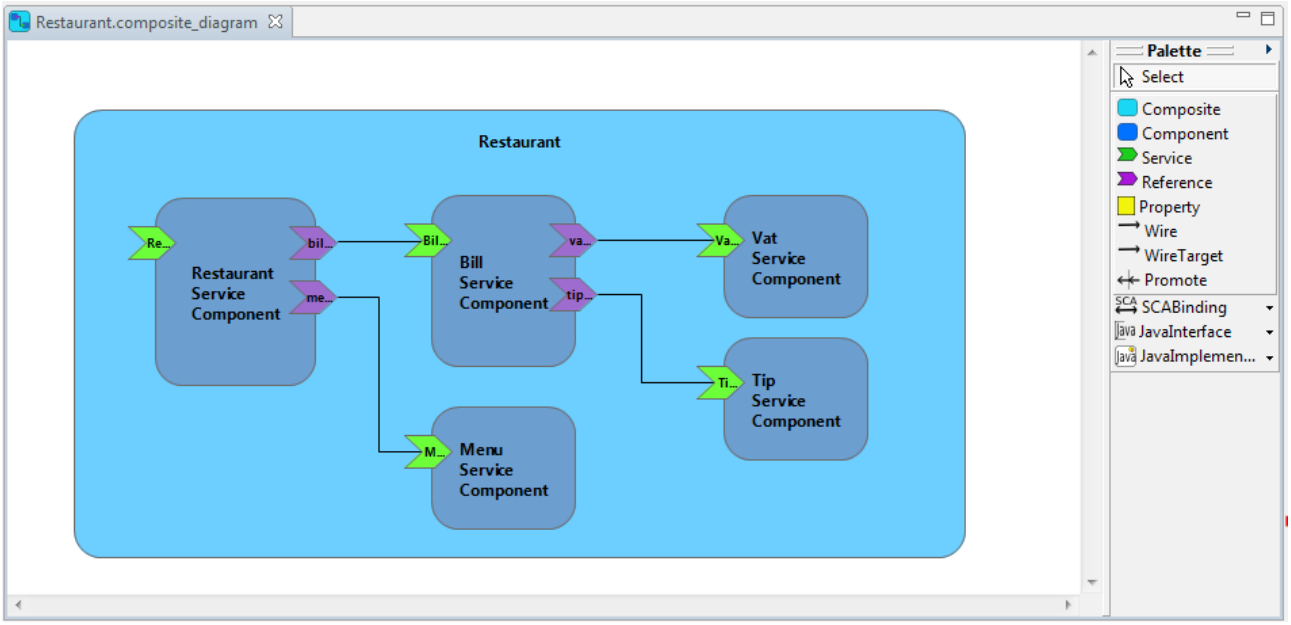
In the next step add the following references:

- A reference named billService on the restaurant service component
- A reference named menuService on the restaurant service component
- A reference named vatService on the bill service component
- A reference named tip service component



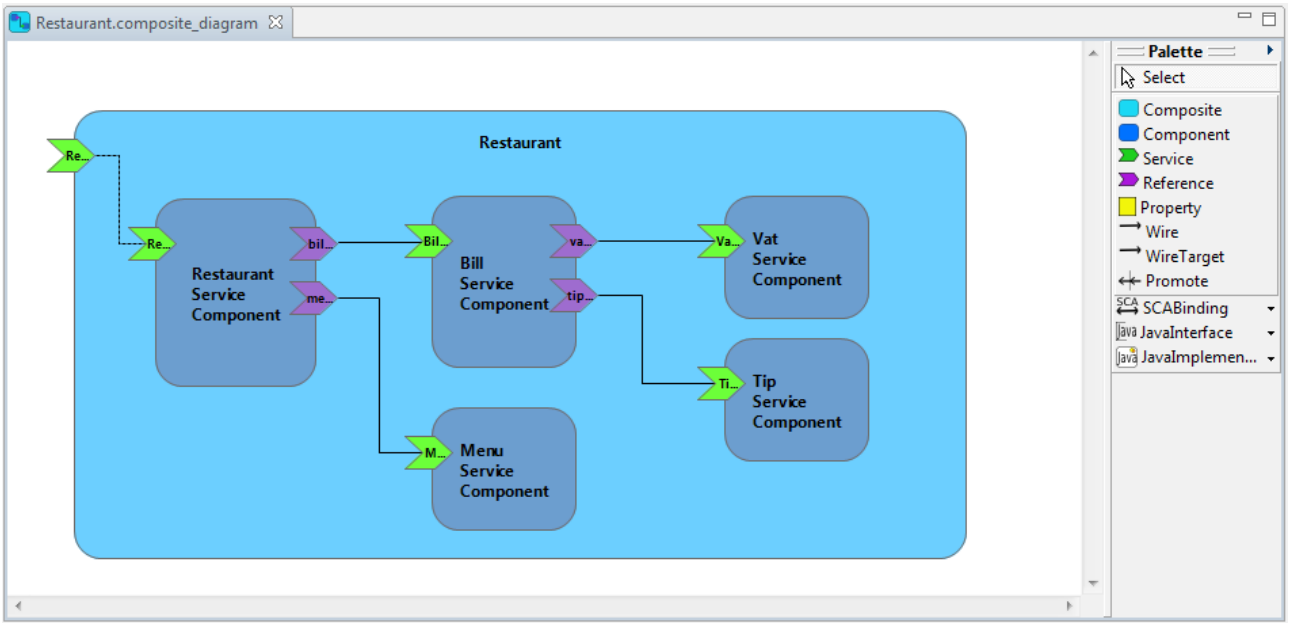
Now you can wire the services and the references. From the palette, you can use:

- the **Wire** creation tool: a Wire element is added,
- the **Wire target**: target attribut of the Reference element is used.



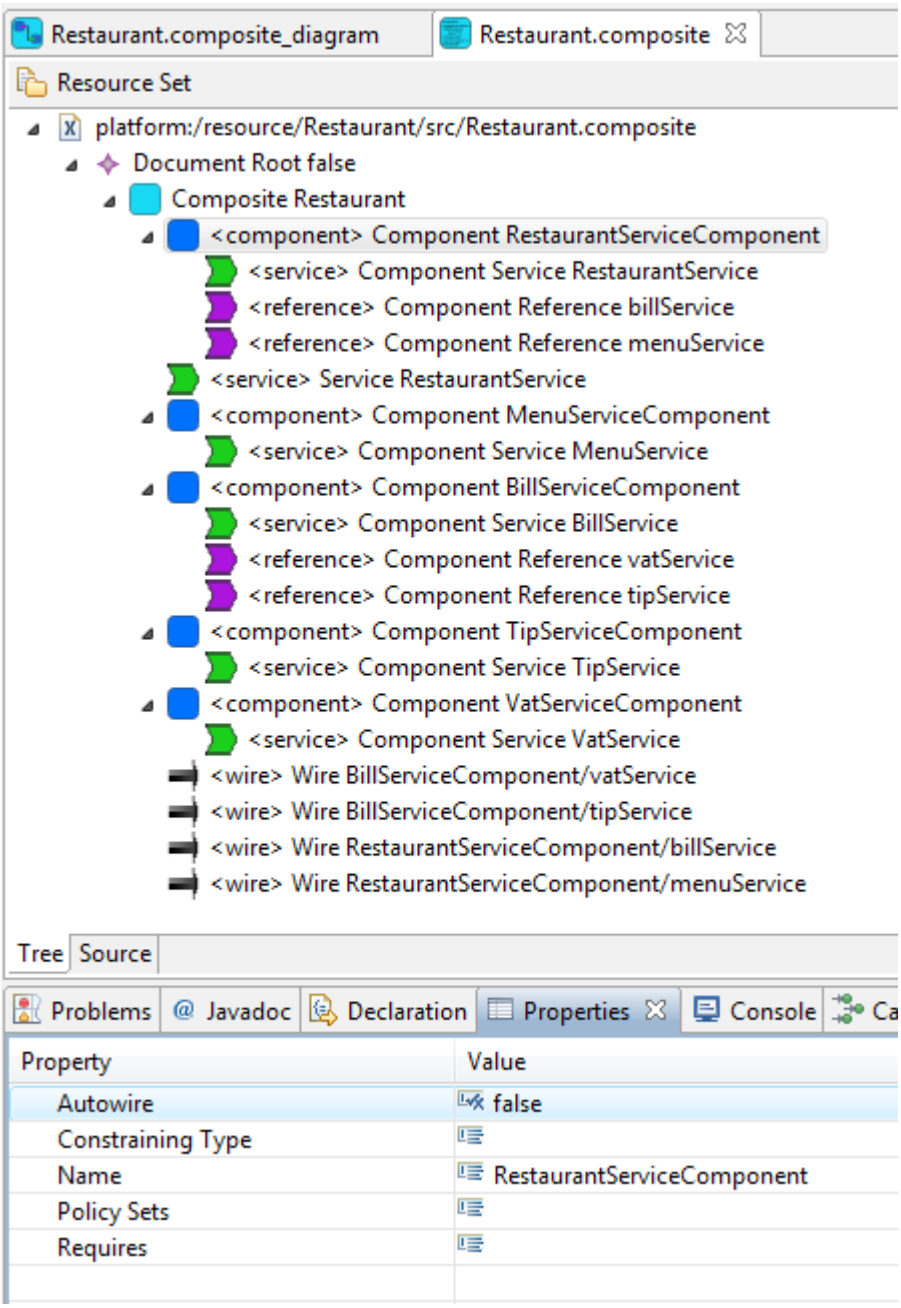
The next step is to promote the restaurant service. Two ways are possible to promote a service (or a reference). You can:

- create a new Service on the composite and then use the **Promote** creation tool from the palette to add a promotion link between the composite service and the promoted component service, or
- do a right click on a component service, and select **Promote** menu item.



Save your diagram. The **Restaurant.composite_diagram** contains the graphical part of your SCA assembly and the **Restaurant.composite** file contains the XML code that describes your SCA assembly.

You can open the **Restaurant.composite** file with the SCA Composite Model Editor: do a right click on the Restaurant.composite file, select **Open with>SCA Composite Model Editor**. A multi page editor is opened. It offers a tree view and the source code of your SCA assembly. You can modify your SCA assembly with these two editors. They are synchronized with the SCA Composite Designer.



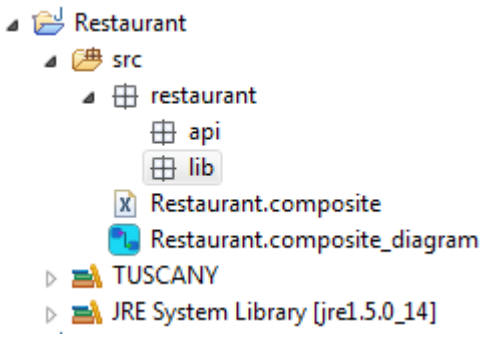
Restaurant.composite_diagram | Restaurant.composite

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<sca:composite xmlns:sca="http://www.osoa.org/xmlns/sca/1.0" name="Restaurant">
  <sca:component name="RestaurantServiceComponent">
    <sca:service name="RestaurantService"/>
    <sca:reference name="billService"/>
    <sca:reference name="menuService"/>
  </sca:component>
  <sca:service name="RestaurantService" promote="RestaurantServiceComponent/RestaurantService"/>
  <sca:component name="MenuServiceComponent">
    <sca:service name="MenuService"/>
  </sca:component>
  <sca:component name="BillServiceComponent">
    <sca:service name="BillService"/>
    <sca:reference name="vatService"/>
    <sca:reference name="tipService"/>
  </sca:component>
  <sca:component name="TipServiceComponent">
    <sca:service name="TipService"/>
  </sca:component>
  <sca:component name="VatServiceComponent">
    <sca:service name="VatService"/>
  </sca:component>
  <sca:wire source="BillServiceComponent/vatService" target="VatServiceComponent/VatService"/>
  <sca:wire source="BillServiceComponent/tipService" target="TipServiceComponent/TipService"/>
  <sca:wire source="RestaurantServiceComponent/billService" target="BillServiceComponent/BillService"/>
  <sca:wire source="RestaurantServiceComponent/menuService" target="MenuServiceComponent/MenuService"/>
</sca:composite>

```

Tree | Source

Define interfaces and implementations

First, in the src directory of the Restaurant project create a new package named restaurant with two sub-packages named api and lib.



Define the interfaces of the services (**restaurant.api** package).

– Restaurant service:

```
package restaurant.api;

public interface RestaurantService {
      Menu[] getMenus();
      double getBill(Menu menu);
}
```

– Menu service:

```
package restaurant.api;

public interface MenuService {
```

```
	Menu[] getMenu();
	double getPrice(Menu menu);
}
```

- Bill service:

```
package restaurant.api;

public interface BillService {
	double getBill(double menuPrice);
}
```

- Vat service:

```
package restaurant.api;

public interface VatService {
	double getPriceWithVat(double price);
}
```

- Tip service:

```
package restaurant.api;

public interface TipService {
	double getPriceWithTip(double price);
}
```

Define also the interface of the Menu (Data Transfert Object):

```
package restaurant.api;

import java.io.Serializable;

public interface Menu extends Serializable {
	String printMenu();
}
```

Next, define the implementations (**restaurant.lib** package).

- Restaurant service

```
package restaurant.lib;

import org.osoa.sca.annotations.Reference;
import org.osoa.sca.annotations.Service;

import restaurant.api.BillService;
import restaurant.api.Menu;
import restaurant.api.MenuService;
import restaurant.api.RestaurantService;

@Service(RestaurantService.class)
public class RestaurantServiceImpl implements RestaurantService {

	private MenuService menuService;
	private BillService billService;

	@Reference
	public void setMenuService(MenuService menuService) {
```

```
		this.menuService = menuService;
	}

	@Reference
	public void setBillService(BillService billService) {
		this.billService = billService;
	}

	public double getBill(Menu menu) {
		double menuPrice = this.menuService.getPrice(menu);
		return this.billService.getBill(menuPrice);
	}

	public Menu[] getMenus() {
		return this.menuService.getMenu();
	}

}
```

- Menu service

```
package restaurant.lib;

import org.osoa.sca.annotations.Init;
import org.osoa.sca.annotations.Service;

import restaurant.api.Menu;
import restaurant.api.MenuService;

@Service(MenuService.class)
public class MenuServiceImpl implements MenuService {

	private Menu[] menus;

	private double[] prices;

	@Init
	public void init() {
		this.menus = new Menu[] {
				new MenuImpl(0, "Grilled hamburger with French fries" ),
				new MenuImpl(1, "Roasted chicken with vegetables"),
				new MenuImpl(2, "Duck breast in an orange sauce"),
				new MenuImpl(3, "Duck foie gras & mango chutney") };
		this.prices = new double[] { 10, 15, 35, 50 };
	}

	public Menu[] getMenu() {
		return this.menus;
	}

	public double getPrice(Menu menu) {
		return this.prices[((MenuImpl) menu).getId()];
	}

}
```

- Bill service

```
package restaurant.lib;

import org.osoa.sca.annotations.Reference;
```

```
import org.osoa.sca.annotations.Service;

import restaurant.api.BillService;
import restaurant.api.TipService;
import restaurant.api.VatService;

@Service(BillService.class)
public class BillServiceImpl implements BillService {

	private VatService vatService;
	private TipService tipService;

	@Reference
	public void setVatService(VatService vatService) {
		this.vatService = vatService;
	}

	@Reference
	public void setTipService(TipService tipService) {
		this.tipService = tipService;
	}

	public double getBill(double menuPrice) {
		double pricewithTaxRate =
				this.vatService.getPriceWithVat(menuPrice);
		double priceWithTipRate =
				this.tipService.getPriceWithTip(pricewithTaxRate);
		return priceWithTipRate;
	}

}
```

- Vat service

```
package restaurant.lib;

import org.osoa.sca.annotations.Service;

import restaurant.api.VatService;

@Service(VatService.class)
public class VatServiceImpl implements VatService {

	public double vatRate;

	public VatServiceImpl(){
		this.vatRate=19.6;
	}

	public double getPriceWithVat(double price) {
		return price * this.vatRate/100 + price;
	}

}
```

- Tip service

```
package restaurant.lib;

import org.osoa.sca.annotations.Property;
import org.osoa.sca.annotations.Service;
```

```
import restaurant.api.TipService;

@Service(TipService.class)
public class TipServiceImpl implements TipService {

	@Property
	public double tipRate;

	public TipServiceImpl(){
		this.tipRate=10;
	}
	public double getPriceWithTip(double price) {
		return price *  this.tipRate/100 + price;
	}

}
```

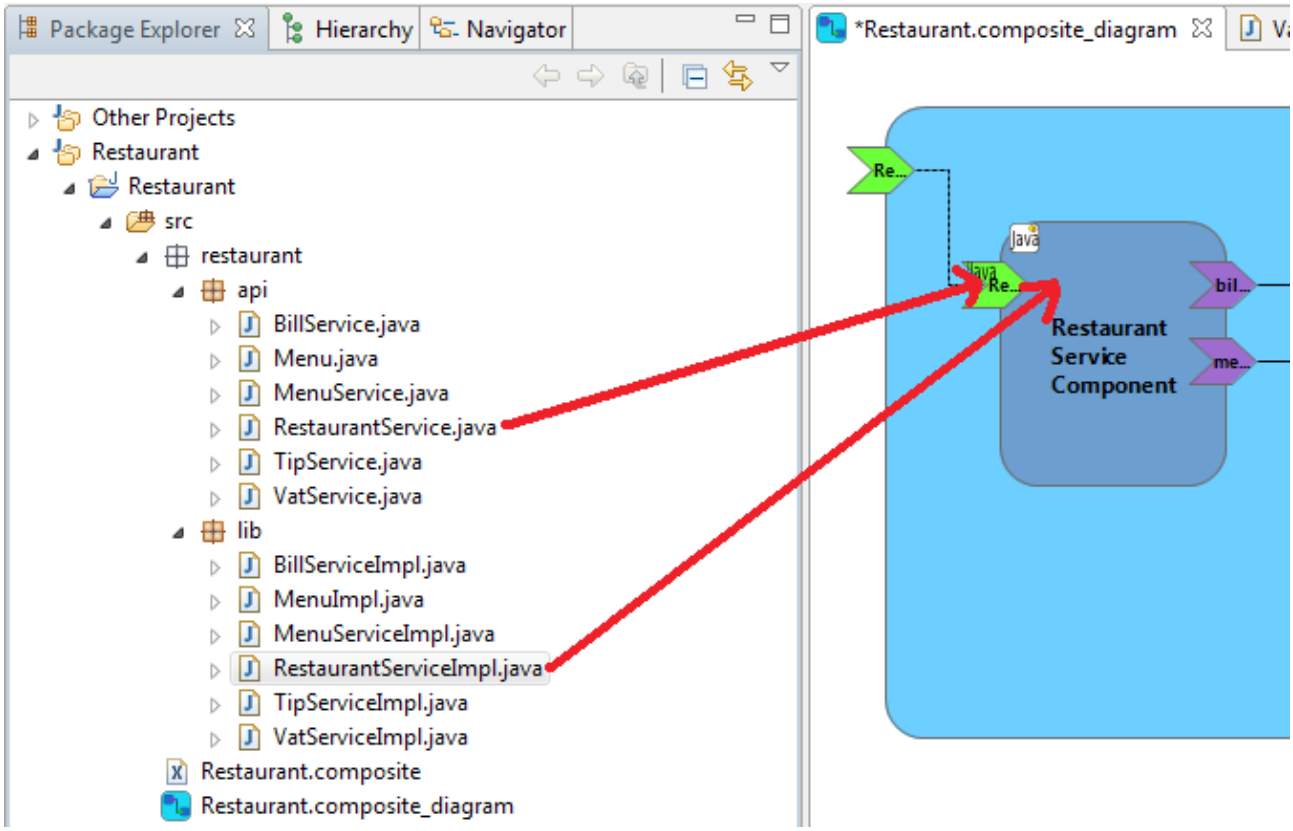
- Menu

```
package restaurant.lib;

import restaurant.api.Menu;

public class MenuImpl implements Menu {

	private int id;
	private String details;

	MenuImpl(int idC, String detailsC) {
		this.id = idC;
		this.details = detailsC;
	}

	public String printMenu() {
		return this.details;
	}

	int getId() {
		return this.id;
	}

}
```

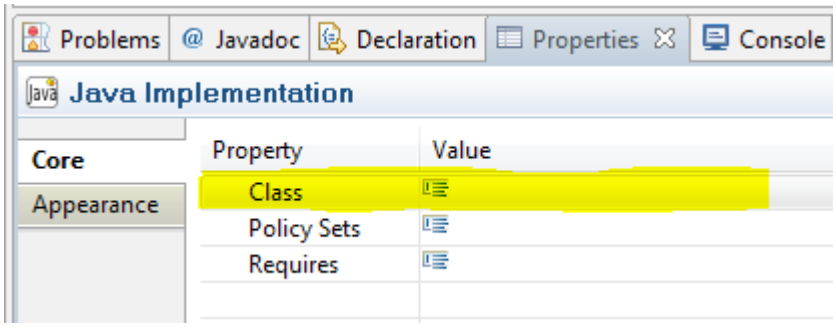
Now, you can fill your SCA assembly file with the interfaces and implementations that you defined. Open the **Restaurant.composite_diagram** with the SCA Composite Designer.

You have two ways to fill your SCA assembly file:

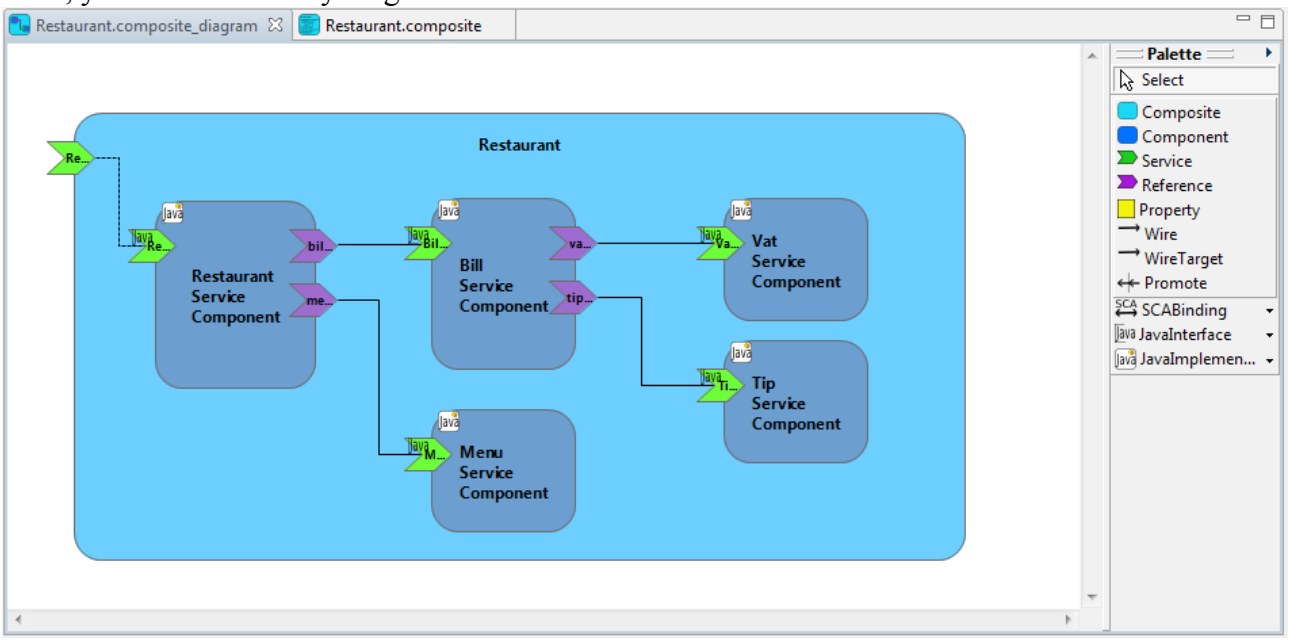
- Drag and drop the Java interface files on the services of your diagram and the Java implementation files on the components. The name of these elements are set automatically.



- Use the JavaInterface and the JavaImplementation creation tools from the palette or the contextual menu. For each created element, you must set (in the **Properties** view):
  - the Interface attribute for a Java interface and
  - the Class attribute for a Java implementation.



Now, your SCA assembly diagram looks like:



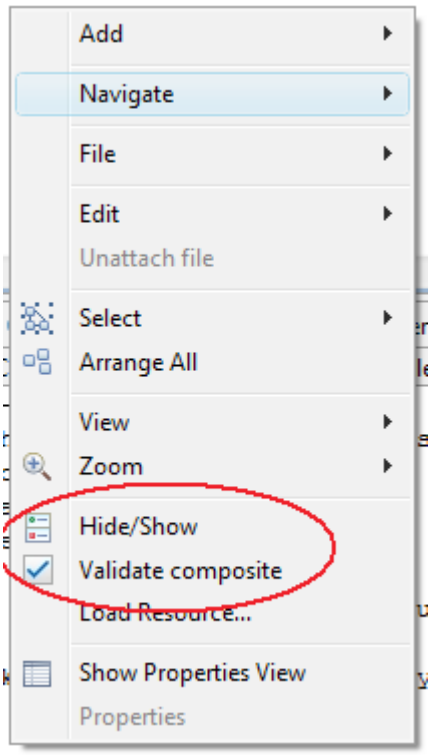
and your SCA assembly file looks like:

Restaurant.composite_diagram | Restaurant.composite

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<sca:composite xmlns:sca="http://www.osoa.org/xmlns/sca/1.0" name="Restaurant">
  <sca:component name="RestaurantServiceComponent">
    <sca:implementation.java class="restaurant.lib.RestaurantServiceImpl"/>
    <sca:service name="RestaurantService">
      <sca:interface.java interface="restaurant.api.RestaurantService"/>
    </sca:service>
    <sca:reference name="billService"/>
    <sca:reference name="menuService"/>
  </sca:component>
  <sca:service name="RestaurantService" promote="RestaurantServiceComponent/RestaurantService"/>
  <sca:component name="MenuServiceComponent">
    <sca:implementation.java class="restaurant.lib.MenuServiceImpl"/>
    <sca:service name="MenuService">
      <sca:interface.java interface="restaurant.api.MenuService"/>
    </sca:service>
  </sca:component>
  <sca:component name="BillServiceComponent">
    <sca:implementation.java class="restaurant.lib.BillServiceImpl"/>
    <sca:service name="BillService">
      <sca:interface.java interface="restaurant.api.BillService"/>
    </sca:service>
    <sca:reference name="vatService"/>
    <sca:reference name="tipService"/>
  </sca:component>
  <sca:component name="TipServiceComponent">
    <sca:implementation.java class="restaurant.lib.TipServiceImpl"/>
    <sca:service name="TipService">
      <sca:interface.java interface="restaurant.api.TipService"/>
    </sca:service>
  </sca:component>
  <sca:component name="VatServiceComponent">
    <sca:implementation.java class="restaurant.lib.VatServiceImpl"/>
    <sca:service name="VatService">
      <sca:interface.java interface="restaurant.api.VatService"/>
    </sca:service>
  </sca:component>
  <sca:wire source="BillServiceComponent/vatService" target="VatServiceComponent/VatService"/>
  <sca:wire source="BillServiceComponent/tipService" target="TipServiceComponent/TipService"/>
  <sca:wire source="RestaurantServiceComponent/billService" target="BillServiceComponent/BillService"/>
  <sca:wire source="RestaurantServiceComponent/menuService" target="MenuServiceComponent/MenuService"/>
</sca:composite>
```

Do a right click on your diagram (on the Canvas part = white part of your diagram) open a contextual menu, this allows:

- to validate your diagram and
- to hide/show implementation, interface and binding icons.



You have finished developping your first SCA application with the SCA Composite Designer. It is time to test.

Test the Restaurant

Create a new package, named test, in the src folder of your project. Create a new **Java class** named Client.

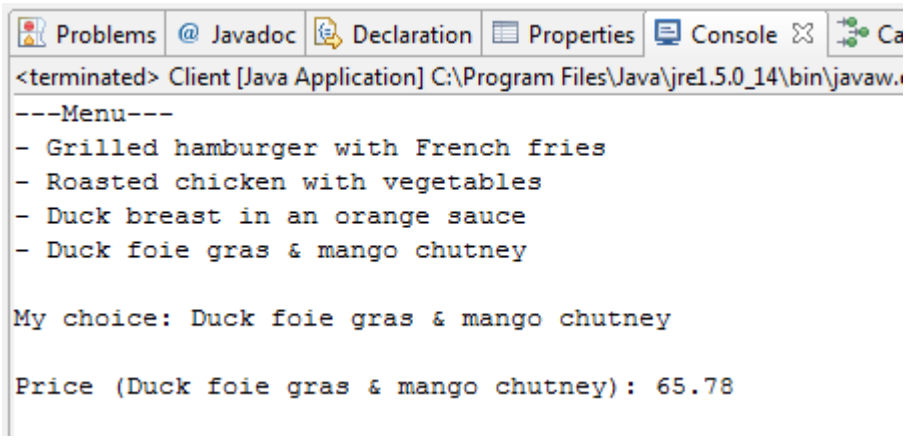
```
package test;

import org.apache.tuscany.sca.host.embedded.SCADomain;

import restaurant.api.Menu;
import restaurant.api.RestaurantService;

public class Client {

      public static void main(String[] args) throws Exception {
            SCADomain scaDomain = SCADomain.newInstance("Restaurant.composite");
            RestaurantService restaurantService = scaDomain.getService(
                        RestaurantService.class, "RestaurantServiceComponent");

            Menu[] menus = restaurantService.getMenus();
            System.out.println("--- Menu ---");
            for (Menu m : menus) {
                  System.out.println("- " + m.printMenu());
            }
            System.out.println();
```

```
            Menu menu = menus[3];
            System.out.println("My choice: " + menu.printMenu());
            System.out.println();

            double price = restaurantService.getBill(menu);
            System.out.println("Price (" + menu.printMenu() + "): " + price);

            scaDomain.close();
      }

}
```

Launch the client:

- Do a right click on the Client class
- Select **Run as > Java Application**

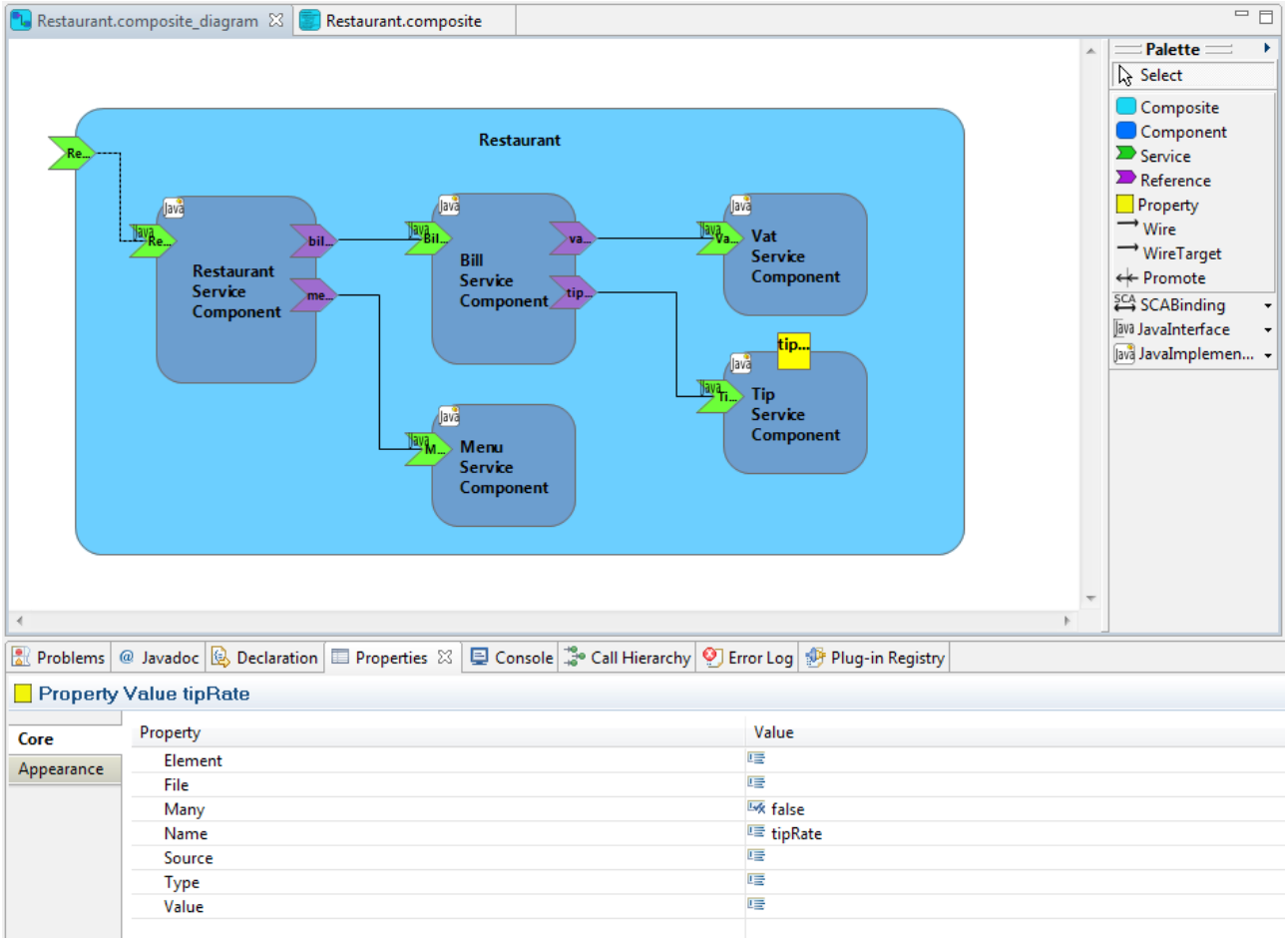
In the Console view you should see:



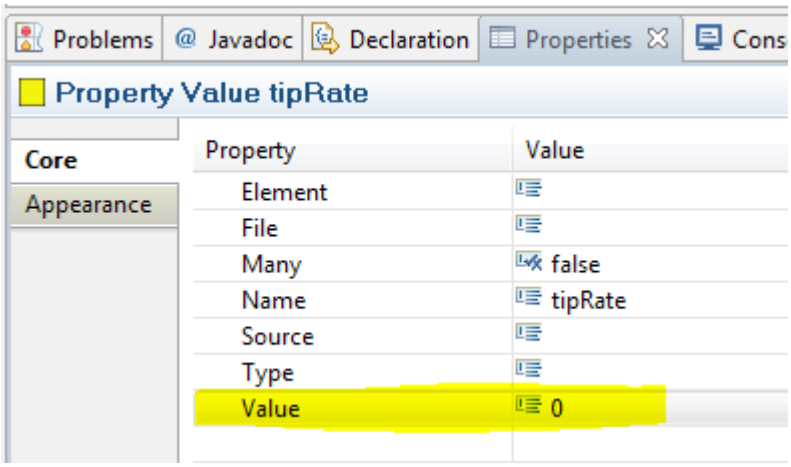
Refine a property

In some countries there is no tip. SCA allows to simply refine an implementation property in the configuration file.

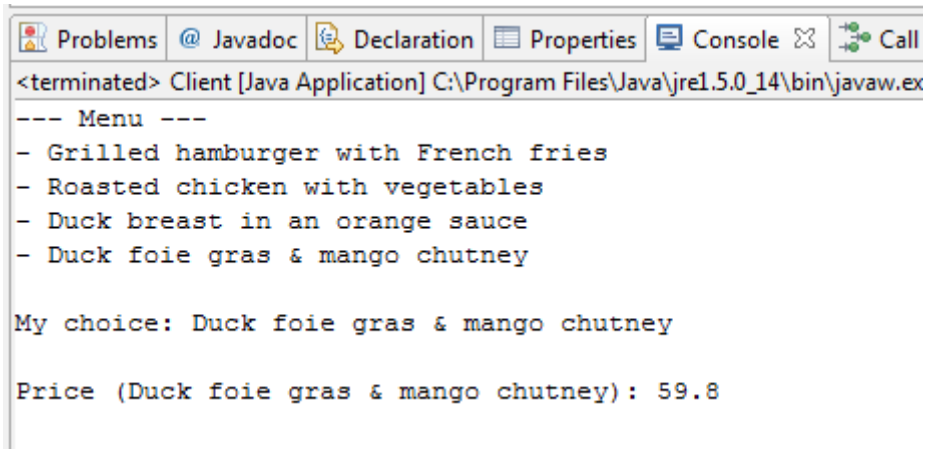
Add a new property to the TipServiceComponent named tipRate (use the Property creation tool from the Palette or the contextual menu on the TipServiceComponent).



In the **Properties** view set the value to 0.



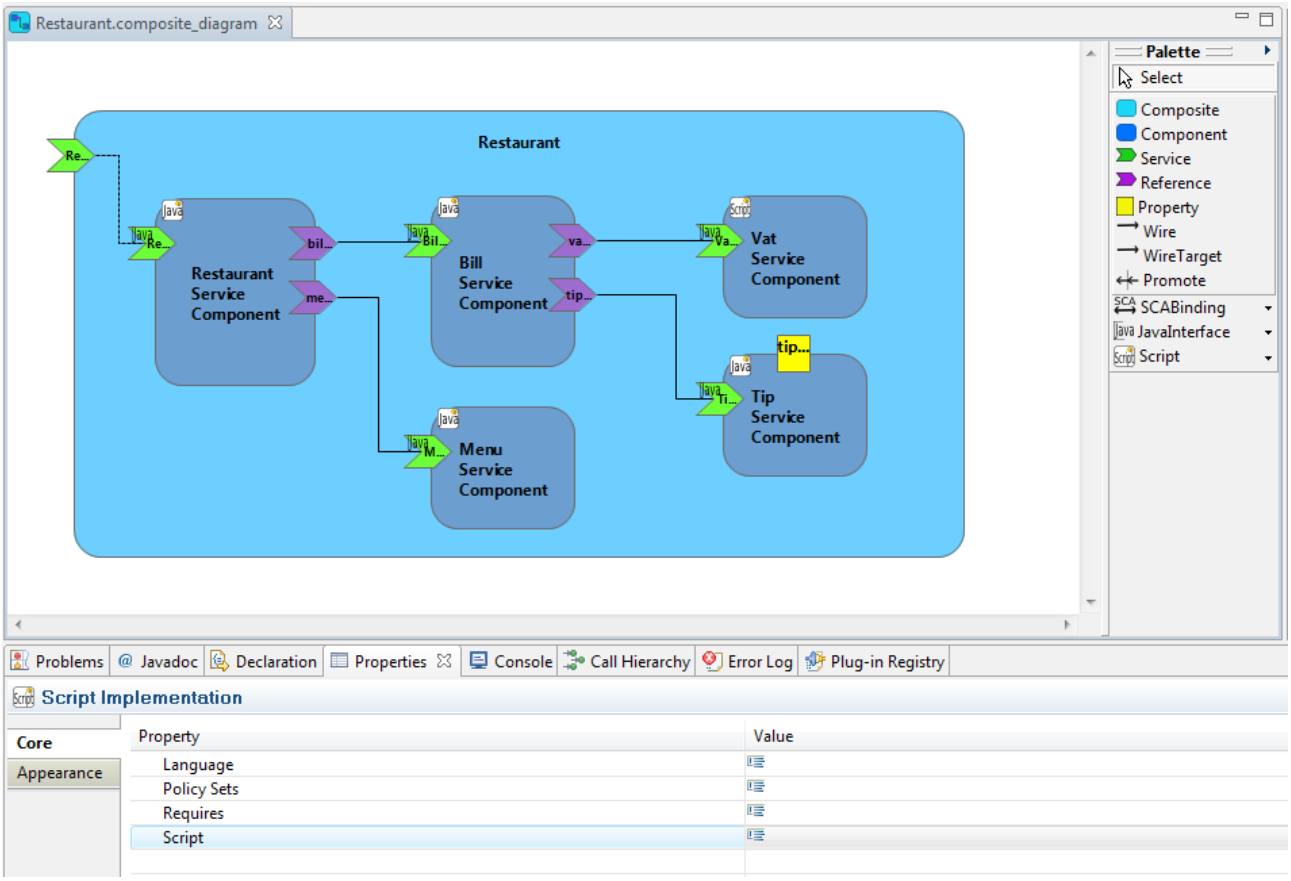
Launch the Client. Now you should see:



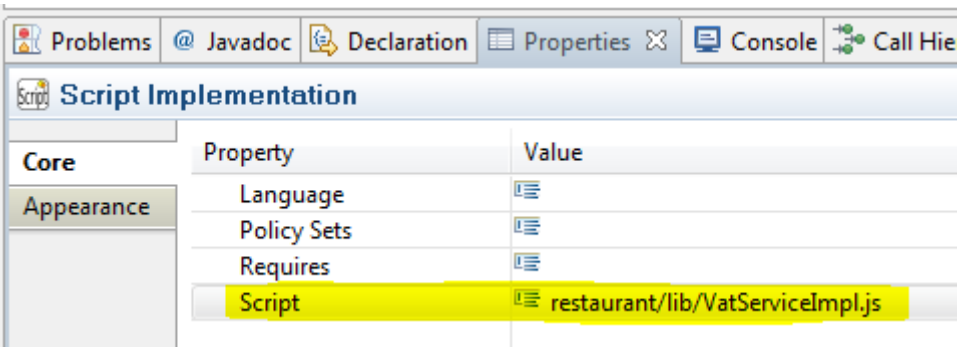
Change implementations and bindings

Use JavaScript for your implementation

Remove the java implementation of the component VatServiceComponent. Add, as new component implementation, a script implementation (use ScriptImplementation creation tool from Palette or contextual menu on VatServiceComponent).



Set (in the **Properties** view) as **Script** with the value restaurant/lib/VatServiceImpl.js.

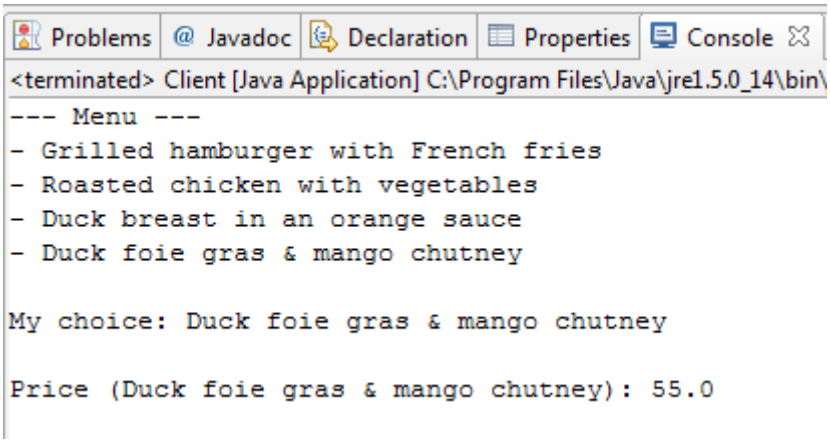


In directory src/restaurant/lib add a **new File** named VatServiceImpl.js. Open the new created file. Set the following code as implementation:

```
var vatRate=10

function getPriceWithVat(price) {
```

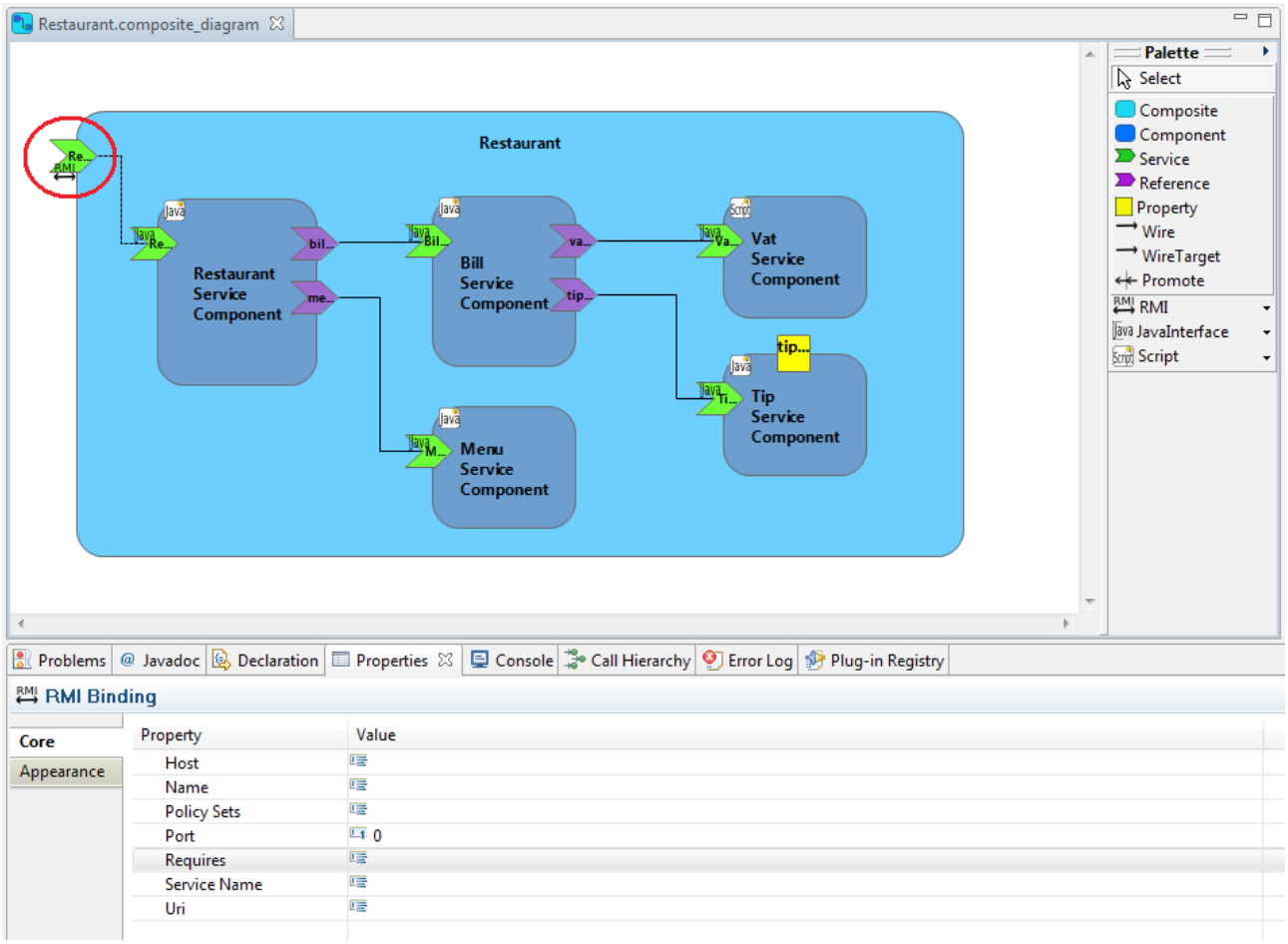
```
            return price *  vatRate/100 + price;
}
```

Launch the Client. You should see:



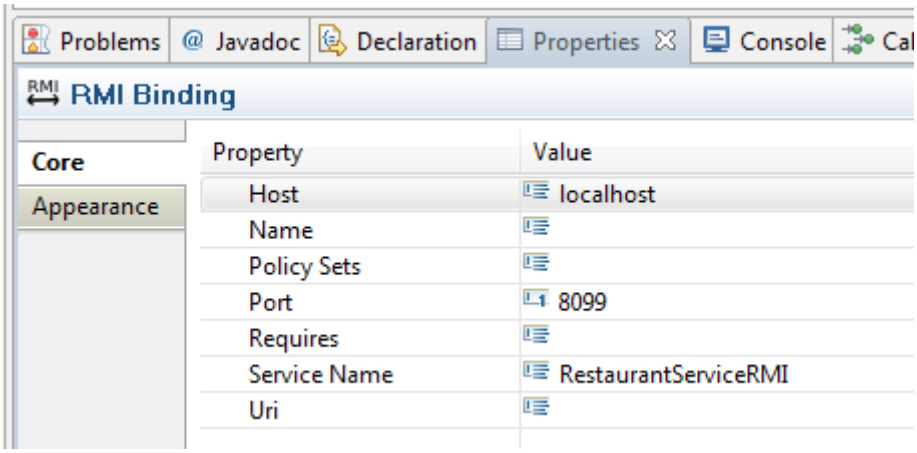
Use an RMI Binding

Add a **RMIBinding** on the composite service named RestaurantService.



Set (in the **Properties** view) as

- **ServiceName**: RestaurantServiceRMI,
- **Port**: 8099 and
- **Host**: localhost.



In the package test add a **new Java Class** named RestaurantServiceServer.

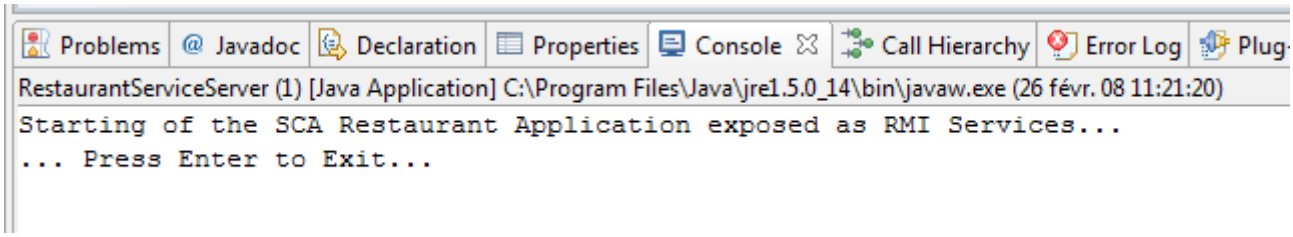
```
package test;

import org.apache.tuscany.sca.host.embedded.SCADomain;

/**
 * A restaurant service server. Starts up the SCA runtime which will start
 * listening for RMI service requests.
 */
public class RestaurantServiceServer {

	public static void main(String[] args) throws Exception {
		System.out
				.println("Starting of the SCA Restaurant Application
exposed as RMI Services...");
		SCADomain scaDomain = SCADomain
				.newInstance("Restaurant.composite");
		System.out.println("... Press Enter to Exit...");
		System.in.read();
		scaDomain.close();
		System.out.println("Exited...");
		System.exit(0);
	}

}
```

Create a **new Java Class** named Client2 in the package test.

```
package test;

import java.rmi.Naming;

import restaurant.api.Menu;
import restaurant.api.RestaurantService;

public class Client2 {

	public static void main(String[] args) throws Exception {
```

```
            RestaurantService restaurantService = (RestaurantService) Naming
                        .lookup("//localhost:8099/RestaurantServiceRMI");

            Menu[] menus = restaurantService.getMenus();
            System.out.println("--- Menu ---");
            for (Menu m : menus) {
                  System.out.println("- " + m.printMenu());
            }
            System.out.println();

            Menu menu = menus[3];
            System.out.println("My choice: " + menu.printMenu());
            System.out.println();

            double price = restaurantService.getBill(menu);
            System.out.println("Price (" + menu.printMenu() + "): " + price);
      }

}
```

Launch the RestaurantServiceServer application.



Launch Client2. You should see: