Otsopack Documentation

Release 1.1.0

Aitor Gómez-Goiri

December 05, 2013

Contents

Otsopack is a Triple Space solution which has different incarnations for heterogeneous devices. Triple Space Computing (TS) is a paradigm based on Tuple Spaces where Semantic Web techniques are used to dene the knowledge which is exchanged using a distributed shared space (for more information check this presentation in Spanish).

Otsopack runs on...

Otsopack is designed to be run in a wide variety of devices with limited computational resources. More specifically, nowadays th

- OtsoSE Java SE
- OtsoDroid Android
- FoxG20 & XBee (Python)
- Sunspots

Use cases

In order to understand the scope of the project, the following use cases can be considered:

- As part of the ISMED project (2008-2010), heterogeneous mobile devices interacted with others, using the JXTA peer-to-peer protocol
  - The considered scenarios were mainly domotic, as detailed here.
- As part of the ACROSS project, rich mobile devices interact with each other and with the infrastructure
  - We have published two articles describing the supermarket scenario and the hospital scenario (check it here).
- As part of the TALIS+ENGINE project, embedded devices in Ambient Intelligent scenarios (check it here).
- As part of the THOFU project, mobile and embedded devices in Ambient Intelligent scenarios within a Hotel (check it here).

CHAPTER 3

Authors

This solution has been developed by members of the MORElab Research Group at DeustoTech - Deusto Institute of Technology in the University of Deusto.

Check related publications here.

CHAPTER 4

Contents

4.1 How to build Otsopack

The official releases of Otsopack can be found here. In this page, we explain how to generate the necessary jars from the code on the repository.

4.1.1 Java SE version

Otsopack uses maven . It is divided in several subprojects (otsoAuthentication, otsoCommons, etc.). Each subproject has its own *pom.xml*. Besides, the **root directory** has another **pom.xml** to generate all the modules in the correct order.

For those not familiarized with *Maven*, consider using:

- *mvn eclipse:eclipse*: to generate **'Eclipse IDE<http://www.eclipse.org/>'_**'s descriptors for each module
- *mvn build*: to compile all the subprojects
- *mvn clean*: to clean the files and directories generated by a previous build in all the subprojects
- *mvn package*: to generate jars for all the subprojects
  - add *-Dmaven.test.skip=true* to skips the tests
  - add *-am -pl <module-name>* to generate just one module and its dependencies (e.g. *-am -pl otsoCommons*)
- *mvn install*: to install the jars in your local maven repository * add *-Dmaven.test.skip=true* to skips the tests

After the jar generation (i.e. *mvn package*), you may want to check the following directories:

- *otsoSE/target* to find Otsopack's **Java SE version**
- *otsoSE/target/lib* to find all its (direct or transitive) dependencies

4.1.2 Android version

The **Android release** is generated using the ant buildfile located in otsoDroid/otsoDroid/build.xml.

OtsoDroid depends on *otsoCommons* and its dependencies. The dependencies of *otsoCommons* are copied to *otsoDroid/otsoDroid/libs/* whenever *otsoCommons* is built. *otsoCommons* itself is copied to *libs* folder during the *ant* run. Therefore, prior to the *otsoDroid* build, we need to build *otsoCommons* as explained in the previous section:

1. Go to the root of the project.
2. Execute: *mvn package -Dmaven.test.skip=true -am -pl otsoCommons*

After building *otsoCommons*, we can simply go to *otsoDroid/otsoDroid* and run any of the targets defined in the *ant buildfile*. If it is the first time you use that *buildfile*, copy *local.properties.dist* to *local.properties* and edit it to specify the path of your Android SDK.

To generate a *jar* from the source run:

- *ant clean*: to clean the files and directories generated by a previous build
- *ant release*: to generate a *jar* with *otsoDroid*

After the jar generation, you may want to check the following directories:

- *otsoDroid/otsoDroid/bin* to find Otsopack's **Android version**
- *otsoDroid/otsoDroid/libs* to find all its (direct or transitive) dependencies

4.2 Dependencies

Otsopack uses maven . Therefore, using the *pom.xml* of the root directory, you can easily build and generate distribution packages. For more information on how to do it, check this page.

To use Otsopack, you need to add the version you want to use and its dependencies to your *classpath*.

The build process will copy all the dependencies into a folder (once again, is detailed here). Anyway, they are listed in the following sections.

4.2.1 Common dependencies

Name	Last tested version	Optional
rdf2go.api	4.8.3	No
cglib-nodep	2.2	No
objenesis	1.2	No
apache commons io	2.4	No
restlet	2.0.13	No
restlet.ext.jackson	2.0.13	No
jackson-all	1.7.3	No

4.2.2 Android version

Name	Last tested version	Optional
restlet (Android version)	2.0.13	No
restlet.ext.net (Android version)	2.0.13	No

4.2.3 Java SE version

Name	Last tested version	Optional
otsoCommons	N/A	No
restlet (SE version)	2.0.13	No
restlet.ext.simple	2.0.13	No
rdf2go.impl.sesame	4.8.3	Yes
sesame-runtime	2.7.1	Yes
simple	4.1.21	Recommended
slf4j-api	1.7.5	No
slf4j-nop	1.7.5	Recommended
sqlitejdbc	056	Recommended

4.3 REST Services

This pages details the REST services provided by each node divided into different categories.

Contents

4.3.1 Prefix management

GET /prefixes/

Retrieves the prefixes used by this node.

Accepted content-types: html, json

Status Codes

- **200** – 200 OK

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "http://www.w3.org/1999/02/22-rdf-syntax-ns#":"rdf",
  "http://www.w3.org/2002/07/owl#":"owl",
  "http://www.w3.org/2001/XMLSchema#":"xsd",
  "http://www.w3.org/2000/01/rdf-schema#":"rdfs"
}
```

GET /prefixes/(**uri:** *prefix*)
Retrieves the prefix for an URI provided it is registered.

Accepted content-types: json

Parameters

- **prefix** (*URI*) – URI whose prefix one is looking for (must be encoded)

Status Codes

- **200** – 200 OK
- **404** – When the given uri has not a prefix associated (404 Not Found)

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

"prefix1"
```

4.3.2 Space management

GET /spaces/
Retrieves the spaces a node is connected to.

Accepted content-types: html, json

Status Codes

- **200** – 200 OK

GET /spaces/(**uri:** *space*)
Retrieves a list of the REST services (TSC primitives) which can be consumed on that space. The purpose of showing a representation of this resource is to enable browsing.

Accepted content-types: html, json

Parameters

- **space** (*URI*) – the URI which identifies the space (must be encoded)

Status Codes

- **200** – 200 OK

4.3.3 Authorization

GET /login
Checks whether the user is logged or not.

Accepted content-types: html, json

Status Codes

- **401** – Unauthorized client cannot read this graph. (401 Unauthorized)

4.3.4 Triple Space primitives

Graph level operations (read and take)

In this subsection we describe the primitives related to the RDF Graphs on a space.

`GET /spaces/`(**uri:** *space*)`/graphs`

Retrieves a list of the graphs written into that space on that node.

Accepted content-types: html

Parameters

- **space** (*URI*) – the URI of the space where the graph is written (must be encoded)

Status Codes

- **200** – 200 OK
- **406** – The requested content-type cannot be retrieved (406 Not Acceptable).

`POST /spaces/`(**uri:** *space*)`/graphs`

write({space},{graph}): graphURI
(pending to determine whether it makes sense offering this service or not)

Accepted content-types: semantic formats

Parameters

- **space** (*URI*) – the URI of the space where the graph is written (must be encoded)

Status Codes

- **200** – 200 OK
- **404** – The node has not joined to the *space* provided (404 Not Found).
- **406** – The requested content-type cannot be retrieved (406 Not Acceptable).
- **500** – The information cannot be stored (500 Internal Server Error).

`GET /spaces/`(**uri:** *space*)`/graphs/`
uri: *graph* read({space},{graph})

Accepted content-types: semantic formats, html

Parameters

- **space** (*URI*) – the URI of the space where the graph is written (must be encoded)
- **graph** (*URI*) – the URI of the graph to be read (must be encoded)

Status Codes

- **200** – 200 OK

- **402** – Unauthorized client cannot read this graph (402 Payment Required).
- **403** – The client has not permissions to read this graph (403 Forbidden).
- **404** – When the node has not joined to the {space} provided (starts with SpaceNotExistException.HTTPMSG) or the graph with {graph} URI does not exist (404 Not Found).
- **406** – The requested content-type cannot be retrieved (406 Not Acceptable).

`DELETE /spaces/`(**uri:** *space*)`/graphs/`
uri: *graph* take({space},{graph})

Accepted content-types: semantic formats, html

Parameters

- **space** (*URI*) – the URI of the space where the graph is written (must be encoded)
- **graph** (*URI*) – the URI of the graph to be read (must be encoded)

Status Codes

- **200** – 200 OK
- **402** – Unauthorized client cannot read this graph (402 Payment Required).
- **403** – The client has not permissions to read this graph (403 Forbidden).
- **404** – When the node has not joined to the {space} provided (starts with SpaceNotExistException.HTTPMSG) or the graph with {graph} URI does not exist (404 Not Found).
- **406** – The requested content-type cannot be retrieved (406 Not Acceptable).

`GET /spaces/`(**uri:** *space*)`/graphs/wildcards/`
uri: *subject*`/`**uri:** *predicate*`/`**uri:** *object* read({space},{template}), where {template} is made up of {subject}, {predicate} and {object}

Accepted content-types: semantic formats, html

Parameters

- **space** (*URI*) – the URI of the space where the graph is written (must be encoded)
- **subject** (*URI*) – the URI of the subject or "*" (must be encoded)
- **predicate** (*URI*) – the URI of the predicate or "*" (must be encoded)
- **object** (*URI*) – the URI of the object or "*" (must be encoded)

Status Codes

- **200** – 200 OK
- **400** – The template cannot be created with the provided arguments (400 Bad Request).
- **404** – When the node has not joined to the {space} provided (starts with SpaceNotExistException.HTTPMSG) or the graph with {graph} URI does not exist (404 Not Found).

- **406** – The requested content-type cannot be retrieved (406 Not Acceptable).
- **500** – A non-existing prefix was used in the template (500 Internal Server Error).

`DELETE /spaces/`(**uri:** *space*)**`/graphs/wildcards/`**
uri: *subject***`/`uri:** *predicate***`/`uri:** *object* take({space},{template}), where {template} is made up of {subject}, {predicate} and {object}

Accepted content-types: semantic formats, html

Parameters

- **space** (*URI*) – the URI of the space where the graph is written (must be encoded)
- **subject** (*URI*) – the URI of the subject or "*" (must be encoded)
- **predicate** (*URI*) – the URI of the predicate or "*" (must be encoded)
- **object** (*URI*) – the URI of the object or "*" (must be encoded)

Status Codes

- **200** – 200 OK
- **400** – The template cannot be created with the provided arguments (400 Bad Request).
- **404** – When the node has not joined to the {space} provided (starts with SpaceNotExistException.HTTPMSG) or the graph with {graph} URI does not exist (404 Not Found).
- **406** – The requested content-type cannot be retrieved (406 Not Acceptable).
- **500** – A non-existing prefix was used in the template or the information could not be removed from the store (500 Internal Server Error).

`GET /spaces/`(**uri:** *space*)**`/graphs/wildcards/`**
uri: *subject***`/`uri:** *predicate***`/(object-type)/(object-value)`** read({space},{template}), where {template} is made up of {subject}, {predicate}, {object-type} and {object-value}

Accepted content-types: semantic formats, html

Parameters

- **space** (*URI*) – the URI of the space where the graph is written (must be encoded)
- **subject** (*URI*) – the URI of the subject or "*" (must be encoded)
- **predicate** (*URI*) – the URI of the predicate or "*" (must be encoded)
- **object-type** – the XSD type for the given literal
- **object-value** – the string representation of the literal

Status Codes

- **200** – 200 OK
- **400** – The template cannot be created with the provided arguments (400 Bad Request).

- **404** – When the node has not joined to the {space} provided (starts with SpaceNotExistException.HTTPMSG) or the graph with {graph} URI does not exist (404 Not Found).
- **406** – The requested content-type cannot be retrieved (406 Not Acceptable).
- **500** – A non-existing prefix was used in the template (500 Internal Server Error).

DELETE /spaces/(**uri:** *space*)**/graphs/wildcards/**
uri: *subject***/uri:** *predicate***/(object-type)/(object-value)** take({space},{template}), where {template} is made up of {subject}, {predicate}, {object-type} and {object-value}

Accepted content-types: semantic formats, html

Parameters

- **space** (*URI*) – the URI of the space where the graph is written (must be encoded)
- **subject** (*URI*) – the URI of the subject or "*" (must be encoded)
- **predicate** (*URI*) – the URI of the predicate or "*" (must be encoded)
- **object-type** – the XSD type for the given literal
- **object-value** – the string representation of the literal

Status Codes

- **200** – 200 OK
- **400** – The template cannot be created with the provided arguments (400 Bad Request).
- **404** – When the node has not joined to the {space} provided (starts with SpaceNotExistException.HTTPMSG) or the graph with {graph} URI does not exist (404 Not Found).
- **406** – The requested content-type cannot be retrieved (406 Not Acceptable).
- **500** – A non-existing prefix was used in the template (500 Internal Server Error).

Space level operations (query)

In this subsection we describe the query primitive, which cares about the RDF triples written into a space. In other words, it does not care to which graph each returned RDF triple belongs to.

GET /spaces/(**uri:** *space*)**/query/wildcards/**
uri: *subject***/uri:** *predicate***/uri:** *object* query({space},{template}), where {template} is made up of {subject}, {predicate} and {object}

Accepted content-types: semantic formats, html

Parameters

- **space** (*URI*) – the URI of the space where the graph is written (must be encoded)
- **subject** (*URI*) – the URI of the subject or "*" (must be encoded)
- **predicate** (*URI*) – the URI of the predicate or "*" (must be encoded)
- **object** (*URI*) – the URI of the object or "*" (must be encoded)

Status Codes

- **200** – 200 OK
- **400** – The template cannot be created with the provided arguments (400 Bad Request).
- **404** – When the node has not joined to the {space} provided (starts with SpaceNotExistException.HTTPMSG) or the graph with {graph} URI does not exist (404 Not Found).
- **406** – The requested content-type cannot be retrieved (406 Not Acceptable).
- **500** – A non-existing prefix was used in the template (500 Internal Server Error).

`GET /spaces/`(**uri:** *space*)**`/query/wildcards/`**
uri: *subject***`/`uri:** *predicate***`/(object-type)/(object-value)`** query({space},{template}), where {template} is made up of {subject}, {predicate}, {object-type} and {object-value}

Accepted content-types: semantic formats, html

Parameters

- **space** (*URI*) – the URI of the space where the graph is written (must be encoded)
- **subject** (*URI*) – the URI of the subject or "*" (must be encoded)
- **predicate** (*URI*) – the URI of the predicate or "*" (must be encoded)
- **object-type** – the XSD type for the given literal
- **object-value** – the string representation of the literal

Status Codes

- **200** – 200 OK
- **400** – The template cannot be created with the provided arguments (400 Bad Request).
- **404** – When the node has not joined to the {space} provided (starts with SpaceNotExistException.HTTPMSG) or the graph with {graph} URI does not exist (404 Not Found).
- **406** – The requested content-type cannot be retrieved (406 Not Acceptable).
- **500** – A non-existing prefix was used in the template (500 Internal Server Error).

Contents

4.4 Discovery

Otsopack manages the discovery of nodes in the space, in a seamless way for the developer. In order to do so, Otsopack defines a set of components that will be deployed in different ways depending on the scenario.

4.4.1 Scope

In Otsopack there are different primitives to retrieve and modify triples:

- read
- take
- query
- write

They all work in a *space*: "read a graph which matches this template in space A", "query on all the graphs of the space B".

The discovery process manages two things:

- **Discovering** which nodes are in these spaces
- **Maintaining** which nodes are alive at this moment

When programming with Otsopack, developers are not aware of this information. But it is important that this information is updated. If the node where it is being executed has not discovered that another node exists in the same space, it will not retrieve information from it. If the node does not know that this node has disappeared, it will take longer since it will wait for some information that will never arrive.

4.4.2 Hierarchy

Otsopack supports a wide range of devices: very small devices (sensors), rich mobile phones (Android), regular computers and servers. The scenarios covered where otsopack is used cover very different topologies, where very few devices are used -even 2!- without infrastructure to big deployments with hundreds of potential devices.

Therefore, the design of the discovery system must be hierarchical, so smaller nodes can rely on bigger nodes to find the information.

Node

There are nodes. Every node implements the primitives.

- It must have a UUID (which can be temporal, since it is only used in discovery)
- It can be checkable (if the space manager can directly call a "check" method to check that it is in fact available)
- It can poll a space manager (if it will call every few seconds the space manager to ensure that it still exists)
- It can be permanent (not checkable and it does not poll, but is always there). This makes sense if the space manager is the same node as the node itself.

Space Manager

There are *Space Managers*. They maintain a list of alive nodes per space.

- A Node can join to one or more Space Managers. Since they all provide a UUID, a node should not call more than once to the same node.
- A Space Manager will maintain the list, knowing which nodes are available and which nodes are down, thanks to the check/poll mechanisms.

Discovery

There are Discovery components. They find Space Managers in the environment.

- When requested, it returns the Space Manager it can find (in a file, in memory, in a HTTP server, through multicast **(not implemented)**)
- It is assumed that these space managers are already handling the nodes they support.

The main difference between Space Managers and Discovery components is that the Space Manager maintains a list of final nodes, while the Discovery component searches in every call for the Space Managers. This can be clarified in the following section.

4.4.3 Sample scenarios

Many devices, infrastructure provided

Consider a mobile phone entering in a pub where there are other 50 users with their mobile phones connected. If all the phones tried to maintain the list of available phones through polling, every phone would perform requests to the rest and therefore every phone would need to process 50 requests every few seconds. In order to delegate the maintaining of the list, the pub could provide some kind of infrastructure (such as a desktop machine connected to the network) that would act as a *Space Manager*. It would check every few seconds if the mobile phones are still there (or it would expect them to perform a request every few seconds). Any mobile phone would also perform a call every few seconds to get the latest version of the list of alive mobile phones. The amount of requests performed or processed by each mobile phone is far smaller. When the user enters in the pub, the Discovery will check what Space Managers are available, and it will call them, automatically retrieving the full list of devices.

Few devices, infrastructure not provided

Consider a small set of sensors in a room. They all could have their own Space Manager, pointing only to the current node. Whenever a new sensor appears in the room, the Discovery components will find the Space Manager of the device, and therefore they will find the node it manages.

Future possible scenarios

A dynamic system could be implemented if there are many devices and no infrastructure is provided. The first node could start a Space Manager that the rest would use. When the number of nodes managed by the Space Manager is increased (for instance, 5 in a mobile phone), the Space Manager could start rejecting nodes trying to be added, so a rejected node could start serving as alternative Space Manager. It would take few seconds to check that there is a new space manager to ask for maintained nodes. Also, whenever a node with a Space Manager becomes outreachable, the managed nodes will notice because they are not checked anymore or because they have problems when polling. They could find other Space Manager to be maintained again in the space.

4.4.4 Development

Node

- Node

Space Manager

- ISpaceManager
- Implementations
  - Simple (stores it in memory)
  - File
  - Http (to an external server)
  - Multiplexer (takes more than one Space Manager)

Discovery

- IDiscovery
- Implementations
  - Simple (stores it in memory)
  - Http (to an external server)
  - Multiplexer (takes more than one Space Manager)

Registry

In order to avoid dealing with the Discovery directly, there is an interface called IRegistry. Its implementations can be found here.

4.5 Subscriptions

This page describes in detail the subscription system of Otsopack.

Contents

4.5.1 Subscription primitives

The subscription primitives the developer should use are the following ones:

- *subscribe*
- *unsubscribe*
- *notify*

subscribe

The node subscribes to the given template returning an URI which identifies the subscription.:

```
public String subscribe(String spaceURI, NotificableTemplate template, INotificationListener listene
```

unsubscribe

Unsubscribes to a subscription given its subscription URI.:

```
public void unsubscribe(String spaceURI, String subscriptionURI) throws SpaceNotExistsException, Subs
```

notify

Notifies a template so the subscribed nodes will be warned.:

```
public void notify(String spaceURI, NotificableTemplate template) throws SpaceNotExistsException,  Su
```

4.5.2 Deployment

Bulletin boards

The nodes responsible of handling subscriptions and notifications are called *bulletin boards*. Other *Otsopack* nodes belonging to the same space, discover them through a *registry* and publish their subscriptions and notifications using their HTTP API.

Each *bulletin board*:

- Belongs to a space.
- Exposes a subscription API (see [Subscriptions#Subscription_HTTP_API below]).
- Shares its subscriptions with other bulletin boards which belong to the same space.
- Propagates the notifications to the relevant nodes using the *callback url* provided by them.

To manage the bulletin boards we use a ''IRemoteBulletinBoardsManager'‘. It can be created independently or in conjuntion with a TSC kernel.

To create it independently, we should do::

```
final IRemoteBulletinBoardsManager bbm = BulletinBoardsManager.createPlain(registry);
bbm.startup();
...
bbm.shutdown();
```

To get the ''IRemoteBulletinBoardsManager'' of an already created TSC kernel, we use:

```
((HttpKernel)kernel).getNetworkService().getBulletinBoardsManager();
```

Once we have the ''IRemoteBulletinBoardsManager'' object, if we want to create a bulletin board, we use::

```
bbm.createRemoteBulletinBoard(Description.spaceuri, PORT);
```

On the other hand, if we want to use a remote ''bulletin board'' in the kernel, it will be transparently created the first time a subscription primitive is used for a given space.

Practical example

In the image shown below, we can see a graphic representation of the following example:

1. *N1* subscribes to *BB1* with a template *t1*.
2. *BB1* propagates the subscription provided by *N1* to *BB2* and *BB3*
3. *N3* notifies to *BB3* about *t2*.
4. Since *t1* matches *t2*, *BB3* tries to notify to *N1* using the *callback URI* provided during the subscription process.
5. Unfortunately, the *BB3* cannot notify to *N1* due to unexpected network problems.
6. *BB3* propagates the notification of *t2* to *BB2*.

7. *BB2* reaches *N1*, so it notifies it about *t2* using the *callback URI*.

Check the implementation of this example here.

4.5.3 Subscription HTTP API

Each *bulletin board* exposes a RESTful API which enables the creation of subscriptions on the *bulletin boards* and to trigger notifications to end-nodes.

`GET /subscriptions`

Returns all the subscriptions.

Accepted content-types: json

Status Codes

- **200** – 200 OK

`POST /subscriptions`

subscribe({subscriptionURI}, {template}, {callbackurl})

Accepted content-types: json

Status Codes

- **200** – 200 OK

Example request:

```
POST /subscriptions HTTP/1.1
Host: example.com
Content-Type: text/javascript

{
  "id":"http://space/subscriptions/24534",
  "expiration":1200,
  "callbackURL":"http://callbackuri",
  "tpl":{"object":"http://object","predicate":"http://predicate","subject":"http://subject"}
  "nodesWhichAlreadyKnowTheSubscription":[],
}
```

`GET /subscriptions/`(**uri:** *subscription*)

Returns the subscription.

Accepted content-types: json

Parameters

- **subscription** (*URI*) – the URI of the subscription (must be encoded)

Status Codes

- **200** – 200 OK

DELETE /subscriptions/(uri: *subscription*)

unsubscribe({subscription})

Accepted content-types: json

Parameters

- **subscription** (*URI*) – the URI of the subscription to be deleted (must be encoded)

Status Codes

- **200** – 200 OK

PUT /subscriptions/(uri: *subscription*)

Updates an existing subscription. This method is used to extend the expiration time for a remote subscription.

Accepted content-types: json

Parameters

- **subscription** (*URI*) – the URI of the subscription to be deleted (must be encoded)

Status Codes

- **200** – 200 OK

Example request:

```
PUT /subscriptions/http%3A%2F%2Fmysubscription HTTP/1.1
Host: example.com
Content-Type: text/javascript

{
  "id":"http://space/subscriptions/24534",
  "expiration": 1200,
  "callbackURL": "http://callbackuri",
  "nodesWhichAlreadyKnowTheSubscription":[],
  "tpl": {"object":"http://object","predicate":"http://predicate","subject":"http://subject"}
}
```

POST /notifications

notify({template})

Accepted content-types: json

Status Codes

- **200** – 200 OK

Example request:

```
POST /subscriptions/http%3A%2F%2Fmysubscription HTTP/1.1
Host: example.com
Content-Type: text/javascript

{"subject":"http://subject","predicate":"http://predicate","object":"http://object"}
```

4.6 Acknowledgements

This project is supported or has been supported by the following projects:

- ISMED
- ACROSS
- THOFU
- TALIS+ENGINE

4.7 Publications related with Otsopack

4.7.1 Miscellaneous

Aitor Gómez-Goiri, Diego López-de-Ipiña. *On the complementarity of Triple Spaces and the Web of Things.* In Proceedings of the Second International Workshop on Web of Things, WoT‘11, pages 12:1–12:6. ISBN: 978-1-4503-0624-9. New York, NY, USA, 2011.

4.7.2 ACROSS

Aitor Gómez-Goiri, Pablo Orduña and Diego López-De-Ipiña. *RESTful Triple Spaces of Things*. Third International Workshop on the Web of Things (WoT 2012). Newcastle, UK, June 2012.

Aitor Gómez-Goiri, Pablo Orduña, David Ausín, Mikel Emaldi and Diego López-de-Ipiña. *Collaboration of Sensors and Actuators through Triple Spaces*. IEEE Sensors 2011. Limerick, Ireland, October 2011.

Xabier Laiseca, Eduardo Castillejo, Pablo Orduña, Aitor Gómez-Goiri, Diego López-de-Ipiña, Ester Gonzalez-Aguado. *Distributed Tracking System for Patients with Cognitive Impairments.* III International Workshop on Ambient Assisted Living - IWAAL 2011. Málaga, Spain, June 2011.

Aitor Gómez-Goiri, Eduardo Castillejo, Pablo Orduña, Xabier Laiseca, Diego López-de-Ipiña, Sergio Fínez. *Easing the Mobility of Disabled People in Supermarkets Using a Distributed Solution.* III International Workshop on Ambient Assisted Living - IWAAL 2011. Málaga, Spain, June 2011.

4.7.3 ISMED

Aitor Gómez-Goiri, Mikel Emaldi-Manrique, Diego López-de-Ipiña. *A Semantic Resource Oriented Middleware for Pervasive Environments*. UPGRADE journal, 2011, Issue No. 1: 5-16. ISSN: 1684-5285. February 2011.

Aitor Gómez-Goiri, Mikel Emaldi-Manrique, Diego López-de-Ipiña. *Middleware Semántico Orientado a Recursos para Entornos Ubicuos*. NOVATICA journal, (209): 9–16. ISSN: 0211-2124. February 2011.

Aitor Gómez-Goiri, Diego López-de-Ipiña. *A Triple Space-Based Semantic Distributed Middleware for Internet of Things.* In Florian Daniel and Federico Facca, editors, Current Trends in Web Engineering, volume 6385 of Lecture Notes in Computer Science, pages 447-458. Springer Berlin / Heidelberg, July 2010. 10.1007/978-3-642-16985-4 43.

4.7.4 TALIS+ENGINE

Aitor Gómez-Goiri and Diego López-de-Ipiña. *Assessing data dissemination strategies within Triple Spaces on the Web of Things*. Extending Seamlessly to the Internet of Things (esIoT). Palermo, Italy, July 2012. (to be published)

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

HTTP Routing Table