**Setting-up**

**Dependencies**

To use the Java you need two dependencies installed:

* [core-utils.7.jar](https://forge.soa4d.org/scm/viewvc.php/*checkout*/Common%20Design%20Repository/00.%20DEVELOPMENT/05.%20Prototypes/Core-utils%20JavaAPI/Binaries/core-utils-1.7.jar?root=arrowhead)
* [dnssdjava-1.0-beta7.jar](https://forge.soa4d.org/scm/viewvc.php/*checkout*/Common%20Design%20Repository/00.%20DEVELOPMENT/05.%20Prototypes/UNIBOSimpleJavaAPI/ArrowheadJavaSimpleAPIs/SimpleHelloWorld/lib/dnssdjava-1.0-beta7.jar?root=arrowhead)

These can be found in ”/Dependencies/” and installed in your local Maven repository with this command:

* mvn org.apache.maven.plugins:maven-install-plugin:2.5.2:install-file -Dfile=<file-name>

**Compilation**

The source codes can be compiled using the standard Maven command:

* mvn clean compile package

**Testing FlexOfferManager (Prosumer)**

The FlexOfferManager (to be deployed at the Prosumer-side) can be launched using the following command:

* java -cp .\flexoffer-manager\target\flexoffer-manager-1.0-SNAPSHOT-jar-with-dependencies.jar org.arrowhead.wp5.fom.main.FlexOfferManager

Type ”start()” in the terminal window to start the local HTTP server. The GUI can then be accessed at ”localhost:9997”.

**Testing AggregatoManager (Aggregator)**

The AggregatorManager (to be deployed at the Aggregator-side) can be launched using the following commands:

* cd aggregator-manager
* mvn exec:java -Dexec.mainClass="org.arrowhead.wp5.aggmanager.impl.AggregatorManager"

Type ”start()” in the terminal window to start the local HTTP server. The GUI can then be accessed at ”localhost:9998”. Type ”demo(10)” in the terminal window at the bottom of the web-GUI to simulate adding 10 flex-offers for consumption and 10 flex-offers for production.

**Using the Java library**

In our implementation, there are several classes that can be useful to use FlexOffers. This section will show how to use them

**Using the source code:**

Import the "demonstrator" maven project inside your IDE.

Create a new maven module. The flexoffer-tutorial module contains the code used in this tutorial.

**Create a Distributed Energy Resource (DER):**

A DER is a device or appliance that generates FlexOffers. It implements the AbstractDER abstract class:

public abstract class AbstractDER implements DERProviderIf, Serializable {

private static final long serialVersionUID = -8899409979845372732L;

/\* A DER Id \*/

protected String id = "0";

/\* A DER type description \*/

protected String type = "Abstract DER Type";

/\* A DER instance name \*/

protected String name = "No-name DER";

/\* A DER management URL \*/

protected String url = null;

public AbstractDER() {}

public AbstractDER(String type) {

this.type = type;

}

public AbstractDER(String type, String name) {

this(type);

this.name = name;

}

@XmlElement(name = "id")

public String getId() {

return id;

}

@XmlElement(name = "url")

public String getUrl() {

return url;

}

public void setUrl(String url) {

this.url = url;

}

public void setId(String id) {

this.id = id;

}

@XmlElement(name = "type")

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

@XmlElement(name = "name")

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

/\* Abstract methods \*/

public abstract void generateFlexOffer(Date start, Date end);

}

The following example is provided in the tutorial source code. It makes use of a FlexOfferAgent, that handles the management of FlexOffers.

public class MyDER extends AbstractDER {

private static final long serialVersionUID = -8899409979845372732L;

FlexOfferAgent foa;

public MyDER(FlexOfferAgent foa) {

super("myname", "mytype");

this.foa = foa;

this.foa.registerDer(this);

}

/\*\*

\* Example of a method to generate a FlexOffer

\* @param start

\* @param end

\* @return

\*/

public void generateFlexOffer(Date start, Date end) {

FlexOffer flexOffer = new FlexOffer();

/\*

\* Here we just use static value to populate the slices of the flexoffer

\*/

List slices = new ArrayList();

int loadLow[] = new int[] { -500, 1700, -1050, 1100, 500, 1400, 400 };

int loadHigh[] = new int[] { 500, 1732, -500, 1168, 552, 1471, 453 };

int sliceDuration = 4;

for (int i = 0; i < loadLow.length; i++) {

slices.add(new FlexOfferSlice(sliceDuration, 1, loadLow[i], loadHigh[i]));

}

flexOffer.setSlices(slices.toArray(flexOffer.getSlices()));

/\*Need to set some time parameter of the flexoffer\*/

Date ct = new java.util.Date();

flexOffer.setCreationTime(ct);

Calendar cal = Calendar.getInstance();

cal.setTime(ct);

cal.add(Calendar.MINUTE, 10);

cal.add(Calendar.MINUTE, 1);

/\*Time before aggregator needs to send back acceptance\*/

flexOffer.setAcceptanceBeforeTime(cal.getTime());

cal.add(Calendar.MINUTE, 1);

/\*Time before aggregator needs to send back schedule\*/

flexOffer.setAssignmentBeforeTime(cal.getTime());

cal.add(Calendar.MINUTE, 10);

/\*Minimum time at which the consumption pattern can start\*/

flexOffer.setStartAfterTime(cal.getTime());

cal.add(Calendar.HOUR, 1);

/\*Maximum time for the consumption pattern to start\*/

flexOffer.setStartBeforeTime(cal.getTime());

try {

foa.createFlexOffer(id, flexOffer);

} catch (FlexOfferException e) {

System.out.println("Something went wrong!");

}

}

@Override

public void updateSchedule(FlexOffer fo) {

/\*Apply the schedule to the object so that it follows

\*the assigned consumption pattern\*/

FlexOfferSchedule schedule = fo.getFlexOfferSchedule();

}

}

Note that the two important part are the foa.createFlexOffer(id, flexOffer); that will handle the registration and sending of the FlexOffer (assuming the rest of the code is as shown thereafter) and the updateSchedule(FlexOffer fo) that will be called when a schedule is sent back.

**Create a FlexibleResource:**

The FlexibleResource implements the functionalities to communicate with an Aggregator, and uses the Arrowhead services for service discovery. The implementation provided below should provide most of the required functionalities.

public class MyFlexibleResource implements FlexOfferUpdateListener {

final Logger logger = LoggerFactory.getLogger(this.getClass());

/\*\* Encapsulated flex-offer agent, the entity managing the flexoffers \*/

private FlexOfferAgent agent;

/\*\* The FlexibleDER that needs to generate the FlexOffers \*\*/

private FlexibleDER flexDER;

/\*\*

\* The id of this flexoffer manager. Used both for the XMPP network and the

\* uniqueness of flex offers

\*/

private String id;

/\*\* The id of the aggregator found by using Arrowhead service discovery \*\*/

private String aggId;

/\*\* The password used to connect to the XMPP server \*/

private String password;

/\*\* XMPP Server Host \*/

private String xmppServer = "delling.dpt.cs.aau.dk";

/\*\* XMPP Service \*/

private String xmppService = "delling";

/\*\* XMPP resource \*/

private String xmppResource = "demo";

/\*\* The object that encapsulates HTTP over XMPP functionalities \*/

private HOXTWrapper hoxtWrapper;

/\*\*

\* The object that provides the functionalities to send flexoffers to the

\* aggregator

\*/

private XFlexOfferSubscriberClient xfosc;

/\*\* Interface to use the Arrowhead Service discovery \*\*/

private ArrowheadXMPPServiceManager foServiceManager;

public MyFlexibleResource() {

this.id = "requestit";

this.agent = new FlexOfferAgent(id, this);

this.foServiceManager = new ArrowheadXMPPServiceManager("alpha.jks", "abc1234", "alpha.jks", "abc1234");

this.flexDER = new MyDER(this.agent);

}

public static void main(String[] args) {

MyFlexibleResource flexResource = new MyFlexibleResource();

flexResource.doAggregatorSD();

}

private void doAggregatorSD() {

this.foServiceManager.start();

if (this.foServiceManager.fetchInfo()) {

aggId = this.foServiceManager.getAggId();

xmppServer = this.foServiceManager.getHostname();

xmppResource = this.foServiceManager.getResource();

logger.info("Aggregator ID set from Arrowhead framework: {}", aggId);

} else {

logger.warn("Failed to discover aggregator through Arrowhead framework. Using default config");

}

}

public void initXmpp() {

/\*\*

\* Need to load the trustStore to connect to the XMPP server Can be done

\* in the code like here or using runtime parameter

\* \*\*/

System.setProperty("javax.net.ssl.trustStore",

"resources/clientstore.jks");

/\*Manages REST resources for HTTP over XMPP\*/

ResourceManager resourceManager = new ResourceManager();

/\*Register the interface to the flex offer (implements FlexOfferAgentProviderIf)\*/

resourceManager.registerInstance(new XFlexOfferResource(agent));

/\*Smack connection, set up password, server...\*/

XMPPTCPConnectionConfiguration config = XMPPTCPConnectionConfiguration

.builder()

.setUsernameAndPassword(id, password)

.setServiceName(xmppService)

.setHost(xmppServer)

.setSecurityMode(SecurityMode.required)

.setResource(xmppResource)

.setCompressionEnabled(false).build();

/\*A wrapper for Http over XMPP connection\*/

this.hoxtWrapper = new HOXTWrapper(config, resourceManager);

try {

/\*Initialize the wraper, true means that we will be using

\* both server and client functionalities.

\*/

this.hoxtWrapper.init(true);

} catch (XMPPException | SmackException | IOException | InterruptedException e) {

logger.error("Failed to initialize XMPP", e);

}

/\*This is the client interface to the aggregator, to send flexoffer for example\*/

this.xfosc = new XFlexOfferSubscriberClient(aggId, hoxtWrapper);

}

public void deinitXmpp() {

/\*Clean up the wrapper\*/

this.hoxtWrapper.destroy();

this.xfosc = null;

}

@Override

public void onFlexOfferScheduleUpdate(FlexOffer fo) {

/\* Forwards the FlexOffer with the Schedule to the FlexibleDER \*/

flexDER.updateSchedule(fo);

}

/\* Callback for the FlexOfferAgent when it creates a flexoffer

\* We send the flexoffer to the aggregator throught the XMPP interface

\* \*/

@Override

public void onFlexOfferCreate(FlexOffer fo) {

try {

if (this.xfosc != null) {

this.xfosc.createFlexOffer(id, fo);

}

} catch (FlexOfferException e) {

// TODO Handle error

logger.error("Error creating flexoffer.", e);

}

}

}

The code is similar to what was shown using the HOXTWrapper and XFlexOfferSubscriberClient in the first part. In addition, a FlexOfferAgent is used, and the XFlexOfferResource makes use of it. It should be possible to reuse most of this code as such, changing the parameters for username, passwords and using your own "jks" files for the Arrowhead service discovery.

**Using the HTTP interface**

The last possibility is to use the HTTP interface of the Aggregator to send FlexOffers. FlexOffers can be sent serialized in JSON or XML format to the URI /api/flexoffers/{foaid} where {foaid} is the id the the FlexibleResource sending the FlexOffer. Note that this id must be consistent with the offeredById of the FlexOffer otherwise it will be rejected. The submitted FlexOffer can then be accessed at the URI /api/flexoffers/{foaid}/{foid} where {foid} is the id of the submitted FlexOffer. To detect when a schedule is assigned, poll this resource and check whether the returned FlexOffer has a schedule. The wadl and associated xsd files are available [here](file:///C:\_Laurynas_Postdoc\Projects\Arrowhead_project\SVN_WP5\FlexTutorial\resources\application.wadl) and [here](file:///C:\_Laurynas_Postdoc\Projects\Arrowhead_project\SVN_WP5\FlexTutorial\resources\xsd0.xsd). The following code snippet shows an example of an HTTP client creating a FlexOffer and polling the server until a schedule is assigned:

public class HTTPClientTest {

public static String id = "myid";

public static void main(String[] args) {

FlexOffer flexOffer = new FlexOffer();

/\*

\* Here we just use static value to populate the slices of the flexoffer

\*/

List slices = new ArrayList();

int loadLow[] = new int[] { -500, 1700, -1050, 1100, 500, 1400, 400 };

int loadHigh[] = new int[] { 500, 1732, -500, 1168, 552, 1471, 453 };

int sliceDuration = 4;

for (int i = 0; i < loadLow.length; i++) {

slices.add(new FlexOfferSlice(sliceDuration, 1, loadLow[i], loadHigh[i]));

}

flexOffer.setSlices(slices.toArray(flexOffer.getSlices()));

/\*Need to set some time parameter of the flexoffer\*/

Date ct = new java.util.Date();

flexOffer.setCreationTime(ct);

Calendar cal = Calendar.getInstance();

cal.setTime(ct);

cal.add(Calendar.MINUTE, 10);

cal.add(Calendar.MINUTE, 1);

/\*Time before aggregator needs to send back acceptance\*/

flexOffer.setAcceptanceBeforeTime(cal.getTime());

cal.add(Calendar.MINUTE, 1);

/\*Time before aggregator needs to send back schedule\*/

flexOffer.setAssignmentBeforeTime(cal.getTime());

cal.add(Calendar.MINUTE, 10);

/\*Minimum time at which the consumption pattern can start\*/

flexOffer.setStartAfterTime(cal.getTime());

cal.add(Calendar.HOUR, 1);

/\*Maximum time for the consumption pattern to start\*/

flexOffer.setStartBeforeTime(cal.getTime());

flexOffer.setOfferedById(id);

flexOffer.setId(1);

Client client = ClientBuilder.newClient();

WebTarget target = client.target("http://localhost:9998").path("api").path("flexoffers").path(id);

System.out.println(target.getUri());

Response resp = target.request().post(Entity.xml(flexOffer));

System.out.println(resp.getStatus());

if (resp.getStatus() == 200) {

FlexOfferSchedule fos = null;

while ((fos = flexOffer.getFlexOfferSchedule()) == null) {

target = client.target("http://localhost:9998").path("api").path("flexoffers").path(id).path(Integer.toString(flexOffer.getId()));

resp = target.request().get();

System.out.println(resp.getStatus());

flexOffer = resp.readEntity(FlexOffer.class);

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

System.out.println("This should not have happened!");

}

}

/\*Do something with the schedule (apply it)\*/

System.out.println("Total Energy: " + fos.getTotalEnergy());

}

}

}

**Test tool tutorial**

**FlexOffer Tutorial**

You can use the Core Services with the flex-offer tutorial.

* First you need to activate Arrowhead within the code: Simply update the variable ARROWHEAD\_COMPLIANT to be true.
* Second you need to add the following to the arguments for the Java VM. In Eclipse this is done by right-clicking the executable (MyFlexibleResource.java) -> Run as -> Run configurations -> Arguments -> VM arguments.
  + -Ddns.server=10.200.0.10
  + -Ddnssd.domain=test.bnearit.arrowhead.eu
  + -Ddnssd.tsig=tsig

If you want to see an example of how to register a service and how to lookup a service using the service registry and orchestration look in the "arrowhead-common" project.

NOTICE: You need the tsig and alpha.jsk files in your project if you want to use Arrowhead on another project

**Test tool**

To simplify implementation and testing, a simple aggregator is available online. The XMPP interface can be discovered using the Arrowhead SD, or it can be accessed directly through XMPP using the same id. The HTTP interface is available at http://iliving.cs.aau.dk (see [below](file:///C:\_Laurynas_Postdoc\Projects\Arrowhead_project\SVN_WP5\FlexTutorial\flextuto.html#httpway) for details). The test tool simply receives a FlexOffer, waits 5 seconds and assigns it a schedule, which is sent back if the client is using XMPP (polling is required for HTTP, see [below](file:///C:\_Laurynas_Postdoc\Projects\Arrowhead_project\SVN_WP5\FlexTutorial\flextuto.html#httpway) for details).

**Connecting to Arrowhead VPN**

In order to use the Arrowhead Core Services, see the separate explanation [here](file:///C:\_Laurynas_Postdoc\Projects\Arrowhead_project\SVN_WP5\FlexTutorial\arrowhead-benearit.html).

**Using the XMPP wrapper**

Start by downloading the source code from the svn repository [here](https://forge.soa4d.org/svn/arrowhead/WP5/Prototypes/trunk).

Import the "demonstrator" maven project inside your IDE.

The module of interest here is "communication-xmpp". The first class of interest is HOXTWrapper. Here is an example on how to initialize an instance:

System.setProperty("javax.net.ssl.trustStore",

"resources/clientstore.jks");

ResourceManager resourceManager = new ResourceManager();

resourceManager.registerInstance(new XFlexOfferResource());

XMPPTCPConnectionConfiguration config = XMPPTCPConnectionConfiguration

.builder()

.setUsernameAndPassword(id, password)

.setServiceName("delling")

.setHost("delling.dpt.cs.aau.dk")

.setSecurityMode(SecurityMode.required)

.setResource("demo")

.setConnectTimeout(30000)

.setCompressionEnabled(false).build();

HOXTWrapper hoxtWrapper = new HOXTWrapper(config, resourceManager);

try {

hoxtWrapper.init(true);

} catch (XMPPException | SmackException | IOException | InterruptedException e) {

logger.error("Failed to initialize XMPP", e);

}

The first line sets the ssl trustore required to establish secure communication with the XMPP server (it can also be set as a VM argument). Please request it from us (or take it from SVN at https://forge.soa4d.org/svn/arrowhead/WP5/Prototypes/trunk/flexoffer-tutorial/resources/clientstore.jks). Then a ResourceManager is created. This will be the object handling the incoming request from the Aggregator (essentially receiving the FlexOfferSchedule. How to create the resource will be discussed after. Next an XMPPTCPConnectionConfiguration is created. It needs a username and password (request them from us), a service name and host (use the same), sets the security, a resource (this is an XMPP resource, our libraries expect "demo" so use that, see [here](http://wiki.xmpp.org/web/Jabber_Resources) for details), a connection timeout and finally disable compression. The HOXTWrapper can then be instantiated using the built configuration and resource manager, and then initialized.

Now the following code snippet shows an example of the resource that needs to be implemented to received a FlexOfferSchedule back from the Aggregator:

@Path("/flexoffers")

public class XFlexOfferResource implements FlexOfferAgentProviderIf{

public XFlexOfferResource(){}

@Override

@Path("/{foid}/schedule")

@POST

public void createFlexOfferSchedule(@PathParam("foid") int flexOfferId,

FlexOfferSchedule flexOfferSchedule) throws FlexOfferException {

/\*Store the schdule in the corresponding FlexOffer or apply it\*/

}

}

To send FlexOffers, an XFlexOfferSubscriberClient object can be used. As shown in the following snippet, the constructor takes the id of an aggregator to send the FlexOffers, and a HOXTWapper instance:

XFlexOfferSubscriberClient xfosc = new XFlexOfferSubscriberClient(aggId, hoxtWrapper);

Then the method createFlexOffer can be used to send a FlexOffer to the aggragtor with the id specified in the constructor. The method takes two parameters. The first the id of the sender, which needs to correspond to the name used in the Jabber id (the part before the "@"), which you have to request from us. The second is the FlexOffer to be sent:

xfosc.createFlexOffer(id, flexOffer);

This is the basics to be able to send FlexOffers and receive back schedules.