MSDA User Guide

DRAFT

V 0.9

20/01/2006

authors:
agnes.de_la_chapelle@inria.fr
pierre-guillaume.raverdy@inria.fr

Table of content

1 Introduction

The pervasive/ubiquitous computing vision can now be realized, thanks to the deployment of wirelessly accessible services (i.e., software components that can be accessed remotely through a specific interface) into the environment to support mobile users in their daily life. Mobile devices are also now powerful enough not only to access these services, but also to host applications providing similar services to their environment. Unfortunately, pervasive computing environments are highly heterogeneous, thus limiting actual interactions between users and services:

- The use of various middleware platforms (e.g., UPnP, Jini, Web services) results in the concurrent use of discovery and access protocols that do not directly interoperate with each other, further limiting the number of accessible services.
- The use of various wireless technologies (e.g., cellular networks, WiFi, or Bluetooth) and network management models (e.g., ad hoc- or infrastructure-based) results in many independent networks being available to users at a location. As users can only be connected to a limited number of networks at the same time (often a single one), many services may not be accessible (i.e., not IP reachable).

The *Multi-Protocols Service Discovery and Access* (MSDA) middleware platform aims to support service discovery and access in heterogeneous networks. The MSDA allows clients using a specific SD protocol (supported by MSDA), to discover services in any of the supported protocols and in any of the networks connected to the Global Network. MSDA supports both wired and wireless networks, as well as infrastructure-based (managed) and infrastructure-less (ad hoc) networks. MSDA provides also a simple remote access to the discovered services assuming that the clients know the services' stubs.

1.1 Service Discovery and Access

Over the years, many academic and industry-supported *Service Discovery Protocols* (SDPs) have been proposed. These SDPs differ on their organization (i.e., centralized vs. decentralized vs. hierarchical), methods (push vs. pull vs. symmetric), and timing (aperiodic vs. periodic). SDPs usually have their own proprietary format for service description and service query. SDPs usually target a specific area network (e.g., PAN, LAN, or WAN) and are tied to specific technologies (e.g., SDP for Bluetooth).

Leading SDPs use a pull-based approach, which may be: (i) centralized where one or a few repositories store the descriptions of the services available in the network (e.g., Jini, SLP, and UDDI), (ii) distributed where service providers store locally service descriptions and clients send service requests using the broadcast functionality provided by the network (e.g., SSDP and JXTA-Search), or (iii) a combination of both.

Many access protocols have also been proposed, with the early one based on binary formats (e.g., CORBA CDR), while recent ones are based on XML, such as XML-RPC or SOAP.

In MSDA, we recognize the prevalence of existing service discovery and access protocols. Our goal is to enable clients to discover and access the MSDA platform with the most appropriate set of protocols based on their needs and environment. We also assume that services are registered using existing protocols.

1.2 Heterogeneous Networks

In MSDA, we consider an all-IP networking environment (i.e., all components use IP protocols to provide transport), with heterogeneity arising from:

- Variations in the sets of IP-level configuration and communication functionalities supported in the networks (e.g., IP multicast, DHCP),

- The different administrative domains of the networks (e.g., public/open versus restricted/secure),
- The different classes of targeted applications.

In this context, we define an *Elementary Network* (EN) to be a set of devices that belong to the same administrative domain and share the same IP-level set of communication functionalities (e.g., home network, hotspot). Depending on their underlying characteristics, ENs may vary in terms of size and/or dynamics (i.e., stable versus highly dynamic membership). We further define a *Global Network* (GN) to be a dynamic composition of ENs, with some of the devices acting as bridges/gateways between two or more ENs. Devices in the same GN, but in different ENs may not be able to communicate (e.g., no global IP routing, security restrictions). We consider the Internet and cellular networks as conduits enabling the interconnection of various ENs and also consider that a *discovery domain* (i.e., the set of services being available to a client using a given SD protocol) is contained within an EN. In our scenario, the music labels' hotspots, the ad hoc network, and Ben's home network are all ENs (See Figure 1).

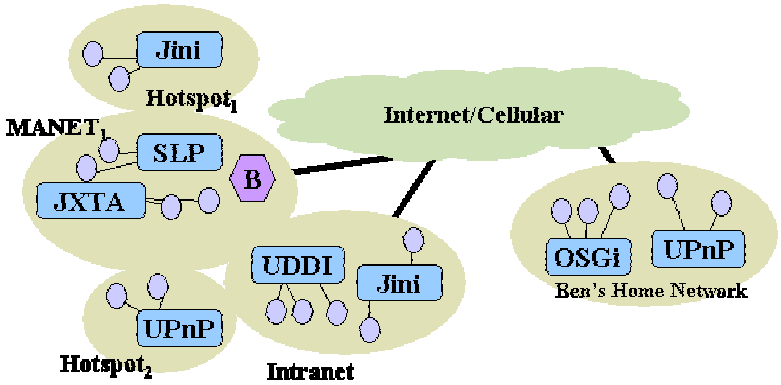

Figure 1: Global Network Example

1.3 Outline

This document describes the installation, configuration, and use of the MSDA middleware platform. In Section 2, we list the different software packages required to compile and use the MSDA platform. In Section 3, we details the installation and compilation of the MSDA platform under Eclipse. In Section 4 we detail how to start the MSDA platform and the format of the startup/configuration files. Finally in Section 5 we list the most common problems encountered during the compilation or the startup of the MSDA platform.

2 MSDA Software Overview

2.1 System requirements

In order to compile and run the MSDA middleware platform, the following components are required.

- Windows or Linux operating systems
- Java 2 Standard Edition v1.4.2 or higher
- Eclipse IDE

Furthermore, a number of Java packages are also required:

- **JaxMe packages** (http://ws.apache.org/jaxme/):
  - jaxme2.jar
  - jaxme2-rt.jar
  - jaxmeapi.jar
  - jaxmejs.jar
  - jaxmepm.jar
  - jaxmexs.jar
  - jaxrpc-api.jar
  - qname.jar
  - saaj.jar
- **Axis/Xerces packages** (http://ws.apache.org/axis/)
  - wsdl4j.jar
  - xercesImpl.jar
  - xml-apis.jar
  - xmlParserAPIs.jar
- **Misc**.:
  - csoap.jar (provided with MSDA)
  - httpParser.jar (provided with MSDA)
  - secureMulti.jar (optional package for secure election, provided with MSDA)
  - soaptools.jar (package from the SOAPTools project)
  - Ant (ant-1.5.4.jar)
  - Log4j (log4j-1.2.8.jar)

A number of Java packages are also required to support various SD protocols

- **Siemens UPnP package**[1] (required for UPnP protocol support, must be downloaded):
  - upnp101.jar
- **PDP packages**[2] (required for PDP protocol support)
  - PDP.jar
  - PDPClient.jar
- **SLP packages**[3] (required for SLP protocol support):
  - openSlp.jar
  - slp.jar
- **Ariadne package**[4] (required for Ariadne protocol support):

[1] available at http://www.plug-n-play-technologies.com/

[2] available at http://www.it.uc3m.es/celeste/pdp/):

[3] available at http://www.openslp.org/download.html

- osdp4m.jar

- **OSGi packages**[5] (required for OSGi home gateway interaction):
  - oscar.jar
  - oscar-1.0.5.jar
  - osgi.jar
  - axisBundle.jar
  - moduleloader.jar

2.2 MSDA packages description

The following files should be downloaded and imported into the Eclipse workspace for compilation:

- **MSDACore.zip** for the **MSDACore project** that contains all the Java source code for the MSDA platform.
- **MSDAPlugins.zip** for the **MSDAPlugins project** that contains implementations of MSDA plugins for various discovery protocols (i.e., UPnP, SLP, PDP, and Ariadne).
- **MSDATests.zip** for the **MSDATests project** that contains various tests and sample programs.
- **SOAPExt.zip** for the **SOAPExt project** that is used to automatically generate additional code in the Java classes generated by JaxMe.
- **SOAPTools.zip** for the **SOAPTools project** that implements the SOAP processing required by the SOAP implementation of the MSDA Service.
- **MSDASupportingFiles.zip** that contains various configuration files and additional Java packages used for the deployment and use of the MSDA platform.

We now detail the content of the different Java projects.

2.2.1 MSDACore Project

This project contains the Java classes of the MSDA platform in the *org.msda.* class hierarchy. It contains the following packages:

- **org.msda.components.bridge** contains all the classes related to the MSDA Bridge component.
- **org.msda.components.core** contains MSDADaemon that manages the election process, NetworkMonitor that manages the emission/reception of presence beacons and MSDA messages between the different components in an elementary network, and MSDANetwork that manages the registration of MSDA components into specific communication groups (each is managed by the CommunicationManager inner class).
- **org.msda.components.interfaces** contains all the component interfaces used in MSDA (i.e., plugin, bridge, manager, network group…).
- **org.msda.components.manager** contains all the classes related to the MSDA Manager component.
  - **org.msda.components.manager.client** contains specific classes from the MSDA Manager related to the management of MSDA clients (through the MSDA Service).

[4] available at http://www-rocq.inria.fr/arles/download/

[5] available at http://www.oscar.org/

- **org.msda.components.manager.object** contains specific classes from the MSDA Manager representing the objects used in the different tasks of the manager (discovery and access).
- **org.msda.components.manager.task** contains specific classes from the MSDA Manager for the processing of the different tasks of the manager (discovery and access).
- **org.msda.components.plugins** contains the parent MSDA plugin class (AbstractPlugin) as well as generic methods to be implemented by all plugins.
- **org.msda.components.plugins.http** contains the HTTP server and processing methods for incoming client messages received by the MSDA service associated with the SD plugin.
- **org.msda.components.plugins.msdaservice** contains two implementations of the MSADA service (SOAP-based and plain XML MSDA messages). These two implementations use HTTP as a transport protocol.

- **org.msda.components.utils** contains manager and thread classes for local components within the stack.
- **org.msda.context** contains all the classes for the management (collection, exchange, and processing) of context information.
- **org.msda.framework** contains the main class that process the startup/configuration file and the start the MSDA platform.
- **org.msda.gui** (and its subpackages) contains all the classes for the GUI panels used by the MSDA platform to provide information on the MSDA Manager and Bridges.
- **org.msda.msd** and **org.msda.msd.impl** are automatically generated by JaxMe and contains all the description of all the information exchanged by MSDA components as well as marshalling/unmarshalling methods between Java and XML.
- **org.msda.MSDClientAPIService** contains the API of the MSDA service registered in the different SD protocols.
- **org.msda.util** (and its subpackages) contains various classes to support MSDA components (e.g., global definitions, processing of propagation lists, management of multicast communications, security…).

In addition to the source folder, this project also contains a **jaxme** folder that contains two XML schemas (*msd.xsd* and *net_profile.xsd*) as well as two ant script (*build_msd.xml* and *build_netprofile.xml*):

- The *msd.xsd* XML schema is used to automatically generate with JaxMe the Java classes located in org.msda.msd.*. These classes corresponds to the objects exchanged between MSDA components, and contains the marshalling and unmarshalling code for XML <> Java serialization.
- The *profile_network.xsd* schema is for the automatic generation of the Java classes related to the network profile.
- Each ant script (build_msd.xml and build_netprofile.xml) is used to launch JaxMe and generate the relevant Java classes for the two above schemas.

2.2.2 MSDAPlugins

The MSDAPlugins project contains several implementations of SDA Plugins:

- **org.msda.plugins.ariadne** for the Ariadne SD protocols for MANETS.
- **org.msda.plugins.ariadne.osgi** for the OSGi home gateway.

- **org.msda.plugins.ariadne.pdp** for the PDP SD protocol for 1-hop ad hoc network.
- **org.msda.plugins.ariadne.slp** for the SLP protocol.
- **org.msda.plugins.ariadne.upnp** for the SSDP protocol that is part pf UPnP. This plugin uses the Siemens implementation of the UPnP stack.

Each plugin extends the *AbstractPlugin* class in the *org.msda.components.plugins* package.

2.2.3 MSDATests

The MSDATest project contains multiple implementations of client applications that interact with MSDA to discover and access services in the Global Network. Implementations are provided for the Ariadne, PDP, SLP, and UPnP discovery protocols (in the current implementation, all services are SOAP services).

- **org.msda.clientsamples.ariadne** (and its subpackages)
- **org.msda.clientsamples.pdp.rfid**
- **org.msda.clientsamples.slp** (and its subpackages)
- **org.msda.clientsamples.upnp** (and its subpackages)

In addition, several packages are provided to test internal components of the MSDA platform as well as group communications.

- **org.msda.tests** (and its subpackages)

Finally, a UPnP servce using the Siemens UPnP stack is provided.

- **org.msda.upnpdevice.sample** (and its subpackages)

2.2.4 SOAPExt

SOAPExt is a compile-time application used to automatically extend the source code generated by JaxMe (located in org.msda.msd). Specifically, this project adds the necessary methods to request the MSDA type associated to the Java class. This additional code is necessary for the SOAP-MSDA conversion.

2.2.5 SOAPTools

SOAPTools iare run-time tools used by the AbstractPlugin class to process incoming SOAP messages from a client accessing the MSDA Service. SOAPTools convert the SOAP messages (e.g., SOAP data structures) into the corresponding MSDA message (e.g., Java classes).

Note: If the MSDA data schema is updated (see Section 3.2) with new data structures, this project must be updated accordingly otherwise incoming SOAP messages may be rejected.

3 Getting Started

In this section, we detail the installation, compilation, and startup of the MSDA platform.

3.1 MSDA Installation (Eclipse)

The following four projects should be created: MSDACore, MSDAPlugins, SOAPExt, SOAPTools. For each project, the source code file system should be imported from the corresponding archive file. Each project classpath should be configured (in particular the location of the additional packages). To avoid duplication and potential version conflicts, is it recommended to copy all additional jar components in the MSDACore project's lib directory.

In this documentation, we assume that the MSDA executable packages are installed in the $INSTALLDIR directory and that the MSDA platform is started from the command line.

The following folders/files from the MSDA*SupportingFiles.zip* archives should also be copied in the execution folder.

- MSDDeploy.wsdd should be copied into the $INSTALLDIR/configuration folder
- MSDA startup files should be copied in the $INSTALLDIR folder
- Context/profile files should be copied in the $INSTALLDIR folder

3.2 MSDA Compilation

To ease the development of the MSDA platform, we use the Open Source project JaxMe[6] to automatically generate the Java classes (along with the XML <> Java marshalling methods) for the objects exchanged between MSDA components (i.e., discovery requests, service descriptions, control messages, access requests and response messages).

The *msd.xsd* schema file located in the *jaxme/schema* folder of the *MSDACore* project contains the XML description of all these data types and the code generated by JaxMe is located in the packages org.msda.msd amd org.msda.msd.impl. If you want to change the content of MSDA messages, it is necessary to change the XML schema and process it. Directly changing the automatically generated classes in `org.msda.msd.*` may generate errors (e.g., invalid name or type in the unamrshaler of another class).

Do not directly modify the classes in the packages org.msda.msd and org.msda.msd.impl.

3.2.1 Basic compilation

If no change is made to the MSDA schema, only the MSDACore and MSDAPlugins need to compiled and exported as jar files into the $INSTALLDIR directory.

3.2.2 MSDA schema update (simple)

If the MSDA schema is updated but (i) no new data type is created, and (ii) names of existing data types are not changed, then only a simple automatic code generation is necessary.

The Ant script `jaxme/schema/build_msd.xml in the MSDACore project` is used to automatically generate the Java classes based on this schema. However, these classes require additional processing (some methods need to be added to the automatically generated classes). The SOAPExt project is used to modify the generated Java classes and make them compatible

[6] http://ws.apache.org/jaxme/

with CSOAP, which is used for the client access to the MSDA Service. That project uses Java reflection to add some required information.

We assume you use Eclipse to open the MSDA project and the SOAPExt project. Both project are also in the same workspace

Modifying the MSDA schema requires the following steps:

1. **Modify** the schema (msd.xsd);
2. **Execute** the JaxMe Ant script (build_msd.xml) with the "Ant build" function.
3. **Refresh** the `org.msda.msd` package (recompile the project and regenerate the *.class files);
4. **Export** the `org.msda.msd` and `org.msda.msd.impl` subpackages in the jaxmeClasses.jar library of the SOAPExt project;
5. **Execute** the soap.SOAPExt main class of the SOAPExt project with the arguments `../MSDACore/src/org/msda/msd/impl urn:MSDClientAPIService`
6. **Refresh** the `org.msda.msd` package (SOAPExt transparently updated the generated classes, therefore the MSDACore project needs to be recompiled.

To check that the code generation has been completed, open a Java class file from the `org.msda.msd.impl` package that ends with "…`Impl.java`", (e.g., `AddressImpl.java`). Then checks that a `//Type metadata` section has been added to the code similar to the one below:

```
   //Type metadata
   private static csoap.wsdl.TypeDesc typeDesc=new csoap.wsdl.TypeDesc(AddressImpl.class);

   static {
   typeDesc.setXmlType(new javax.xml.namespace.QName("urn:MSDClientAPIService","AddressImpl"));
   csoap.wsdl.ElementDesc elementField;

   elementField=new csoap.wsdl.ElementDesc();
   elementField.setFieldName("_location");
   elementField.setXmlName(new javax.xml.namespace.QName("","_location"));
   elementField.setXmlType(new
javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema","string"));
   typeDesc.addFieldDesc(elementField);

   elementField=new csoap.wsdl.ElementDesc();
   elementField.setFieldName("_port");
   elementField.setXmlName(new javax.xml.namespace.QName("","_port"));
   elementField.setXmlType(new
javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema","integer"));
   typeDesc.addFieldDesc(elementField);
   }

   /**
    * Return type metadata object
    */
   public static csoap.wsdl.TypeDesc getTypeDesc() {
       return typeDesc;
   }
```

3.2.3 MSDA schema update (complex)

If new elements are added to the MSDA schema, or if the names of existing elements are changed, additional changes (to the ones described in the previous section) are necessary. Specifically, all the classes processing SOAP messages in MSDA need to be updated to take into account the new names or elements.

- In the package org.msda.MSDClientAPIService
  - MSDClientAPIBindingStub
  - MSDClientAPIPortType
- In the configuration folder:
  - MSDDeploy.wsdd
  - MSDServiceInterface.wsdl
  - wsdd.conf

3.3 MSDA Startup

In this section, we assume that the MSDA platform is started from the console (and from the execution directory).

3.3.1 MSDA components

The MSDA platform can easily be started with the following command line:

```
>java.exe -Djava.ext.dirs=. -cp MSDACore.jar org.msda.framework.FrameworkMain
config_dev1.xml
```

All the setup information (i.e., network interfaces and multicast addresses to use, startup of MSDA Manager and/or Bridge, SD Plugins to use…) is located in the XML configuration file (e.g., config_dev1.xml). The format of the configuration file is detailed in Section 4.

Figure 2 Election panel

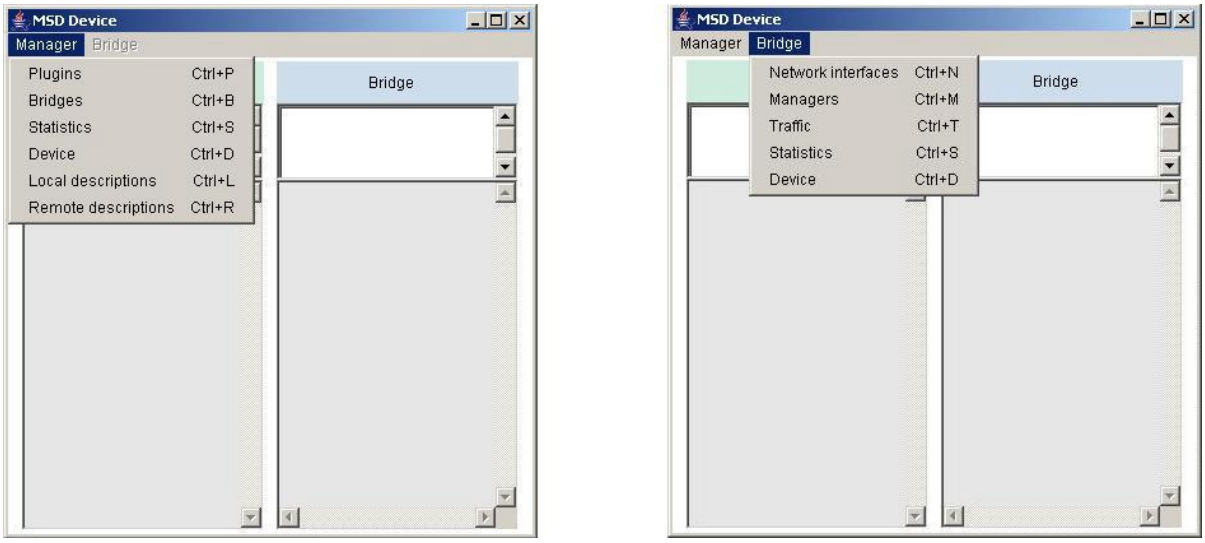

Figure 3 : Manager and bridge panels

When the MSDA stack is started, a first GUI window (see figure 4) is displayed providing feedback on the election process. Once the election process is completed, a new GUI window (see figure 5) containing two panels appear. The left panel provides information on the MSDA Manager component (if started) while the right panel provides information on the MSDA Bridge components (if any).

3.3.2 Starting a service

3.3.2.1 UPnP Service

A sample UPnP Service can be started from the MSDATest project.

```
>java.exe -Djava.ext.dirs=. -cp MSDATests.jar
org.msda.upnpdevice.sample.UpnPSampleDevices
```

3.3.2.2 SLP Service

In order to start an SLP service (and the SLP plugin as well), an active SLP registry is required in the local area network. An open source implementation of the SLP registry can be found at http://www.openslp.org/

An SLP service can then be registered directly from the command line or from an application using the SLP API.

3.4 Demonstration

The *MSDASupportingFile.zip* archive contains an *MSDA* folder with different libraries, scripts and configuration files for testing the MSDA framework (the UPnP library must be downloaded from Siemens):

- *Confif_dev1.xml* configures an MSDA Manager for network Network1 with an SLP SD Plugin
- *Confif_dev2.xml* configures an MSDA Bridge between networks Network1 and Network2.
- *Confif_dev1.xml* configures an MSDA Manager for network Network2 with a UPnP SD Plugin

- *start_client_slp.bati* that starts an SLP graphical client
- *start_client.bat* that starts a UPnP graphical client.
- *start_msda.bat* that starts an MSDA stack based on a configuration file given in parameter.
- *start_service.bat* that starts a UPnP test service.

The demonstration steps are:

1. Start the SLP registry (slpd.exe).
2. Launch 3 MSDA stacks with the three configuration files mentioned above.
3. Launch the UPnP service.
4. Launch the SLP graphical client.

After these three steps, 4 windows should be created as shown in Figure 4. The Manager for Network 2 should also have (at least) 3 service descriptions (see Figure 5).

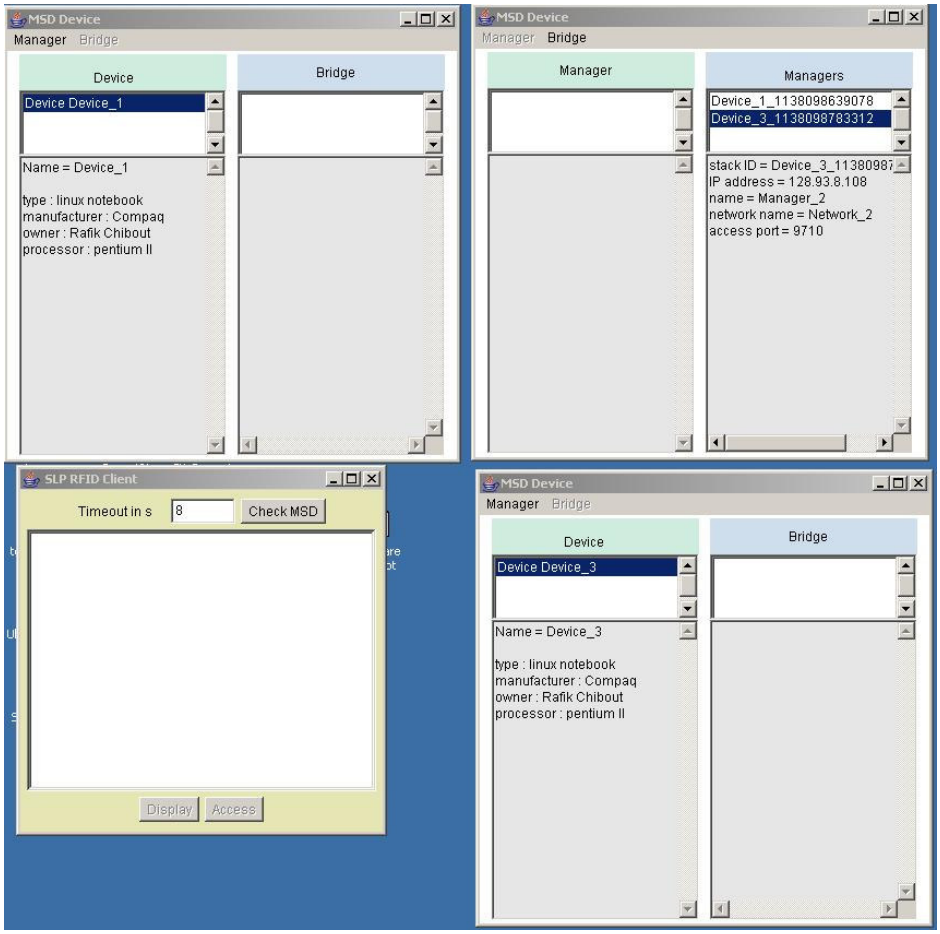

Figure 4: Demonstration setup

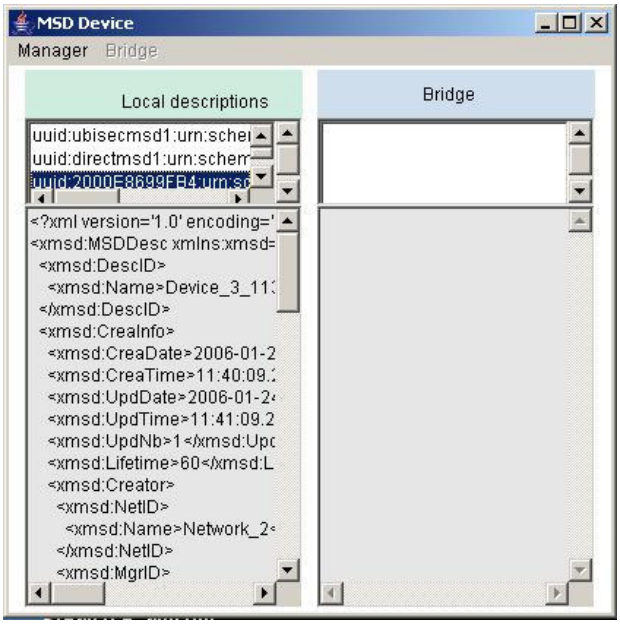

Figure 5: UPnP services discovered

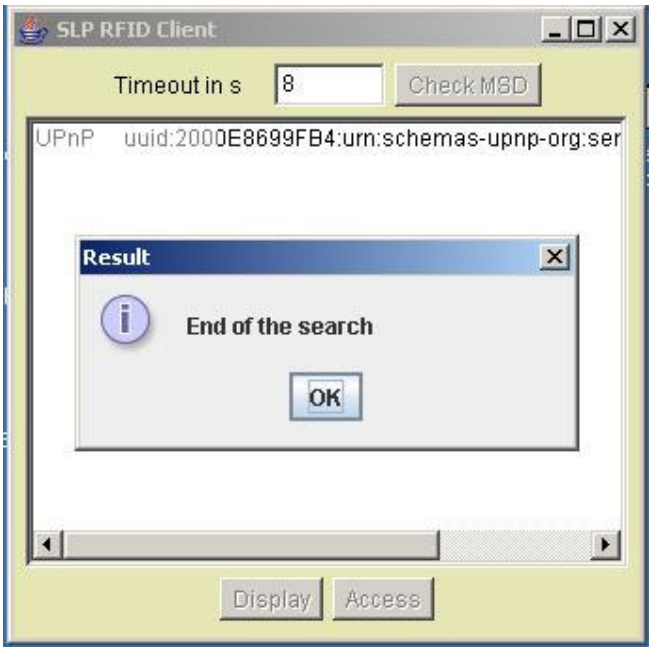

Figure 6: MSDA Service Discovery

After all applications have started, press the button "Check MSD" on the SLP graphical client. The client then connects to the MSDA Service recorded in the SLP registry and sends a discovery request for an RFIDService. After the discovery timeout expires, one service description should be available (see Figure 6). Press the "Access" button to go to the access panel (Figure 7). Pressing the "Get Infos" button will then send an MSDA service access request to the remote service through the two MSDA Managers and the MSDA Bridge and return some information.

MSDA execution logs are available for the three stacks (Device_1.log, Device_2.log, Device_3.log).

Figure 7: MSDA Service Access

4 MSDA Startup File

The MSDA platform is provided an XML configuration file at startup that contains the information related to the device, the network interfaces and if needed, the manager properties. In the following, we describe in details the format of this file.

The XML configuration file is composed of a Device section providing profile information on the device, a Framework section that contains all the MSDA configuration information, and multiple Networks that provide different network configuration that *may* be used by this MSDA stack.

```
<Device>
    <Name>Device_1</Name>
    <Params type="type" value="linux notebook"/>
    <Params type="manufacturer" value="Compaq"/>
    <Params type="owner" value="R.C."/>
    <Params type="processor" value="pentium II"/>
</Device>
```

The `<Device>` node first contains a device name that should be unique within the elementary network, and then an unbounded list of Params. Each Param contains two string (type and value). This enables the definition of the device profile.

```
<Framework>
    <ActiveNetwork>Network_1</ActiveNetwork>
    <agreeBridge>0</agreeBridge>
    <contextMgmt>true</contextMgmt>
    <Manager>
         <Name>Manager_1</Name>
         <NetworkName>Network_1</NetworkName>
         <RemoteAccessPort>9009</RemoteAccessPort>
         <DirectMSDPort>8509</DirectMSDPort>
         <Plugins>
             <Name>Plugin UPnP Siemens 1.0</Name>
             <Creator>Siemens</Creator>
             <ClassName>plugins.upnp.MSDUPnPPlugin</ClassName>
             <HTTPServerPort>9506</HTTPServerPort>
         </Plugins>
         <Params type="getServicesTimeout" value="8000"/>
         <Params type="serviceAccessTimeout" value="5000"/>
         <NetworkCtx>./configuration/ctxHS.xml</NetworkCtx>
    </Manager>
</Framework>
```

The `<Framework>` node first lists the names of the networks this device is connected to (for the list of networks at this end of the configuration file). At least one network is mandatory. Then the configuration file indicates if an MSDA Bridge component should be started or not, and then whether this stack should manage context information..The MSDA Bridge comonent interconnect all active networks.

If this device accepts to be an MSDA Manager, the <Framework> node then contains a <Manager> node. This node first assign a unique name (within the elementary network) to this manager and then indicates the network potentially managed by this MSDA Manager. Only one network should be listed, but several <Manager> nodes may exist within a <framework> node.

Then two port numbers are provided for managing remote access connection requests and direct MSDA access (plain XML MSDA messages over TCP sockets). Then a list of <Plugins> nodes is provided that lists the different SD plugins this MSDA manager handles. Each <Plugin> node first contains the plugin name, its creator, the class name to instantiate, and the HTTP port for accepting access requests.

Finally, the Manager node contains a list of parameters and optionally the location of the XML file containing profile information about the network handled by this MSDA Manager.

A stack may have both one or more MSDA Manager components and one MSDA Bridge component. In the example below, a simple MSDA stack is defined that only contains an MSDA Bridge component for two networks.

```
<Framework>
    <ActiveNetwork>Network_1</ActiveNetwork>
    <ActiveNetwork>Network_2</ActiveNetwork>
    <agreeBridge>1</agreeBridge>
    <contextMgmt>true</contextMgmt>
</Framework>
```

If an MSDA Manager can provide several plugins, the file would contain several `<Plugins>` node as it is shown below.

```
<Plugins>
    <Name>Plugin UPnP Siemens 1.0</Name>
    <Creator>Siemens</Creator>
    <ClassName>plugins.upnp.MSDUPnPPlugin</ClassName>
    <HTTPServerPort>9504</HTTPServerPort>
    </Plugins>
<Plugins>
    <Name>Plugin OpenSLP</Name>
    <Creator>OpenSLP</Creator>
    <ClassName>plugins.slp.MSDSLPPlugin</ClassName>
    <HTTPServerPort>9002</HTTPServerPort>
</Plugins>
```

The `<Framework>` node also enables some parameters to be defined with a `<Params>` node.

- `getServicesTimeout` : defines the timeout to perform a “getServices”
- `serviceAccessTimeout` : defines the timeout to perform a “serviceAccess”

```
<Params type="getServicesTimeout" value="8000"/>
<Params type="serviceAccessTimeout" value="5000"/>
```

In the configuration files, several networks can be defined but only the active networks will be taken into account..

```
<Networks>
    <Name>network_2</Name>
    <Interface>eth0</Interface>
    <ControlGroup>239.255.255.1</ControlGroup>
    <ControlPort>9101</ControlPort>
```

```
      <DataGroup>239.255.255.101</DataGroup>
      <DataPort>9101</DataPort>
      <BridgeRemoteAccessPort>9201</BridgeRemoteAccessPort>
      <Params type="type" value="WiFi"/>
      <Params type="management" value="ad hoc"/>
      <Params type="owner" value="none"/>
      <Params type="bandwidth" value="11 Mbps"/>
  </Networks>
```

Each network node first provides the name of the networks, then the identification of the network interface to use followed by two pairs of IP addresse and port for the control and data communication groups. The two IP addresses should be multicast IP addresses and the two ports may either be identical or different. This is followed by the port number for incoming remote access requests and a list of parameters.

The tables below summarize the elements required for each node of the startup file.

Framework node	
`ActiveNetwork`	The name of the active network
`agreeBridge`	:If the value is 1, then the device agrees to become a bridge
`contextMgmt`	If the value is true, then the bridge or manager will provide context management, otherwise no context will be verify and everything will be allowed.
`Manager`	Optional definition(s) of MSDA Managers

Manager node	
`Name`	the name of the manager
`NetworkName`	the name of the network
`RemoteAccessPort`	the port for the remote access
`DirectMSDPort`	the port of the direct MSDA service for socket connections
`Plugins`	the children of this node provide all the require information to start a plugin
`Params`	any needed parameter with its type and name
`NetworkCtx`	Filename for device context information

Plugin node	
`Name`	the name of the plugin
`Creator`	the creator name of the plugin
`ClassName`	the name of the class to instantiate dynamically
`HTTPServerPort`	the port number used by the HTTP server

Network node	
`Name`	the name of the network

`Interface`	the identification of the network interface
`ControlGroup`	the IP address used by the multicast group for the election
`ControlPort`	the port number used by the multicast group for the election
`DataGroup`	the IP address used by the multicast group for the MSDA
`DataPort`	the port number used by the multicast group for the MSDA
`BridgeRemoteAccessPort`	the port number used for service access
`Params`	any needed parameter with its type and name

5 MSDA-aware Client Operations

5.1 Service Discovery

After configuring the manager and bridges, you can search for services with an MSDA-aware client. The client can connect to the MSDA service using socket connections or SOAP connections.

Case socket connection :

- searching for the MSDA service in the client's protocol (UPnP, SLP, …);
- making a MSDA request for service discovery;
- making a MSDA message containing the request ;
- sending the message to the MSDA service;

Case SOAP connection :

- searching for the MSDA service in the client's protocol (UPnP, SLP, …);
- making a MSDA request for service discovery;
- invoking the getServices method of the MSDA service.

5.1.1 MSDA request

The MSDA request contains :

- the client's timeout to perform the request (`ReqCreationInformation.setLifetime(..)`);
- the name of the searched service (`RequestService.setName(..)`);
- the methods of the service if you want to select more specifically a service;
- a client ID and a request ID sets to -1 if it is the first time the request is sent;
- some context rules/information

```
ReqProcessingInformation rpi = new ReqProcessingInformationImpl();
ReqCreationInformation rci = new ReqCreationInformationImpl();
Creator creator = new CreatorImpl();
UbisecID id = new UbisecIDImpl();
id.setName( "Client" );
creator.setMgrID( id );
creator.setNetID( id );
Calendar cal = Calendar.getInstance();
rci.setDate( cal );
rci.setTime( cal );
rci.setLifetime( BigInteger.valueOf( timeout ) );// in secondes
rci.setCreator( creator );
rpi.setCrea( rci );

// Name of the searched service
RequestService requestService = new RequestServiceImpl();
requestService.setName("serviceName");

ReqDescriptionInformation reqDescription = new ReqDescriptionInformationImpl();
reqDescription.setService(requestService);

MSDRequest msdRequest = new MSDRequestImpl();

msdRequest.setDescInfo(reqDescription);
msdRequest.setProcInfo( rpi );

PropagationInformation pi = new PropagationInformationImpl();
msdRequest.setPropaInfo( pi );
```

```
msdRequest.setClientID( BigInteger.valueOf( -1 ) );
msdRequest.setReqID( BigInteger.valueOf( -1 ) );
```

MSDA message to send to the MSDA service for **socket connections**

```
MessageContent content = new MessageContentImpl();
content.setRequest( msdRequest );

MSDMessage mess = new MSDMessageImpl();
mess.setSender( id );
mess.setType( BigInteger.valueOf( InfoType.Message.CLIENT_NEW_REQUEST ) );
mess.setContent( content );
```

Invoking the MSDA service for a discovery request using **SOAP messages**.

```
MSDDescriptionImpl[] results = msdService.getServices( msdRequest );
```

5.1.2 Getting the results

Whatever the client connection (socket or SOAP), the MSDA manager will return the first results as soon as at least one is received or is in its cache. If the client's timeout has expired, the result will be null. But if the client's timeout has not expired, then either the client is satisfied with the results, or the client can still retrieve periodically the results until the request timeout expires.

In that goal, the client has to send a request with the correct client ID and request ID that identify the running request. These two identifiers can be retrieved from a previous result.

```
MSDDescription[] res = msdService.getServices( msdRequest );

// Hack : <DeviceID> contains clientId#requestId
CreationInformation creation = res[i].getCreaInfo();
UbisecID deviceId = creation.getCreator().getDvcID();
String deviceName = deviceId.getName();
int index = deviceName.indexOf("#");
clientId = Integer.parseInt( deviceName.substring(0, index) );
requestId = Integer.parseInt( deviceName.substring( index+1 ) );
```

For a socket connection, keeping on retrieving the results will be done sending the request with the client ID and request ID to the socket with a MSDA message of type CLIENT_RUNNING_REQUEST :

```
MessageContent content = new MessageContentImpl();
content.setRequest( msdRequest );

MSDMessage mess = new MSDMessageImpl();
mess.setSender( id );
mess.setType( BigInteger.valueOf( InfoType.Message.CLIENT_RUNNING_REQUEST ) );
mess.setContent( content );
```

For a SOAP connection, keeping on retrieving the results will be done sending the request with client ID and request ID :

```
MSDDescriptionImpl[] results = msdService.getServices( msdRequest );
```

5.1.3 Context information / context rules

If the network supports context management, the MSDA request can be completed by some context rules and/or context information.

A request can contain user context information, service context information and/or network context information.

```
ContextInformationTable ciNet = new ContextInformationTableImpl();
ContextElement elt = ContextUtil.getInstance().createCtxElement("Hop", 0 );
ciNet.getCITable().add( elt );

ContextInformationTable ciSvc = new ContextInformationTableImpl();
elt = ContextUtil.getInstance().createCtxElement("Quality", "High" );
ciUser.getCITable().add( elt );

ContextInformation ci = new ContextInformationImpl();
ci.setCINet( ciNet );
ci.setCISvc( ciSvc );

msdRequest.setCtxInfo(ci);
```

A request can contain user context rules, service context rules and/or network context rules.

```
// Network ContextRules   HOP < 5
TypeValue compValue = new TypeValueImpl();
compValue.setInt( BigInteger.valueOf(5) );
ComparisonNode compNode = new ComparisonNodeImpl();
compNode.setName(NetworkTemplate.HOP);
compNode.setOp("L");
compNode.setVal(compValue);
ContextNode ctxNode = new ContextNodeImpl();
ctxNode.setCa(compNode);

ContextRulesTree crNet = new ContextRulesTreeImpl();
crNet.setRoot(ctxNode);

// Service context rules  (AND)

TypeValue v1 = new TypeValueImpl();
V1.setInt( BigInteger.valueOf(2) );
ComparisonNode comp1 = new ComparisonNodeImpl();
comp1.setName("CtxValue");
comp1.setOp("!=");
comp1.setVal(v1);

TypeValue v2 = new TypeValueImpl();
V2.setStr("Medium");
ComparisonNode comp2 = new ComparisonNodeImpl();
comp1.setName("CtxQuality");
comp1.setOp("=");
comp1.setVal(v1);
```

```
ContextNode n1 = new ContextNodeImpl();
n1.setCa(comp1);
ContextNode n2 = new ContextNodeImpl();
n2.setCa(comp2);
CombinationNode rootCbi = new CombinationNodeImpl();
rootCbi.setOp("AND");
rootCbi.setCN1( n1 );
rootCbi.setCN2( n2 );

ContextNode rootNode = new ContextNodeImpl();
rootNode.setCi( rootCbi );

ContextRulesTree crSvc = new ContextRulesTreeImpl();
crSvc.setRoot(rootNode);

ContextRules ctxRule = new ContextRulesImpl();
ctxRule.setCRSvc(crSvc);
ctxRule.setCRNet(crNet);

msdRequest.setCtxRules( ctxRule );
```

5.2 Service Access

The current iprototype implementation of the MSDA platform only supports service access for SOAP services. Furthermore, the client application also needs to have the service stub. However, MSDA does not require a specific implementation of the service and is not required to have a service stub..

Accessing a service is only possible after an MSDA search has returned the MSDA description of the service to the client. Indeed, the client's MSDA Manager maintains a temporary association between the MSDA description returned to the client and a (temporary, local) service entry point. As the client sends its access request to this custom service entry point, the MSDA Manager automatically knows all the information (e.g., propagation list, service SDP, …) required to send the access message to the end service. This temporary association permits the direct retransmission of the access messages by the MSDA Manager without requiring the explicit creation by the client of a communication channel to the end service. The client must therefore first perform an MSDA discovery request, and must start to access the service before the association timeout expires.

The client initialize the SOAP service stub with the temporary endpoint reference found in the MSDA service description. This endpoint reference is composed of the MSDA service IP address, the service endpoint reference in its network and the identification of the returned MSDA Description. We use the "/" to separate the different strings. The MSDA Manager separate all these information and, with the description identification, knows where to find the required information.

MSDAAddress+"/serviceaccess/"+ServiceAddress+"/"+descID

6 Most common problems

6.1 Missing MSDDeploy.wsdd

Problem: When launching MSDA an error is reported claiming that MSDDeploy.wsdd is missing.

```
   [java] The args attribute is deprecated. Please use nested arg elements.
   [java] Will start MSDDeamon
   [java] Not an MSD Bridge
   [java] - Total SD time
   [java] 0 [main] INFO total_msd_processing_time.log  - Total SD time
   [java] Found Network context 1
   [java] START PLUGIN
   [java] CSOAP_HOME not defined
   [java] configuration/wsdd.conf doesn't exist
   [java] java.io.FileNotFoundException: configuration/MSDDeploy.wsdd (No such file or directory)
   [java] at java.io.FileInputStream.open(Native Method)
   [java] at java.io.FileInputStream.<init>(FileInputStream.java:106)
   [java] at java.io.FileInputStream.<init>(FileInputStream.java:66)
   [java] at ubisec.soap.SOAPTools.<init>(SOAPTools.java:43)
   [java] at org.ubisec.plugins.AbstractPlugin.<init>(AbstractPlugin.java:65)
   [java] at org.ubisec.plugins.slp.MSDSLPPlugin.<init>(MSDSLPPlugin.java:71)
   [java] at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
   [java]                                                                                  at
sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:39)
   [java]                                                                                  at
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:27
)
   [java] at java.lang.reflect.Constructor.newInstance(Constructor.java:494)
   [java] at java.lang.Class.newInstance0(Class.java:350)
   [java] at java.lang.Class.newInstance(Class.java:303)
   [java] at org.ubisec.components.MSDManager.startPlugin(MSDManager.java:979)
   [java] at org.ubisec.components.MSDDaemon.<init>(MSDDaemon.java:179)
   [java] at org.ubisec.framework.FrameworkMain.main(FrameworkMain.java:80)
```

Solution: When starting MSDA under Eclipse, you have to copy the `configuration` directory that contains the `MSDDeploy.wsdd` file in the $MSDACORE_DIR/bin directory. When cleaning/recompiling a project, this directory ($MSDACORE_DIR/bin) is sometimes deleted. It is therefore necessary to copy the directory again there.

6.2 NullPointerException when getting the IP Address

Problem: When launching MSDA, a NullPointerException is raised when trying to get an IP address.

```
   [java] The args attribute is deprecated. Please use nested arg elements.
   [java] Will start MSDDeamon
   [java] Not an MSD Bridge
   [java] - Total SD time
   [java] 0 [main] INFO total_msd_processing_time.log  - Total SD time
   [java] Found Network context 1
   [java] Could not initialize java.lang.NullPointerException
   [java] java.lang.NullPointerException
   [java] at org.ubisec.components.MSDDaemon.getIPAddress(MSDDaemon.java:245)
   [java] at org.ubisec.components.MSDManager.<init>(MSDManager.java:179)
   [java] at org.ubisec.components.MSDDaemon.<init>(MSDDaemon.java:160)
   [java] at org.ubisec.framework.FrameworkMain.main(FrameworkMain.java:80)
```

Solution: The configuration file contains an interface name which does not exist on the device. Check the names of the network interfaces and then check the configuration file.

6.3 MSDA component is not elected

Problem: MSDA component is not elected as a MSDA Manager and nothing happen

Solution: The network name defined in the <Manager> tag is not the same than the one defined in the <activeNetwork> tag. You have to put the correct one in both tags (see below).

```
<Framework>
    <ActiveNetwork>Network_2</ActiveNetwork>
          <agreeBridge>0</agreeBridge>
          <contextMgmt>true</contextMgmt>
          <Manager>
                <Name>Manager_1</Name>
                <NetworkName>Network_1</NetworkName>
                <RemoteAccessPort>9310</RemoteAccessPort>
                <DirectMSDPort>8310</DirectMSDPort>
```

6.4 SLP Client cannot connect to MSDA or any SLP service

Problem: client cannot connect to the MSDA service registered in SLP

Solution: The MSDA service is in fact not correctly registered in SLP. This happens when a previous MSDA service with another port or IP address has already been registered in the current SLP daemon. You have just had to kill the “slpd” process and restart it.

6.5 Wrong element

Problem: MSDA component is not elected as an MSDA Manager and nothing happens

Solution: The network name defined in the <Manager> tag is not the same than the one defined in the <activeNetwork> tag. You have to put the correct one in both tags (see below).

```
<Framework>
    <ActiveNetwork>Network_2</ActiveNetwork>
          <agreeBridge>0</agreeBridge>
          <contextMgmt>true</contextMgmt>
          <Manager>
                <Name>Manager_1</Name>
                <NetworkName>Network_1</NetworkName>
                <RemoteAccessPort>9310</RemoteAccessPort>
                <DirectMSDPort>8310</DirectMSDPort>
```

References

[1] Bluetooth. http://www.bluetooth.com
[2] Ariadne. http://www-rocq.inria.fr/arles/download/ariadne/index.html
[3] Jini. http://www.sun.com/software/jini/
[4] SLP. http://www.openslp.org/doc/rfc/rfc2608.txt
[5] UPnP. http://www.upnp.org
[6] UDDI. Universal description discovery and integration. http://www.uddi.org/
[7] CSOAP. http://www-rocq.inria.fr/arles/download/ozone/index.html