WSAMI Middleware User Guide

Author: Daniele Sacchetti – daniele.sacchetti@inria.fr

Table of contents

1.Introduction

The objective of the WSAMI Distributed Middleware Infrastructure is to extend the Web Services architecture (WSA) towards mobile distributed systems, addressing in particular:

- dynamic service composition
- service provisioning over resource-constrained mobile nodes with unstable network connectivity

The WSA's standards on which WSAMI Middleware is based are:

- **WSDL (Web Services Description Language)** that is a language based on XML that is proposed by the W3C for describing the interfaces of Web Services
- **SOAP (Simple Object Access Protocol)** that defines a lightweight protocol for information exchange

WSA is further conveniently complemented by **UDDI (Universal Description, Discovery and Integration)** that is a specification of a registry for dynamically locating and advertising Web Services.

The WSAMI Distributed Middleware Infrastructure provides a platform based on WSAMI language and some middleware-related services providing essential functionalities to allow the users to develop and run their own application-related services on the top of the WSAMI Middleware.

We introduce the WSAMI (Web Services for AMbient Intelligence) declarative language, based on WSDL, for the specification of both application- and middleware-related services (see section 6 for details about WSAMI specification). Given the WSAMI specification of a service, an instance is automatically composed upon a user request, according to the services (instances) that may be retrieved from the environment, regarding in particular network connectivity.

The WSAMI Middleware is based on the following middleware-related services:

Core middleware infrastructure

offering a Core Broker and the WSAMI language for the declarative specification of services, for enabling interaction among services distributed over heterogeneous terminals (see 5.1 for details about Core Broker).

Service for naming, discovery and lookup (ND Service)

permitting the retrieval of a service both in the local and in the wide area, according to the service specification (in terms of either a name or a declarative specification) and to the environment in which the service is requested (see 5.2 for details about Naming and Discovery Service).

We introduce some further middleware-related services:

Universal Repository (UR) Service

as in WSA specification, this service is used as a registry for dynamically locating and advertising application-related services, but in our implementation instead of the UDDI specification, we have adapted this service to WSAMI Middleware spefic requirements (see 5.3 for details about UR Service).

Gateway Service

permitting a multihomed station connected both to an infrastructure-based network and to a wireless ad-hoc network to enable the retrieval of services on the two networks through the ND Service.

It further allows all the stations on the wireless ad-hoc network to be connected to the infrastructure-based network and to have an interaction with services running outside the ad-hoc network (see 5.4 for details about Gateway Service).

In this document we describe how you can install the WSAMI Middleware in section 2 (see section 2.1 for installation on a PC (in this guide, the term PC refers to an Intel x86 architecture) and section 2.2 for installation on a resource-constrained mobile device such as a Personal Digital Assistant (PDA: in this guide, the term PDA refers to a Strong-ARM architecture), how to develop and deploy your application-related services using WSAMI Tools in see section 3, we provide a reference guide to WSAMI Tools in section 4, some details about the internals of the main components of the WSAMI Middleware in section 5 and finally the specification of our language WSAMI in section 6.

2.Installation

2.1. Installation on PC station

The WSAMI Middleware version for PC is based on Axis SOAP implementation and all the libraries implementing Axis version 1.1 have been included in the distribution provided with WSAMI Middleware.

2.1.1 System Requirements

- Operating System: Linux and Windows with Cygwin
- Java 2 Standard Edition 1.3.1 (or higher)
- Jakarta Tomcat version 4.x
- WSAMI Middleware

2.1.2 Installation

The first step is the installation of J2SE and Jakarta Tomcat. The directory where Tomcat is installed will be referenced as `$TOMCATDIR`.

The second step is to extract the `wsami-src-doc-1.0.0.tar.gz` in a directory that will be referenced as `$WSAMIDIR`.

Then, you must copy the content of `$WSAMIDIR/dist/tomcat/webapps` to the directory `$TOMCATDIR/webapps` and the content of `$WSAMIDIR/dist/tomcat/shared/lib` to the directory `$TOMCATDIR/shared/lib`.

Finally you must add the content of `$WSAMIDIR/dist/tomcat/conf/server.xml` to `$TOMCATDIR/conf/server.xml` Tomcat server configuration file in the following position:

```
<Server ... >
  <Service ... >
    <Engine ... >
      <Host ... >
        <Context path="/wsami" docBase="wsami" debug="1"
                        crossContext="true"/>
        <Context path="/services" docBase="services" debug="1"
                         crossContext="true"/>
        <Context path="/examples" ... >
      </Host>s
    </Engine>
  </Service>
</Server>
```

The structure of `$TOMCATDIR` and `$WSAMIDIR` directories will be the following:

```
$WSAMIDIR  +--+ dist +--+ tomcat +--+ bin

$TOMCATDIR +--+ bin
           |
           + conf   --- server.xml
           |
           + shared +--+ lib
           |
           + webapps --+ wsami--+ deploy.xsd
                       |        + WSAMI.xsd
                       |        + install.xsd
                       |        + deploy_install.xsd
                       |        + NDConcrete.wsdl
                       |        + NDInterface.wsdl
                       |        + NDAbstract.wsami
                       |        + NDService.wsami
                       |        + GW ---+ GWAbstract.wsami
                       |        |       + GWService.wsami
                       |        |       + GWInterface.wsdl
                       |        |       + GWConcrete.wsdl
                       |        |       + wsami.xml
                       |        |       + wsamiInstalledServices.xml
                       |        + UR ---+ URAbstract.wsami
                       |        |       + URService.wsami
                       |        |       + URInterface.wsdl
                       |        |       + URConcrete.wsdl
                       |        |       + wsami.xml
                       |        |       + wsamiInstalledServices.xml
                       |        |
```

```
|           + ( wsami.xml )
|           + ( wsamiInstalledServices.xml )
|           |
|           + WEB-INF ---+ web.xml
|                        + web.xml.WIN
|                        + lib
|                        |
|                        + classes
|
+ services-+ WEB-INF ---+ web.xml
                         + lib
                         |
                         + classes
```

Where:

- `$WSAMIDIR/dist/tomcat/bin` directory contains all WSAMI scripts
- `$TOMCATDIR/bin` directory contains all Tomcat scripts
- shared/lib contains jar files implementing the Core Broker and Naming and Discovery Service
- in webapps/wsami directory:
  - all *.xsd are use for XML schema validation
  - NDConcrete.wsdl, NDInterface.wsdl, NDAbstract.wsami, NDService.wsami define the Naming and Discovery Service
  - wsami.xml and wsamiInstalledServices.xml contain information about the services that must be deployed at WSAMI server startup. (These two files are not installed in this installation phase but are automatically generated while the system is running and you deploy services)
  - WEB-INF/web.xml and WEB-INF/web.xml.WIN contain the WSAMI configuration for Linux and for Windows with Cygwin
  - GW and UR directories contain files (wsdl, wsami and xml) that define the Gateway Service (GW) and UR Service (UR) and can be used to add these services to your WSAMI installation (see section 2.1.3).
- in webapps/services directory:
  - WEB-INF/classes contains class files implementing services
  - WEB-INF/web.xml contains WSAMI configuration required by Tomcat

2.1.3 Configuration

The first step in the configuration of your system is to edit the file `$TOMCATDIR/webapps/wsami/WEB-INF/web.xml`. This file contains some parameters that you can modify or remove:

- wsamiBinDir (mandatory): is the directory `$WSAMIDIR/dist/tomcat/bin` (for Windows+cygwin users: see `WEB-INF/web.xml.WIN` as an example of the path to be used for this parameter).
- UR (not mandatory): is the address of the UR service (on your local machine or on a remote host) that must be used as repository by your WSAMI server (for example: http://1.2.3.4:8080/services/UR). If not used you can remove this parameter.

Your standard WSAMI configuration includes WSAMI Core Broker and Naming and Discovery service. If you want to enable UR or Gateway services on your host you must add

the definition of the specific service you want to add to your WSAMI configuration files (`wsami.xml` and `wsamiInstalledServices.xml`).
For example to enable UR service on your host:

- copy the content of `$TOMCATDIR/webapps/wsami/UR/wsami.xml` into `$TOMCATDIR/webapps/wsami/wsami.xml`.
- copy the content of `$TOMCATDIR/webapps/wsami/UR/wsamiInstalledServices.xml` into `$TOMCATDIR/webapps/wsami/wsamiInstalledServices.xml`.
- copy all wsdl and wsami files in `$TOMCATDIR/webapps/wsami/UR` directory to `$TOMCATDIR/webapps/wsami`.

2.1.4 Server startup and shutdown

The first operation to be executed is to run the SLP manager script `$WSAMIDIR/dist/tomcat/bin/SLPMGR.sh`. To execute this command you must be logged as root user.
Then you can start the WSAMI server with the command `$TOMCATDIR/bin/startup.sh`. When the WSAMI server starts up the system reads the file `$TOMCATDIR/webapps/wsami/wsami.xml` to initialize the Core Broker. This file contains the definition of all the services that must be deployed at system startup. The configuration file `$TOMCATDIR/webapps/wsami/WEB-INF/web.xml` contains the location and name of this file. You can change the name of the file used for initialization by modifying the init-param `config` of AdminServlet.
The command to stop the WSAMI server is `$TOMCATDIR/bin/shutdown.sh`. While the server is running, you can see the list of deployed services with a web browser at the following address `http://localhost:8080/wsami/AdminServlet`.

2.2. Installation on PDA

The WSAMI Middleware version for PDA is based on Jetty Web Server and all the libraries implementing Jetty version 4.2.9 (the use of this version of Jetty is mandatory for compatibility both with WSAMI Middleware and J2ME CDC/FP or PP) have been included in the distribution provided with WSAMI Middleware. For the Web Services (SOAP and WSDL) implementation, WSAMI is based on CSoap.

2.2.1 System requirements

- Architecture and Operating System: ARM/Linux. The Familiar Linux distribution for PDA can be found at:

  http://www.handhelds.org

  Sharp Zaurus PDA is provided with Linux:

  http://www.sharpusa.com/zaurus

- Java 2 Micro Edition Connected Device Configuration (CDC) with Foundation Profile (FP) – without graphic interface API:

  http://java.sun.com/j2me/download.html

  or Java 2 Micro Edition Connected Device Configuration (CDC) with Personal Profile (PP) - with graphic interface API:

  http://java.sun.com/products/personalprofile/downloads/ea/index.html

  http://www.sharpusa.com/zaurus

2.2.2 Installation and configuration

The first step is to download the WSAMI Middleware and uncompress `wsami-src-doc-1.0.0.tar.gz`. Then you can copy the directory `wsami-x.y.z/dist/pda` to the PDA. The directory on the PDA where you have installed WSAMI will be referenced as `$WSAMIDIR/dist/pda`.
In `$WSAMIDIR/dist/pda/bin/setvars.sh` the JVM used by WSAMI is set to `/opt/QtPalmtop/j2me/bin/cvm`. If you want to use a different VM you must modify the value of `CVM` variable.
If you want to enable the use of a UR service from the WSAMI server on your host you must set the value of `UR` variable in `$WSAMIDIR/dist/pda/bin/runcvm.sh` to the location of the UR service (see also section 2.1.3).

2.2.3 Server startup and shutdown

The first operation to be executed is to run the SLP manager script `$WSAMIDIR/dist/pda/bin/SLPMGR.sh`. To execute this command you must be logged as root user.
Then you can start the WSAMI server with the command `$WSAMIDIR/dist/pda/bin/runcvm.sh`.
When the WSAMI server starts up the system reads the file `$WSAMIDIR/dist/pda/wsami/wsami.xml` to initialize the Core Broker. This file contains the definition of all the services that must be deployed at system startup.
The command to stop the WSAMI server is `$WSAMIDIR/dist/pda/bin/runcvm.sh stop`.
While the server is running, you can see the list of deployed services with a web browser at the following address `http://localhost:8080/wsami/AdminServlet`.

3.Tutorial - developing and deploying a service and a client

We present now the process of development and deployment of a service and the related client. Some examples may be found in the `$WSAMIDIR/samples` and `$WSAMIDIR/services` directory. In this section we make use of the tools provided by WSAMI Middleware to

generate and install all the files that are required to develop and deploy a service on the server. These tools may be found in `$WSAMIDIR/dist/tomcat/bin` if you are working on the PC version and in `$WSAMIDIR/dist/pda/bin` if you are working on the PDA version and are described in more detail in section 4.

In the examples provided in `$WSAMIDIR/samples` and `$WSAMIDIR/services` we also make use of the ant utility and we provide a `build.xml` file (based on the same WSAMI tools used in this section) for each example that can be used to compile and deploy the service and client examples.
If you want to make use of the ant tool and of our framework, you must adapt your service directory to that proposed by our examples that can be summarized by the following schema:

```
$SERVICEDIR +--- build.xml
            |
            +--+ src --+ $SERVICEService --+ $SERVICEImpl.java
            |          + $SERVICEClient  --+ $SERVICEClient.java
            |
            +--+ lib
            |
            +--+ build
            |
            +--+ wsdl
            |
            +--+ wsami
```

Where:

- The `wsdl` directory contains the wsdl files defining the service
- The `src/$SERVICEService/$SERVICEImpl.java` is the file implementing the service and the `src/$$SERVICEClient/$SERVICEClient.java` is the file implementing the client
- The `lib` directory contains the jar files that must be used to compile and/or deploy the service
- The `wsami` directory contains the wsami and deployment files generated during the compilation task
- The `build` directory contains the class and jar files implementing the service that are generated during the compilation task
- The `build.xml` file is the file used by ant to execute the different tasks. This file is only used to set up the parameters related to the specific service, while all the tasks' definitions are contained in `$WSAMIDIR/services/commonLinux.xml` (for Linux) and in `$WSAMIDIR/services/commonWindows.xml` (for Windows+Cygwin).
  The developer must include the right xml file into the build.xml and set the parameters related to the different tasks (the tasks and the associated parameters are detailed in the following sections) paying attention to the paths of the files that must be compliant with the target system he is working on.

3.1. WSDL interface and concrete files

Every service is defined by two WSDL files: one for the service interface and the other for the definition of the service location.

In the *Example1* example these two files are `Example1Interface.wsdl` and `Example1Concrete.wsdl`.

To develop a new service *Agenda*, you must create two new files `AgendaInterface.wsdl` and `AgendaConcrete.wsdl` starting from *Example1* wsdl files and replace all the occurences of *Example1* with *Agenda* both in `AgendaInterface.wsdl` and `AgendaConcrete.wsdl`. In `AgendaConcrete.wsdl` you must:

- modify the address `localhost:8080` with the address and port of your host where your Tomcat server is running and you must specify the same value for the service name in `<soap:address location="...">` and in `<port name="...">`. For example if you specify:

  ```
  <soap:address location="http://myhost:80/wsami/services/Agenda">
  ```

  the other tag must be:

  ```
  <port name="Agenda">
  ```

  The value in `<service name="...">` and in `<port name="...">` must be different. If they have the same value, some errors will raise during service compilation.

- modify the import file name in `AgendaConcrete.wsdl` with `AgendaInterface.wsdl`

In `AgendaInterface.wsdl` you must add the definition of the operations you want to implement in Agenda service. We refer to the directory where these two files are stored as `$AGENDADIR`. See 4.2 for restrictions on WSDL language for service interface definition.

3.2. Service development

Now you must generate the skeleton and all the other files that are required to develop your service. To accomplish this step you need the two wsdl files created in section 3.1. The command is the following:

```
compileService.sh AgendaConcrete.wsdl
```

This command will generate some java files in the directory `$AGENDADIR/AgendaService`. To implement your *Agenda* service you must now implement a class named for example `AgendaService.AgendaImpl`. As an example to write this class, you can take the class `$AGENDADIR/AgendaService/AgendaSOAPBindingImpl.java` (generated by the above command `compileService.sh`).
You must add the java code that implements your service methods. When you finish writing your service you must compile it to generate the .class files, with the following command:

```
compileService.sh AgendaConcrete.wsdl . AgendaService
```

Where the first parameter is the name of the wsdl concrete file, the second is the classpath to add to the default WSAMI classpath in order to compile the service and the last parameter is the directory containing java files that implement the service. All the .class files will be generated in the current directory.

Compile with ant

```
ant compileService
ant compileServicePDA
```

The .class and .jar files will be created in `$AGENDADIR/build/buildService` and the .wsami and .xml files will be created in `$AGENDADIR/wsami`.
If the compilation of the service requires some extra libraries you must set the colon separated list of jar files and directory to be added to the compilation classpath in variable `service.extra.compileclasspath` in file `$AGENDADIR/build.xml`. The path of each file must be an absolute path (you can use the `${basedir}` variable to refer to the service directory).

3.3. WSAMI and deploy/undeploy files

Now you must generate all the XML/WSAMI files required to describe and deploy the service on the WSAMI server using the command `WSDL2WSAMI.sh`. For the `Agenda` service the command is the following:

```
WSDL2WSAMI.sh Agenda
```

And the default values used to generate the WSAMI and xml files are the following:

```
outputDir: local directory
packageName: AgendaService
java className implementing the service: AgendaService.AgendaImpl
type: javaclass
deployXML filename: AgendaDeploy.xml
undeployXML filename: AgendaunDeploy.xml
scope: Application

WAD filename: AgendaAbstract.wsami
WSD filename: AgendaService.wsami
WAU address: http://localhost:8080/wsami/AgendaAbstract.wsami
WSU address: http://localhost:8080/wsami/AgendaService.wsami
WSDLinterface filename: AgendaInterface.wsdl
WSDLconcrete filename: AgendaConcrete.wsdl
HTTPBaseAddr: http://localhost:8080/wsami/
URI of WSDLInterface: http://localhost:8080/wsami/AgendaInterface.wsdl

WCD: not defined
hasCustomizer: false
WCU: not defined
isCustomizer: not defined
instanceURL: not defined
```

And the generated files are the following:
AgendaAbstract.wsami

```
<?xml version="1.0" encoding="UTF-8"?>
<wsami xmlns="http://www-rocq.inria.fr/arles/wsami"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www-rocq.inria.fr/arles/wsami WSAMI.xsd">
   <Definition name="Agenda" targetNamespace="http://localhost:8080/wsami/">
     <Abstract name="Agenda">
       <Interface hrefSchema="http://localhost:8080/wsami/AgendaInterface.wsdl"/>
     </Abstract>
   </Definition>
</wsami>
```

AgendaService.wsami

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<wsami xmlns="http://www-rocq.inria.fr/arles/wsami"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www-rocq.inria.fr/arles/wsami WSAMI.xsd">
   <Definition name="Agenda" targetNamespace="http://localhost:8080/wsami/">
     <Service name="Agenda">
       <Abstract hrefSchema="http://localhost:8080/wsami/AgendaAbstract.wsami"/>
       <Concrete hrefSchema="http://localhost:8080/wsami/AgendaConcrete.wsdl"/>
     </Service>
   </Definition>
</wsami>
```

AgendaDeploy.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<deploy xmlns="http://www-rocq.inria.fr/arles/wsami"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www-rocq.inria.fr/arles/wsami deploy_install.xsd">
   <service name="Agenda">
      <wad name="AgendaAbstract.wsami"/>
      <wsd name="AgendaService.wsami"/>
      <wau name="http://localhost:8080/wsami/AgendaAbstract.wsami"/>
      <implementation>
         <class name="AgendaService.AgendaImpl"/>
         <type name="javaclass"/>
         <package name="AgendaService"/>
      </implementation>
      <scope name="Application"/>
   </service>
</deploy>
```

AgendaunDeploy.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<undeploy xmlns="http://www-rocq.inria.fr/arles/wsami"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www-rocq.inria.fr/arles/wsami deploy.xsd">
   <service name="Agenda">
      <wau name="http://localhost:8080/wsami/AgendaAbstract.wsami"/>
   </service>
</undeploy>
```

The *Agenda* service will be identified by WSAMI Middleware with the WAU identifier `http://localhost:8080/wsami/AgendaAbstract.wsami`. The clients that want to find an instance of this service will use the method `getService` of Naming and Discovery service (see sections 3.5 and 5.2).
The WAU identifier is the same for all the services implementing the same abstract interface defined by the WSAMI abstract document. The WAU identifier is mandatory for a service to be deployed and identified by WSAMI Middleware.

The WSAMI Middleware provides you with the possibility to associate an identifier to a specific service instance, through the option `--instanceURL` of `WSDL2WSAMI.sh`. All the services deployed with the same `--instanceURL` value will be considered as the same service instance and will be interchangeable, so they will have the same WAU identifier (i.e., the same abstract interface) and the same implementation.
If you want to enable the identification of the service by instance URL, you must specify the option `--instanceURL` of `WSDL2WSAMI.sh`. The specified value will be added to the `AgendaDeploy.xml` file and when the service will be deployed, a client will be able to find it with the method `getService_byInstanceURL` of Naming and Discovery service (see sections 3.5 and 5.2) in addition to the seach by WAU with method method `getService` of Naming and Discovery.

3.4. Service deployment

Deploying (and undeploying) a service requires the WSAMI server to be up and running. To deploy the service *Agenda* on the server, the command is the following:

```
deployer.sh --deploy AgendaDeploy.xml --impl Agenda.jar
    --descr AgendaConcrete.wsdl AgendaInterface.wsdl
       AgendaAbstract.wsami AgendaService.wsami
```

After the execution of this command:

- The files `AgendaAbstract.wsami`, `AgendaService.wsami`, `AgendaInterface.wsdl` and `AgendaConcrete.wsdl` are available at the addresses specified in section 3.3:

```
http://localhost:8080/wsami/AgendaAbstract.wsami
http://localhost:8080/wsami/AgendaService.wsami
http://localhost:8080/wsami/AgendaInterface.wsdl
http://localhost:8080/wsami/AgendaConcrete.wsdl
```

- all the files .class included in `Agenda.jar` have been copied to the service directory of the WSAMI server
- The service is available on the Core Broker and the Naming and Discovery Service at the address:

```
http://localhost:8080/services/Agenda
```

The command to undeploy the *Agenda* service is:

```
deployer.sh --deploy AgendaunDeploy.xml
```

Deploy with ant

```
ant deploy
ant undeploy
```

If the execution of the service requires some extra libraries you must set the space separated list of jar files in variable `service.extra.deployjars` in file `$AGENDADIR/build.xml`. The path of each file must be an absolute path (you can use the `${basedir}` variable to refer to the service directory).
The service will be deployed (undeployed) on the WSAMI server.

```
ant deployPDA
```

If the execution of the service requires some extra libraries you must set the space separated list of jar files in variable `service.extra.deployjars` in file `$AGENDADIR/build.xml`. The path of each file must be an absolute path (you can use the `${basedir}` variable to refer to the service directory).
A tar file containing all wsami, wsdl, xml (deploy/undeploy) and jar files for the service will be created in `$AGENDADIR/build/buildService`. Then you can copy this file to the PDA and after having extracted all the files in a PDA's local directory and you can deploy the service with the `deployer.sh` command seen above.

3.5. Client development

The first step in client development is the generation of the java file implementing the service stub (and the java files implementing complex types, if defined by the service). To accomplish this step you need the wsdl file that defines the service interface (in the example, `AgendaInterface.wsdl`). The command is the following:

```
compileClient.sh AgendaInterface.wsdl
```

The second step is the implementation of the java client. You can refer to the client example `$WSAMIDIR/samples/Example1/Exampl1Client/Example1Client.java` to implement `AgendaClient.java`.
In the following piece of code you can see the main steps in the basic operation a minimal client must do to have an interaction with a service.

In lines 1-2 the client gets a stub to call the local Naming and Discovery Service (ND)
In lines 4-5 the client calls the ND asking the list of services *Agenda* available on the network
In lines 10-13 the client uses the service endpoint to create a stub and make a call to the service's method `method1`.

```
1   java.net.URL localNDURL = new java.net.URL("http://localhost:8080/service/ND");
    NDService.NDPortType nd = new NDService.NDBindingStub(localNDURL,null);

    String wsami = "http://localhost:8080/wsami/AgendaAbstract.wsami";
5   NDService.GetServiceData[] services = nd.getService(wsami);
    if (services != null && services.length > 0) {
      for (int i=0; i<services.length; i++) {
        String serviceEndPoint = services[i].getServiceEP();

10      URL url = new URL(serviceEndPoint);
        AgendaService.AgendaInterface service =
                        new AgendaService.AgendaSOAPBindingStub( url, null );
        service.method1(...);
      }
15  }
```

You can now compile your client code with the command:

```
compileClient.sh AgendaInterface.wsdl . AgendaClient.java AgendaService
```

Where the first parameter is the name of the wsdl interface file, the second is used to add the local directory to the compilation classpath, the third is the client implemention file and the last one is the directory containing the java files generated from wsdl and used to implement the client (service stub and interface and complex types).

In the following piece of code we show how a client can retrieve a specific service instance by using its instance URL address.
The only diffenerence from the above code is in lines 4-5 where the client calls the method `getService_ByInstanceURL` instead of `getService` of the local Naming and Discovery Service (ND) to get the list of services' instances identified by the instance URL `"http://www.inria.fr/arles/WSAMIService1"`.

```
1   java.net.URL localNDURL = new java.net.URL("http://localhost:8080/service/ND");
    NDService.NDPortType nd = new NDService.NDBindingStub(localNDURL,null);

    String instanceURL = "http://www.inria.fr/arles/WSAMIService1";
```

```
5   NDService.GetServiceData[] services = nd.getService_ByInstanceURL(instanceURL);
    if (services != null && services.length > 0) {
      for (int i=0; i<services.length; i++) {
        String serviceEndPoint = services[i].getServiceEP();

10      URL url = new URL(serviceEndPoint);
        AgendaService.AgendaInterface service =
                        new AgendaService.AgendaSOAPBindingStub( url, null );
        service.method1(...);
      }
15  }
```

When the invocation of a service method raises an exception because the service is no longer available, you can catch the exception and ask the ND service to get a new instance of the service. If you are interested in a service implementing the same abstract interface you can use the `getService` method as shown in the first piece of code, while if you are interested in the same service implementation you can use the method `getService_ByInstanceURL` as in the second piece of code.
To prevent the ND from giving you the same service address that you are using as service endpoint, before calling the `getService` (or `getService_ByInstanceURL`) method you must invalidate this address with the ND method `invalidate`.

```
    java.net.URL localNDURL = new java.net.URL("http://localhost:8080/service/ND");
    NDService.NDPortType nd = new NDService.NDBindingStub(localNDURL,null);

    String wsami = "http://localhost:8080/wsami/AgendaAbstract.wsami";
    NDService.GetServiceData[] services = nd.getService(wsami);
    if (services != null && services.length > 0) {
        String serviceEndPoint = services[0].getServiceEP();

        URL url = new URL(serviceEndPoint);
        AgendaService.AgendaInterface service =
                        new AgendaService.AgendaSOAPBindingStub( url, null );
        try {
            service.method1(...);
        } catch (Exception ex) {
            String instanceURL = services[0].getInstanceURL();
            nd.invalidate(serviceEndPoint);
            services = nd.getService_ByInstanceURL(instanceURL);
            if (services != null && services.length > 0) {
                serviceEndPoint = services[0].getServiceEP();

                url = new URL(serviceEndPoint);
                AgendaService.AgendaInterface service =
                        new AgendaService.AgendaSOAPBindingStub( url, null );

                service.method1(...);
        }
    }
```

Compile with ant

```
    ant compileClient
    ant compileClientPDA
```

If the compilation of the client requires some extra libraries you must set the colon separated list of jar files and directory to add to the compilation classpath in variable `client.extra.compileclasspath` in file `$AGENDADIR/build.xml`. The path of each file must be an absolute path (you can use the `${basedir}` variable to refer to the service directory).

The .class and .jar files will be created in `$AGENDADIR/build/buildClient`.

If you want to develop a client with a web interface on PDA there are some limitations you must take into account in writing your client. Since the WSAMI server running on the PDA doesn't support JSP because of performance concerns, the only way to have a web interface is to use HTML and servlets.
In this case the ant/build.xml files provided for the examples can help you:

- set the variable `client.isJSP` in `$AGENDADIR/build.xml`
- write your jsp files in `$AGENDADIR/src/AgendaClient/jsp`
- execute `ant compileClientPDA`
- the jsp will be compiled into `$AGENDADIR/build/buildClient` as servlet classes and a tar file including servlets and service related files (stub, interface and complex types) will be generated
- copy the tar file to the PDA and extract its content into `$WSAMIDIR/dist/pda/root/servlet`.
- The address of the servlet will be:
  `http://localhost:8080/root/servlet/Agenda.servletClassName`.
- if you want to add html files you can copy in `$WSAMIDIR/dist/pda/root` directory and they will be available at the address `http://localhost:8080/root/`.

3.6. Client execution

You can finally execute your client with the following command:

```
runClient.sh AgendaClient.jar AgendaClient.AgendaClient param1 ... paramN
```

Where the first argument is the jar file implementing the client, the second parameter is the name of the .class file to execute to run the client and the following parameters are used as argument of the client class.

Execute with ant

```
ant runClient
```

If the execution of the client requires some parameters, you must set the variable `client.params` in file `$AGENDADIR/build.xml` as a string of parameters separated by a space. The value of the parameters must take into account the target system (Linux or Windows+Cygwin). You can use the `${basedir}` variable to refer to the service directory.

3.7. Customization support

The WSAMI abstract interface specification is extended with the definition of the QoS properties (for example: security, resource-saving) required from a service. In the service WSAMI definition, these properties are defined by the `QoSCriterion` element inside `ServiceQoS` and `ConnectorQoS` elements (see WSAMI definition in 6).

Our approach in service QoS implementation is based on the use of a pair of content customizers at both ends, as presented in [STEI]. A Customizer decomposes into a local and a remote service. Any message exchanged between the client and the service goes through the

Customizer.
In the WSAMI specification, the `QoSCriterion` part specifies the specific QoS criterions that are enforced by the embedding customizer. The `Local` and `Remote` parts specify the abstract interfaces of the customizer services.
Given the above WSAMI QoS elements (`QoSCriterion`, `Local` and `Remote`), a service instance matches a requested service if they both specify the same document URI for the service's abstract interface and if the a service with the URI of the Local Customizer service's abstract interface is available in the environment of the client and the a service with the URI of the Remote Customizer service's abstract interface is available in the environment of the service (if `Colocation` is set to true, they must be available respectively, on the client and on the service nodes).

3.7.1 WSDL interface and concrete files

As the interaction between the client and the service through a customizer must be transparent, the client must not be aware of the service it is interacting with (either directly with the service or with the Local Customizer). This requires that the service and the local customizer must have the same interface and, as a result, also the same WSDL interfaces. The remote customizer can have a WSDL interface different from the local customizer and the service.
The Local and Remote Customizer are considered regular WSAMI application-related services and there are no special rules and restrictions to apply in their development and deployment.
As an example of customization we consider the Agenda service described in tutorial in section 3.1 - 3.6 and we consider a local customizer with a service name `AgendaLC` and a remote customizer `AgendaRC`.

3.7.2 Service development

Local Customizer Implementation
The Local Customizer implementation class must extend the `CoreBroker.BasicCustomizer` class (line 3) provided by Core Broker and providing some functionalities aimed at making the customization process transparent for the service and customizer developer. All the customization logic is implemented and hidden inside the core broker.
The Local Customizer class must further implement the interface generated from wsdl (line 4) and all the same methods of the service. The method `getRC()` inherited from class `CoreBroker.BasicCustomizer` returns the address enpoint of the Remote Customizer (line 7) and must be used to build a stub (line 10-11) to make a call the Remote Customizer'method `methodX` (line 12).

```
1   package AgendaLCService;

    public class AgendaLCImpl extends CoreBroker.BasicCustomizer
        implements AgendaLCInterface{
5
    public void method1(...) throws java.rmi.RemoteException {
      String rc = getRC();
      try{
        java.net.URL rcURL = new java.net.URL(rc);
10      AgendaRCService.AgendaRCInterface remoteCust =
              new AgendaRCService.AgendaRCSOAPBindingStub(rcURL, null);
```

```
        remoteCust.methodX(...);
      } catch (Exception ex) {
        ex.printStackTrace();
    }
  }
```

Remote customizer Implementation

The Remote Customizer implementation class must extend the `CoreBroker.BasicCustomizer` class (line 3) provided by Core Broker and providing some functionalities aimed at making the customization process transparent for the service and customizer developer. All the customization logic is implemented and hidden inside the core broker.

The Remote Customizer class must further implement the interface generated from wsdl (line 4) and all the same methods of the service. The method `getWS()` inherited from class `CoreBroker.BasicCustomizer` returns the address enpoint of the Service (line 7) and must be used to build a stub (line 10-11) to make a call to the service's method `method1` (line 12) that must have the same specification of the method `method1` of `AgendaLCService.AgendaLCInterface` seen above.

```
1   package AgendaLCService;

    public class AgendaRCImpl extends CoreBroker.BasicCustomizer
        implements AgendaRCInterface{
5
    public void methodX(...) throws java.rmi.RemoteException {
      String ws = getWS();
      try{
        java.net.URL wsURL = new java.net.URL(ws);
10      AgendaService.AgendaInterface service =
              new AgendaService.AgendaSOAPBindingStub(wsURL, null);
        service.method1(...);
      } catch (Exception ex) {
        ex.printStackTrace();
15    }
    }
```

3.7.3 WSAMI and deploy/undeploy files

If you want the Agenda service to be customized, the wsami and deployment files must be built with the following command:

```
WSDL2WSAMI.sh --hasCustomizer http://localhost:8080/wsami/AgendaLCAbstract.wsami
                http://localhost:8080/wsami/AgendRCAbstract.wsami true Agenda
```

The generated AgendaAbstract.wsami is:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsami xmlns="http://www-rocq.inria.fr/arles/wsami"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www-rocq.inria.fr/arles/wsami WSAMI.xsd">
  <Definition name="Agenda" targetNamespace="http://localhost:8080/wsami/">
    <Abstract name="Agenda">
      <Interface hrefSchema="http://localhost:8080/wsami/AgendaInterface.wsdl"/>
       <ConnectorQoS>
         <QoSCriterion name="security"/>
       </ConnectorQoS>
     </Abstract>
   </Definition>
</wsami>
```

A new file AgendaCustomizer.wsami will be generated:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsami xmlns="http://www-rocq.inria.fr/arles/wsami"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www-rocq.inria.fr/arles/wsami WSAMI.xsd">
   <Definition name="AgendaCust" targetNamespace="http://localhost:8080/wsami/">
     <Customizer name="AgendaCust" Colocation="true">
       <QoSCriterion name="security"/>
       <Local hrefSchema="http://localhost:8080/wsami/AgendaLCAbstract.wsami"/>
       <Remote hrefSchema="http://localhost:8080/wsami/AgendRCAbstract.wsami"/>
     </Customizer>
   </Definition>
</wsami>
```

The AgendaDeploy.xml will contain the definition of the wsami file for customization (wcd):

```
<?xml version="1.0" encoding="UTF-8"?>
<deploy xmlns="http://www-rocq.inria.fr/arles/wsami"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www-rocq.inria.fr/arles/wsami
    deploy_install.xsd">
  <service name="Agenda">
      <wad name="AgendaAbstract.wsami"/>
      <wsd name="AgendaService.wsami"/>
      <wau name="http://localhost:8080/wsami/AgendaAbstract.wsami"/>
      <wcd name="AgendaCustomizer.wsami"/>
      <implementation>
         <class name="AgendaService.AgendaImpl"/>
         <type name="javaclass"/>
         <package name="AgendaService"/>
      </implementation>
      <scope name="Application"/>
  </service>
</deploy>
```

To obtain the files for the Local Customizer the command is:

```
WSDL2WSAMI.sh --isCustomizer local --scope Session AgendaLC
```

AgendaLCDeploy.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<deploy xmlns="http://www-rocq.inria.fr/arles/wsami"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www-rocq.inria.fr/arles/wsami
   deploy_install.xsd">
  <service customizer="local" name="AgendaLC">
      <wad name="AgendaLCAbstract.wsami"/>
      <wsd name="AgendaLCService.wsami"/>
      <wau name="http://localhost:8080/wsami/AgendaLCAbstract.wsami"/>
      <implementation>
         <class name="AgendaLCService.AgendaLCImpl"/>
         <type name="javaclass"/>
         <package name="AgendaLCService"/>
      </implementation>
      <scope name="Session"/>
  </service>
</deploy>
```

And for the Remote Customizer the command is:

```
WSDL2WSAMI.sh --isCustomizer remote --scope Session AgendaRC
```

AgendaRCDeploy.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<deploy xmlns="http://www-rocq.inria.fr/arles/wsami"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www-rocq.inria.fr/arles/wsami deploy_install.xsd">
   <service customizer="remote" name="AgendaRC">
      <wad name="AgendaRCAbstract.wsami"/>
      <wsd name="AgendaRCService.wsami"/>
      <wau name="http://localhost:8080/wsami/AgendaRCAbstract.wsami"/>
      <implementation>
         <class name="AgendaRCService.AgendaRCImpl"/>
         <type name="javaclass"/>
         <package name="AgendaRCService"/>
      </implementation>
      <scope name="Session"/>
   </service>
</deploy>
```

3.7.4 Service deployment

The Local Customizer will have to be deployed on a node available in the client's environment (on the client's node if `Colocation` is set to true) and the Remote Customizer will have to be deployed on a node available in the service's environment (on the service's node if `Colocation` is set to true).

3.7.5 Client development

In the following piece of code you can see how a client can set the customization chain composed of local customizer, remote customizer and service and make a call to the service. In lines 1-2 the client gets an instance of the core broker In lines 4-9 the client makes a call to the local Naming and Discovery Service (ND) to retrieve an instance of the service identified by the WSAMI Abstract URI `http://localhost:8080/wsami/AgendaAbstract.wsami`. In lines 10-13 the client gets the address of the service and of customizers In line 15-16 the client sets the customizer chain In lines 17-20 the clinet uses the endpoint receive by the broker to create a stub of service and makes a call to the service's method `method1` already seen above in the local customizer implementation.

```
1   CoreBroker.BrokerWrapper broker =
            CoreBroker.BrokerFactory.getClientInstance();

    java.net.URL localNDURL =
5       new java.net.URL("http://localhost:8080/service/ND");
    NDService.NDPortType nd = new NDService.NDBindingStub(localNDURL,null);

    String wsami = "http://localhost:8080/wsami/AgendaAbstract.wsami";
    NDService.GetServiceData[] services = nd.getService(wsami);
10  if (services != null && services.length > 0) {
        String epService = services[0].getServiceEP();
        String epCustLocal = services[0].getCustLocalEP();
        String epCustRemote = services[0].getCustRemoteEP();

15      epService =
            broker.setCustomizerChain(epCustLocal, epCustRemote, epService);
        URL url = new URL(epService);
        AgendaService.AgendaInterface service =
                new AgendaService.AgendaSOAPBindingStub( url, null );
20      service.method1(...);
    }
```

4.WSAMI Tools Reference

All the tools described in this section are available in directory `$WSAMIDIR/dist/tomcat/bin` for WSAMI version PC and in directory `$WSAMIDIR/dist/pda/bin` for WSAMI version PDA.

4.1 compileService.sh and compileClient.sh

```
compileService.sh <concrete.wsdl> <classpath> <file1>...<fileN> <dir1>...<dirN>
```

where:

```
<concrete.wsdl> is the wsdl file that defines the wsdl abstract interface
        of the service and the endpoint of the service
<classpath> is the classpath required to compile the service and that must
        be added to the default WSAMI classpath used for service
        compilation. The elements must be separated by a colon (semi-colon
        in cygwin???)
<file1>...<fileN> <dir1>...<dirN> is the list of files java or directories
        containing java files that implement the service and must be
        compiled.
```

If it is invoked without java files and dirs, the command will execute only the wsdl generation step, that is it will generated the java files for service stub, interface and skeleton and for defined complex types.
Otherwise, if they are specified, after the first wsdl generation step, all the generated java files along with all the files given in input line are compiled using the WSAMI classpath together with the specified classpath. The .class files are generated in the current directory.

```
compileClient.sh <interface.wsdl> <classpath> <file1.java>...<fileN.java> <dir1>...<dirN>
```

The difference from `compileService.sh` is that `<interface.wsdl>` is the wsdl file that defines the wsdl abstract interface of the service and does not define the endpoint of the service as `<concrete.wsdl>` and the service skeleton file is not generated.

4.2 WSDL service interface restrictions

WSAMI platform fixes some restrictions in the WSDL service interface in the definition of types and exception:

complex types

According to the XML Schema specifications, complex types may be declared as containers of three types: `sequence`, `all` and `choice`. The only type supported by WSAMI for complex types is `sequence`. The following example shows the definition of a complex type containing a string and an integer type:

```
<xsd:complexType name="UserDefinedComplexType">
  <xsd:sequence>
    <xsd:element name="firstPar" type="xsd:string"/>
    <xsd:element name="secondPar" type="xsd:int"/>
  </xsd:sequence>
```

```
</xsd:complexType>
```

arrays

According to WSDL specification, arrays must be declared by `restriction`. In other words, if you have to declare an array of Java String, you define it as an element belonging to a subset (a restriction) of the set of standard SOAP arrays. More formally, an array is derived from a `soapenc:Array` by restriction using the `wsdl:arrayType` attribute, with the `soapenc` prefix associated to the the namespace URI `http://schemas.xmlsoap.org/soap/encoding` and the `wsdl` prefix to `http://schemas.xmlsoap.org/wsdl`.

The following example shows the definition of an array of String derived from the `soapenc:Array` by restriction:

```
<xsd:complexType name="ArrayOfString">
  <xsd:complexContent>
    <xsd:restriction base="soapenc:Array">
      <xsd:sequence>
        <xsd:element name="string_value"
                  minOccurs="0" maxOccurs="unbounded" type="xsd:string"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

It is worth noting that the name of the complex type must start with `ArrayOf`; this limitation derives from the implementation of the WSIF library that we have used to manipulate the XML Schema declaration inside the WSDL file.

exceptions

You should pay particular attention in declaring exceptions; in fact, you must follow the defined standard. Now we focus in particular on service specific exceptions. To sum up, a `fault` element, which is an optional element inside an `operation` block, specifies the format of any error message related to the remote invocation of a particular service method. It is worth noting that according to the WSDL specification, a fault message must have a single part. The name of the Java exception is mapped from the `name` attribute of the `message`. For reference, you can look at the following example:

```
<message name="Example3Exception">
  <part name="exceptionCode" type="xsd:string"/>
</message>

<portType name=... >
  <operation name=... >
    <input message=... />
    <output message=... />
    <fault name="Example3Exception" message="tns:Example3Exception"/>
  </operation>
</portType>

<binding name= ... >
    <operation name= ... >
      <soap:operation ... >
        <input> ... </input>
        <output> ...  </output>
        <fault name="Example3Exception">
          <soap:fault name="Example3Exception" use="literal"/>
        </fault>
     </operation>
```

```
        </soap>
      </binding>
```

4.3 WSDL2WSAMI.sh

```
WSDL2WSAMI.sh [options] <servicename>

Options:
    --packageName <optionvalue>
        the name of the directory where are stored all .class files used by
        the java service implementation (the same of the java package)
    --WSD <optionvalue>
        WSAMI Service Document: the name of the local file that defines the
        WSAMI Service of the service
    --WSDLconcrete <optionvalue>
        the name of the WSDL files that defines the location endpoint of
        the service
    --HTTPBaseAddr <optionvalue> (default="http://localhost:8080/wsami/" )
        The address used to construct the URI (WAU, WSU, WCD, WSDL URI)
        that are used for files generation
    --WSU <optionvalue>
        WSAMI Service URI: the URI address where is stored the WSAMI
        Service Document
    --WAD <optionvalue>
        WSAMI Abstract Document: the name of the local file that defines
        the WSAMI Abstract of the service
    --undeployXML <optionvalue>
        the name of the file that will be used to undeploy the service from
        the Core Broker
    --help
        help
    --WAU <optionvalue>
        WSAMI Abstract URI: the URI address where is stored the WSAMI
        Abstract Document
    --className <optionvalue>
        the name of the java class that implements the service
    --WCD <optionvalue>
        WSAMI Customizer Document: the name of the local file that defines
        the WSAMI Customizer of the service
    --type     [ javaclass ]  ( default = "javaclass" )
        javaclass if service is implemented as a java class.
    --WSDLinterface <optionvalue>
        the name of the WSDL file that defines the service WSDL Interface
    --hasCustomizer <optionvalue> <optionvalue>
        the service implements the Customization feature. If this option is
        specified, the Customization file WCD and all the references to WCU
        will be generated in WAD (first option is Local Customizer WAU and
        second option is Remote Customizer WAU
    --hasCustomizer <optionvalue> <optionvalue> <optionvalue>
        the service implements the Customization feature. If this option
        is specified, the Customization file WCD and all the references to
        WCU will be generated in WAD (first option is Local Customizer WAU
        and second option is Remote Customizer WAU, third option is for
        customizers colocation (true/false)
    --deployXML <optionvalue>
        the name of the file that will be used to deploy the service on the
        Core Broker
    --WCU <optionvalue>
        WSAMI Customizer URI: the URI address where is stored the WSAMI
```

```
    Customizer Document
--isCustomizer     [ local | remote ]
    local if this service implements the local side of the
    customization feature.
    remote if this service implements the remote side of the
    customization feature
--outputDir <optionvalue>
    the name of the directory where all xml and wsami files are
    generated
--WSDLconcrete <optionvalue>
    the name of the WSDL files that defines the location endpoint of
    the service
--scope[ Request | Session | Application ] ( default = "Application" )
    Application: only one instance of the service.
    Session: one instance for every client.
    Request: one instance for every request.
--instance_URL <optionvalue>
    the instance_URL of the service instance deployed with this
    deployment descriptor
```

The default behavior of the tool is to create in the current directory the following files: `<servicename>Deploy.xml`, `<servicename>unDeploy.xml`, `<servicename>Abstract.wsami`, `<servicename>Service.wsami`, using the following standard values:

```
packageName: <servicename>Service
java className implementing the service:
<servicename>Service.<servicename>Impl
type: javaclass
deployXML filename: <servicename>Deploy.xml
undeployXML filename: <servicename>unDeploy.xml
scope: Application

WAD filename: <servicename>Abstract.wsami
WSD filename: <servicename>Service.wsami
WAU address: http://localhost:8080/wsami/<servicename>Abstract.wsami
WSU address: http://localhost:8080/wsami/<servicename>Service.wsami
WSDLinterface filename: <servicename>Interface.wsdl
WSDLconcrete filename: <servicename>Concrete.wsdl
HTTPBaseAddr: http://localhost:8080/wsami/
URI of WSDLInterface:
http://localhost:8080/wsami/<servicename>Interface.wsdl
```

If you want to deploy a customized service (for more details about customization see section 3.7) you must specify the `--hasCustomizer` option. The `<servicename>Customizer.wsami` will be generated and the following standard values will be used:

```
WCD: <servicename>Customizer.wsami
WCU: http://localhost:8080/wsami/<servicename>Customizer.wsami
```

If you want to deploy a service as a local or remote customizer (for more details about customization see section 3.7) you must specify the `--isCustomizer [ local | remote ]` option. The `<servicename>Deploy.wsami` will be generated taking into account the information.

4.4 deployer.sh

```
deployer.sh [options]

Options:
   --impl
       The name of the jar files that implement the service
   --help
       help
   --deploy <optionvalue>
       The name of the (un)deployment file
   --CoreBrokerAddr <optionvalue>
       ( default = "http://localhost:8080/wsami/AdminServlet" )
       The address of the Core Broker on the local machine
   --descr
       The name of the files used to describe the service(wsami and wsdl
       files)
```

When you execute this command the WSAMI server must be running (see section 2.1.4 for PC and section2.2.3 for PDA). If you want to deploy a service the command will be like the following:

```
deployer.sh --deploy <Service>Deploy.xml --impl <Service>.jar
    --descr <Service>Concrete.wsdl <Service>Interface.wsdl
        <Service>Abstract.wsami <Service>Service.wsami
```

To undeploy the service the command will be:

```
deployer.sh --deploy <Service>unDeploy.xml
```

The wsami, wsdl, `<Service>Deploy.xml` and `<Service>unDeploy.xml` files must be generated using the `WSDL2WSAMI.sh` tool (see section 4.3).

When you deploy a service:

- The wsami and wsdl files specified with the `--descr` option are made available through the WSAMI server at the specified addresses.
- the .class files contained in all the jar files specified by `--impl` option are copied to the `$TOMCATDIR/webapps/services/WEB-INF/classes` directory on the WSAMI PC version and to the `$WSAMIDIR/services/WEB-INF/classes` directory on the WSAMI PDA version.
- The service is made available on the Core Broker and the Naming and Discovery Service.

When you undeploy a service, all the wsami, wsdl files defining and .class files implementing the service are removed from WSAMI server and the service is no longer available on the Core Broker and the Naming and Discovery Service.
If you want to redeploy a service after the undeployment operation, you must stop and restart the WSAMI server.

4.5 runClient.sh

```
runClient.sh <classpath> <classname> <param1> ... <paramN>
```

where:

```
<classpath> is the classpath required to execute the client and that must
        be added to the default WSAMI classpath libraries.
        Elements must be separated by a colon(semi-colon in cygwin???)
<classname> is the name of the class containg the main method and to be
        executed
<param1> ... <paramN> is the list of parameters that will be passed as
        arguments to the java class <classname>
```

5.WSAMI Middleware Internals

5.1 WSAMI Core Broker

The Core Broker provides two main functionalities: the execution and the deployment of services. The former functionality is provided through a servlet running on top of the web server and waiting for SOAP/HTTP RPC messages that correspond to calls to services deployed on the server. The latter is provided, on the server side, as a servlet running on top of the web server and waiting for deployment commands and on the client side, with the `deployer.sh` tool (described in section 4.4) used to send deployment commands. In the execution of a service, the `--scope` deployment option of `deployer.sh` tool plays a central role in deciding how service instances are managed. The possible values are:

Request
: an new instance of the service is created at each call

Session
: at the first call received from a client, a new instance of the service is created and used for all the following calls received from the same client

Application
: at the first call received for a service, a new instance is created and this instance will be used during all the life of the server to reply to calls received from any clients

In your client or service implementation code you can make use of some of the functionalities offered by Core broker's interface to applications.
If you are developing a client you can obtain the instance of the client side Core Broker with the instruction:

```
CoreBroker.BrokerWrapper broker = CoreBroker.BrokerFactory.getClientInstance();
```

If you are developing a client you can obtain the instance of the client side Core Broker with the instruction:

```
CoreBroker.BrokerWrapper broker = CoreBroker.BrokerFactory.getServerInstance();
```

5.2 WSAMI Naming and Discovery Service

The Naming and Discovery Service (ND Service) implementation is based on Service Location Protocol (SLP). The SLP implementation used by WSAMI is OpenSLP. In WSAMI server startup sections (2.1.4 and 2.2.3) the `SLPMGR.sh` script is required to be up and running

before WSAMI server starts because this script has the role to manage the configuration and startup/shutdown of the SLP server.

The ND service is the first service deployed at server startup because it is essential for the system to run. It's always running at the address `http://localhost:8080/services/ND`. Section 3.5 details how a client application can obtain a list of instances of a service identified by a specific WSAMI Abstract URI of by a specific instance URL.

5.2.1 WSAMI and WSDL files

NDAbstract.wsami

```
<?xml version="1.0" encoding="UTF-8"?>
<wsami
        xmlns="http://www-rocq.inria.fr/arles/wsami"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www-rocq.inria.fr/arles/wsami WSAMI.xsd">

 <Definition name="ND" targetNamespace="http://localhost:8080/wsami/">
  <Abstract name="ND">
   <Interface hrefSchema="http://localhost:8080/wsami/NDInterface.wsdl"/>
  </Abstract>
 </Definition>
</wsami>
```

NDService.wsami

```
<?xml version="1.0" encoding="UTF-8"?>
<wsami
        xmlns="http://www-rocq.inria.fr/arles/wsami"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www-rocq.inria.fr/arles/wsami WSAMI.xsd">

 <Definition name="ND" targetNamespace="http://localhost:8080/wsami/">
  <Service name="ND">
   <Abstract hrefSchema="http://localhost:8080/wsami/NDAbstract.wsami"/>
   <Concrete hrefSchema="http://localhost:8080/wsami/NDConcrete.wsdl"/>
  </Service>
 </Definition>
</wsami>
```

NDInterface.wsdl

```
<?xml version="1.0" ?>
<definitions name="urn:NDService"
             targetNamespace="urn:NDService"
             xmlns:tns="urn:NDService"
             xmlns:typens="urn:NDService"
             xmlns:xsd="http://www.w3.org/1999/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
             xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!-- type defs -->
  <types>
    <xsd:schema targetNamespace="urn:NDService">

     <xsd:complexType name="RemoteService">
       <xsd:sequence>
        <xsd:element name="WAU" type="xsd:string"/>
        <xsd:element name="address" type="xsd:string"/>
        <xsd:element name="instanceURL" type="xsd:string"/>
        <xsd:element name="AC" type="xsd:boolean"/>
        <xsd:element name="battery" type="xsd:int"/>
```

```
      <xsd:element name="LCwau" type="xsd:string"/>
      <xsd:element name="RCwau" type="xsd:string"/>
      <xsd:element name="colocation" type="xsd:boolean"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="ArrayOfRemoteService">
    <xsd:complexContent>
     <xsd:restriction base="soapenc:Array">
       <xsd:sequence>
         <xsd:element name="strings" minOccurs="0" maxOccurs="unbounded"
                      type="typens:RemoteService"/>
       </xsd:sequence>
     </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="RemoteServices">
    <xsd:sequence>
     <xsd:element name="list" type="typens:ArrayOfRemoteService"/>
    </xsd:sequence>
  </xsd:complexType>


  <xsd:complexType name="GetServiceData">
    <xsd:sequence>
     <xsd:element name="serviceEP" type="xsd:string"/>
     <xsd:element name="custLocalEP" type="xsd:string"/>
     <xsd:element name="custRemoteEP" type="xsd:string"/>
     <xsd:element name="instanceURL" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="ArrayOfGetServiceData">
    <xsd:complexContent>
     <xsd:restriction base="soapenc:Array">
       <xsd:sequence>
         <xsd:element name="strings" minOccurs="0" maxOccurs="unbounded"
                  type="typens:GetServiceData"/>
       </xsd:sequence>
     </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>

 </xsd:schema>
</types>



<!-- message declns -->
<message name="initializationRequest"/>
<message name="stopRequest"/>

<message name="RegisterRequest">
  <part name="name" type="xsd:string"/>
  <part name="WAU" type="xsd:string"/>
  <part name="WSU" type="xsd:string"/>
  <part name="WCU" type="xsd:string"/>
</message>

<message name="UnRegisterRequest">
  <part name="WAU" type="xsd:string"/>
</message>
<message name="UnRegisterResponse">
  <part name="Error" type="xsd:string"/>
</message>

<message name="GetService_ByInstanceURLRequest">
```

```
  <part name="instanceURL" type="xsd:string"/>
</message>
<message name="GetService_ByInstanceURLResponse">
  <part name="serviceEP" type="typens:ArrayOfGetServiceData"/>
</message>

<message name="GetServiceRequest">
  <part name="WAU" type="xsd:string"/>
</message>
<message name="GetServiceResponse">
  <part name="serviceEP" type="typens:ArrayOfGetServiceData"/>
</message>

<message name="QueryService">
  <part name="WAUquery" type="xsd:string"/>
</message>
<message name="ReplyService">
  <part name="EP" type="typens:RemoteServices" />
</message>

<message name="QueryService_ByInstanceURL">
  <part name="instanceURLquery" type="xsd:string"/>
</message>
<message name="ReplyService_ByInstanceURL">
  <part name="EP" type="typens:RemoteServices" />
</message>

<message name="invalidateRequest">
  <part name="serviceEP" type="xsd:string"/>
</message>

<!-- port type declns -->
<portType name="NDPortType">
  <operation name="initialization">
    <input message="tns:initializationRequest"/>
  </operation>
  <operation name="stop">
    <input message="tns:stopRequest"/>
  </operation>

  <operation name="Register">
    <input message="tns:RegisterRequest"/>
  </operation>

  <operation name="UnRegister">
    <input message="tns:UnRegisterRequest"/>
    <output message="tns:UnRegisterResponse"/>
  </operation>
  <operation name="getService">
    <input message="tns:GetServiceRequest"/>
    <output message="tns:GetServiceResponse"/>
  </operation>
  <operation name="getService_ByInstanceURL">
    <input message="tns:GetService_ByInstanceURLRequest"/>
    <output message="tns:GetService_ByInstanceURLResponse"/>
  </operation>
 <operation name="QueryService">
    <input message="tns:QueryService"/>
    <output message="tns:ReplyService"/>
  </operation>
 <operation name="QueryService_ByInstanceURL">
    <input message="tns:QueryService_ByInstanceURL"/>
    <output message="tns:ReplyService_ByInstanceURL"/>
  </operation>
  <operation name="Invalidate">
    <input message="tns:invalidateRequest"/>
  </operation>
```

```
</portType>


<!-- binding declns -->
<binding name="NDBinding" type="tns:NDPortType">

  <soap:binding style="rpc"
                transport="http://schemas.xmlsoap.org/soap/http" />

  <operation name="initialization">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="encoded"
                 namespace="urn:NDService"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
  </operation>


  <operation name="stop">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="encoded"
                 namespace="urn:NDService"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
  </operation>


  <operation name="Register">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="encoded"
                 namespace="urn:NDService"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
  </operation>
  <operation name="UnRegister">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="encoded"
                 namespace="urn:NDService"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
   <output>
      <soap:body use="encoded"
                 namespace="urn:NDService"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
  </operation>
  <operation name="getService">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="encoded"
                 namespace="urn:NDService"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <output>
      <soap:body use="encoded"
                 namespace="urn:NDService"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
  </operation>

  <operation name="getService_ByInstanceURL">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="encoded"
```

```
                    namespace="urn:NDService"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
                    namespace="urn:NDService"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>

    <operation name="QueryService">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="encoded"
                    namespace="urn:NDService"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
                    namespace="urn:NDService"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>

    <operation name="QueryService_ByInstanceURL">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="encoded"
                    namespace="urn:NDService"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
                    namespace="urn:NDService"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>

    <operation name="Invalidate">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="encoded"
                    namespace="urn:NDService"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
    </operation>
  </binding>
</definitions>
```

NDConcrete.wsdl

```
<definitions name="urn:NDService"
             targetNamespace="urn:NDService"
             xmlns:tns="urn:NDService"
             xmlns:typens="urn:NDService"
             xmlns:xsd="http://www.w3.org/1999/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

<import namespace="urn:NDService" location="NDInterface.wsdl" />
 <!-- service decln -->
  <service name="NDServiceX">
    <port name="ND" binding="tns:NDBinding">
      <soap:address location="http://localhost:8080/services/ND"/>
    </port>
  </service>
</definitions>
```

5.3 WSAMI Univeral Repository (UR) Service

The Universal Repository service is a centralized repository where references to services distributed across the network are registered to be retrieved by ND services when the service discovery process (see ND method `getServices` in section 5.2) in the local area leads to find an empty list of services.

If an address of a universal repository is provided by the user in the configuration file upon WSAMI Middleware startup (see section 2.1.3), the UR service will be used by the local ND service to make available local running services and to let local application discover remote services that could not be discovered without UR.

The UR service provides functionalities for service registration and unregistration. These methods are called by ND service that is charged to register (unregister) on the UR each locally deployed (undeployed) service.

The WSAMI Middleware also provides a client for UR service allowing to search for services registered on a UR service and add the retrieved services and their related information (e.g., the service endpoint) to the local Web services directory that will be used by the Naming and Discovery service in `getService` method (see [ARCHGUIDE]).

5.3.1 WSAMI and WSDL files

URAbstract.wsami

```
<?xml version="1.0" encoding="UTF-8"?>
<wsami xmlns="http://www-rocq.inria.fr/arles/wsami"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www-rocq.inria.fr/arles/wsami WSAMI.xsd">
   <Definition name="UR" targetNamespace="http://localhost:8080/wsami/">
      <Abstract name="UR">
         <Interface hrefSchema="http://localhost:8080/wsami/URInterface.wsdl"/>
      </Abstract>
   </Definition>
</wsami>
```

URService.wsami

```
<?xml version="1.0" encoding="UTF-8"?>
<wsami xmlns="http://www-rocq.inria.fr/arles/wsami"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www-rocq.inria.fr/arles/wsami WSAMI.xsd">
   <Definition name="UR" targetNamespace="http://localhost:8080/wsami/">
      <Service name="UR">
         <Abstract hrefSchema="http://localhost:8080/wsami/URAbstract.wsami"/>
         <Concrete hrefSchema="http://localhost:8080/wsami/URConcrete.wsdl"/>
      </Service>
   </Definition>
</wsami>
```

URInterface.wsdl

```
<?xml version="1.0" ?>
<definitions name="urn:URService"
             targetNamespace="urn:URService"
             xmlns:tns="urn:URService"
             xmlns:typens="urn:URService"
             xmlns:xsd="http://www.w3.org/1999/XMLSchema"
```

```
            xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
            xmlns="http://schemas.xmlsoap.org/wsdl/">


<!-- type defs -->
  <types>
    <xsd:schema targetNamespace="urn:URService">

     <xsd:complexType name="GetServiceDataUR">
       <xsd:sequence>
        <xsd:element name="WAU" type="xsd:string"/>
        <xsd:element name="serviceEP" type="xsd:string"/>
        <xsd:element name="instanceURL" type="xsd:string"/>
        <xsd:element name="LCWAU" type="xsd:string"/>
        <xsd:element name="RCWAU" type="xsd:string"/>
        <xsd:element name="colocation" type="xsd:boolean"/>
       </xsd:sequence>
     </xsd:complexType>

     <xsd:complexType name="ArrayOfGetServiceDataUR">
       <xsd:complexContent>
        <xsd:restriction base="soapenc:Array">
          <xsd:sequence>
            <xsd:element name="strings" minOccurs="0" maxOccurs="unbounded"
                         type="typens:GetServiceDataUR"/>
          </xsd:sequence>
        </xsd:restriction>
       </xsd:complexContent>
     </xsd:complexType>

     <xsd:complexType name="ArrayOfString">
       <xsd:complexContent>
        <xsd:restriction base="soapenc:Array">
          <xsd:sequence>
            <xsd:element name="strings" minOccurs="0" maxOccurs="unbounded"
                     type="xsd:string"/>
          </xsd:sequence>
        </xsd:restriction>
       </xsd:complexContent>
     </xsd:complexType>
    </xsd:schema>
  </types>


<!-- message declns -->

  <message name="registerRequest">
    <part name="WAU" type="xsd:string"/>
    <part name="serviceEP" type="typens:ArrayOfString"/>
    <part name="instanceURL" type="xsd:string"/>
    <part name="LCWAU" type="xsd:string"/>
    <part name="RCWAU" type="xsd:string"/>
    <part name="colocation" type="xsd:boolean"/>
  </message>
  <!-- message name="registerResponse"/-->

  <message name="unRegisterRequest">
    <part name="WAU" type="xsd:string"/>
    <part name="serviceEP" type="xsd:string"/>
  </message>
  <!-- message name="unRegisterResponse"/-->

  <message name="unRegisterAllRequest">
    <part name="ips" type="typens:ArrayOfString"/>
  </message>

  <message name="updateAllAddressesRequest">
```

```
    <part name="ipsOld" type="typens:ArrayOfString"/>
    <part name="ipsNew" type="typens:ArrayOfString"/>
  </message>

  <message name="getServicesRequest">
    <part name="WAU" type="xsd:string"/>
  </message>
  <message name="getServicesResponse">
    <part name="services" type="typens:ArrayOfGetServiceDataUR"/>
  </message>

  <message name="getServicesRequest_ByInstanceURL">
    <part name="instanceURL" type="xsd:string"/>
  </message>
  <message name="getServicesResponse_ByInstanceURL">
    <part name="services" type="typens:ArrayOfGetServiceDataUR"/>
  </message>


<!-- port type declns -->

  <portType name="URInterface">
    <operation name="register">
      <input message="tns:registerRequest"/>
      <!--output message="tns:registerResponse"/-->
    </operation>

    <operation name="unRegister">
      <input message="tns:unRegisterRequest"/>
      <!--output message="tns:unRegisterResponse"/ -->
    </operation>

    <operation name="unRegisterAll">
      <input message="tns:unRegisterAllRequest"/>
    </operation>

    <operation name="updateAllAddresses">
      <input message="tns:updateAllAddressesRequest"/>
    </operation>

    <operation name="getServices">
      <input message="tns:getServicesRequest"/>
      <output message="tns:getServicesResponse"/>
    </operation>

    <operation name="getServices_ByInstanceURL">
      <input message="tns:getServicesRequest_ByInstanceURL"/>
      <output message="tns:getServicesResponse_ByInstanceURL"/>
    </operation>

  </portType>


<!-- binding declns -->

  <binding name="URSOAPBinding" type="tns:URInterface">
    <soap:binding style="rpc"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="register">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="encoded"
                   namespace="http://www.UR.Service"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
    </operation>

    <operation name="unRegister">
```

```
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="encoded"
                   namespace="http://www.UR.Service"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
    </operation>

    <operation name="unRegisterAll">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="encoded"
                   namespace="http://www.UR.Service"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
    </operation>

    <operation name="updateAllAddresses">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="encoded"
                   namespace="http://www.UR.Service"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
    </operation>

    <operation name="getServices">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="encoded"
                   namespace="http://www.UR.Service"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
                   namespace="htto://Service.org"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>

    <operation name="getServices_ByInstanceURL">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="encoded"
                   namespace="http://www.UR.Service"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
                   namespace="htto://Service.org"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>
</definitions>
```

URConcrete.wsdl

```
<definitions name="urn:URService"
             targetNamespace="urn:URService"
             xmlns:tns="urn:URService"
             xmlns:typens="urn:URService"
             xmlns:xsd="http://www.w3.org/1999/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

<import namespace="urn:URService" location="URInterface.wsdl" />
 <!-- service decln -->
```

```
  <service name="UR_Service">
    <port name="UR" binding="tns:URSOAPBinding">
      <soap:address location="http://localhost:8080/services/UR"/>
    </port>
  </service>
</definitions>
```

5.4 WSAMI Gateway Service

This service can be deployed on multihomed hosts to enable the routing from an ad-hoc wireless network to an infrastructure-based network (both wireless or wired). Only Linux PC stations are supported by WSAMI Middleware.

To enable the routing in a station, the WSAMI Middleware requires two operations:

- The host administrator must run the script `$WSAMIDIR/dist/tomcat/bin/WSAMIRouter.sh`. The content of this file can be for example:

```
modprobe ipt_MASQUERADE
iptables -F; iptables -t nat -F; iptables -t mangle -F
iptables -t nat -A POSTROUTING -o wlan0 -d ! 192.168.24.0/24
            -j SNAT --to-source 128.93.135.14
echo 1 > /proc/sys/net/ipv4/ip_forward
```

  where:

  wlan0
  : the network interface connected to the infrastructure-based network

  192.168.24.0/24
  : the network IP address/netmask of the ad-hoc network

  128.93.135.14
  : the IP address of the network interface connected to the infrastructure-based network (wlan0)

- Deploy the GW service on the station (see section 2.1.3)

When deployed, the GW service cannot be called directly from a client, but it can be used only by the service discovery algorithm implemented in method `getService` of ND Service. In this discovery algorithm, if a service is not found in the local ad-hoc wireless network, the GW service will be retrieved and its address will be set as routing table default gateway to allow the node to access services running outside the local network.

5.4.1 WSAMI and WSDL files

GWAbstract.wsami

```
<?xml version="1.0" encoding="UTF-8"?>
<wsami
        xmlns="http://www-rocq.inria.fr/arles/wsami"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www-rocq.inria.fr/arles/wsami WSAMI.xsd">
 <Definition name="GW" targetNamespace="http://localhost:8080/wsami/">
  <Abstract name="GW">
   <Interface hrefSchema="http://localhost:8080/wsami/GWInterface.wsdl"/>
```

```
  </Abstract>
 </Definition>
</wsami>
```

GWService.wsami

```
<?xml version="1.0" encoding="UTF-8"?>
<wsami
        xmlns="http://www-rocq.inria.fr/arles/wsami"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www-rocq.inria.fr/arles/wsami WSAMI.xsd">

 <Definition name="GW" targetNamespace="http://localhost:8080/wsami/">
  <Service name="GW">
   <Abstract hrefSchema="http://localhost:8080/wsami/GWAbstract.wsami"/>
   <Concrete hrefSchema="http://localhost:8080/wsami/GWConcrete.wsdl"/>
  </Service>
 </Definition>
</wsami>
```

GWInterface.wsdl

```
<?xml version="1.0" ?>

<definitions name="urn:GWService"
             targetNamespace="urn:GWService"
             xmlns:tns="urn:GWService"
             xmlns:typens="urn:GWService"
             xmlns:xsd="http://www.w3.org/1999/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

<!-- message declns -->
  <message name="GetGWIPAddrRequest"/>
  <message name="GetGWIPAddrResponse">
   <part name="addr" type="xsd:string"/>
  </message>

<!-- port type declns -->
  <portType name="GWInterface">
    <operation name="GetGWIPAddr">
      <input message="tns:GetGWIPAddrRequest"/>
      <output message="tns:GetGWIPAddrResponse"/>
    </operation>
  </portType>

<!-- binding declns -->
  <binding name="GWSOAPBinding" type="tns:GWInterface">
    <soap:binding style="rpc"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetGWIPAddr">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="encoded"
                   namespace="http://www.GW.Service"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
                   namespace="htto://Service.org"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>
</definitions>
```

GWConcrete.wsdl

```
<definitions name="urn:GWService"
```

```
            targetNamespace="urn:GWService"
            xmlns:tns="urn:GWService"
            xmlns:typens="urn:GWService"
            xmlns:xsd="http://www.w3.org/1999/XMLSchema"
            xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
            xmlns="http://schemas.xmlsoap.org/wsdl/">
<import namespace="urn:GWService" location="GWInterface.wsdl" />
 <!-- service decln -->
  <service name="GW_Service">
    <port name="GW" binding="tns:GWSOAPBinding">
      <soap:address location="http://localhost:8080/services/GW"/>
    </port>
  </service>
</definitions>
```

6.WSAMI

```
<xsd:schema
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www-rocq.inria.fr/arles/wsami"
        targetNamespace="http://www-rocq.inria.fr/arles/wsami"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >


<xsd:element name="wsami">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Definition" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Definition">
  <xsd:complexType >
    <xsd:choice minOccurs="1" maxOccurs="1" >
       <xsd:element ref="Abstract" />
       <xsd:element ref="Service" />
       <xsd:element ref="Customizer"/>
    </xsd:choice >
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="targetNamespace" type="xsd:anyURI"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Abstract">
  <xsd:complexType >
    <xsd:sequence>
       <xsd:element name="Interface"  type="XMLDocumentType"/>
       <xsd:element name="Conversation" type="XMLDocumentType"  minOccurs="0"/>
       <xsd:element ref="ServiceQoS" minOccurs="0" />
       <xsd:element ref="ConnectorQoS" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" />
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="XMLDocumentType">
  <xsd:attribute name="hrefSchema" type="xsd:anyURI"/>
</xsd:complexType>

<xsd:element name="Service">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Abstract" type="XMLDocumentType"/>
```

```
      <xsd:element name="Concrete" type="XMLDocumentType"/>
      <xsd:element ref="Required" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Required">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="ReqService" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element  name="ReqService">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Abstract" type="XMLDocumentType"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="optional" type="xsd:boolean" default="false"/>
    <xsd:attribute name="instance" type="xsd:boolean" default="false"/>
    <xsd:attribute name="multiple" type="xsd:boolean" default="false"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Customizer">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="QoSCriterion" maxOccurs="unbounded"/>
      <xsd:element name="Local" type="XMLDocumentType"/>
      <xsd:element name="Remote" type="XMLDocumentType"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
   <xsd:attribute name="Colocation" type="xsd:boolean" default="false"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="ServiceQoS">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="QoSCriterion" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="ConnectorQoS">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="QoSCriterion" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="QoSCriterion">
  <xsd:complexType>
    <xsd:attribute name="name" type="QoSType"/>
  </xsd:complexType>
</xsd:element>

<xsd:simpleType name="QoSType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="transactionalService"/>
    <xsd:enumeration value="security"/>
    <xsd:enumeration value="saveBandwidth"/>
  </xsd:restriction>
</xsd:simpleType>
```

```
</xsd:schema>
```

7.FAQ

- **When I execute the command $WSAMIDIR/dist/pda/bin/runcvm.sh to start the WSAMI server on the PDA, the Java VM generates the error "`/opt/QtPalm/j2me/bin/cvm: relocation error in mknod: libifacemgr.so`".**

  If file /dev/apm_bios exists, delete it. Otehrwise, if it does not exist, create it with the following command:

  ```
  mknod -m a=rw /dev/apm_bios c 10 134
  ```

Bibliography

STEI
: J. Steinberg and J. Pasquale. A Web middleware architecture for dynamic customization of content for wireless clients. In Proceedings of the WWW'02 Conference, 2002

ARCHGUIDE
: WSAMI Middleware Architecture guide