INMIDIO User's Guide

Table of Contents

1 Overview

Provider

INRIA

Introduction

The role of the INteroperable MIddleware for service Discovery and service InteractiOn (INMIDIO) is to identify the discovery and interaction middleware protocols that execute on the network and to translate the incoming/outgoing messages of one protocol into messages of another, target protocol. The system parses the incoming/outgoing message and, after having interpreted the semantics of the message, it generates a list of semantic events and uses this list to reconstruct a message for the target protocol, matching the semantics of the original message. The INMIDIO middleware acts in a transparent way with regard to discovery and interaction middleware protocols and with regard to services running on top of them. The supported service discovery protocols are UPnP, SLP and WS-Discovery, while the supported service interaction protocols are SOAP and RMI.

Intended audience

System developers that seek to integrate heterogeneous middleware platforms and their supported service-oriented architectures inside dynamic environments.

License

INMIDIO is available under the LGPL license terms.

Language

C

Environment (set-up) info needed if you want to run this sw (service)

INMIDIO requires running a web server on the machine.

Platform

Linux

2 Deployment

2.1 System requirements

Operating System: Linux

2.2 Download

Source code files and executable file are currently available either on the AMIGO GForge site[1] under the *mdwcore/sdi_sii* structure (for Subversion users) or at the following web page: http://www-rocq.inria.fr/arles/download/inmidio/index.html. Two separate files are available at the latter: a zipped file containing the source code and the executable for the middleware and a zipped file including the examples explained in the Tutorial section (3).

2.3 Install

Unzip the downloaded file in a directory `$DIR`.

Running the middleware requires a web server installed on the machine. If it is not already installed, your can download and install Jakarta Tomcat[2].

Copy `$DIR/ib/libnanohttp.so` to `/usr/lib`.

2.4 Run

To execute the middleware, run

```
$DIR/amigo_monitor $webserver_dir
```

where `$webserver_dir` is the directory where the web server used by the middleware is installed.

[1] http://gforge.inria.fr/projects/amigo/

[2] http://tomcat.apache.org/

3 Tutorial

In this section we describe some samples of clients and services provided as examples of interoperability between service discovery (SD) and service interaction (SI) for the different protocols supported by the INMIDIO.

```
samples/
samples/lib
samples/lib/org
samples/lib/slpapi
samples/res
samples/res/conf

samples/build.xml

samples/clients
samples/clients/defaultPkg
samples/clients/slprmi_samples
samples/clients/remotectrl

samples/services
samples/services/upnp-sample-tv
samples/services/slprmi_washer
samples/services/slprmi_washer/slpd
samples/services/WSDiscovery_sample_client_and_service
samples/services/WSDiscovery_sample_client_and_service/WSDiscoveryFramework-Build10003.msi
samples/services/WSDiscovery_sample_client_and_service/Client
samples/services/WSDiscovery_sample_client_and_service/Server
```

Figure 3-1: samples directory structure

Figure 3-1 shows the structure of the `samples` directory of INMIDIO. Below, for each client and service, we describe the discovery and interaction process. The `samples/res` directory contains shared resource and configuration files and the `samples/lib` directory contains the libraries used by the different client and service samples.

The examples proposed for SLP, RMI and UPnP are based on Java and those based on WS-Discovery/SOAP are based on C#. All the Java samples can be recompiled and executed using the tool `ant`[3]: we provide the associated `samples/build.xml` file that implements the compile and run target operations.

3.1.1 SLP/RMI client

The code for the SLP/RMI clients is available at the directories:

```
samples/clients/defaultPkg
samples/clients/slprmi_samples
```

These directories contain two examples: the first example will discover and interact with a UPnP `tv` service (see section 3.1.4) while the second example discover and interact with the WS-Discovery/SOAP `hello` service (see section 3.1.6). They are both based on the Openslp[4] SLP library and on the Java JDK RMI implementation.

The middleware assumes a fixed association between the SLP discovery protocol and the RMI interaction protocols for the development of clients and services. The assumed SLP/RMI schema for discovery and interaction process for clients can be summarized as follows:

[3] http://ant.apache.org/

[4] http://www.openslp.org/

- The client multicasts an SLP request for the `service-type` and receives an SLP response with the `address` of the service.
- The `address` of the service is the used to access the RMI Registry: the client executes an RMI lookup on the RMI Registry at the `address` and obtains an instance of the stub of the service.
- The client finally invokes method call on the stub.

The `ant` target operations to compile and execute the first example are:

```
compile_slprmi_client_tv
run_slprmi_client_tv
```

The service interface expected by the client is the following:

```
package defaultPkg;
  public interface tv extends java.rmi.Remote,defaultPkg.power {} ;


package defaultPkg;
  public interface power {
        public void SetPower(Integer intX, rmiholders.IBooleanHolder Result)
                throws java.rmi.RemoteException, java.lang.NoSuchMethodException ;

        public void GetPower(rmiholders.IBooleanHolder Power)  throws
                java.rmi.RemoteException, java.lang.NoSuchMethodException ;
  }
```

The `ant` target operations to compile and execute the second example are:

```
run_slprmi_client_hello
compile_slprmi_client_hello
```

The service interface expected by the client is the following:

```
package defaultPkg;
  public interface ServiceSoap extends java.rmi.Remote,defaultPkg.helloservice {} ;


package defaultPkg;
  public interface Service {
        public void HelloWorld(java.lang.String a ,rmiholders.IStringHolder c)
            throws java.rmi.RemoteException;
        public void AddOne(java.lang.Integer a, rmiholders.IIntegerHolder c )
            throws java.rmi.RemoteException;
        public void isTrue(java.lang.Integer a, rmiholders.IBooleanHolder c )
                throws java.rmi.RemoteException;
  }
```

3.1.2 SLP/RMI service

The code for SLP/RMI service is available at the directory:

```
samples/services/slprmi_washer
samples/services/slprmi_washer/slpd
```

The directory contains a service `washer` that will be discovered and interact with the UPnP client in section 3.1.3 and the WS-Discovery/SOAP client in section 3.1.5. The service is based on the provided Openslp[5] library and on the Java JDK RMI implementation. It also requires a running SLP daemon.

[5] http://www.openslp.org/

The middleware assumes a fixed association between the SLP discovery protocol and the RMI interaction protocols for the development of clients and services. The assumed SLP/RMI schema for discovery and interaction process for services can be summarized as follows:

- Start slpd (as a root/Administrator user).
- The service starts an instance of an RMI Registry and registers itself with a given `address`.
- The service registers itself on the slpd daemon with its `service-type` and the same `address` used for the registration on the RMI Registry.

The `ant` target operations to compile and execute the example are:

```
compile_slprmi_service_washer
run_slprmi_service_washer
```

The interface provided by the service is the following:

```
package washer_rmislp;

public interface WasherInterface extends Remote {
       public void start(int x, String y) throws RemoteException;
       public String stop(int a) throws RemoteException;
       public int SetState(int state) throws RemoteException;
}
```

3.1.3 UPnP client

The code for the UPnP client is available at the directory:

```
samples/clients/remotectrl
```

This directory contains a graphical client that allows at the same time, to discover and interact with the SLP/RMI `washer` service (see section 3.1.2) and to discover and interact with the WS-Discovery/SOAP `hello` service (see section 3.1.6). The UPnP client code is based on the CyberLink UPnP library[6].

The `ant` target operations to compile and execute the example are:

```
compile_upnp_client_remotectrl
run_upnp_client_remotectrl
```

3.1.4 UPnP service

The code for the UPnP service is available at the directory:

```
samples/services/upnp-sample-tv
```

The directory contains a service `tv` that will be discovered and interact with the RMI/SLP client in section 3.1.1 and WS-Discovery/SOAP client in section 3.1.5. The UPnP service code is based on the CyberLink UPnP library[7].

The `ant` target operations to compile and execute the example are:

---

[6] http://www.cybergarage.org/net/upnp/java/index.html

[7] http://www.cybergarage.org/net/upnp/java/index.html

```
run_upnp_service_tv
compile_upnp_service_tv
```

The interface provided by the UPnP device is the following:

```
<?xml version="1.0" ?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
   <specVersion>
        <major>1</major>
        <minor>0</minor>
   </specVersion>
   <device>
        <deviceType>urn:schemas-upnp-org:device:tv:1</deviceType>
        <manufacturer>Amigo</manufacturer>
        <manufacturerURL>http://www.extra.research.philips.com/euprojects/amigo</manufacturerURL>
        <modelDescription>Amigo TV Device</modelDescription>
        <modelName>TV</modelName>
        <modelNumber>1.0</modelNumber>
        <serialNumber>1234567890</serialNumber>
        <UDN>uuid:AmigoTvDevice</UDN>
        <UPC>1234567890</UPC>
        <iconList>
                <icon>
                        <mimetype>image/gif</mimetype>
                        <width>48</width>
                        <height>32</height>
                        <depth>8</depth>
                        <url>icon.gif</url>
                </icon>
        </iconList>
        <serviceList>
                <service>
                        <serviceType>urn:schemas-upnp-org:service:power:1</serviceType>
                        <serviceId>urn:schemas-upnp-org:serviceId:power:1</serviceId>
                        <SCPDURL>/service/power/description.xml</SCPDURL>
                        <controlURL>/service/power/control</controlURL>
                        <eventSubURL>/service/power/eventSub</eventSubURL>
                </service>
        </serviceList>
    </device>
</root>
```

The interface provided by the UPnP service is the following:

```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0" >
        <specVersion>
                <major>1</major>
                <minor>0</minor>
        </specVersion>
        <actionList>
                <action>
                        <name>SetPower</name>
                        <argumentList>
                                <argument>
                                        <name>PowerInt</name>
                                        <relatedStateVariable>PowerInt</relatedStateVariable>
                                        <direction>in</direction>
                                </argument>
                                <argument>
                                        <name>Result</name>
                                        <relatedStateVariable>Result</relatedStateVariable>
                                        <direction>out</direction>
                                </argument>
                        </argumentList>
                </action>
                <action>
                        <name>GetPower</name>
                        <argumentList>
                                <argument>
                                        <name>Power</name>
                                        <relatedStateVariable>Power</relatedStateVariable>
                                        <direction>out</direction>
                                </argument>
                        </argumentList>
```

```
                </action>
        </actionList>
        <serviceStateTable>
                <stateVariable sendEvents="yes">
                        <name>Power</name>
                        <dataType>boolean</dataType>
                </stateVariable>
                <stateVariable sendEvents="no">
                        <name>Result</name>
                        <dataType>boolean</dataType>
                </stateVariable>
                <stateVariable sendEvents="yes">
                        <name>PowerInt</name>
                        <dataType>int</dataType>
                </stateVariable>
                <stateVariable sendEvents="yes">
                        <name>PowerString</name>
                        <dataType>string</dataType>
                </stateVariable>
        </serviceStateTable>
</scpd>
```

3.1.5 WS-Discovery/SOAP client

The code for the WS-Discovery/SOAP client is available at the directory:

```
samples/services/WSDiscovery_sample_client_and_service/WSDiscoveryFramework-Build10003.msi
samples/services/WSDiscovery_sample_client_and_service/Client
```

This directory contains a client that allows at the same time, to discover and interact with the SLP/RMI `washer` service (see section 3.1.2) and to discover and interact with the UPNP `tv` service (see section 3.1.4). The WS-Discovery client code is based on `WSDiscoveryFramework-Build10003.msi`[8] that implements WS-Discovery protocol and that is developed as part of the Amigo IST-FP6 Project[9]. The installation and execution of the client requires a Windows machine, Microsoft .NET Framework[10] and Microsoft Visual Studio C#[11].

3.1.6 WS-Discovery/SOAP service

The code for the WS-Discovery/SOAP service is available at the directory:

```
samples/services/WSDiscovery_sample_client_and_service/WSDiscoveryFramework-Build10003.msi
samples/services/WSDiscovery_sample_client_and_service/Server
```

The directory contains a service `HelloWorldService` that will be discovered and interact with the RMI/SLP client in section 3.1.1 and UPnP client in section 3.1.3. The WS-Discovery service code is based on `WSDiscoveryFramework-Build10003.msi`[12] that implements WS-Discovery protocol and that is developed as part of the Amigo IST-FP6 Project[13]. The installation and execution of the client requires a Windows machine, Microsoft .NET Framework[14] and Microsoft Visual Studio C#[15].

---

[8] http://amigo.gforge.inria.fr/home/components/wp3/NET_Framework/download/index.html

[9] http://www.extra.research.philips.com/euprojects/amigo/

[10] http://www.microsoft.com/net/default.mspx

[11] http://msdn.microsoft.com/vstudio/express/visualcsharp/

[12] http://amigo.gforge.inria.fr/home/components/wp3/NET_Framework/download/index.html

[13] http://www.extra.research.philips.com/euprojects/amigo/

[14] http://www.microsoft.com/net/default.mspx

The scope, service type and interface defined for the service are the following:

```
[ScopeAttribute("urn:AmigoDemo")]
[TypeAttribute("HelloWorldService")]
    public Boolean isTrue(int x)
    public int AddOne(int parameter)
    public string HelloWorld(string parameter)
```

[15] http://msdn.microsoft.com/vstudio/express/visualcsharp/

4 Appendix

4.1 Description of tools/languages provided by component

4.2 FAQ