

## Automotive Case Study On Road Assistance Demonstrator

### Tutorial

Authors: Rong Xie (CIR), Nora Koch (CIR,LMU)

Period covered: from 01.09.2008 to 28.02.2010

Date of preparation: 15.2.2010

Revision: Draft

Dissemination level: PU

Contract start date: September 1, 2005 Duration: 48 months

Project coordinator: Martin Wirsing (LMU)

Partners: LMU, UNITN, ULEICES, UWARSAW, DTU, PISA, DSIUF,  
UNIBO, ISTI, FFCUL, UEDIN, ATX, TILab, FAST, BUTE,  
S&N, LSS-Imperial, LSS-UCL, MIP, ATXT, CIR

Integrated Project funded by the  
European Community under the  
"Information Society Technologies"  
Programme (2002—2006)

## **Executive Summary**

This report provides a detailed documentation on the Automotive Demonstrator implemented in the SENSORIA project in order to demonstrate the use of a set of techniques, methods, and tools developed within the scope of the project. The techniques used are UML4SOA, various model transformations from UML to BPEL and WSDL, the CASE tool MagicDraw, and the SENSORIA Development Environment (SDE). In particular, the Demonstrator focuses on the development process of service-oriented software. It demonstrates how a model-driven process can work, which automatically generates and deploys a service based on a model of the service. This service is defined as an orchestration of web services.

## Contents

<b>1</b>	<b>Introduction .....</b>	<b>7</b>
<b>2</b>	<b>System Requirements of the Demonstrator .....</b>	<b>9</b>
2.1	Software .....	9
2.2	JDK1.6 (for SDE) .....	9
2.2.1	JDK1.5 (for Tomcat5) .....	9
2.2.2	Eclipse3.4 .....	9
2.2.3	Eclipse Plug-ins .....	11
2.2.4	Tomcat 5.5.....	14
2.2.5	ActiveBPEL Engine (ActiveBPEL4.1 Final Release).....	14
2.2.6	Dino 0.2.2.....	14
2.2.7	MySQL5.1 .....	14
2.2.8	MagicDraw .....	14
2.3	Hardware.....	15
2.4	Standards.....	15
<b>3</b>	<b>Architecture of the Automotive Demonstrator .....</b>	<b>16</b>
3.1	Overview.....	16
3.2	Service Orchestration.....	18
3.3	View Manager.....	19
3.4	Web Services .....	19
3.4.1	Position Service .....	20
3.4.2	Bank Service (Bank Card Charge) .....	20
3.4.3	Garage Service.....	21
3.4.4	Rental car service .....	21
3.5	Semantic Dynamic Service Discovery (Dino).....	22
3.5.1	Integrating Dino in the Automotive Demonstrator.....	22
3.5.2	Registration of Ontology Description in Dino.....	23
3.5.3	Registration of Capability Documents in Dino .....	23
3.5.4	Requirement Document of Request to Dino.....	24
3.6	Map Display and Route Plan .....	24
<b>4</b>	<b>MDD4SOA Plug-ins for Generation of Executable BPEL.....</b>	<b>26</b>
4.1	Integration of MDD4SOA_Extension in SDE.....	26
4.2	Functions of the MDD4SOA Extension .....	27
4.2.1	Transformation to Executable BPEL.....	27
4.2.2	Transformation to Interactive BPEL .....	28
4.2.3	Deployment to a Web Server.....	29
4.3	Configuration of the Converter.....	29
<b>5</b>	<b>Demonstration Process.....</b>	<b>30</b>
5.1	Model-Driven Development .....	30
5.2	Building UML4SOA Orchestration Model.....	31
5.3	Transformation Chain .....	32
5.3.1	Orchestration of Eclipse Plug-in Functions with SDE Graphical Orchestration .....	33
5.3.2	Result of the transformation .....	36
5.4	Run Deployed Application .....	39
5.5	Running the Demonstrator after Modifying the UML4SOA Model .....	43
<b>6</b>	<b>Lessons Learned and Future Work .....</b>	<b>50</b>
<b>7</b>	<b>References .....</b>	<b>51</b>
7.1	Documents .....	51
7.2	Tools .....	51

<b>Appendix</b> .....	<b>53</b>
A.1 Configuration of MDD4SOA_Extension plug-in projects .....	53
A.1.1 Configuration of Plug-in Project .....	53
A.1.2 Configuration of feature project .....	55
A.1.3 Configuration of Update Site Project .....	56
A.2 Deploy a Web Service with Eclipse Web Developer Tools .....	58

## Index of Figures

Figure 1: Screenshot of properties of eclipse icon in windows desktop .....	10
Figure 2: Installed JRE .....	10
Figure 3: Configure VM Arguments .....	11
Figure 4: Update plug-ins.....	12
Figure 5: Select to installed plug-ins of Ganymede .....	12
Figure 6: Select plug-ins to be installed for SDE and MDD4SOA.....	13
Figure 7: Architecture of the Demonstrator .....	17
Figure 8: Sequence diagram of a standard process in the Demonstrator .....	18
Figure 9: Sequence BPEL process for the service orchestration.....	19
Figure 10: Dino integration .....	23
Figure 11: Map display.....	24
Figure 12: Route in map .....	25
Figure 13: BPEL2ActiveBPEL/WSDL converter and deployment.....	26
Figure 14: Directory structure of a BPEL deployment archive.....	29
Figure 15: Model-driven development process.....	30
Figure 16: Activity diagram of the orchestration .....	32
Figure 17: Sequence Activity diagram of the orchestration.....	32
Figure 18: Creating a graphical orchestration (.god file) .....	33
Figure 19: Chain tool script demochain.god .....	34
Figure 20: Set arguments of functions .....	36
Figure 21: In ActiveBPEL engine executable BPEL process of scenario of Figure 16.....	37
Figure 22: Interactive BPEL process of scenario of Figure 16 .....	38
Figure 23: Start page of the demonstration .....	39
Figure 24: Car location.....	40
Figure 25: Find Garage.....	41
Figure 26: Find best garage .....	41
Figure 27: Find rental car station .....	42
Figure 28: Best rental car station.....	43
Figure 29: Payment with credit card .....	43
Figure 30: Activity diagram of the orchestration for parallel scenario .....	44
Figure 31: BPEL process of scenario of Figure 30 .....	45
Figure 32: Interactive BPEL process of scenario of Figure 30 .....	46
Figure 33: Start page .....	47
Figure 34: Payment with credit card .....	47
Figure 35: Result of payment .....	48
Figure 36: Location service.....	48
Figure 37: Find service providers.....	49
Figure 38: Find best provider .....	49
Figure 39: Configure dependences .....	53
Figure 40: Specify the libraries and folders that constitute the plug-in classpath.....	54
Figure 41: Select folders for build.....	54
Figure 42: Testing and debug .....	55
Figure 43: Feature project .....	56
Figure 44: Create update site project.....	56
Figure 45: Create update site project with web resources.....	57
Figure 46: Update Site Map .....	57
Figure 47: Create web service .....	58

Figure 48: Publish web service ..... 59  
Figure 49: Select methods of web service..... 59  
Figure 50: Server location ..... 60

## 1 Introduction

This report provides a detailed documentation on the Automotive Demonstrator implemented in the SENSORIA project in order to demonstrate the use of a set of techniques, methods, and tools developed within the scope of the project.

The main goal of the Automotive Demonstrator is to show the power of the SENSORIA approach based on the application of model-driven architecture (MDA) principles in the area of service-oriented computing. Implementation following the model-driven approach is based on the construction of models and model transformations. In particular, it demonstrates how a model-driven development process can work, which automatically generates and deploys a service based on a model of the composition of services. This service is defined as an orchestration of web services.

The Automotive Demonstrator implements the *On Road Assistance* scenario [D1.4a, Koch07, BK07]. In this scenario, the diagnostic system reports a severe failure in the car engine; for example, the vehicle's oil lamp reports a low oil level. This triggers the diagnostic system of the vehicle to perform an analysis of the sensor values. The diagnostic system reports, for example, a problem with the pressure in one cylinder head, and therefore the car is no longer drivable. It sends a message with the diagnostic data as well as the vehicle's GPS data to the car manufacturer or service centre.

Based on the car position, the service discovery system identifies and selects the appropriate services in the area: repair shop (garage), tow truck and car rental. When the driver makes an appointment with the garage; the diagnostic data is automatically transferred to the garage, which could then be able to identify the spare parts needed to perform the repair. The service discovery system identifies as well a car rental service, providing the GPS data of the stranded vehicle. The driver makes an appointment with the car rental service to pick up the rental car. The current version of the Demonstrator is limited to the orchestration of garage and rental car services, but could easily be extended to include also the tow truck service. We assume that the owner of the car has to deposit a security payment in order to be able to use the services.

The services involved then in the implementation of the *On Road Assistance* are the following:

- *Position Service* providing the GPS data of the stranded vehicle
- *Bank Service* for charging a credit card
- *Garage Services* for the localization and selection of garages
- *Rental Car Services* for the localization and selection of car rental stations.

For the demonstration, a first UML model is built as the sequential orchestration of the required services for determining the car position, finding garages in nearby the car and selecting the most convenient garage first, finding rental car stations nearby and selecting one afterwards. The orchestration process finalizes with the credit card charge service. Using a chain of model transformations the model is transformed to an executable service implemented in WSDL and BPEL and deployed to a web server. Model transformations and deployment are performed in a fully automatic way.

The Automotive Demonstrator is designed in such a way that the invocation of each service is visualized in the web browser and expects a user interaction, almost all just a click on a continue button. In fact, the position of the car, asset of garages and car rentals nearby the car position, and then the selected garage and car rental station are visualized in Google maps API. For the implementation of the interactions and the visualization dynamic generated web pages are associated to each service and the BPEL process is enriched with additional interactive features by a complementary model transformation.

The power of the model-driven development approach is shown by a second run of the Demonstrator that consists of changing the orchestration model. The changes are twofold: (1) the credit charge service is invoked at the beginning of the process, and (2) the localization of garages and rental car stations as well as the selection of the most appropriate garage and rental car station are parallelized. Again, the model transformations and deployment are performed automatically with the tool chain defined for this purpose.

The models are built using the UML4SOA notation, which is an UML extension that has been defined to model service-oriented systems. The model transformations convert the UML4SOA models to BPEL and WSDL in

several steps. The model transformation from UML to BPEL and WSDL defined so far within the scope of the SENSORIA project were not sufficient for supporting the model-driven process, i.e. the automatic generation based on the UML4SOA models built of the application. In addition, deployment of the application only could be performed manually.

To overcome these limitations additional model-to-model and model-to-code transformations, and a function supporting automatic deployment of a web application onto a web application server were implemented. The first additional model transformation is needed to provide BPEL and WSDL code that is executable by a BPEL engine (in our case ActiveBPEL). The second model transformation is needed to allow user interactions and to visualize results step by step during the demonstration. The third one is to deploy the resulting web application.

Note that the model transformations are independent of the scenario, even more they are independent of the BPEL/WSDL application, i.e. they are totally generic and reusable for other services modeled as orchestration of other services.

The CASE tool MagicDraw was used for the modelling activities and the demonstration process was implemented using the SENSORIA Development Environment (SDE), which is an Eclipse-based framework for the integration and use of the tools developed in the project for the analysis and development of service-oriented software. The resulting web application is run under a Tomcat v5 server.

The remainder of the document is organized as a reference guide for the installation of the software required for the Automotive Demonstrator and the information needed for a successful run of the demonstration. Section 2 describes the system requirements of the Automotive Demonstrator. Section 3 presents the architecture of the Demonstrator. Section 4 presents the newly defined model transformations. Section 5 describes the demonstration process and the results of running the demonstration twice. First run is based on the sequential orchestration of services and the second run on the modified model changing order of services and introducing parallelism. Finally, section 6 describes lessons learned and sketches some future work.



## 2 System Requirements of the Demonstrator

The following section describes the system requirement and development environment of the Demonstrator. The section covers the software- and the hardware configuration as well as the standards used in the implementation.

### 2.1 Software

A set of software tools must be installed to develop and run the Demonstrator. At first, a UML CASE tool like MagicDraw or IBM Rational Software Modeler (RSM) is needed to modeling the UML model. After modeling the Eclipse Development Environment v3.4 and a set of Eclipse plug-ins (SDE, MDD4SOA) must be installed for running the model transformations and the implementation of the web services. The application data of web services should be stored in a database (e.g. MySQL v5). The web services and service orchestration (BPEL process) must be deployed in an application server like Tomcat v5.5. A BPEL engine (e.g. ActiveBPEL engine) must be installed to run the BPEL process. Java Development Kit (JDK) v1.6 is necessary to use all feature of the SDE. Further more DINO [DINO07a, Dino07b] is required to realization of the semantic dynamic service registry and discovery. More details on required software components and tools that the Demonstrator needs are described in the following subsections.

### 2.2 JDK1.6 (for SDE)

The Automotive Demonstrator is written mainly in Java, for which implementation the SUN's JDK 1.6 was used. The SDE is built on cutting-edge technology. It requires Java JDK 1.6 to be installed. Former versions, including 1.5, will NOT work as the SENSORIA Development Environment (SDE) uses the Java 6 Scripting Engine. Note that you will need the JDK. Only the JRE isn't sufficient for running the all feature of SDE.

Download: <http://java.sun.com/javase/downloads/index.jsp>

#### 2.2.1 JDK1.5 (for Tomcat5)

JDK1.5 must also be installed, because the ActiveBPEL Engine 4.1 runs only with Apache Tomcat5. Tomcat5 requires the version 1.5 of JDK.

Download: [http://java.sun.com/javase/downloads/index\\_jdk5.jsp](http://java.sun.com/javase/downloads/index_jdk5.jsp)

#### 2.2.2 Eclipse3.4

SENSORIA SDE needs the new features of Eclipse3.4. Therefore, the Eclipse3.4 must be installed as IDE. It is recommended to explicitly use the Java VM when running Eclipse. This is achieved with the -vm command line argument (for example: -vm "D:\program files\Java\jdk1.6.0\_06\bin\javaw.exe"). Without -vm, Eclipse uses the first Java VM found on the O/S path.

For example for Windows, Figure 1 indicates how to configure Java VM explicitly. Right mouse click at eclipse icon in windows desktop → properties → link → set target with (D:\EntwicklungsProgramm\eclipse\_je\_ ganymede\_SR13.4.1\eclipse\eclipse.exe -vm "D:\program files\Java\jdk1.6.0\_06\bin\javaw.exe" -Xmx1024M)

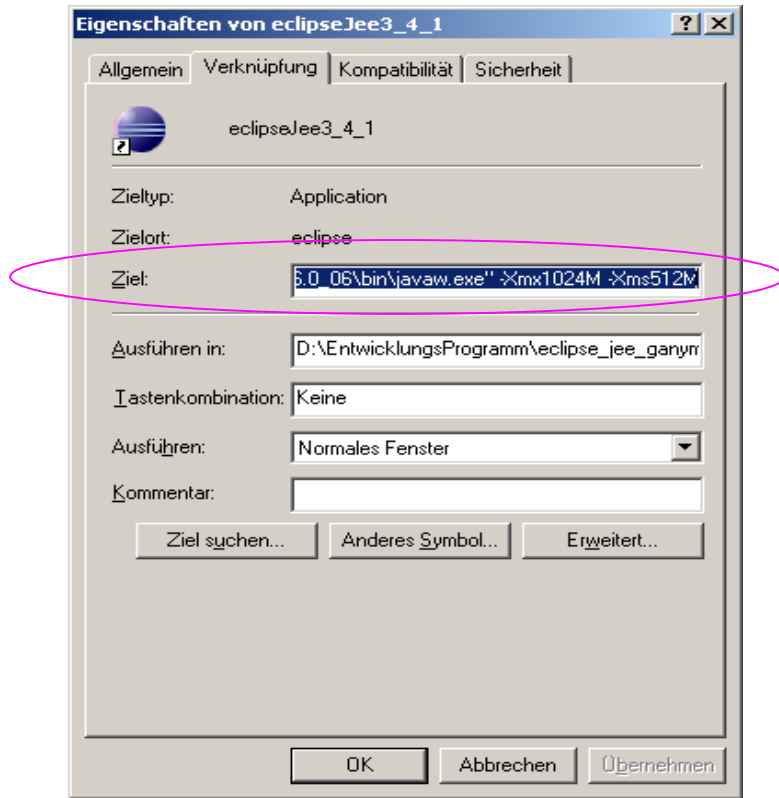


Figure 1: Screenshot of properties of eclipse icon in windows desktop

Also, within Eclipse, the right JDK must be selected in the preferences before starting a Runtime Workbench. The Java JDK 1.6 must be selected as default JRE in Eclipse (Window > Preferences > Java > Installed JREs...) as shown in Figure 2.

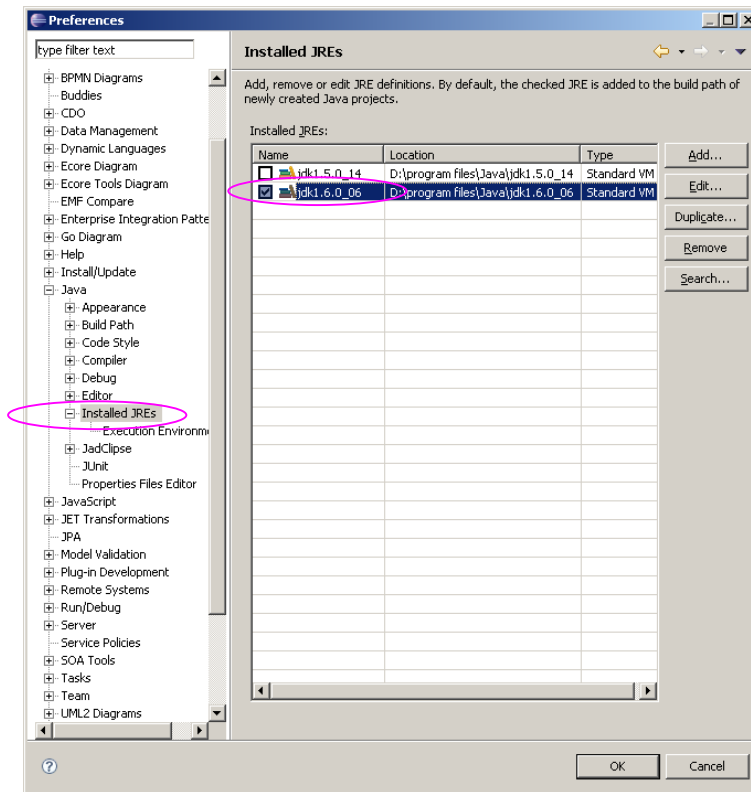


Figure 2: Installed JRE

In addition, set VM Memory for Runtime Workbench as follows: Click “Edit”, configure VM Arguments “-Xmx1024m” in Figure 3.

Download: <http://www.eclipse.org/downloads/>

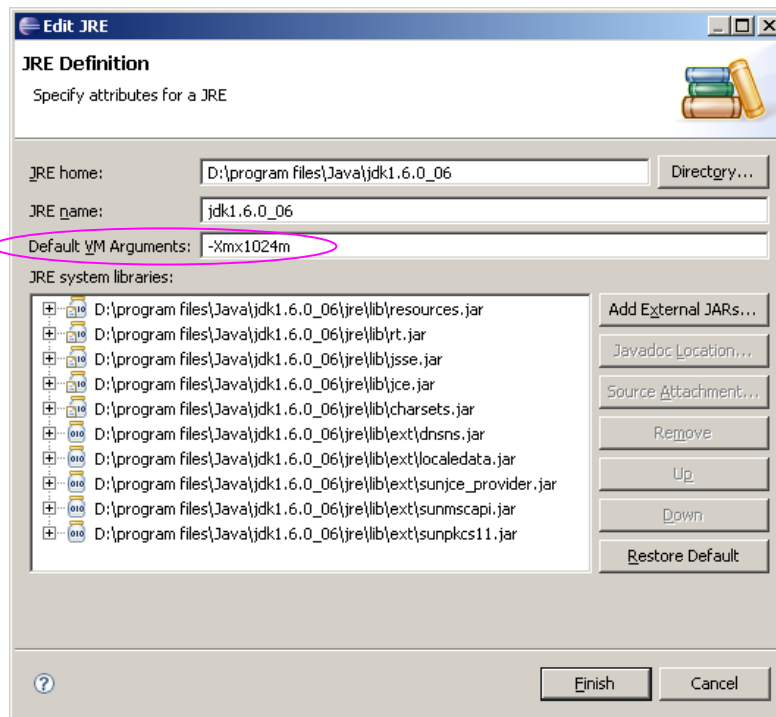


Figure 3: Configure VM Arguments

## 2.2.3 Eclipse Plug-ins

The Eclipse plug-ins (e.g. Graphical Editors and Frameworks, Models and Model Development, SOA Development, Web and Java EE Development, SDE, MDD4SOA and MDD4SOA\_Extension) are necessary for running the Automotive Demonstrator.

### 2.2.3.1 Installation of Plug-ins using Ganymede Update Site

Begin the installation of the plug-ins by selecting the menu path as shown in Figure 4.

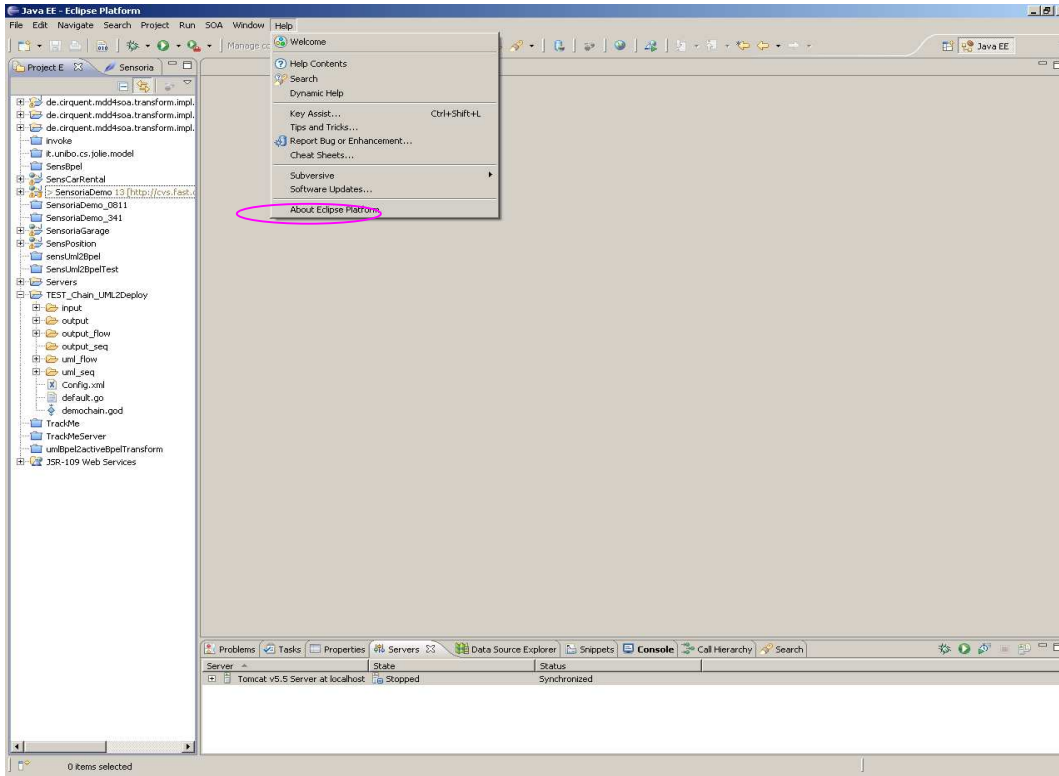


Figure 4: Update plug-ins

Select the “Graphical Editors and Frameworks”, “Models and Model Development”, “SOA Development”, “Web and Java EE Development” of Ganymede Update Site and install them as illustrated in Figure 5.

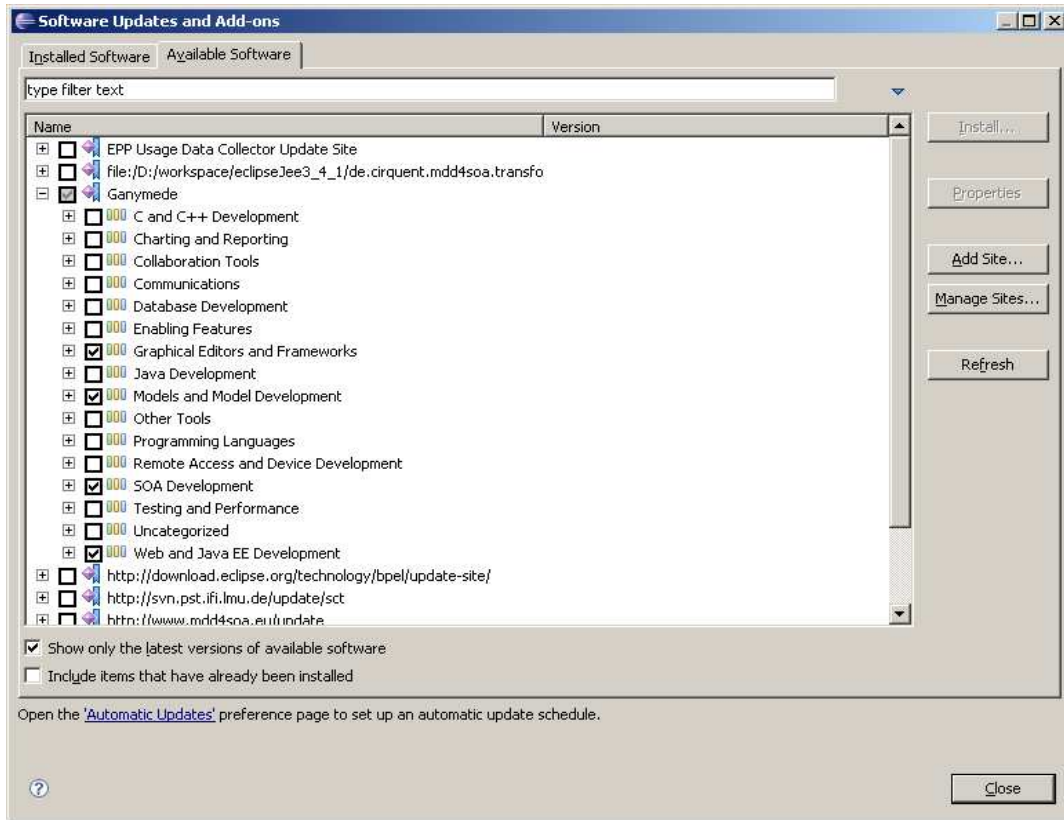


Figure 5: Select to installed plug-ins of Ganymede

### 2.2.3.2 Installation of the SENSORIA Development Environment (SDE)

The SENSORIA Development Environment (SDE) has to be installed as Eclipse plug-in through the Eclipse update site of SDE. Further information about SDE can be found at the SDE website.

SDE website: <http://svn.pst.ifi.lmu.de/trac/sct>

Eclipse update site of SDE: <http://svn.pst.ifi.lmu.de/update/sct>

### 2.2.3.3 MDD4SOA

MDD4SOA is a model-driven development approach for SOA applications. It is based on UML4SOA models and on model transformations. UML4SOA is a UML profile for modeling service-oriented software; in particular, for modeling orchestrations of services inside a SOA-based system [MSKa, MSKb]. The model transformations defined before the Automotive Demonstrator was built, were not sufficient for supporting the model-driven process, i.e. the automatic generation based on the UML4SOA models built of the application. We implemented additional model-to-model and model-to-code transformations to overcome these limitations. Note that these implemented model transformations are independent of the scenario, even more they are independent of the BPEL/WSDL application, i.e. they are totally generic and reusable for other services modeled as orchestration of other services.

MDD4SOA must be installed as an Eclipse plug-in through the Eclipse update site of MDD4SOA.

MDD4SOA website: <http://www.mdd4soa.eu/web/>

Eclipse Update Site of MDD4SOA: <http://www.mdd4soa.eu/update>

### 2.2.3.4 Extending MDD4SOA

MDD4SOA\_Extension is an extension of the MDD4SOA plug-in. It includes two additional model transformations. The objective of these model transformations is to convert BPEL/WSDL files to ActiveBPEL Engine compatible BPEL/WSDL files. In addition it includes a method to deploy BPEL process to web server. This extension could be installed as an Eclipse plug-in through the Eclipse update site of MDD4SOA\_Extension. Figure 6 shows the selection of plug-ins to be installed for SDE and MDD4SOA.

Eclipse updates site of MDD4SOA\_Extension: [http://www.sensoria-ist.eu/cirquent/eclipse\\_Plug-in\\_update](http://www.sensoria-ist.eu/cirquent/eclipse_Plug-in_update)

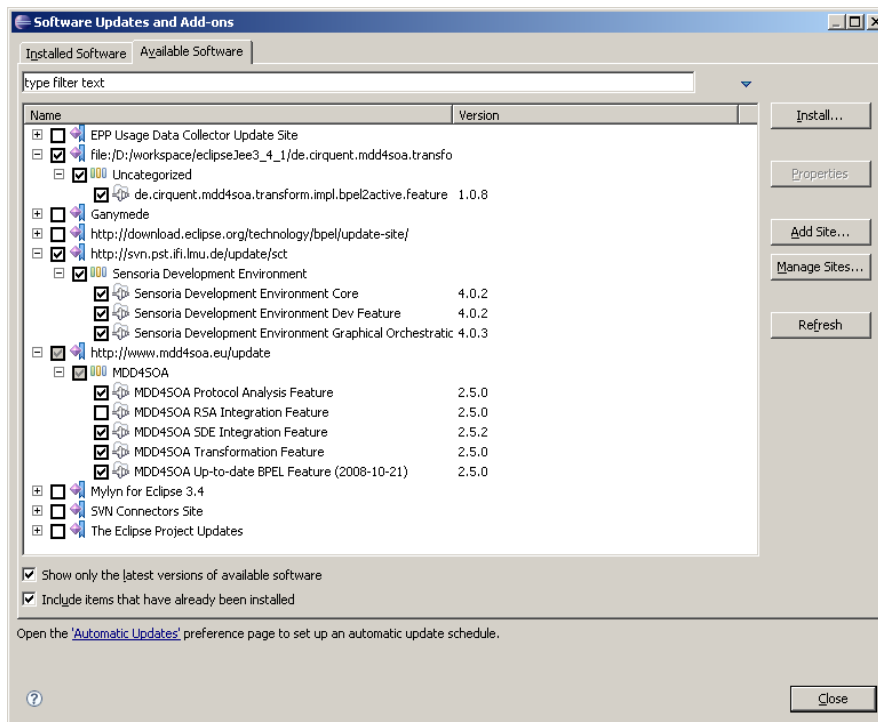


Figure 6: Select plug-ins to be installed for SDE and MDD4SOA

### 2.2.4 Tomcat 5.5

An Apache Tomcat (version 5.5) is used as application server. The different web services and BPEL process must be deployed at the application server. The Apache Tomcat 5 or higher version is required by ActiveBPEL Engine 4.1.

Download: <http://tomcat.apache.org/download-55.cgi>

### 2.2.5 ActiveBPEL Engine (ActiveBPEL4.1 Final Release)

For the orchestration of the services we use the Business Process Execution Language (BPEL). The XML-based BPEL files are executed in the open source execution engine ActiveBPEL, version 4.1. This engine runs web applications in the webapps directory of an Apache Tomcat application server. (We use Tomcat 5.5, because this version has been tested for ActiveBPEL 4.1). With the BPEL engine, the different services could be invoked.

To create and to deploy own web services, the tools of Apache Axis 2, version 1.3, is used. Axis is embedded in the Apache Tomcat similarly to ActiveBPEL. Axis provides tools for creating and deploying new web services or for transforming existing applications into web services.

Requirements of ActiveBPEL engine:

- Apache Tomcat 5.5
- Sun's JDK 1.5

The installation itself is described in the documentation section of the web site (<http://www.activevos.com>). After running the install script the ActiveBPEL container is integrated in the Tomcat server. The BPEL engine would always be started automatically when starting the Tomcat.

Download ActiveBPEL 4.1: <http://www.activevos.com/community-open-source-engine-download.php>

Tutorial for Installing ActiveBPEL:

[http://users.encs.concordia.ca/~yuhong/teaching/UNB/CS6905/Task0\\_GettingUpandRunning-LITEVERSION.doc](http://users.encs.concordia.ca/~yuhong/teaching/UNB/CS6905/Task0_GettingUpandRunning-LITEVERSION.doc)

### 2.2.6 Dino 0.2.2

The Dino tool [DINO07a, Dino07b] (version 0.2.2) is used for the semantic dynamic service discovery. The Dino project provides a tools and a runtime system for enabling dynamic and adaptive composition of autonomous services based on the non-functional requirements of services.

Web page of Dino: <http://www.cs.ucl.ac.uk/staff/a.mukhija/dino/>

### 2.2.7 MySQL5.1

MySQL5.1 was used as database to store the test data of the Demonstrator.

Download of MySQL5.1: <http://dev.mysql.com/downloads/mysql/5.1.html>

### 2.2.8 MagicDraw

We use MagicDraw15.0 to model the orchestration of the services that define the application of the Demonstrator. IBM Rational Software Modeler (RSM) or the IBM Rational Software Architect (RSA) can be used as well. The UML Profile (MDD4SOA Profile) was tested with both CASE tools, MagicDraw and RSM/RSA.

Download of Magicdraw: <http://www.magicdraw.com/>

### **2.3 Hardware**

Basically the Automotive Demonstrator will run on any machine where a standard Java distribution is available. Furthermore, to connect to the different services, which are requested in the scenario, it must be possible to establish an Internet connection. Especially it has to be ensured, that Java applications accessing the Internet are not blocked by a firewall or a proxy.

To provide services, which can be invoked by the Demonstrator, different application servers, running either on one or more physical machines, must be installed. As application servers we will use the Apache Tomcat running on any Tomcat compatible operating system (e.g. Windows, Linux, Mac).

### **2.4 Standards**

Especially in the area of service-oriented architectures it is the aim to use standard methods and tools whenever they are suitable. The Demonstrator mainly relies on web service techniques, such as WSDL, BPEL, SOAP/XML-RPC and REST. In addition, we use the UML standard for modeling the SENSORIA Automotive demonstration application.

### 3 Architecture of the Automotive Demonstrator

Basically the Demonstrator of the “On Road Assistance” scenario has the issue to show the “car breakdown” scenario implemented with a service-oriented approach. The driver of a car has got a vehicle malfunction or an accident somewhere on the road. Now the user wants to get help and road assistance. Therefore he can invoke several services by using the cars built in communication device. This is exactly what the Demonstrator wants to show different workflows for this scenario.

#### 3.1 Overview

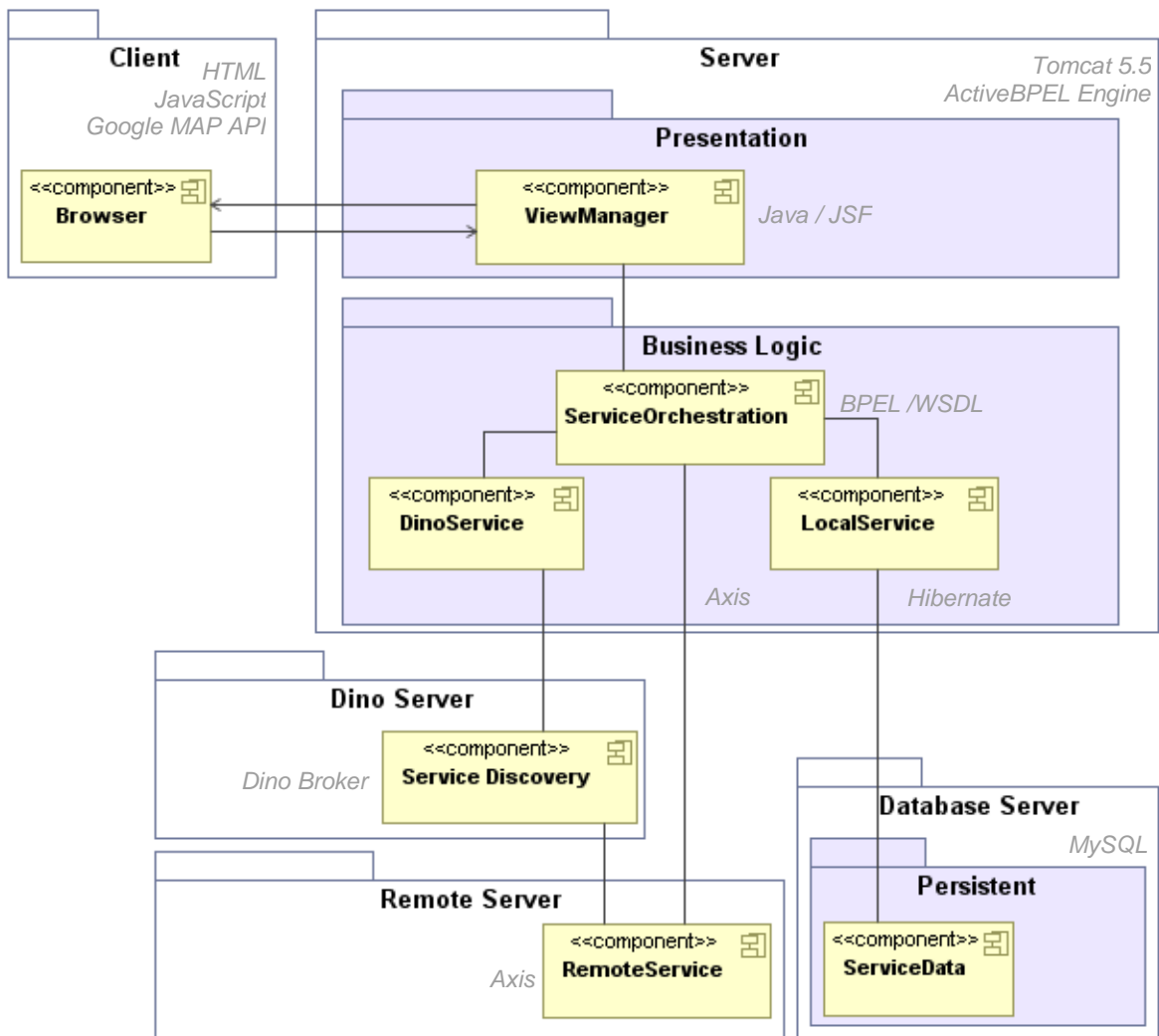
The architecture of the Demonstrator for the automotive scenario is built as typical client/server architecture. On the client side is only a full JavaScript enabled web browser is needed, for example Firefox or Internet Explore. The business logic specified as a service orchestration is deployed on the server side.

The application is defined as a three tier architecture on the server as showed in Figure 7. The first layer is the presentation layer. The “ViewManager” in the presentation layer is a component developed to parser the client request, to call the service orchestration and generate web pages for the client.

The business logic is located in the second layer. A BPEL process lies in this layer, which is in charge of the service orchestration. Several local or remote web services can be called by the BPEL process. A special service is used for the invocation of the Dino broker.

A database lies in the third layer, i.e. the persistent layer. The database will contain all data needed by the services.





**Figure 7: Architecture of the Demonstrator**

The architecture of the Automotive Demonstrator for the “On Road Assistance” scenario consists of two different kinds of components. On the one hand, components, which are located inside the vehicle. On the other hand, components that are arranged somewhere outside the vehicle. They could be spread over the World Wide Web. For the Demonstrator all the web services contained in the same application server as service orchestration are called as local service. All the other web services are called remote services, which are hosted on another remote server.

The standard process in the Demonstrator for requesting a service is shown in the sequence diagram of Figure 8.

The Orchestrator is in charge of controlling the service requesting process that consists of the following steps:

1. The client fills in the form at the web page and presses the button in the web browser.
2. The http-request sends to ViewManager in the web server.
3. The ViewManager parses the request and call the BPEL process (ServiceOrchestration).
4. Local or remote web services are invoked by the BPEL process.
5. Web services get the service data from database and send the result to BPEL process.
6. Then the BPEL process sends the result back to the ViewManager.

7. The ViewManager generates a web page based on the results and sends the web page to client as http-response.
8. After that, the results of service invoke and information of the next task can be shown on the web browser of client side.

If the client clicks the button of next task, the standard process of the Demonstrator will be executed again until reaching the end activity of the BPEL process.

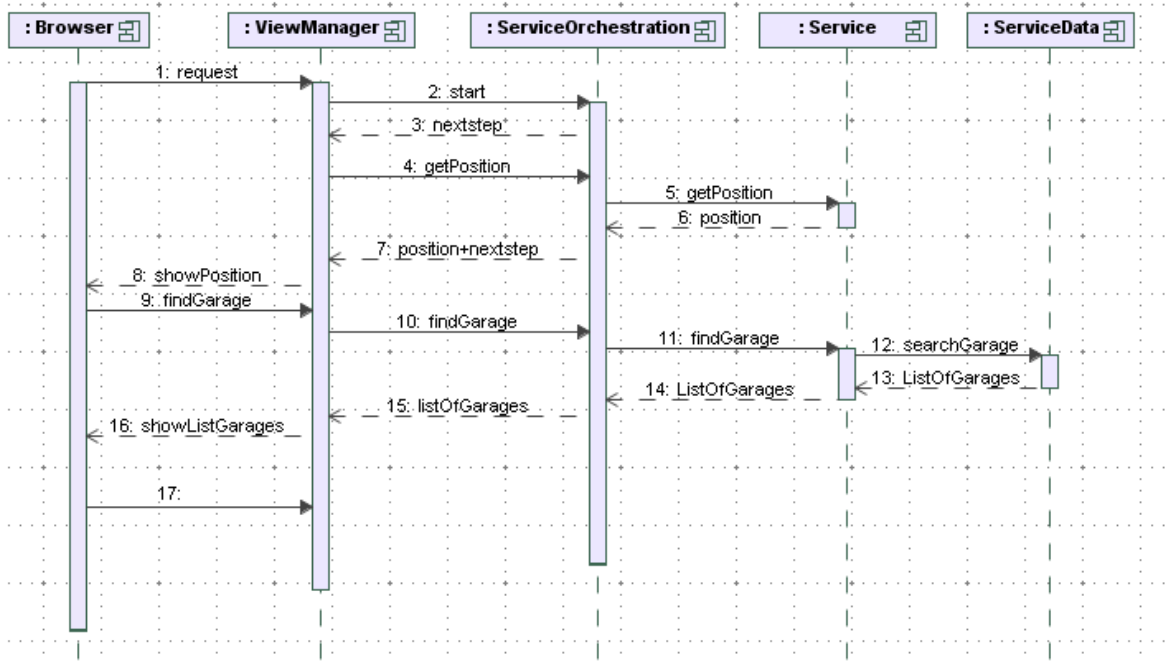


Figure 8: Sequence diagram of a standard process in the Demonstrator

### 3.2 Service Orchestration

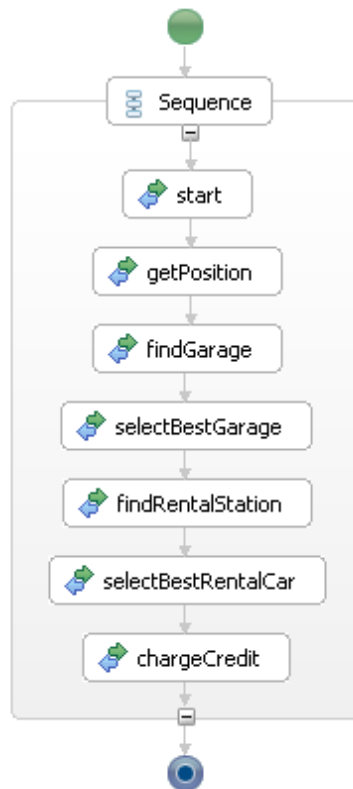
The Orchestrator is the key element of the Demonstrator, as it is the controlling element, which invokes all the services. A very important issue for the Orchestrator is how to build the workflow of the processes. For this the Business Process Execution Language (BPEL) is used. In the BPEL language the workflow for the automotive scenario will be determined. This includes to define, which kind of services are invoked, if different services have to be requested sequentially or parallel and some compensation mechanisms when services are cancelled

At the beginning the BPEL process receives the input from the ViewManager. Depending on this input it forwards the information to the other elements. In the other direction the BPEL process sends the results of the calculations or invoke of other web service back to the ViewManager.

If the orchestration is specified in BPEL process, the BPEL file has to be executed. Therefore a BPEL execution engine is required. For our Demonstrator we focus on open source solutions, which bring us to ActiveBPEL or Apache ODE. Both are open source and can be used for executing BPEL code. For our Demonstrator we will focus on the ActiveBPEL for the following reasons: (1) All in all, this engine seems to be in a further development stage than the ODE. For example ActiveBPEL supports the invocation of REST web services from the BPEL code. This feature is not yet implemented in the ODE engine. (2) We invoke web services, which are provided over the Internet. It is then a useful feature, because many of these services are published in the REST style.

ActiveBPEL engine runs under the most popular application servers. To execute BPEL code, an archive containing the BPEL file itself and its WSDL descriptions must be deployed on the application server. Now the BPEL process can be invoked from other entities just like a web service. Similar to these, the requesters of the BPEL processes are informed how to access the service by using WSDL descriptions.

Figure 9 shows a sequence BPEL process for the first variant of our scenario, which consist of the sequential composition of services.



**Figure 9: Sequence BPEL process for the service orchestration**

### 3.3 View Manager

ViewManager is a server side program to coordinate the BPEL process and user interfaces. It's a bridge between the BPEL process and the user interface.

The user fills-in the form and press the button in the web page. A http-request will be send to the ViewManager of application server. The ViewManager parses the request and prepares the input parameters. Then it calls the BPEL process (orchestration). The BPEL process has the whole business logic. It processes the business plan and decides what should be done in this step and what should be done in the next step.

The BPEL process waits for the next call of the ViewManager at next "receive" activity. ActiveBPEL Engine give a "receive Queue" (list of next "receive" or Tasks) in the URL:

[http://localhost:8080/BpelAdmin/message\\_rec\\_queue.jsp](http://localhost:8080/BpelAdmin/message_rec_queue.jsp).

The ViewManager reads the next tasks with the framework "HttpClient" of Apache from the "receive Queue" URL and updates the list of next tasks in the session variable with the "receive Queue" from URL and uses this session variable to decision the next call of BPEL process. At the same time, the ViewManager gets the results of BPEL process and generates a web page according to the results. After that the ViewManager sends the web page to the client as http-response. Thus the client can see the result and know what the next task is.

### 3.4 Web Services

A web service is defined by the W3C as "a software system designed to support interoperable machine-to-machine interaction over a network" [WSDL07]. Web services are frequently just web application programming interfaces (API) that can be accessed over Internet, and executed on a remote system hosting the requested services.

The W3C web service definition encompasses many different systems, but in common usage the term refers to clients and servers that communicate over the HTTP protocol used on the web. Such services tend to fall into one of two camps: Big web services and RESTful [REST08] web services.

"Web Services" use Extensible Markup Language (XML) messages that follow the Simple Object Access Protocol (SOAP) standard and have been popular with traditional enterprise. In such systems, there is often a machine-readable description of the operations offered by the service written in the Web Services Description Language (WSDL). The latter is not a requirement of a SOAP endpoint, but it is a prerequisite for automated client side code generation in many Java and .NET SOAP frameworks (frameworks such as Spring, Apache Axis2 and Apache CXF). Some industry organizations, such as the WS-I, mandate both SOAP and WSDL in their definition of a web service.

In the SENSORIA Demonstrator the web service is the basic functional element of the orchestration. The orchestration orchestrates several web services together as a new process. This new process can be deployed as a new web service. The orchestration doesn't implement any new function self. It just reuses the exported functions of the web services.

For this Demonstrator there are six concrete web Services separately implemented. All web Services will be implemented in standard Java. Every web service is a web project and could be deployed in different web server. Each web service could be used as a basic invoke activity in BPEL process. Apache Axis2 will be used to develop the web services in this demonstration.

For the deployment of web services with the Eclipse Web Developer Tools see: Appendix A.2.

### 3.4.1 Position Service

Some services in the car assistance scenario implemented in the Automotive Demonstrator require the current position of the car. Therefore information from the Global Positioning System (GPS) is used. The GPS data is taken from the built-in GPS device of the vehicle.

A client program reads the current location data from GPS device periodically, for example in every five minutes and sends the location data to the position service server. Then the position service server updates the location data in the database for the user. In this way the current location of every user will be saved in the database of the position service server.

The "PositionService" is a web service for the orchestration. This web service exports the method "getPosition" to read the data from the database and give the current position of a specific car back. The description is shown in Table 1. The information from this service will be used as input parameter for other requested services.

Web Service	PositionService
Method	getPosition
Input parameters	userID
Output	location of the specified user
WSDL	getPositionPL.wsdl

**Table 1: Position service**

### 3.4.2 Bank Service (Bank Card Charge)

For requesting the car assistance services, first the user's credit card will be charged. Therefore credit card data has to be provided in the Automotive Demonstrator before or after using the services. Then the provided data will be checked by a bank service and the bank service will debit the payment to the user's account. This web service exported a method "chargeCredit" as shown in Table 2.

Web Service	BankService
Method	chargeCredit
Input Parameters	userID, card information
Output	Message of payment
WSDL	chargeCreditPL.wsdl

**Table 2: Bank service**

### 3.4.3 Garage Service

A garage offers the repair services for different car types. There are several configurations: A garage could only repair cars of a certain brand, such as an authorized dealer. Furthermore a garage might be limited due to the lack of technical equipment for repairing certain brands of cars. Two web services for the orchestration are prepared. The first is to find the garages in the near of user. The second is to find the “best” garage for the user, where “best” could be defined as the “nearest”, the “cheapest”, the “fastest” service, etc. In our example it is the nearest.

The web service “findGarageService” exported a method “findGarage” as shown in Table 3.

Web Service	findGarageService
Method	findGarage
Input Parameters	location of user
Output	a list of garage in the near of current user position
WSDL	findGaragePL.wsdl

**Table 3: Service to find garages nearby**

The web service “selectBestGarageService” exported a method “selectBestGarage” as shown in Table 4.

Web Service	selectBestGarageService
Method	selectBestGarage
Input Parameters	location of user
Output	the best garage
WSDL	selectBestGaragePL.wsdl

**Table 4: Service to select the “best” garage service**

### 3.4.4 Rental car service

If the broken vehicle cannot be repaired in time, the user needs to find a car rental station and rent another car. These services could help the user to find a best car rental station in the near of the broken car. Two different web services for orchestration will be implemented. The first is to find a list of rental stations nearby to the stranded car. The second is to find the “best” rental station for the user.

The web service “findRentalStationService” exported a method “findRentalStation” as shown in Table 5.

Web Service	findRentalStationService
Method	findRentalStation

Input Parameters	location of user
Output	a list of rental stations in the near of current user position
WSDL	findRentalStationPL.wsdl

**Table 5: Service to find rental car stations nearby**

The web service “selectBestRentalCarService” exports a method “selectBestRentalCar” as shown in Table 6.

Web Service	selectBestRentalCarService
Method	selectBestRentalCar
Input Parameters	location of user
Output	the best rental station
WSDL	selectBestRentalCarPL.wsdl

**Table 6: Service to select the “best” rental car service**

### 3.5 Semantic Dynamic Service Discovery (Dino)

The realization of semantic dynamic service discovery in the Automotive Demonstrator is possible through the integration of the Dino tool. “Dino provides a runtime infrastructure for supporting all stages of service composition, namely: service discovery, selection, binding, delivery, monitoring and adaptation” [Dino07a].

The services can be registered at the Dino repository in different steps. At first the existing web services must be described semantically. That means, the ontology description file (.owl file) must be created for each service. Secondly the criteria for the selection of services in Dino are the non-functional requirements of a service. So the non-functional requirements of each service must be defined statically as Capability Documents (.qos file). At last the ontology description files (.owl file) and Capability Documents (.qos file) must be copy to the special folders of the Dino broker server, so the services could be registered in Dino.

#### 3.5.1 Integrating Dino in the Automotive Demonstrator

In this Demonstrator a DinoService and a set of variants for the FindRentalStationsService and the Dino tool are integrated to show how to realize semantic dynamic service discovery. These components and how they are related is shown in Figure 10.

The ServiceOrchestration (BPEL) invokes an Axis web service (DinoService), which does not provide the concrete function itself. Instead it sends the request to the Dino broker server. The Dino broker server discovers the services from the Dino repository and selects the appropriate service according to the non-functional requirement of services. The selected FindRentalStationsService must fulfill all details of the quality requirements of the request.

Dino invokes the selected service automatically and delivers the result back to the DinoService. After that DinoService returns the result to the ServiceOrchestration (BPEL).

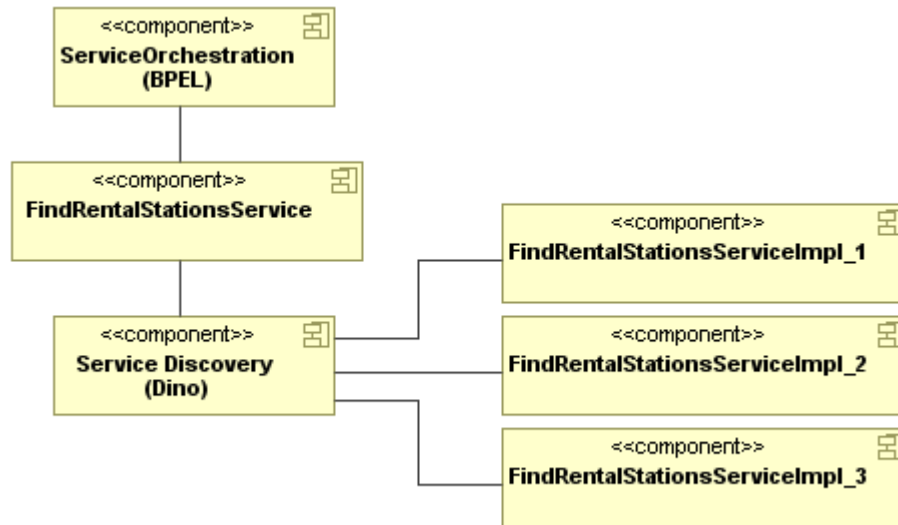


Figure 10: Dino integration

### 3.5.2 Registration of Ontology Description in Dino

The web services must have been registered at the Dino repository. The first step is to create an ontology description (.owl file) for each web service. That document (.owl file) provides a semantic description of the service-operations. Dino must get this information by copying the documents in special folders in the Dino broker Server as below:

- rentalCarAxisServiceFind.owl to ontology-cache
- rentalCarAxisServiceFind1.owl to owls-services/
- rentalCarAxisServiceFind2.owl to owls-services/
- rentalCarAxisServiceFind3.owl to owls-services/

### 3.5.3 Registration of Capability Documents in Dino

Each web service has special non-functional properties. For describing these details Dino has its own format.

The example below shows the syntax of a capability document (.qos file):

```

<QosDoc>
  <and>
    <qos name="PriceLevel" minVal="0.0" maxVal="5.0" avgVal="0.0" confidence="0.5"/>
    <qos name="CustomerSatisfaction" minVal="0.0" maxVal="0.0" avgVal="0.0" confidence="0.5"/>
  </and>
</QosDoc>
  
```

Dino must get this information also by copying of the documents in special folders in the Dino broker Server:

- rentalCarAxisServiceFind1.qos to owls-services/
- rentalCarAxisServiceFind2.qos to owls-services/
- rentalCarAxisServiceFind3.qos to owls-services/

Now Dino has all necessary information and can wait for requests for semantic dynamic service discovery.

### 3.5.4 Requirement Document of Request to Dino

The requests to Dino must be formulated in a special format. That xml-structure is called *ReqDoc*. The by Dino selected service must fulfill all details of the *ReqDoc*.

The example below shows the syntax of a *ReqDoc* file (rentalCarAxisFindReqDoc.xml):

```
<ReqDoc name="rentalCarService" xmlns="http://www.cs.ucl.ac.uk/research/dino/Dino-ReqDoc">
  <mode name="dino">
    <service name="rentalCarService"
      functional-ref="http://localhost:8080/Dino_Reqdoc/rentalCarAxisFind.owl"
      qos-ref="http://localhost:8080/Dino_Reqdoc/rentalCarAxisFindDino.qos"/>
  </mode>
</ReqDoc>
```

“qos-ref” is the URL of the required Quality of Service (qos) file for the request to Dino. Dino compares this qos file of request (qos in ReqDoc) with the qos files of service implementations (the qos files in the folder “owls-services”) and find the best matched service implementations.

There are some limitations of current version of Dino:

- Dino needs a direct Internet access. A proxy cannot be configured.
- The repository of Dino is a file repository. The discovery of service and comparison of non-functional requirement are executed linearly.
- The folder “repository” of Dino broker has to be deleted manually and Dino broker server has to be restarted in case the ontology description (owl file) of a service is changed.
- Dino supports currently only non-functional properties as decision criteria.

### 3.6 Map Display and Route Plan

To provide the driver with useful information, the display of maps is an essential feature for the Demonstrator. These maps could be provided by different map services. For example the requested service could be selected regarding the geographical position of the car.

The Google Maps API (<http://code.google.com/apis/maps/>) is used to show the map and to plan the route in the Automotive Demonstrator.

The screenshot displays the Sensoria On Road Assistance web application. At the top, the logo 'Sensoria On Road Assistance' is visible. The main content area is divided into two columns. The left column features a Google Maps interface showing a city street map with several red and blue pins indicating rental car stations. The right column contains a 'Next step' section with a 'continue' button and a list of nearby rental car stations. The list includes 'Garage Denninger', 'Garage Neckar', 'Garage Riedenburger', and 'Garage Zaubzer', each with a 'get route' link. Below this, there is a section for 'rental car station nearby your car' listing 'Car Rental Gotthelf', 'Car Rental Steinhauser', 'Car Rental Eva', and 'Car Rental Ina', each with a 'get route' link.

Figure 11: Map display



On the map the car position and the positions of the different service providers are separately marked as shown in Figure 11. If the user goes along with the selected choice, then he can request the service. Otherwise he tells the system to choose another service or to modify his request completely.

If the user selects a service provider, the route plan from current car location to service provider could be indicated. Figure 12 shows an example of route plan.



Figure 12: Route in map

## 4 MDD4SOA Plug-ins for Generation of Executable BPEL

The MDD4SOA Eclipse plug-in in its first version was designed for transforming UML models to abstract BPEL processes. The resulting BPEL process file and WSDL files generated were not executable by any BPEL engine. The specification of namespaces, service location and service binding of web services in BPEL/WSDL files were not enough for a running application using a specific BPEL engine.

The ActiveBPEL engine, we used for our Automotive Demonstrator, requires on the one hand a specific directory structure, and on the other hand a catalog file and a process deployment descriptor file (pdd file). Therefore, we implemented an extension of the MDD4SOA Eclipse plug-in called “BPEL2ActiveBPEL/WSDL Converter and Deployment”. It is used for the generation of executable BPEL and it is also implemented as an Eclipse plug-in. The new plug-in comprises two methods to transfer the abstract BPEL/WSDL files to executable BPEL/WSDL files in ActiveBPEL engine and one extra method to deploy the executable BPEL process to a web server.

### 4.1 Integration of MDD4SOA Extension in SDE

In its initial form, the SENSORIA Development Environment (SDE) perspective has the layout as in Figure 13. (Note that in the screenshot of Figure 13, some tools have already been installed by using other update sites available on the SDE web site).

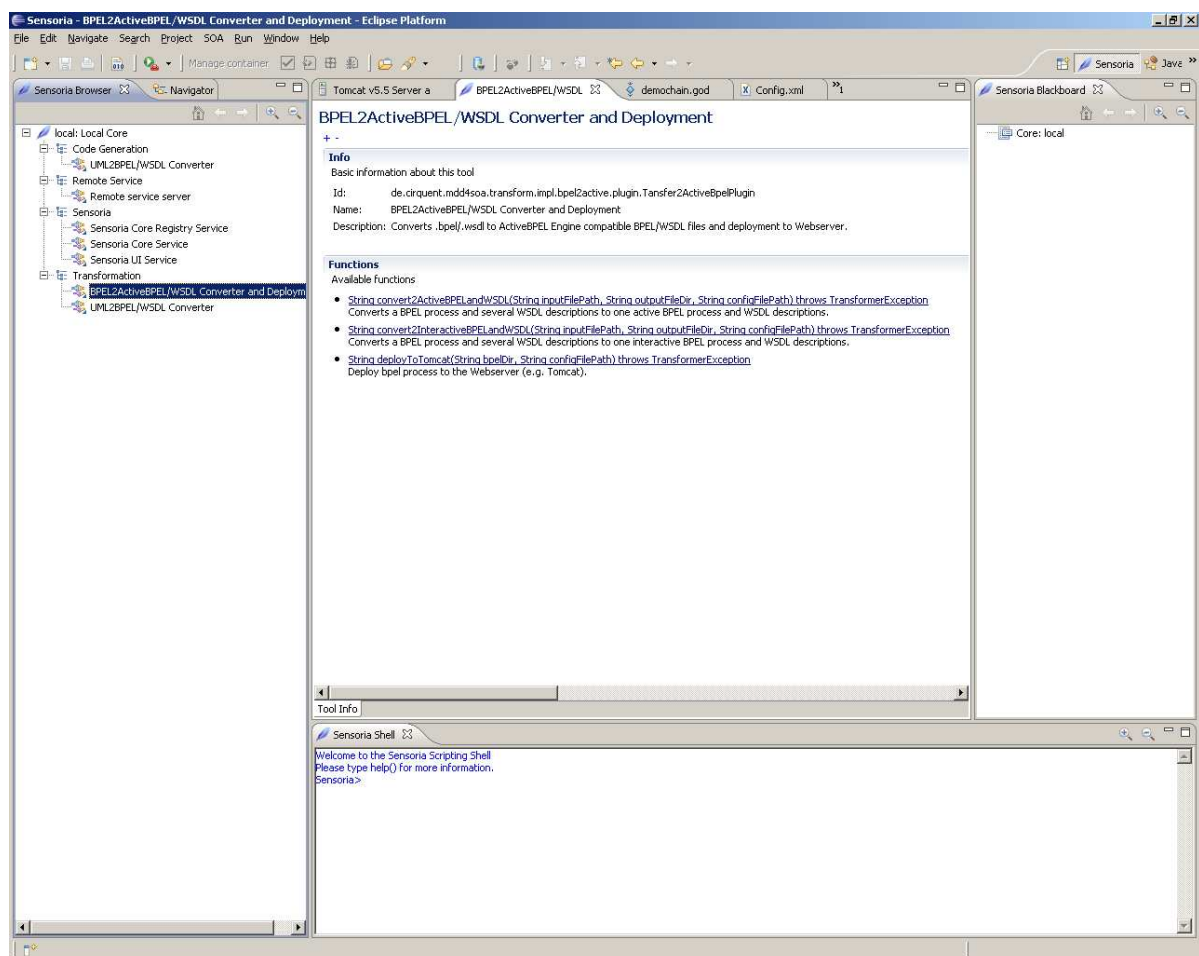


Figure 13: BPEL2ActiveBPEL/WSDL converter and deployment

Three views are visible in SDE:

- On the left-hand side, the **SENSORIA Browser** is displayed. It contains a categorized listing of all tools which are currently available in this particular instance of the SDE.
- On the right-hand side, the **SENSORIA blackboard** is displayed. The blackboard is used to store Java object values in-between service invocations when using the manual generic user interface (UI) to access tool functions.
- At the bottom, the **SENDORIA shell** is displayed. As pointed out above, the shell is a manual orchestrator which can be used to employ JavaScript to call tool (plug-in) functions.

Double-clicking on a tool in the SENSORIA Browser displays more information about the functions of the tool, e.g. of the “BPEL2ActiveBPEL/WSDL Converter and Deployment”.

Besides some general information about the tool in the upper section, all available functions of the tool are listed in the function section. These functions may be directly invoked using the generic wizard UI by selecting the appropriate links. This is detailed in the following sections.

For the development of Eclipse plug-ins like the “BPEL2ActiveBPEL/WSDL Converter and Deployment” the reader is referred to the SDE tutorial (<http://svn.pst.ifi.lmu.de/trac/sct/wiki/Tutorial>) and briefly described in the Appendix A.1.

## 4.2 Functions of the MDD4SOA Extension

There are three new functions in MDD4SOA\_Extension. You can find them in “BPEL2ActiveBPEL/WSDL Converter and Deployment” of the catalog “Transformation” in the SENSORIA browser. Figure 13 is a screenshot of the MDD4SOA\_Extension.

### 4.2.1 Transformation to Executable BPEL

The goal of this function is to convert an abstract BPEL process and several abstract WSDL descriptions to an executable BPEL process in the ActiveBPEL engine. In this function the namespaces, service location and service binding of web services in BPEL/WSDL files are specified based on the information available within the configuration file.

At same time this function generates the for ActiveBPEL engine specified directory structure sketched in Figure 14, WSDL-catalogue file (catalog.xml) and process deployment descriptor file (.pdd file). A Process Deployment Descriptor (.pdd) file describes service binding and, optionally, the process version details of a BPEL process. The WSDL-catalogue file tells the BPEL engine, where the WSDL files can be found in the deployment archive.

The example below shows the syntax of a WSDL-catalogue file:

```
<wsdlCatalog>
  <wsdlEntry location="string" classpath="slash/separated/classpath/filename.wsdl"/>
</wsdlCatalog>
```

The example below shows the syntax of a Process Deployment Descriptor (.pdd) file:

```

<process xmlns="http://schemas.active-endpoints.com/pdd/2006/08/pdd.xsd"
  xmlns:bpelns="http://target.sensoria.cirquent.de/orchestration2_orchestration2_Interactive.bpel"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
  location="bpel/orchestration2_orchestration2_Interactive.bpel"
  name="bpelns:orchestration2_orchestration2_Interactive">
  <partnerLinks>
    <partnerLink name="findGaragePL">
      <partnerRole endpointReference="static">
        <wsa:EndpointReference xmlns:g="http://garage.sensoria.cirquent.de/findGaragePL.wsdl">
          <wsa:Address>http://localhost:8080/SensoriaGarage/services/GarageService</wsa:Address>
          <wsa:ServiceNamePortName="findGaragePLServicePort">
            g:findGaragePLService
          </wsa:ServiceName>
        </wsa:EndpointReference>
      </partnerRole>
      <myRole allowedRoles="" binding="RPC" service="findGaragePLService"/>
    </partnerLink>
    ...
  </partnerLinks>
  <wsdlReferences>
    <wsdl location="project:/findGaragePL.wsdl"
      namespace="http://garage.sensoria.cirquent.de/findGaragePL.wsdl"/>
    ...
  </wsdlReferences>
</process>

```

Function: convert2ActiveBPELAndWSDL (String inputFilePath, String outputFileDir, String configFilePath)

Parameters:

- inputFilePath: the absolute file path of the input abstract BPEL file, the abstract WSDL files have to be saved in the same directory as the input abstract BPEL file.
- outputFileDir: the absolute file path of the output directory for generated BPEL/WSDL files
- configFilePath: the absolute file path of the configuration file

Return value of this function is the absolute file path of the generated BPEL file.

#### 4.2.2 Transformation to Interactive BPEL

In the current version of UML4SOA profile there is no data handling. It will be including in the next version of UML4SOA profile. The user interactions must be modeled explicitly in BPEL. So we give a compromise solution for this problem of the current version of UML4SOA profile. This method inserts a receive activity before every invoke activity and a reply activity after every invoke activity in the BPEL. At the same time an assign activity is inserted before the new reply activity. The result of the invoke activity as output message for reply activity will be specified in this assign activity. So the BPEL process is forced to communicate with client. Data handling and user interactions are controlled in the ViewManager (see section3.3).

Function: convert2InteractiveBPELAndWSDL(String inputFilePath, String outputFileDir, String configFilePath)

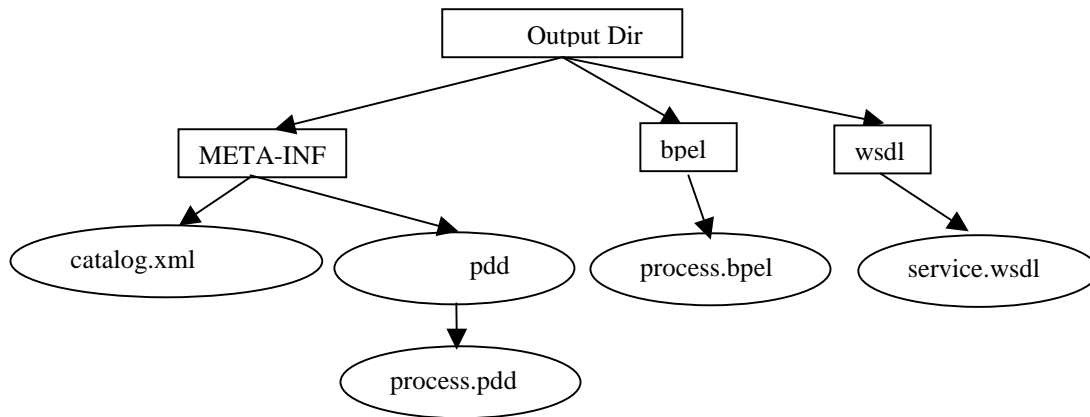
Parameters:

- inputFilePath: the absolute file path of the input BPEL file
- outputFileDir: the absolute file path of the output directory for generated BPEL/WSDL files
- configFilePath: the absolute file path of the configuration file

Return value of this function is the absolute file path of the generated BPEL file.

### 4.2.3 Deployment to a Web Server

This function compresses the output directory of BPEL process as a bpr file (BPEL deployment archive file) and deploys the bpr file to the bpr directory of Tomcat server. The directory structure of a BPEL deployment archive for ActiveBPEL engine is sketched in Figure 14.



**Figure 14: Directory structure of a BPEL deployment archive**

Function: `deployToTomcat(String bpelDir, String configFilePatH)`

Parameters:

- `bpelDir`: the absolute file path of the input directory of to deployed BPEL process
- `configFilePatH`: the absolute file path of configuration file

Return value of this function is a message of deployment status.

More details on the documentation of the deploying of a BPEL Process can be found under:

<http://www.activebpel.org/samples/samples-2/deploy/doc/index.html>

### 4.3 Configuration of the Converter

An orchestration invokes several web services. These web services could be deployed anywhere in the web locally or remotely. Currently, the orchestration gets the information on location of services through the configuration file.

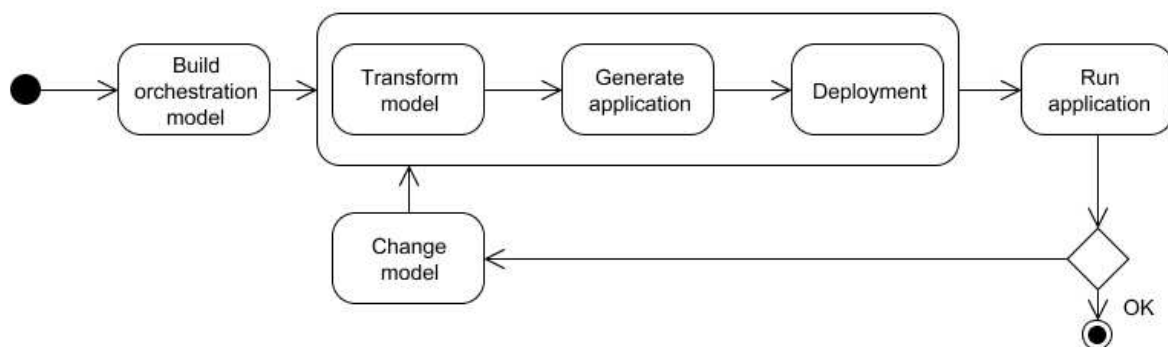
## 5 Demonstration Process

The goal of the Automotive Demonstrator is to show how some SENSORIA techniques can be used to support a real model-driven process with fully automatic generation of a web application based on the orchestration of services. In particular, the Automotive Demonstrator shows how to build a model of orchestration and transform models into other models and code (model2model & model2code).

### 5.1 Model-Driven Development

Model-Driven Development (MDD) is a software development methodology which focuses on creating models, or abstractions, more close to some particular domain concepts rather than computing (or algorithmic) concepts. It is meant to increase productivity by maximizing compatibility between systems, simplifying the process of design, and promoting communication between individuals and teams working on the system.

A modeling paradigm for MDD is considered effective if its models make sense from the point of view of the user and can serve as a basis for implementing systems. MDD gives architects the ability to define and communicate a solution while creating artifacts that become part of the overall solution.



**Figure 15: Model-driven development process**

Figure 15 shows a sample of the SENSORIA model-driven development process (MDD4SOA). It consists of the following steps:

1. At first you should build the UML model for orchestration are built with the UML4SOA Profile in a UML tool (e.g. Magicdraw, Rational Software Modeler or the Rational Software Architecture).
2. The orchestration model is transformed to an abstract BPEL process and a set of WSDL files with a corresponding function of the MDD4SOA Eclipse plug-in.
3. The abstract BPEL process and the WSDL files are transformed to executable BPEL process and WSDL files, i.e. the BPEL process is executable with an ActiveBPEL engine. The transformation is performed with the corresponding function of the MDD4SOA\_Extension.
4. The executable BPEL process and the WSDL files are deployed to a web server (e.g. Tomcat).

If all web services invoked by the orchestration are already implemented and available, the whole application can be run immediately. If there are some web services still not implemented, these web services have to be implemented with the specification of the WSDL generated by the model transformation.

After the model transformation and deployment of applications have been executed successfully, the web server can be started. Then the server applications are ready for client request. If the orchestration model is changed, the whole process should be run again.

### 5.2 Building UML4SOA Orchestration Model

The UML4SOA orchestration model can be built with a UML tool (e.g. Magicdraw, Rational Software Modeler or the Rational Software Architecture). Before starting to build a model, the UML4SOA profile must be installed in the UML tool (details see: <http://www.mdd4soa.eu/web/wiki/InstallUseProfile>). Then the specific UML4SOA model elements can be used to model an activity diagram representing an orchestration model.

Once the model is complete, export the model to UML files (EMF UML2 (v2.x) XMI files) like it is shown in Figure 16. The exported UML files will be used as input files in the model transformation of the MDD4SOA plug-in. Figure 16 is the activity diagram of a sequential composition of services (first variant of our scenario). We used the MagicDraw. Tool.

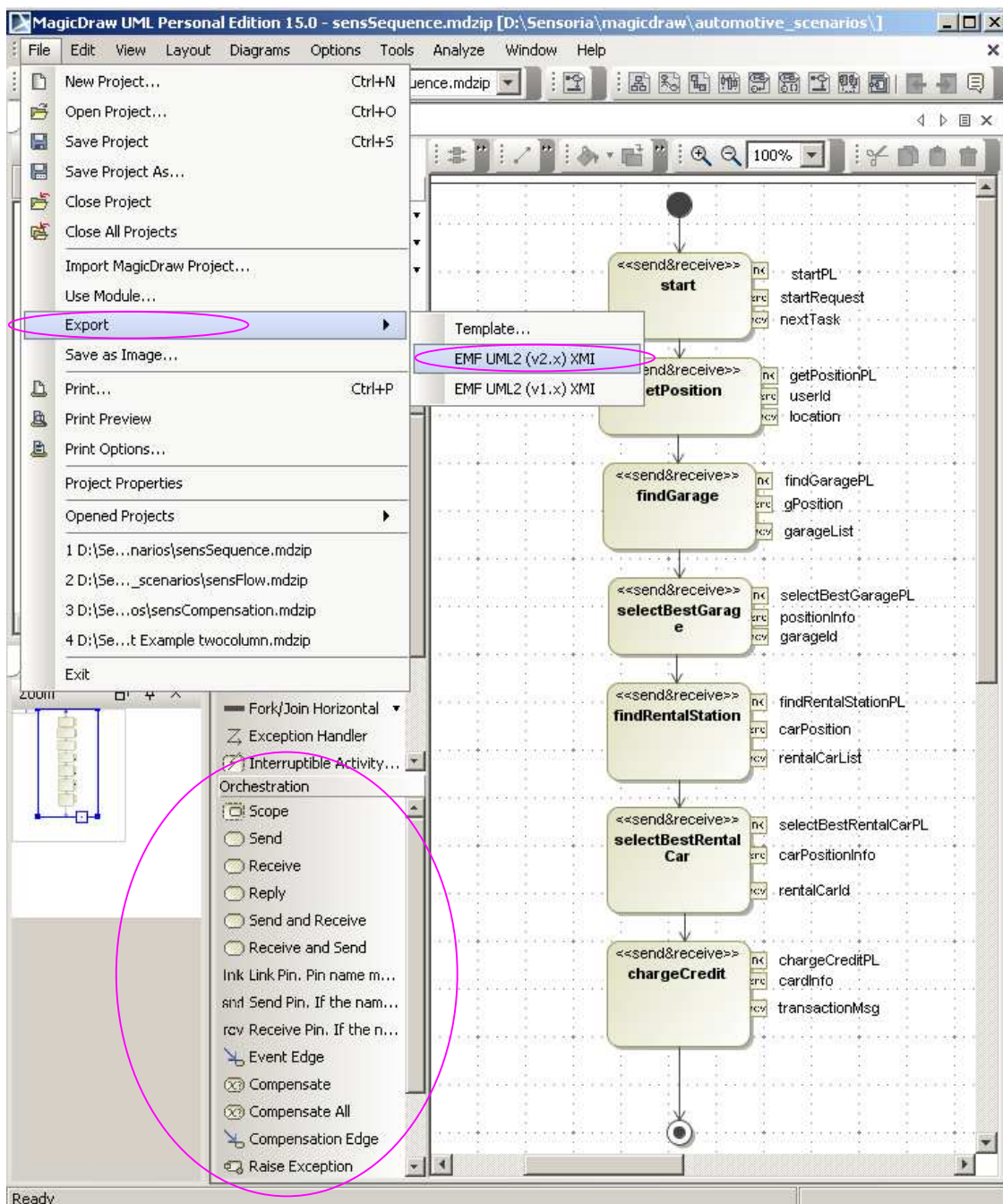




Figure 16: Activity diagram of the orchestration

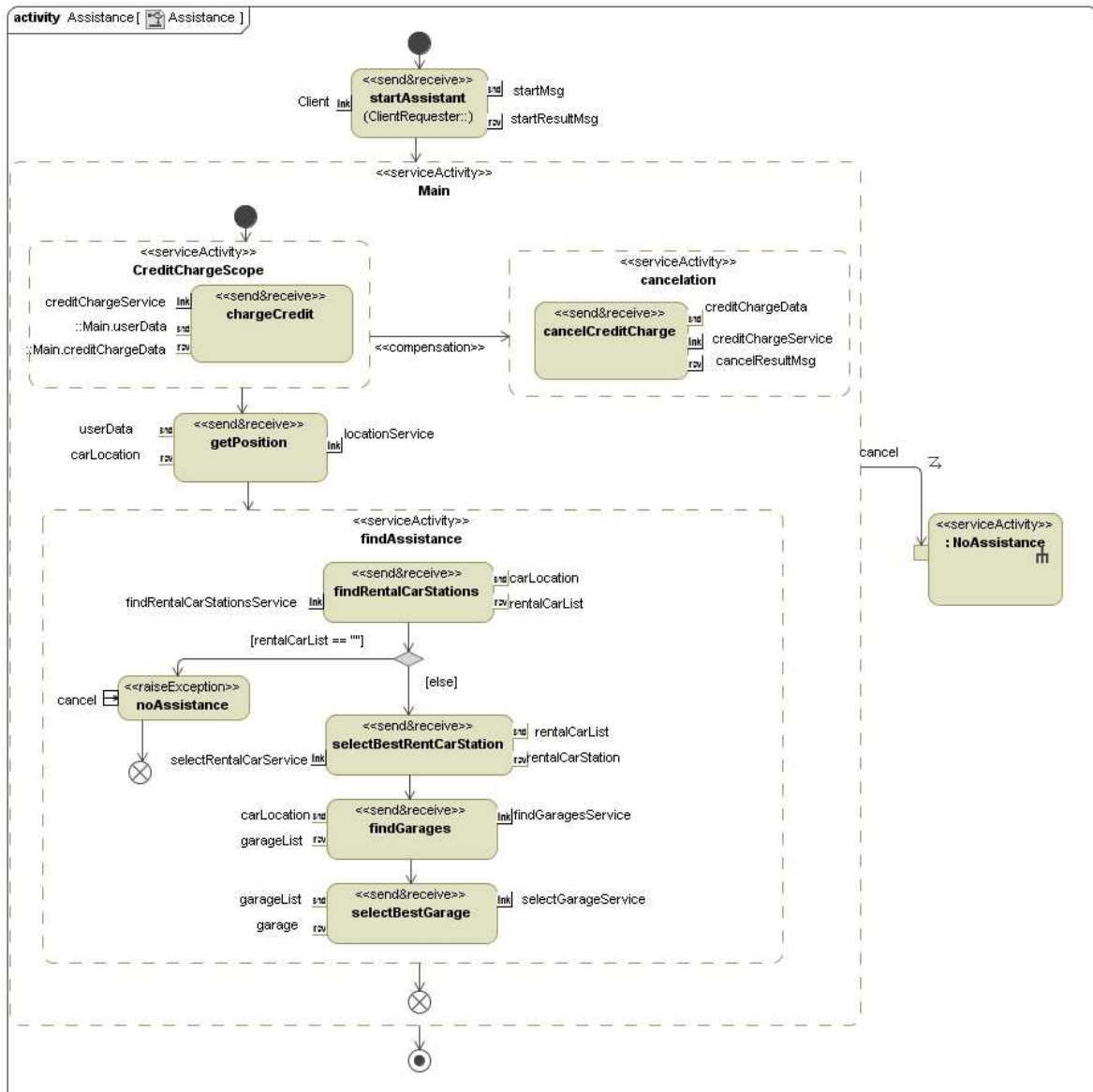


Figure 17: Sequence Activity diagram of the orchestration

### 5.3 Transformation Chain

Once the orchestration has been modeled, then it can be transformed step by step to an executable BPEL process and deployed to the web server. The basic functions of the transformation between different models and the function of deployment are available as Eclipse plug-in functions of the MDD4SOA in the SDE. These functions can be orchestrated manually or with help of the graphical tool chain editor of SDE or as a script. The advantage of the script or the graphic representation of the tool chain is an easy execution of the whole development process.

One of the main aims of the SDE is the enabling of orchestration of tools. As tools can be discovered using the SENSORIA core and their interface functions, orchestration can be provided by arbitrary tools on top of this service layer. The SDE provides three options for orchestrating tools:

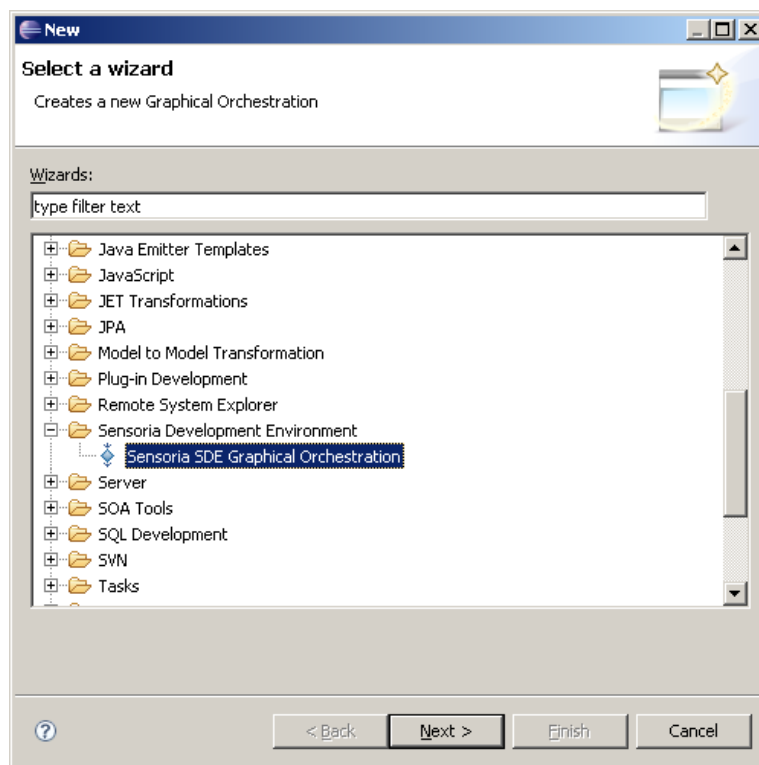


- **SENSORIA Shell** provided within the core is a manual orchestrator which provides such an orchestration mechanism as a UNIX-like shell with the additional ability to store, load, and execute scripts.
- **SENSORIA Scripts** are written in JavaScript and executed without parameterization. In order to create new tools, SENSORIA Tool Scripts may be written which contain JavaScript functions. After converting them to tools, they can be used as any other tool through the tool browser.
- **Graphical Orchestrations (Tool Chain)** is a graphical editor. It allows to write data-driven activity diagrams. Such orchestrations are again tools themselves.

In all orchestrations, the SENSORIA core can be accessed directly; thus any tool can be retrieved from the core and its interface functions executed. In this demonstration the graphical orchestrations tool chain will be applied. Further information can be found at the SDE Tutorial (<http://svn.pst.ifi.lmu.de/trac/sct/wiki/Tutorial>).

### 5.3.1 Orchestration of Eclipse Plug-in Functions with SDE Graphical Orchestrations

Figure 18 shows how to create a graphical orchestration (.god file): *File > New > Other > Sensoria Development > Sensoria SDE Graphical Orchestration*



**Figure 18: Creating a graphical orchestration (.god file)**

The canvas of the editor corresponds to a new tool to be created in the SDE. As each tool can contain multiple functions, a function needs to be added first. To do so, click on Function in the palette on the right hand side, and click onto the canvas to create a new function. Name it appropriately, for example convert2ActiveBPELandWSDL.

Now, tool functions can be dragged into the new function as appropriate from the palette on the right-hand side, which contains all executable function of all installed tools. Additionally, the following Meta-tools can be used:

- Use Link to model data flow from an output of a function to the input of another
- Use Input Pin to add an input parameter to a function
- Use Output Pin to add an output parameter to a function

An example of a complete script is shown in Figure 19.

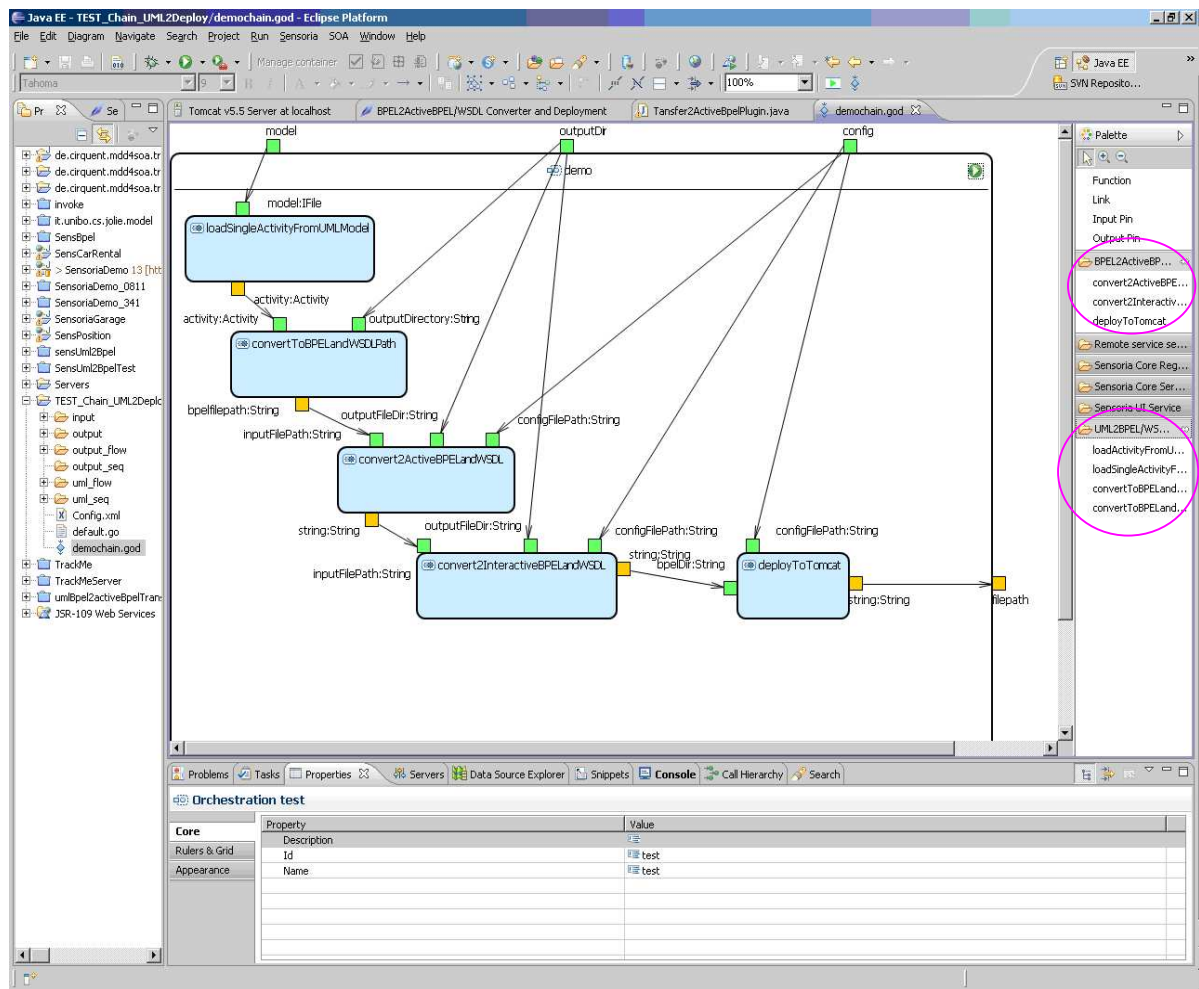


Figure 19: Chain tool script demochain.god

In the Demonstrator the following transformations will be done. The following five functions of SDE will be orchestrated.

**Two functions under “*Converter UMLASOA to BPEL / WSDL*”:**

1. loadSingleActivityFromUMLModel: transforms UML2 models to the Intermediate Orchestration Model (IOM)
  - *Input*: model (IFile of uml model)
  - *Output*: activity (Activity of IOM)
2. convertToBPELandWSDLPath: transform IOM to abstract BPEL / WSDL
  - *Input*: activity (Activity of IOM)
  - *Output*: absolute file path of the generated BPEL file.

**Three functions under “*BPEL2ActiveBPEL/WSDL Converter and Deployment*”**

1. convert2ActiveBPELandWSDL: transform abstract BPEL / WSDL to executable BPEL / WSDL
  - *Input*: inputFilePath (the absolute file path of input abstract BPEL file)


- *Input:* outputFileDir (the absolute file path of the output directory for executable BPEL/WSDL files)
  - *Input:* configFileDir (the absolute file path of configuration file)
  - *Output:* absolute file path of the generated BPEL file.
2. convert2InteractiveBPELandWSDL: transform BPEL to interactive BPEL
- *Input:* inputFileDir (the absolute file path of input BPEL file)
  - *Input:* outputFileDir (the absolute file path of the output directory for generated BPEL/WSDL files)
  - *Input:* configFileDir (the absolute file path of configuration file)
  - *Output:* absolute file path of the generated BPEL file.
3. deployToTomcat: deploy to tomcat web server
- *Input:* bpeDir (the absolute file path of the root directory of the BPEL/WSDL files)
  - *Input:* configFileDir (the absolute file path of configuration file)
  - *Output:* message of deployment

Transformer process: UML model → Intermediate Orchestration Model (IOM) → abstract BPEL/WSDL → for ActiveBPEL engine executable BPEL/WSDL → interactive BPEL → deploy to Tomcat

Name of function	Goal of function
loadSingleActivityFromUMLModel	UML model → Intermediate Orchestration Model (IOM)
convertToBPELandWSDLPath	Intermediate Orchestration Model (IOM) → abstract BPEL/WSDL
convert2ActiveBPELandWSDL	abstract BPEL /WSDL → for ActiveBPEL engine executable BPEL/WSDL
convert2InteractiveBPELandWSDL	ActiveBPEL engine executable BPEL / WSDL → interactive BPEL
deployToTomcat	interactive BPEL → deploy to Tomcat

**Table 7: Functions for building a tool chain**

These five tool functions can be used for dragged into the tool chain.

Finally, to execute a function, simply click on the green play button  in the upper right corner of the outermost function as shown in Figure 20. In addition, Figure 20 indicates how to set the parameters in order to run a tool function.

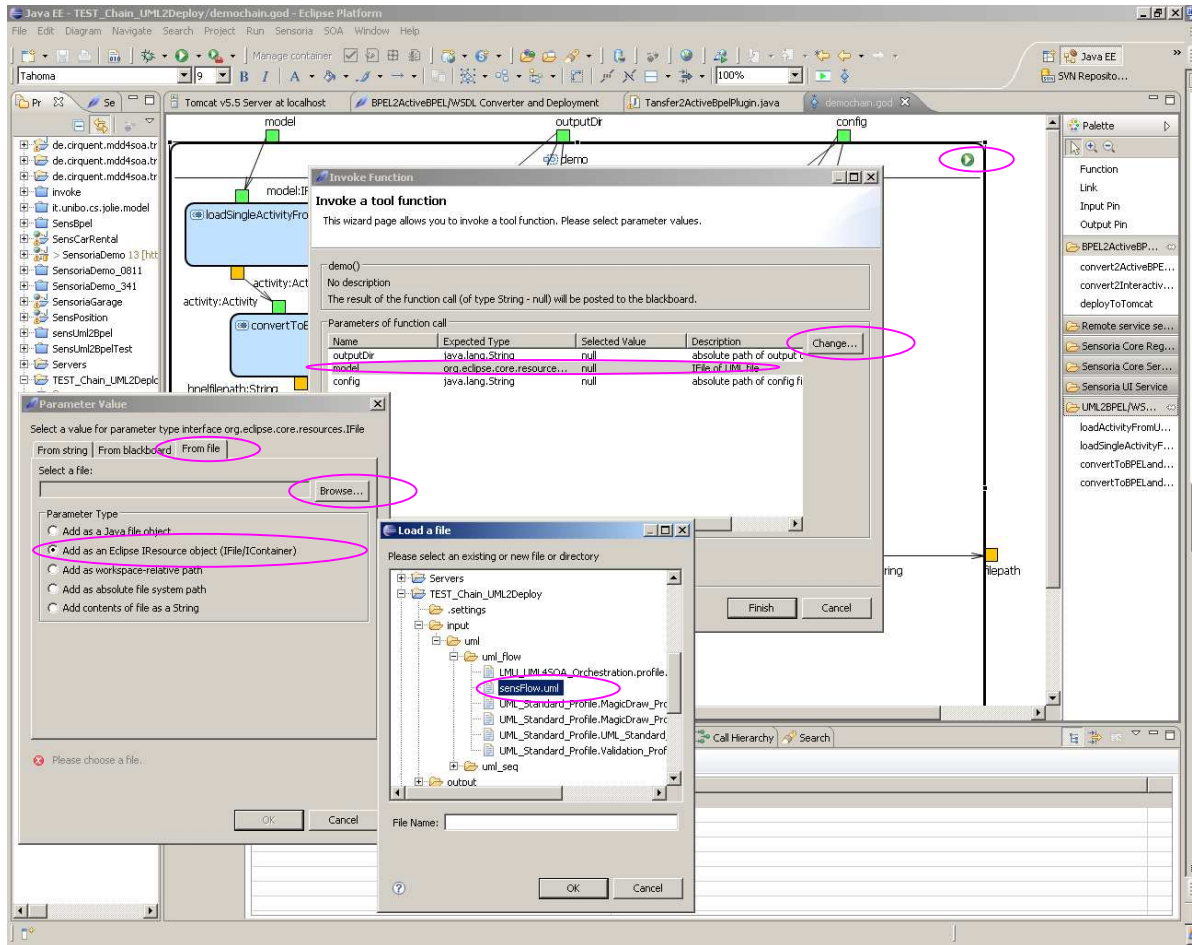


Figure 20: Set arguments of functions

Select a parameter and click “change”. You can set the parameter from string, blackboard or file in the new windows. If you want to set a parameter from file, you should select a file type (e.g. java file object, IFile/Icontainer or absolute file system path) and click “browser” to select a file. (*Change →From file →Browser* in Figure 20)

In this case you should set the following three parameters:

- *model*: Eclipse IResource object (IFile/Icontainer) of an UML file
- *outputDir*: absolute file system path for output directory
- *config*: absolute file system path of configure file

### 5.3.2 Result of the transformation

Figure 21 shows the BPEL process of the first variant of the “On Road Assistance” scenario, which is executable in an ActiveBPEL. The process is a sequential composition of services.

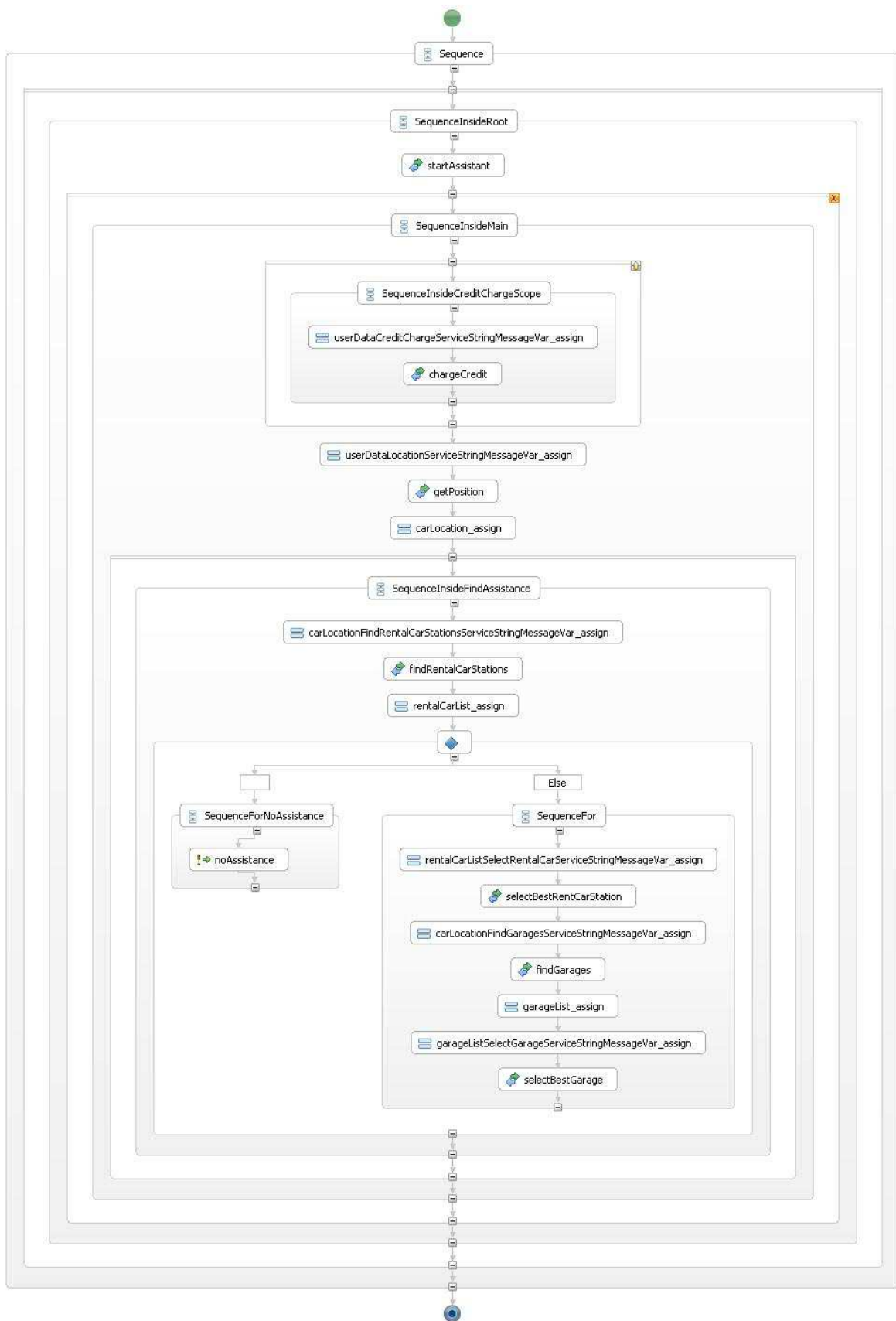


Figure 21: In ActiveBPEL engine executable BPEL process of scenario of Figure 16

Figure 22 shows the BPEL process enriched with interactive elements. Before every invoke activity a receive activity has been automatically inserted. After every invoke activity an assign activity and a reply activity have been inserted. This process is also executable in an ActiveBPEL engine.

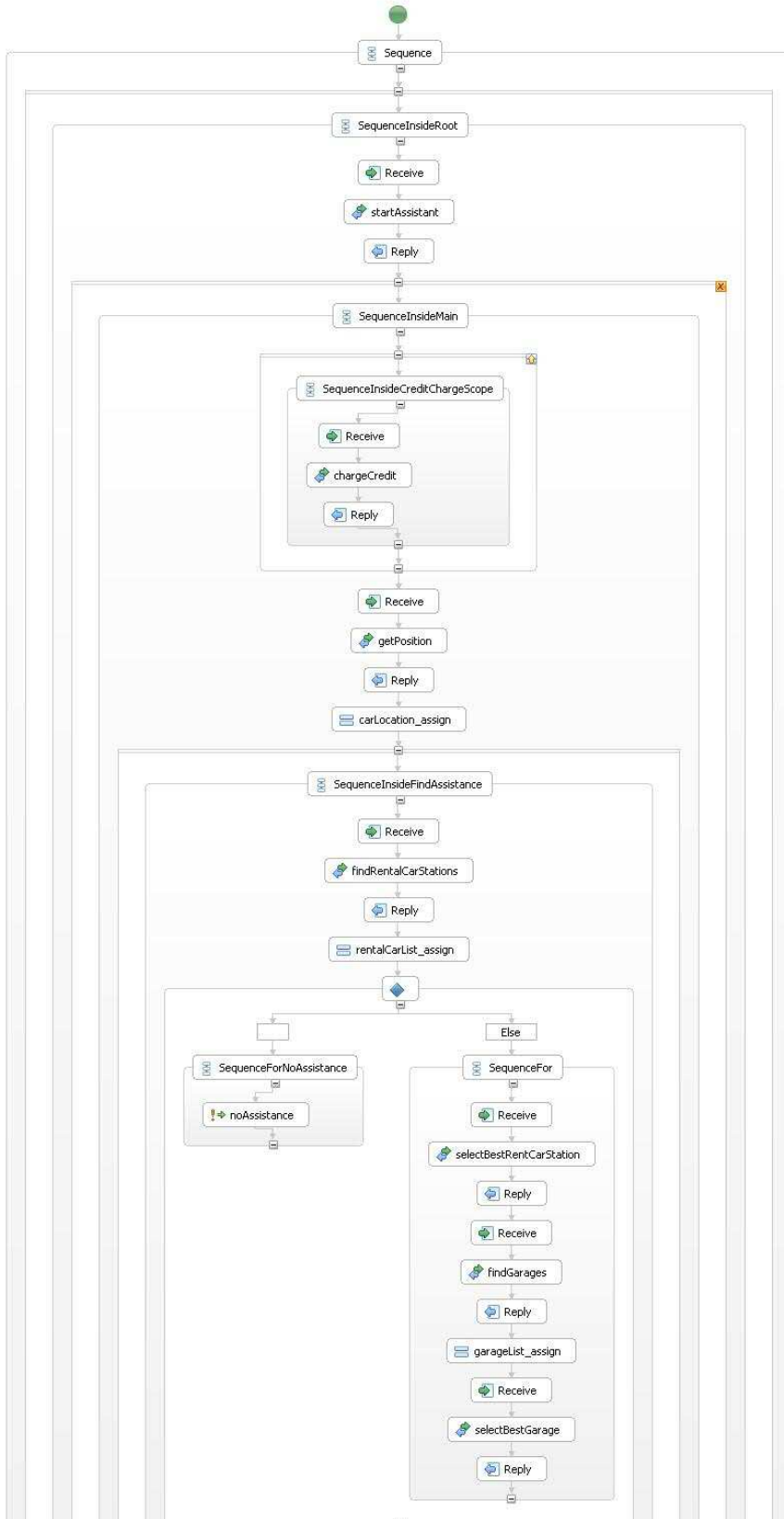


Figure 22: Interactive BPEL process of scenario of Figure 16

## 5.4 Run Deployed Application

If the BPEL process was already deployed in web server, the web server can be started. Afterwards the BPEL process will be available as a web service. The BPEL process can be invoked as any standard web service with Axis.

In the Automotive Demonstrator the client uses the web interface in their web browser to run the application.

- The user fills-in the form and press the button “Start Service” to send an http-request to server.
- The server gets the request and redirects the request to ViewManager.
- ViewMagager parses the request and builds the parameters for next service invoke and calls the BPEL process.
- The BPEL process starts and invokes the web services as modeled in the orchestration. Result is sent to the ViewManager.
- The ViewManager builds a web page according these results and sends the web page to client.

Figure 23 shows the start page of the demonstration.

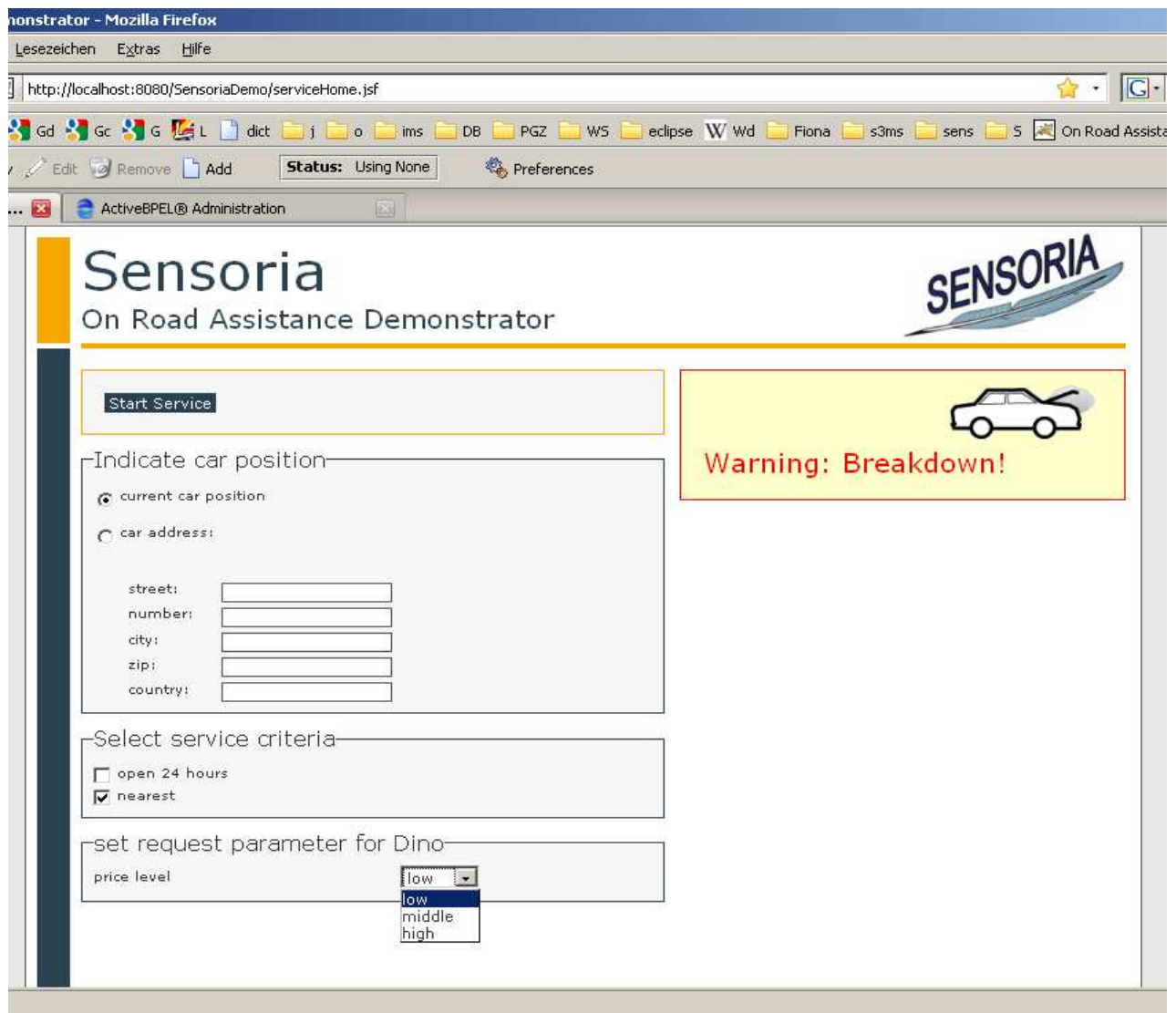


Figure 23: Start page of the demonstration



According to the workflow of the orchestration of our scenario of Figure 16, the PositionService is started in the first step. The current location of the car is shown in the map of Figure 24. The next step is to find the garage nearby the car according to the workflow of orchestration.

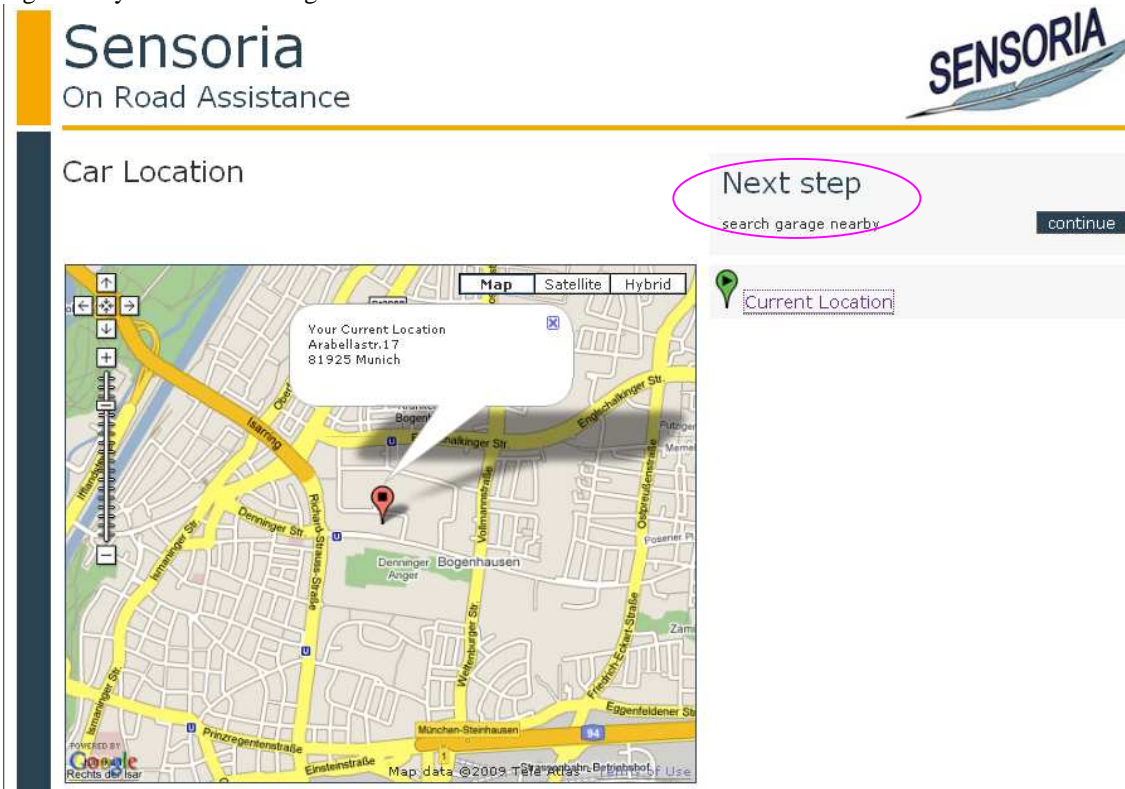



Figure 24: Car location

In this step a list of garages are found by the FindGarageService and shown in the map of Figure 25. The garages are marked in the map. According to the workflow of orchestration in scenario of Figure 16 the next step is to find best garage.



# Sensoria

On Road Assistance



## Garage nearby your car

**Next step**  
search best garage continue

**Current Location**

**Garage nearby your car**

Garage Denninger	get route
Garage Neckar	get route
Garage Riedenburger	get route
Garage Zaubzer	get route





Figure 25: Find Garage

In this step the best garage is found by SelectGarageService and shown in the map of Figure 26. According to the workflow of orchestration in the scenario of Figure 16 the next step is to find rental car station.

# Sensoria

On Road Assistance



## The best Garage

**Next step**  
search rental car station nearby continue

**Current Location**

**The best garage**

Garage Denninger	get route
------------------	-----------

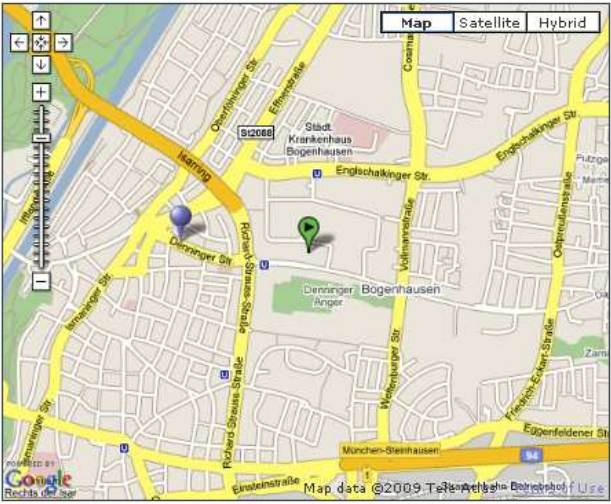


Figure 26: Find best garage

In this step a list of rental car stations is found by FindRentalStationService and shown in the map of Figure 27. According to the workflow of orchestration in the scenario of Figure 16 the next step is to find the best rental car station.

The screenshot shows the Sensoria On Road Assistance interface. At the top left, the logo "Sensoria On Road Assistance" is displayed. On the right, there is a "SENSORIA" logo with a stylized car icon. The main heading is "Rental car station nearby your car". Below this, a map shows the current location (green pin) and several nearby rental car stations (red pins). A "Next step" button is circled in pink, with the text "search best rental car station" and a "continue" button next to it. The map includes a search bar and navigation controls. The list of rental car stations is as follows:

Station Name	Action
Garage Denninger	get route
Car Rental Gotthelf	get route
Car Rental Steinhauser	get route
Car Rental Eva	get route
Car Rental Ina	get route

Figure 27: Find rental car station

In this step the best rental car station is found by SelectBestRentalCarService and shown in the map of Figure 28. According to the workflow of orchestration in the scenario of Figure 16 the next step is to charge the credit card.

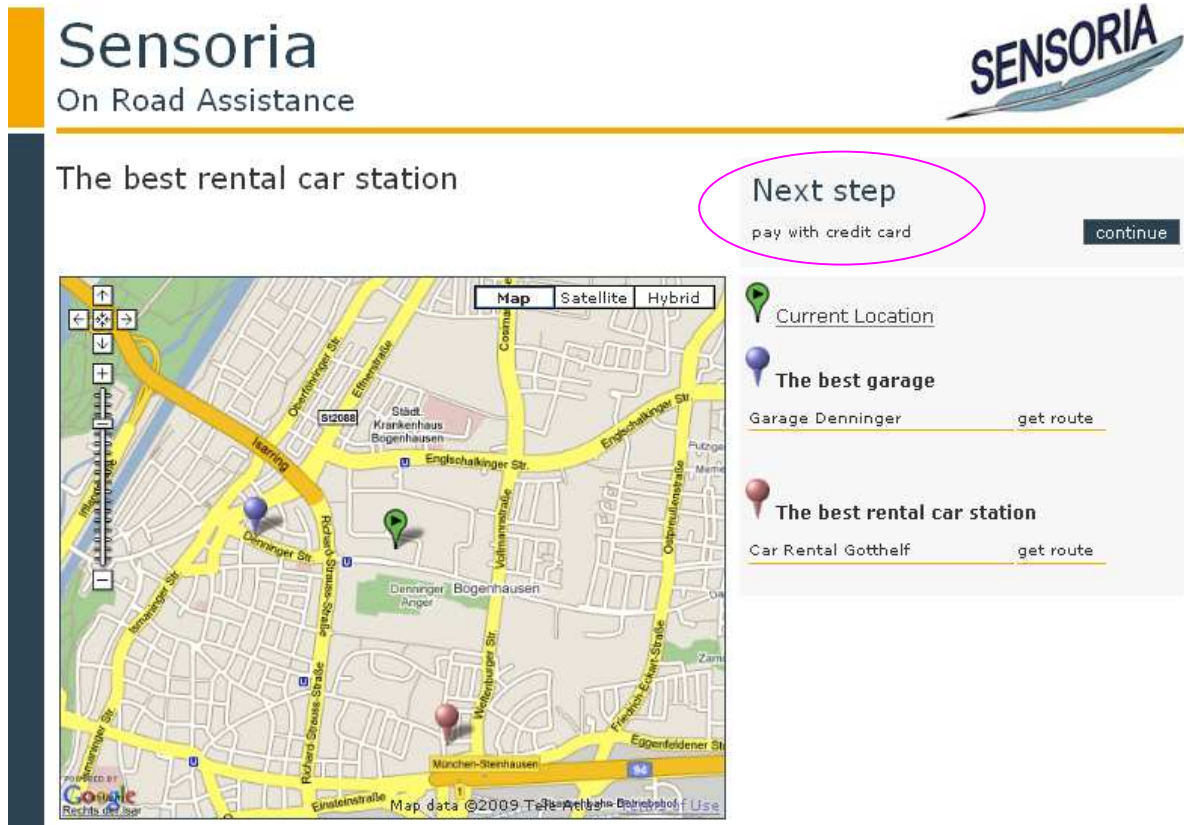


Figure 28: Best rental car station

In this step the services will be paid with credit card as shown in Figure 29. This is the last step of the scenario according to the workflow of orchestration of Figure 16.

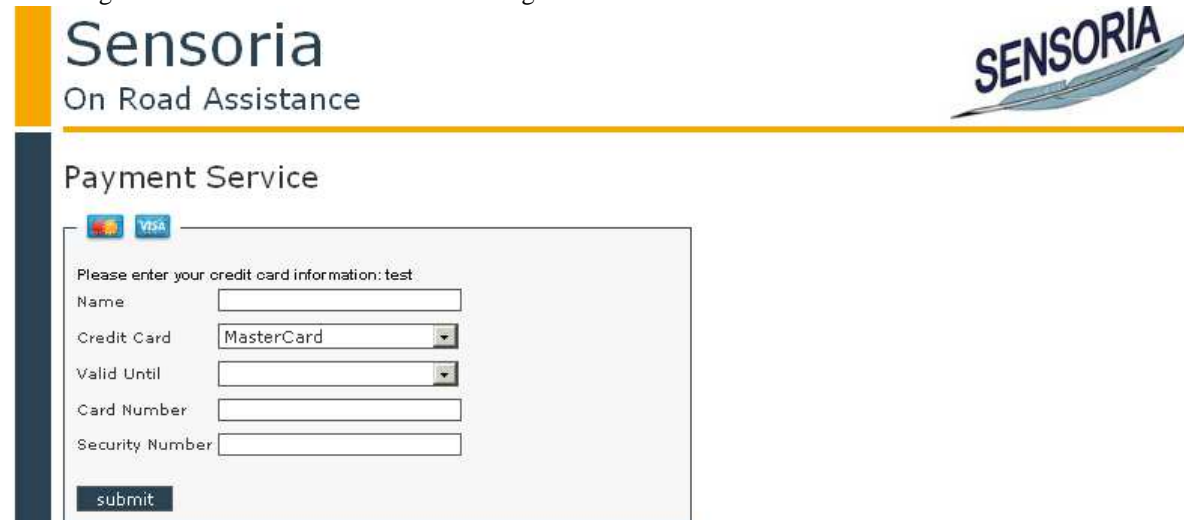


Figure 29: Payment with credit card

## 5.5 Running the Demonstrator after Modifying the UML4SOA Model

If you want to change your business logic, you can change the UML4SOA model and run the transformation process with tool chain function in last section (file demochain.god) again.

In this demonstration we change the sequence process of previous scenario to a parallel process for the second variant of the scenario. In the first scenario the user pays the credit in the last step after the using of all services.

In the current scenario the user must pay the credit at first, and then uses the services. In the first scenario the services are invoked one by one sequentially. In the second the services “findGarage” and “findRentalStation” are invoked in parallel. The results of both services would be shown in the same view. The services “selectBestGarage” and “SelectBestRentalCar” will also be invoked in parallel.

Figure 30 shows the modified parallel UML model of scenario of Figure 30:

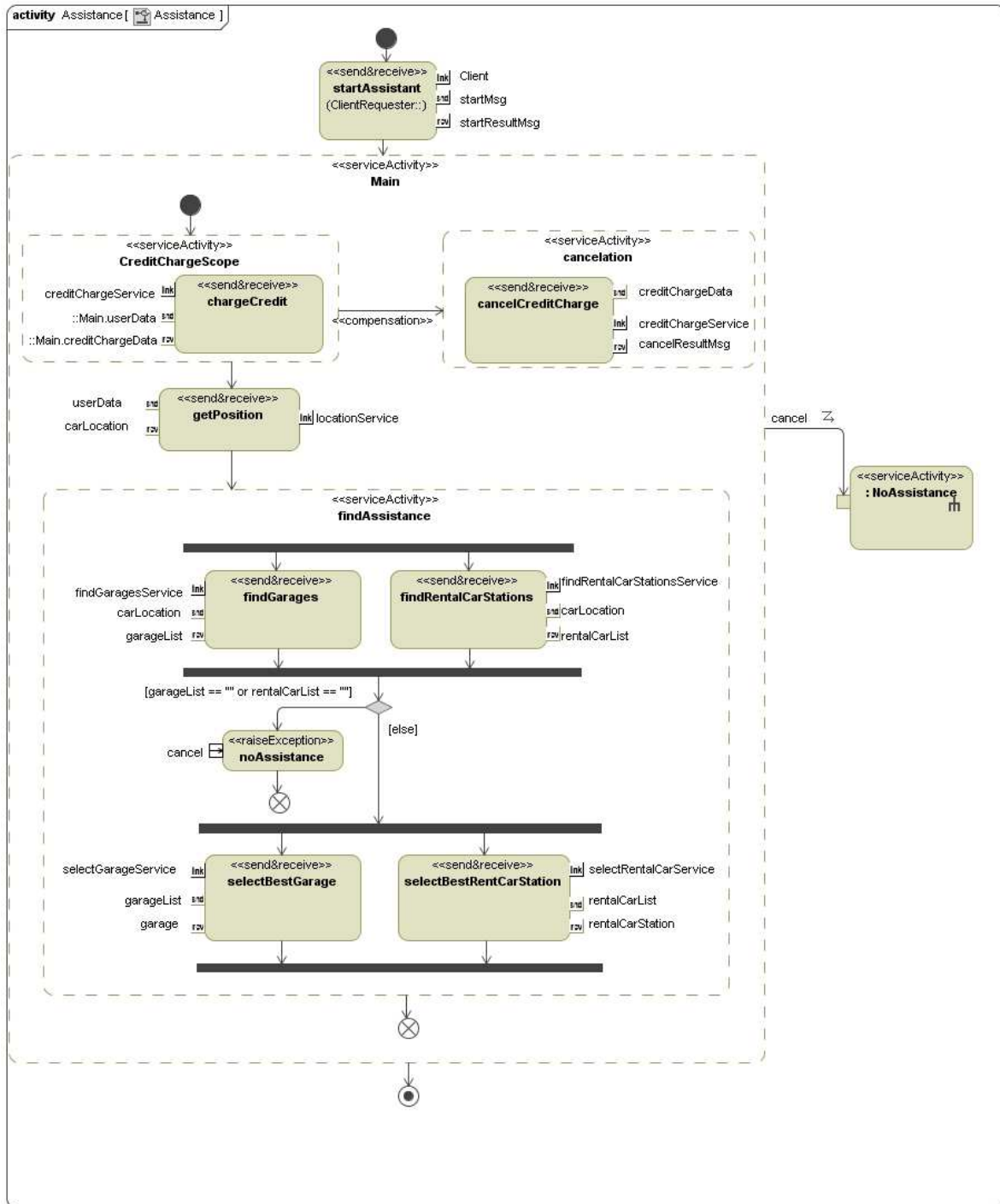
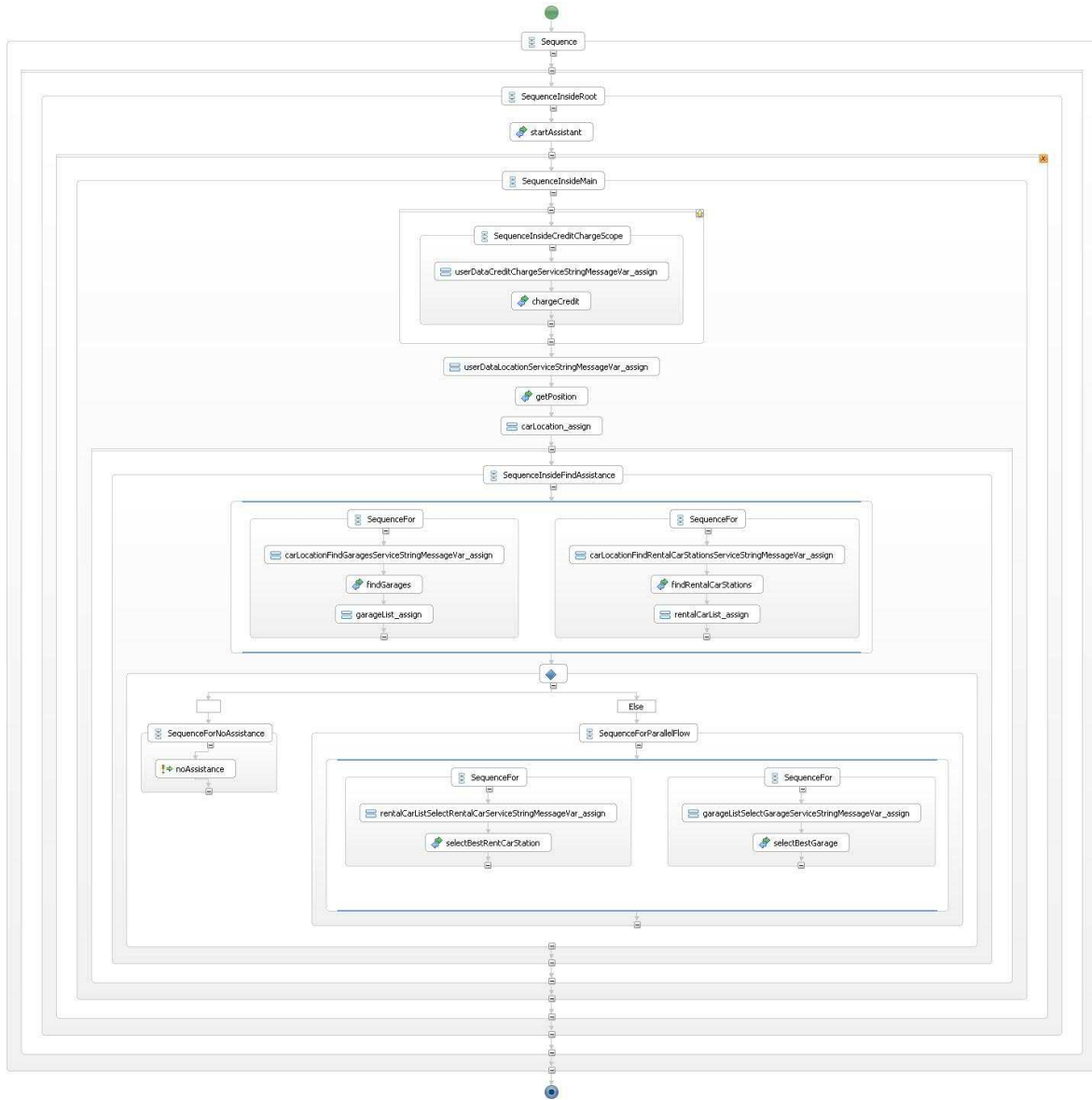


Figure 30: Activity diagram of the orchestration for parallel scenario



After the modification of the UML model you have to export it as UML file and run the tool function in the file tool chain to transfer the model. The following two BPEL process are obtained as results of the transformations.

Figure 31 shows the parallel BPEL process of the second scenario without user interaction. This BPEL process is executable in ActiveBPEL engine. If the BPEL process is started, the process will be executed without interruption until the end.



**Figure 31: BPEL process of scenario of Figure 30**

Figure 32 shows the BPEL process of the parallel scenario of Figure 30. This BPEL process is able to interactively communicate with the client. In comparison with the BPEL process of the scenario of Figure 16, the BPEL process of Figure 30 has a receive activity inserted before each invoke activity. After each invoke activity an assign and a reply activity has been inserted. If the BPEL process is invoked, the process will wait at the receive activity for user input and send the results to the user at a reply activity.

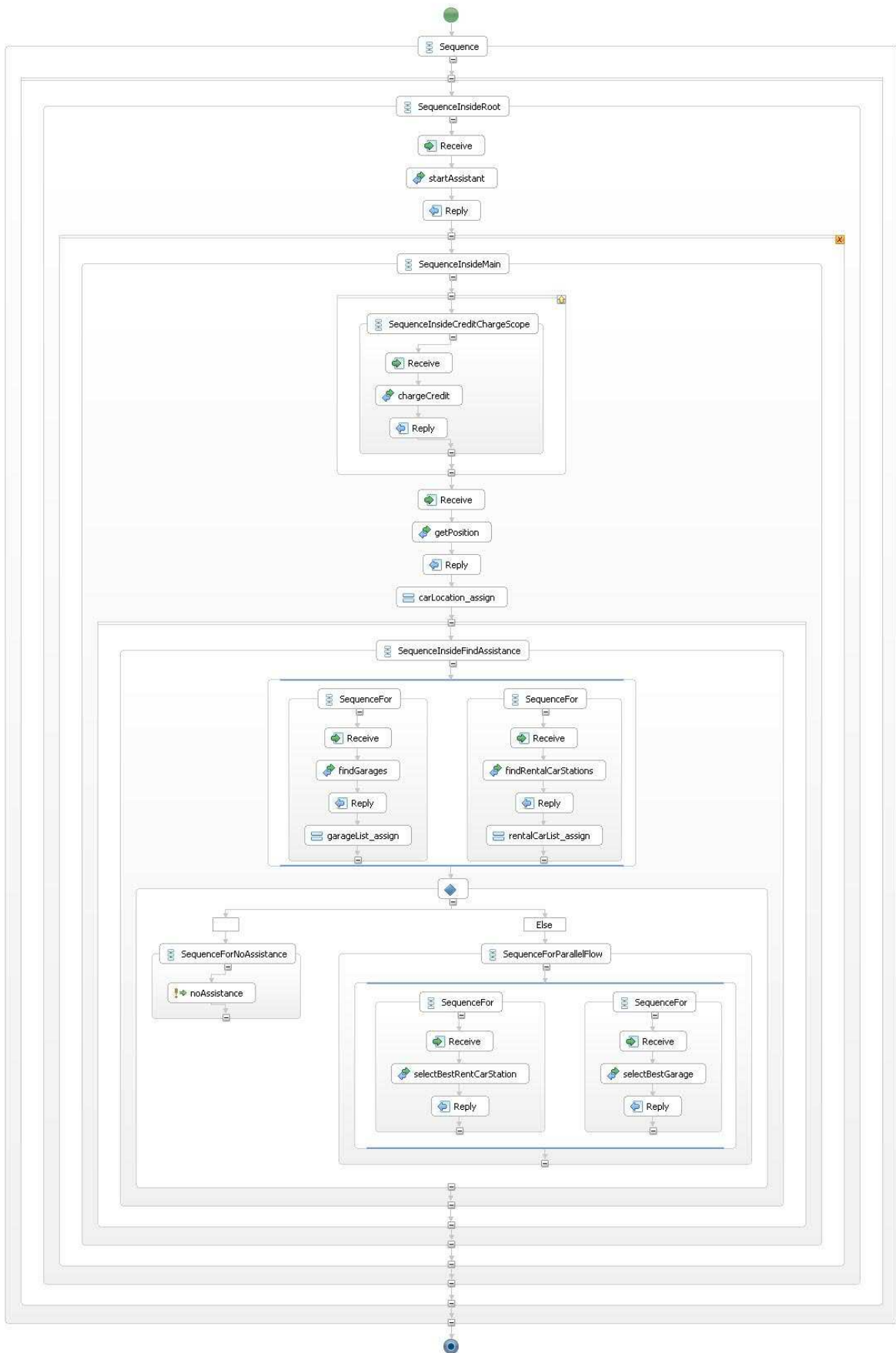


Figure 32: Interactive BPEL process of scenario of Figure 30

If the interactive BPEL process is successfully deployed to a web server, the server application can be started through the web server. The user will fill-in the form and press “Start Service” in Figure 33 to start service request in the web browser (e.g. Firefox or Internet Explore).

Figure 33: Start page

According to the workflow of orchestration in the scenario of Figure 30 the service “ChargeCredit” is started in the first step. The user must fill-in the credit card information to guarantee the payment as shown in Figure 34 before using other services.

Figure 34: Payment with credit card

If the credit card is accepted, the Figure 35 will be shown. According to the workflow of orchestration in scenario of Figure 30 the next step is to find the current location of the car.



Figure 35: Result of payment

In this step the current location of the car is found by `getPositionService` and shown in the map of Figure 36. According to the workflow of orchestration of the scenario of Figure 30 the next step is to find garages and rental car stations in parallel.

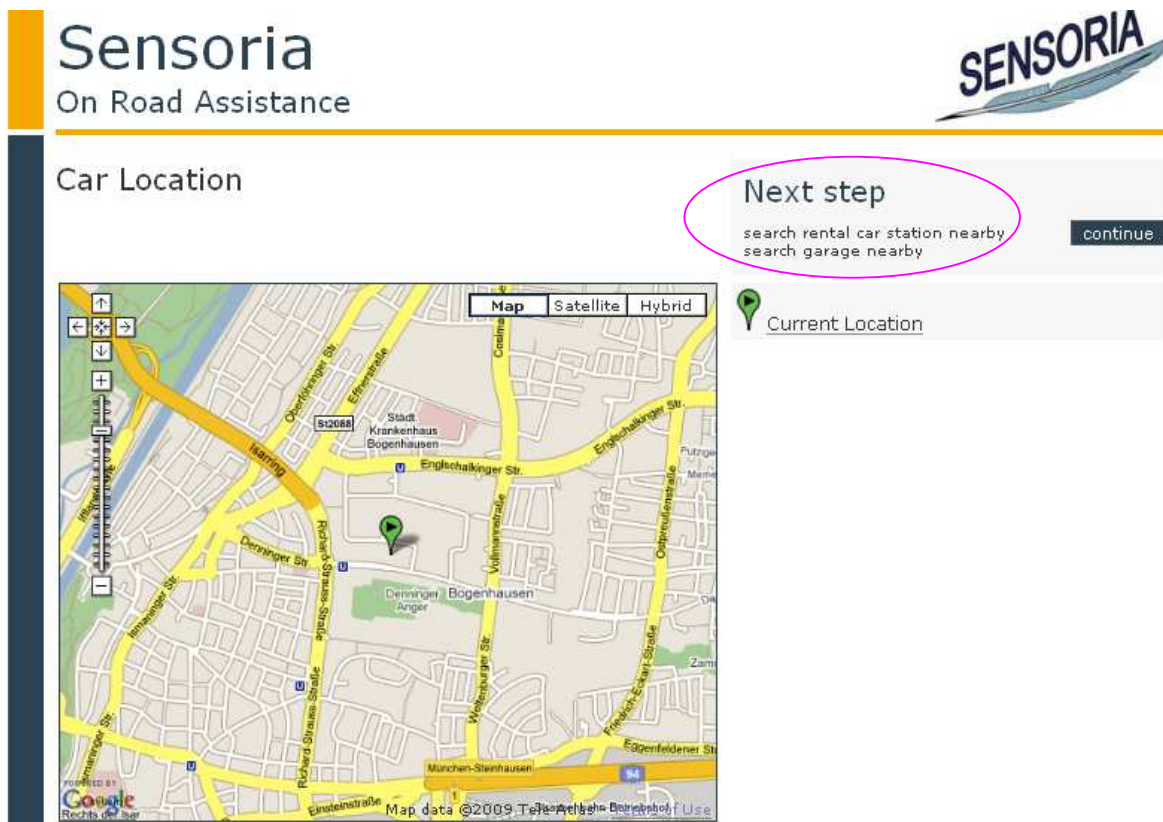



Figure 36: Location service

In this step the `FindGarageService` and `FindRentalcarService` is invoked in parallel in the BPEL process. The results of both services are shown in Figure 37. According to the workflow of orchestration in the scenario of Figure 30 the next step is to find the best garage and the best rental car station in parallel.




# Sensoria

On Road Assistance



Garage nearby your car  
Rental car station nearby your car

**Next step**  
search best garage  
search best rental car station continue



**Current Location**

**Garage nearby your car**

Garage Denninger	get route
Garage Neckar	get route
Garage Riedenburger	get route
Garage Zaubzer	get route

**Rental car station nearby your car**


Car Rental Gotthelf	get route
Car Rental Steinhauser	get route
Car Rental Eva	get route
Car Rental Ina	get route

Figure 37: Find service providers

In this step the FindBestGarageService and FindBestRentalcarService is invoked in parallel in the BPEL process. The results of both services are shown in Figure 38. This is the last step of the scenario of Figure 30 according to the workflow of orchestration.


# Sensoria

On Road Assistance



The best Garage  
The best rental car station

**Next step**  
start new service continue



**Current Location**

**The best garage**

Garage Denninger	get route
------------------	-----------

**The best rental car station**

Car Rental Gotthelf	get route
---------------------	-----------

Figure 38: Find best provider

## 6 Lessons Learned and Future Work

The Automotive Demonstrator implemented so far is a simplified version of the *On Road Assistance* scenario of the *Automotive Case Study*. However, it is used to demonstrate the SENSORIA model-driven approach, which makes use of models built with the UML4SOA extension for service-oriented software and a set of model-transformations defined within the context of the SENSORIA project which allows automatic generation of web service applications.

Many extensions and improvements of this Automotive Demonstrator are possible. On the one hand, extending the functionality of the Demonstrator with additional features of the automotive scenario since the current version is the result of the first step an incremental and iterative development process. On the other hand, improving the development with early analysis, for which different SENSORIA analysis tools that are already integrated in the SDE, can be used, such as WS-Engineer or PEPA.

## 7 References

### 7.1 Documents

- [BK07] Dominik Bernd and Nora Koch. SENSORIA Automotive Scenario: Illustrating Service Specification. pp. 26. Technical Report no. 2. FAST. 2007.
- [D1.4a] Nora Koch, Philip Mayer, Reiko Heckel, László Gönczy and Carlo Montangero. D1.4a: UML for Service-Oriented Systems. Deliverable SENSORIA project.
- [DINO07a] Arun Mukhija, Andrew Dingwall-Smith, and David Rosenblum. DINO - Dynamic and Adaptive Composition of Autonomous Services. Technical report, Department of Computer Science, University College London, London, Januar 2007. White paper on the Dino Approach <http://www.cs.ucl.ac.uk/staff/a.mukhija/dino/DinoWhitePaper.pdf>
- [DINO07b] Arun Mukhija, Andrew Dingwall-Smith, and David Rosenblum. QoS-Aware Service Composition in Dino. In Proceedings of the 5th European Conference on Web Services (ECOWS 2007), pages 3–12, Halle, Germany, November 2007, <http://www.cs.ucl.ac.uk/staff/A.Mukhija/papers/ecows07.pdf>
- [Koch07] Nora Koch. SENSORIA Automotive Case Study: UML Specification of On Road Assistance Scenario. pp. 19. Technical Report no. 1. FAST. 2007.
- [MSK08a] Philip Mayer, Andreas Schroeder and Nora Koch. A Model-Driven Approach to Service Orchestration. *Proceedings of the 2008 IEEE International Conference on Services Computing (SCC 2008)*. 533-536. Vol. 2. IEEE Computer Society. 2008.
- [MSK08b] Philip Mayer, Andreas Schroeder and Nora Koch. MDD4SOA: Model-Driven Service Orchestration. *Proceedings of The 12th IEEE International EDOC Conference (EDOC 2008)*. 203-212. IEEE Computer Society. 2008.
- [REST08] Pautasso, Cesare; Zimmermann, Olaf; Leymann, Frank (2008-04), "RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision", *Proceedings of 17th International World Wide Web Conference (WWW2008)* (Beijing, China) <http://www.jopera.org/docs/publications/2008/restws>
- [WSDL07] Roberto Chinnici, Hugo Haas, Web Services Description Language (WSDL) Version 2.0, W3C Recommendation 26 June 2007, <http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626/>.
- [DINO09] Michel Alessandrini. Finance Case Study: DINO Integration and Intelligent Extension, S&N, Feb. 28th 2009. [http://www.pst.ifi.lmu.de:8080/FinanceCaseStudy/DOWNLOAD/20090225\\_dino-integration.pdf](http://www.pst.ifi.lmu.de:8080/FinanceCaseStudy/DOWNLOAD/20090225_dino-integration.pdf)

### 7.2 Tools

- Demonstrator source [http://www.sensoria-ist.eu/cirquent/automotive\\_demonstrator/on\\_road\\_assistance.zip](http://www.sensoria-ist.eu/cirquent/automotive_demonstrator/on_road_assistance.zip)
- MDD4SOA\_Extention source [http://www.sensoria-ist.eu/cirquent/MDD4SOA\\_Extention.zip](http://www.sensoria-ist.eu/cirquent/MDD4SOA_Extention.zip)
- Eclipse Update Site of MDD4SOA\_Extension [http://www.sensoria-ist.eu/cirquent/eclipse\\_plugin\\_update/](http://www.sensoria-ist.eu/cirquent/eclipse_plugin_update/)
- SDE website <http://svn.pst.ifi.lmu.de/trac/sct>
- Eclipse update site of SDE <http://svn.pst.ifi.lmu.de/update/sct>

---

SDE 4.0 tutorial	<a href="http://svn.pst.ifi.lmu.de/trac/sct/wiki/Tutorial">http://svn.pst.ifi.lmu.de/trac/sct/wiki/Tutorial</a>
MDD4SOA website	<a href="http://www.mdd4soa.eu/web/">http://www.mdd4soa.eu/web/</a>
Eclipse update site of MDD4SOA	<a href="http://www.mdd4soa.eu/update">http://www.mdd4soa.eu/update</a>
Dino web page:	<a href="http://www.cs.ucl.ac.uk/staff/a.mukhija/dino">http://www.cs.ucl.ac.uk/staff/a.mukhija/dino</a>
Java JDK	<a href="http://java.sun.com/javase/downloads/index.jsp">http://java.sun.com/javase/downloads/index.jsp</a>
Eclipse	<a href="http://www.eclipse.org/downloads/">http://www.eclipse.org/downloads/</a>
Tomcat 5.5	<a href="http://tomcat.apache.org/download-55.cgi">http://tomcat.apache.org/download-55.cgi</a>
XAMPP	<a href="http://www.apachefriends.org/en/xampp.html">http://www.apachefriends.org/en/xampp.html</a>
Magicdraw	<a href="http://www.magicdraw.com/">http://www.magicdraw.com/</a>
Google Maps API	<a href="http://code.google.com/apis/maps/">http://code.google.com/apis/maps/</a>
ActiveBPEL	<a href="http://www.activevos.com/community-open-source-engine-download.php">http://www.activevos.com/community-open-source-engine-download.php</a>
Tutorial of Deploying a BPEL Process	<a href="http://www.activebpel.org/samples/samples-2/deploy/doc/index.html">http://www.activebpel.org/samples/samples-2/deploy/doc/index.html</a>
Tutorial for Installing ActiveBPEL	<a href="http://users.encs.concordia.ca/~yuhong/teaching/UNB/CS6905/Task0_GettingUpandRunning-LITEVERSION.doc">http://users.encs.concordia.ca/~yuhong/teaching/UNB/CS6905/Task0_GettingUpandRunning-LITEVERSION.doc</a>

## Appendix

### A.1 Configuration of MDD4SOA\_Extension plug-in projects

#### A.1.1 Configuration of Plug-in Project

MDD4SOA\_Extension is an Eclipse plug-in. So an Eclipse Plug-in project should be created. At first open the plug-in .xml file in the perspective (plug-in development) in eclipse. Then open the tab (Dependencies) and add all other depended eclipse plug-ins like in Figure 39.

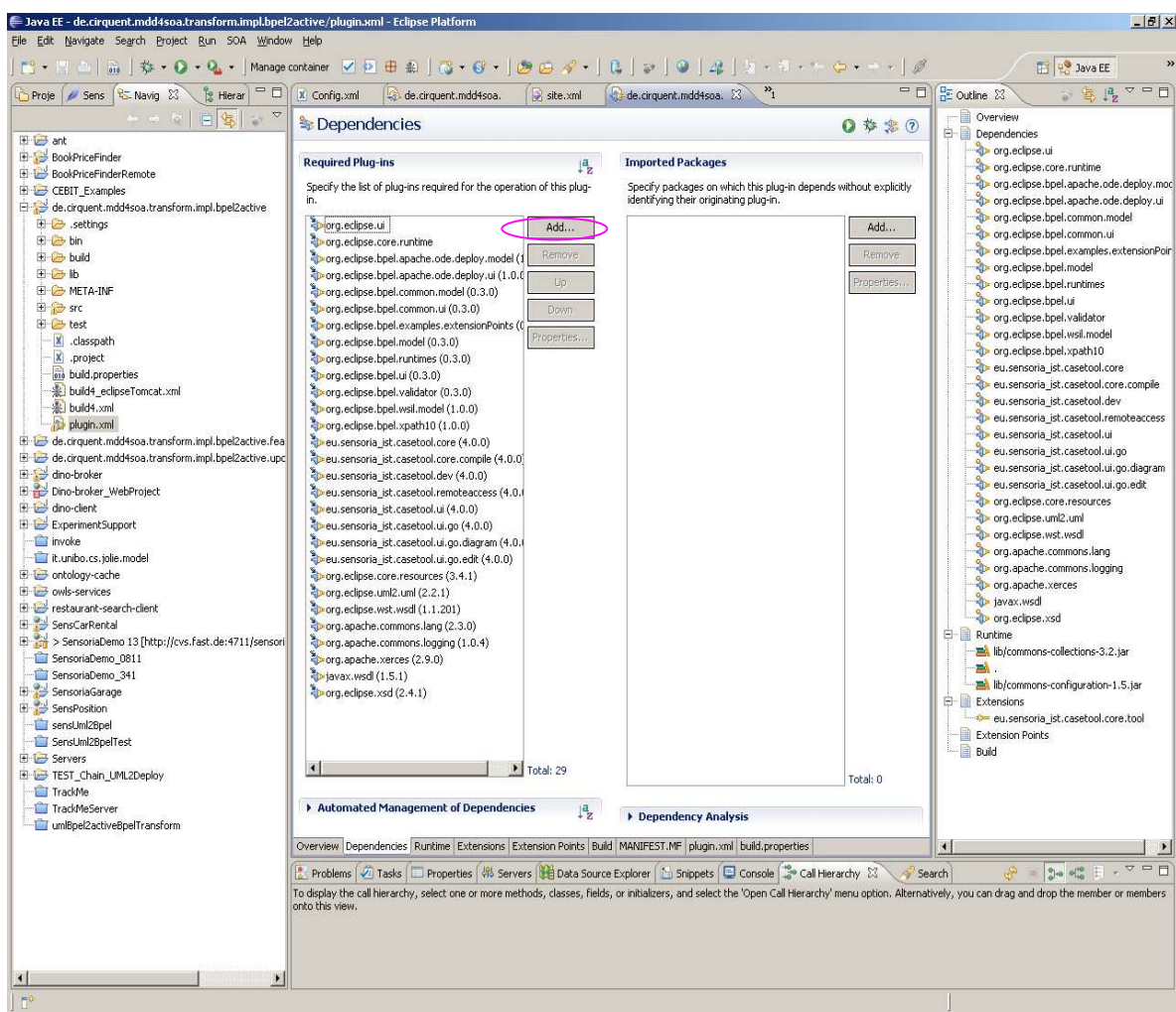


Figure 39: Configure dependencies



Click the tab (Runtime) and specify the libraries and folders that constitute the plug-in classpath as shown in Figure 40. If it is unspecified, the classes and resources are assumed to be at the root of the plug-in.

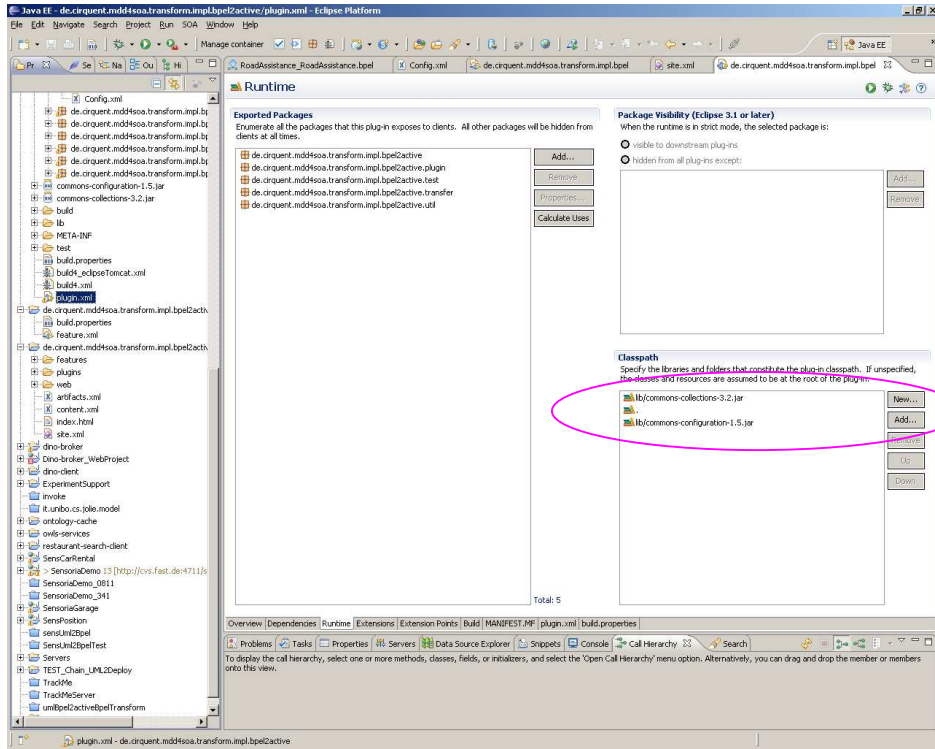


Figure 40: Specify the libraries and folders that constitute the plug-in classpath

Click the tab (Build) and select the folders and files to include in the binary build. Select the folders and files to include in the binary build as shown in Figure 41.

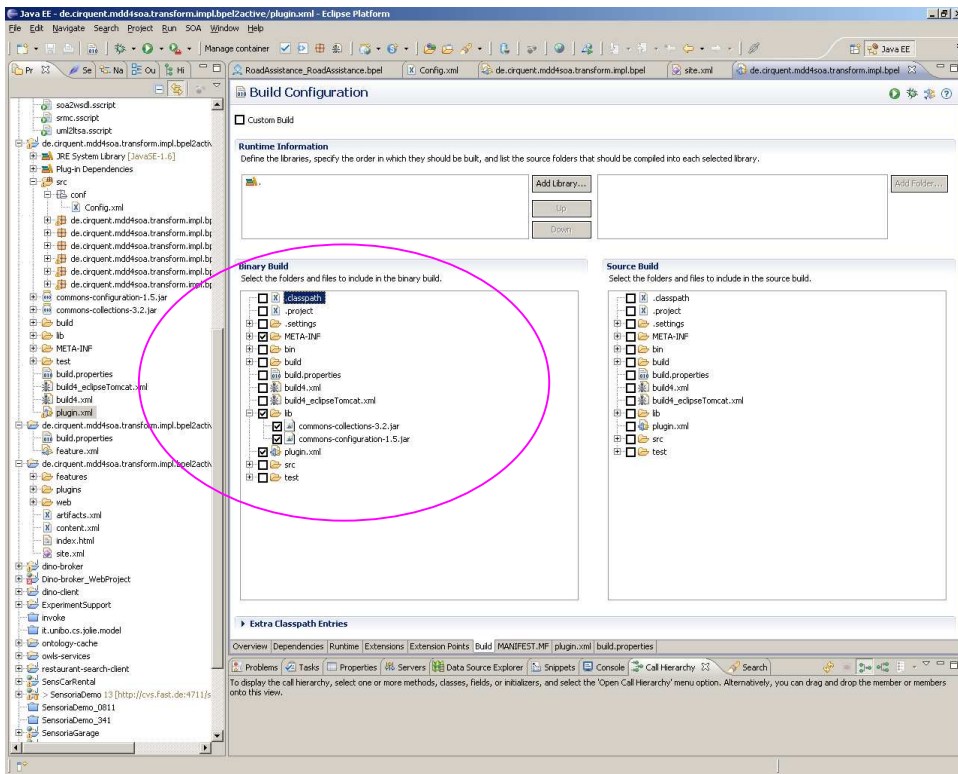


Figure 41: Select folders for build

To test and debug the plug-in, you can click the tab (Overview), then test this plug-in by launching a separate Eclipse application. For debug you can launch an Eclipse application in Debug mode as shown in Figure 42.

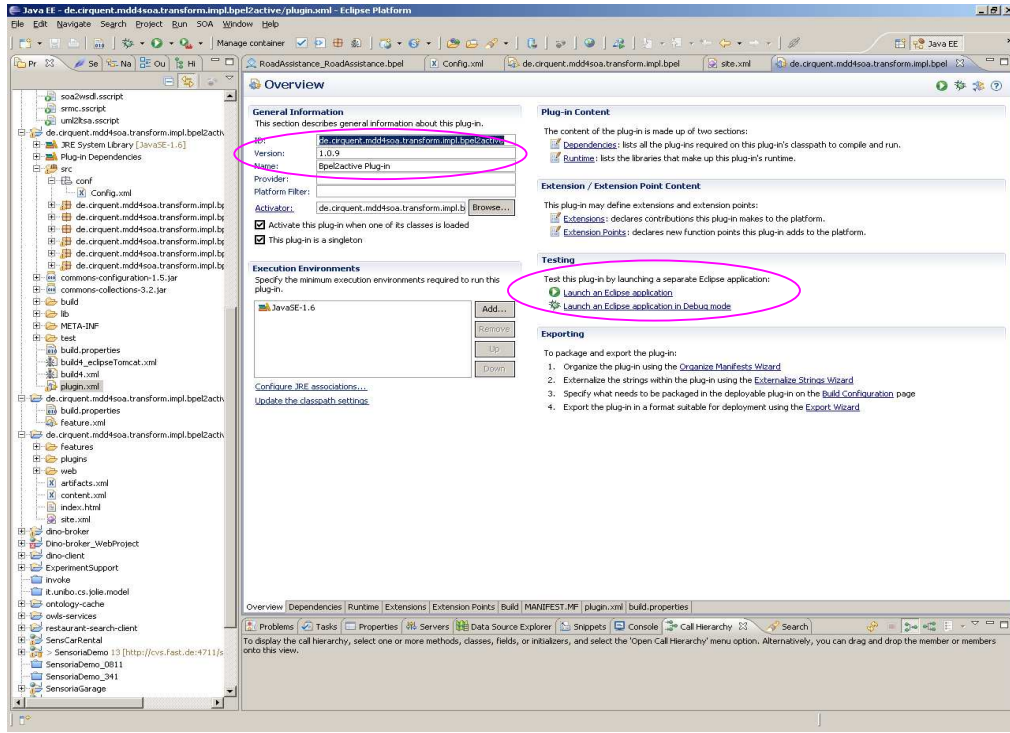


Figure 42: Testing and debug

### A.1.2 Configuration of feature project

Create a feature project for above plug-in project. At first open the feature.xml file in the perspective (Plug-in development) in eclipse. Then open the tab (Overview) and set the same version as the plug-in project as shown in Figure 43.

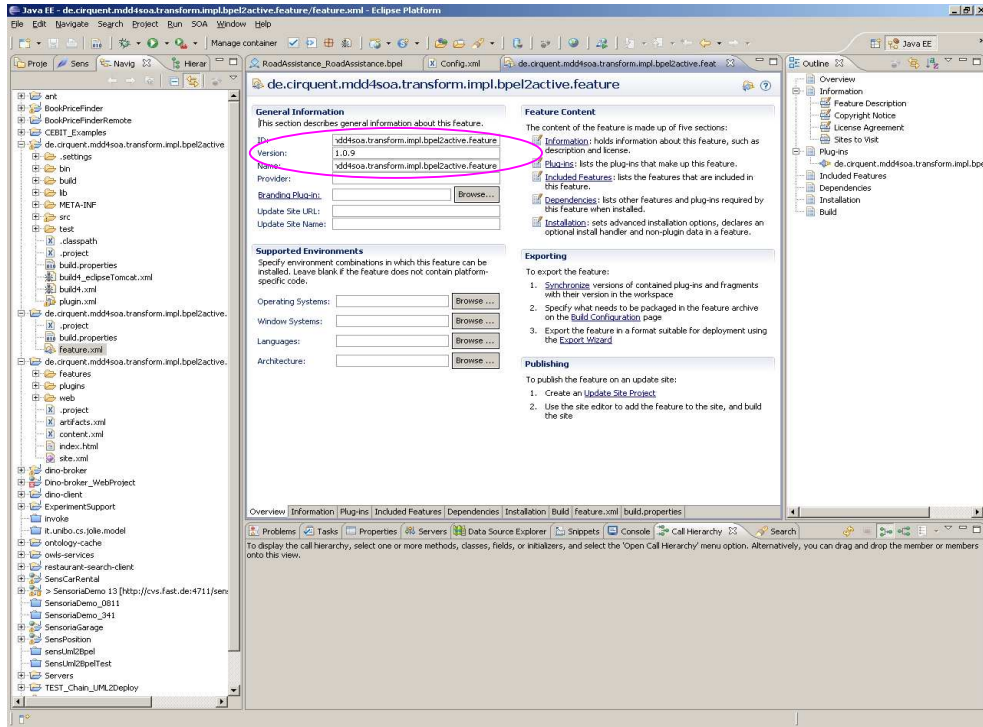


Figure 43: Feature project

### A.1.3 Configuration of Update Site Project

Create an update site project for above feature project as shown in Figure 44.

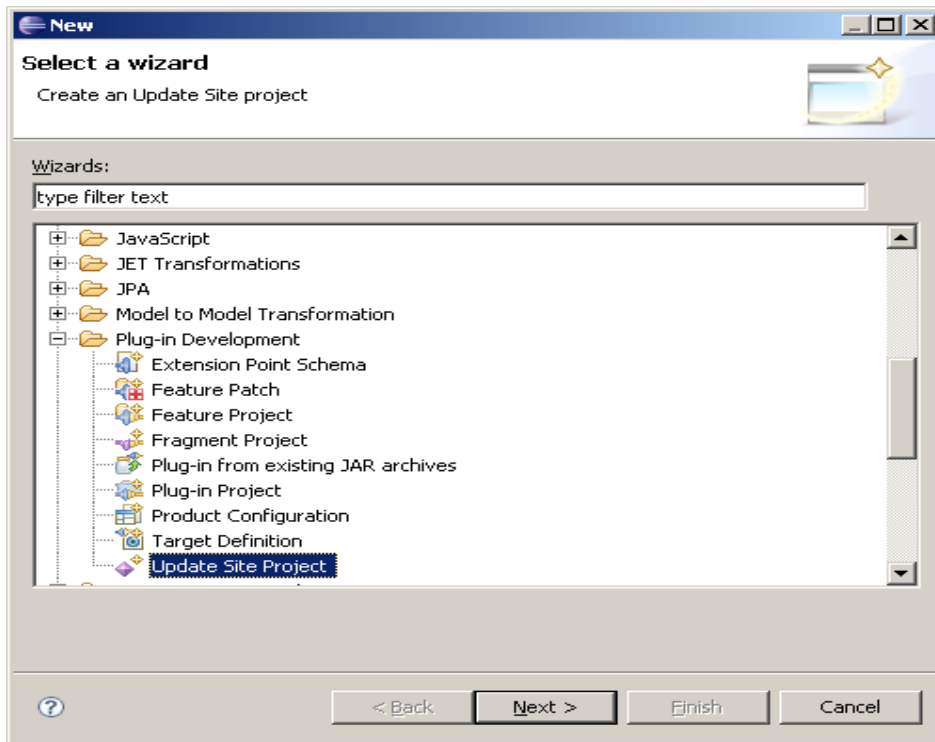
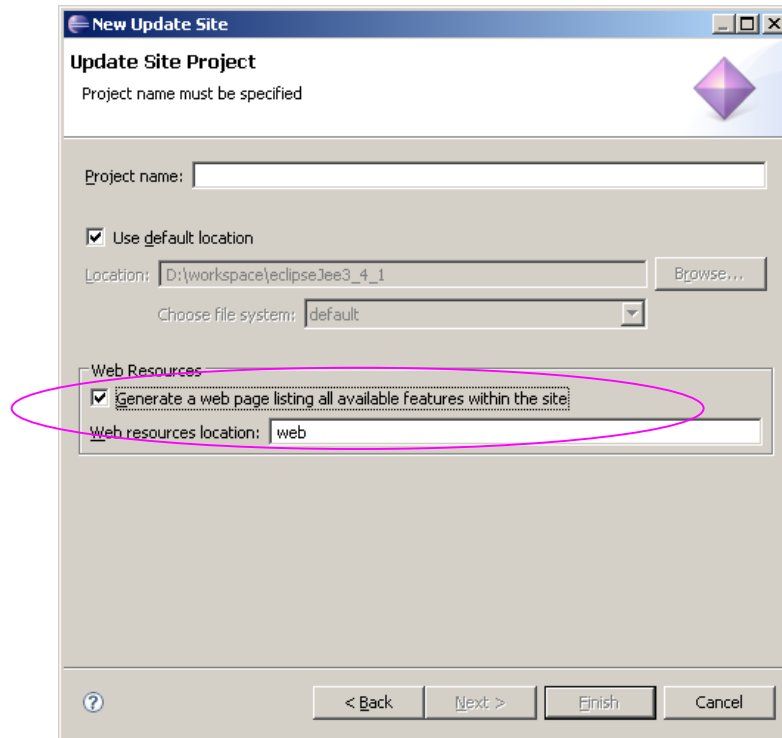


Figure 44: Create update site project

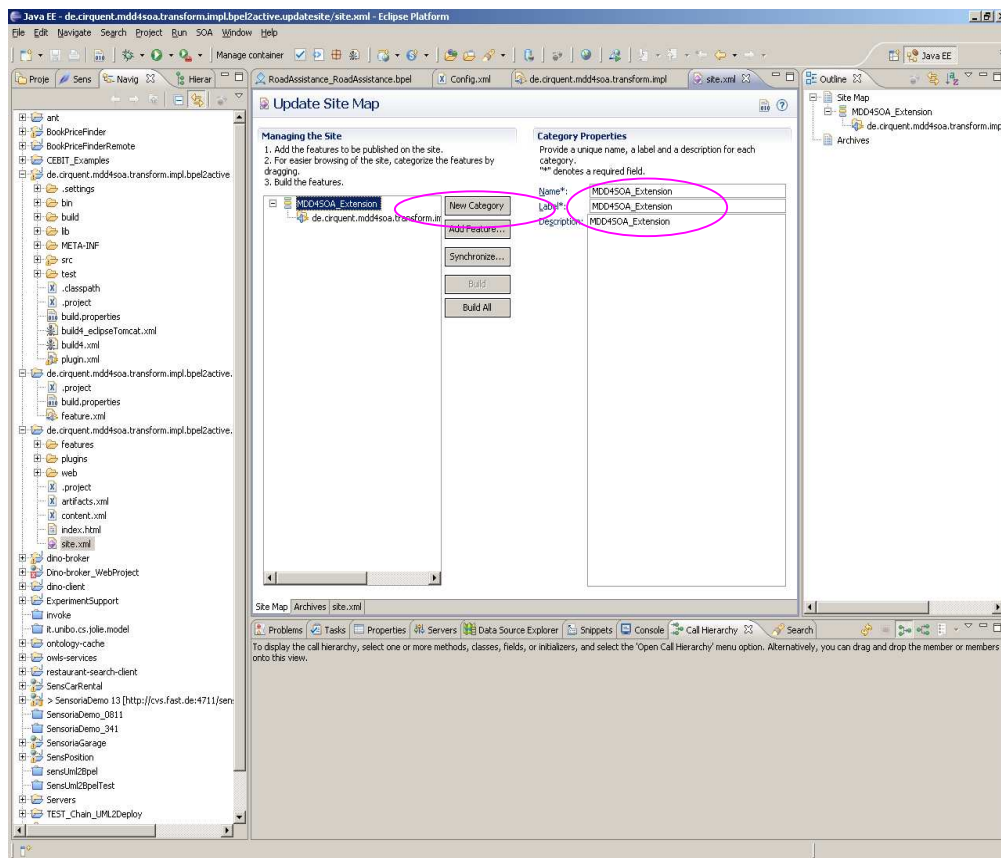
Select “Generate a web page listing all available features within the site” as shown in Figure 45.





**Figure 45: Create update site project with web resources**

At first open the site.xml file in the perspective (plug-in development) in eclipse. Then open the tab (Overview) and set the same version as the plug-in project as shown in Figure 46.



**Figure 46: Update Site Map**

If the version of plug-in project change, the version of feature and update site project must also be changed.

## A.2 Deploy a Web Service with Eclipse Web Developer Tools

You can publish your Java class as a web service with the Eclipse Web Developer Tools. At first the Java class file should be selected. Then right-click the mouse and select “Create Web Service” of “Web Services”.

Java class right click → Web Services → create web service as shown in Figure 47:

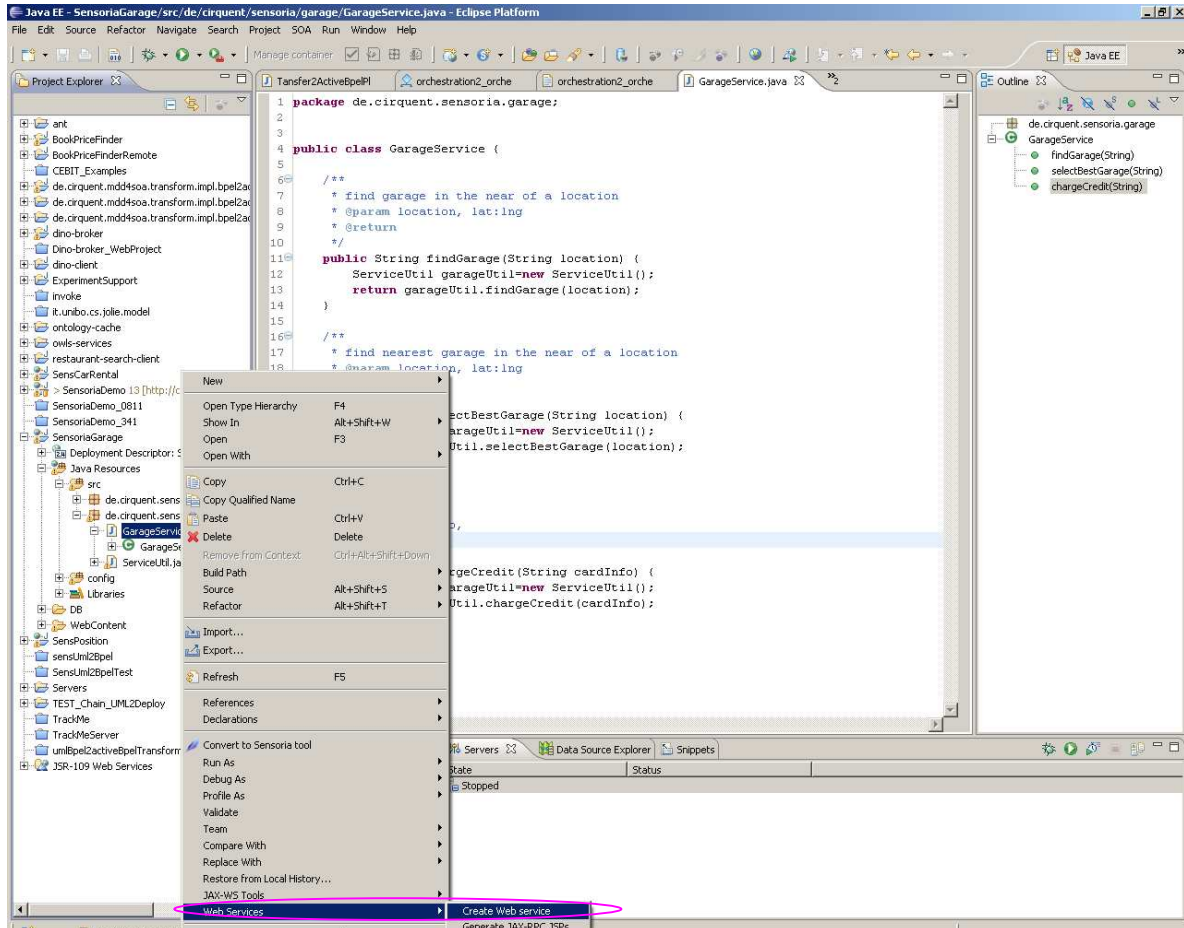


Figure 47: Create web service

After the click a wizard will be opened as following screenshot. Then you can set the configuration for the web service. The web server could be specified. In this demonstration Tomcat v5.5 server should be selected (Server: Tomcat v5.5 server). After that Eclipse will deploy the web service automatically to Tomcat v5.5 server.

The checkbox of “Publish the Web service” should also be selected like below as shown in Figure 48.

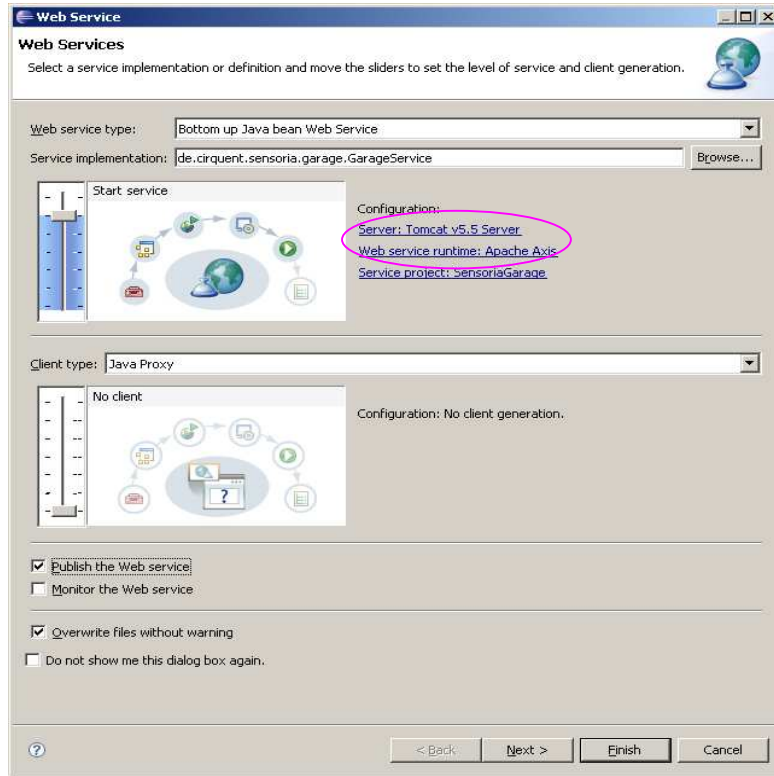


Figure 48: Publish web service

After the configuration you should click “next” button to continue the next wizard.

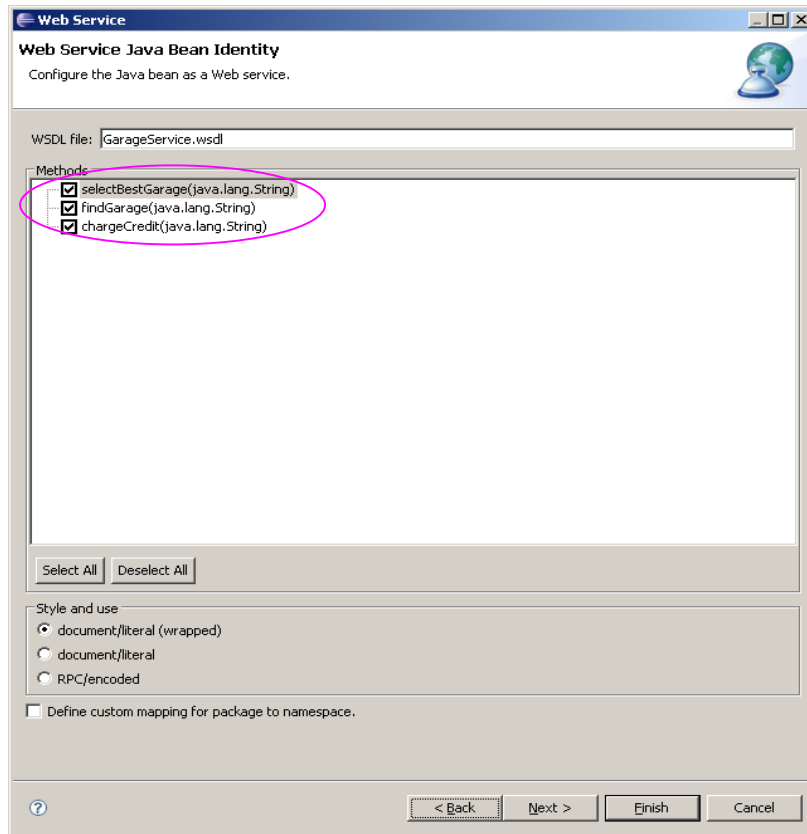


Figure 49: Select methods of web service

Select to deployed methods, then click “finish” button to finish the publication wizard of web service as shown in Figure 49.

Select Use Tomcat installation as shown in Figure 50 to deploy Web Projects to Tomcat in Eclipse.

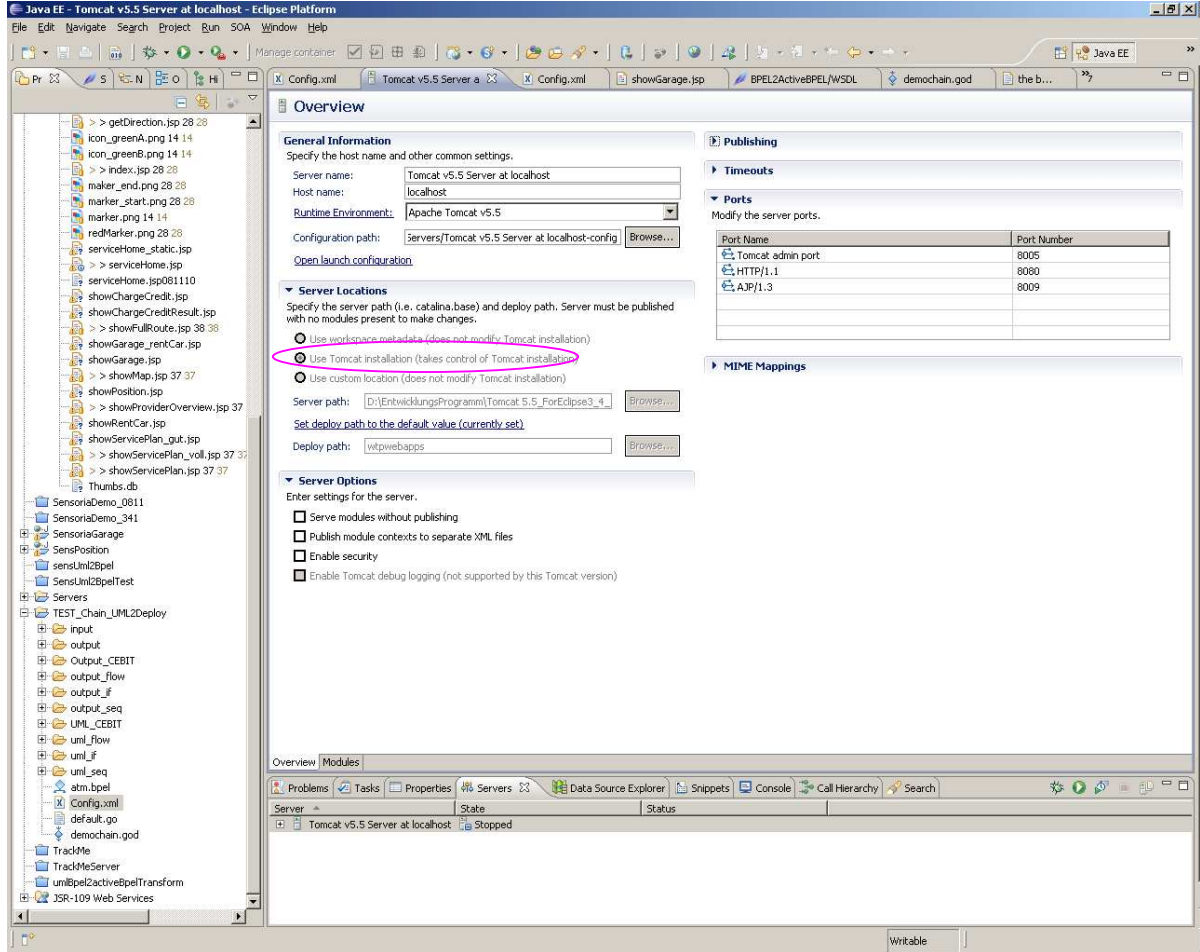


Figure 50: Server location