Deliverable reference:	Date:	Responsible partner:
D05.2	30 April 2015	CNet Svenska AB

Bridging Resources and Agencies in Large-Scale Emergency Management

BRIDGE is a collaborative project co-funded by the European Commission within the Seventh Framework Programme (FP7-SEC-2010-1)
SEC-2010.4.2-1: Interoperability of data, systems, tools and equipment
Grant Agreement No.: 261817
Duration: 1 April 2011 – 31 March 2015

www.sec-bridge.eu

Title:

BRIDGE Middleware

Editor(s):	Approved by:
Peeter. Kool	Dag Ausen
	Classification: Public

Abstract / Executive summary:

This deliverable describes the prototype D05.2 BRIDGE Middleware. The BRIDGE middleware infrastructure provides an abstraction layer for the network-level infrastructure, allowing easy access for higher-level components and GUI-layers. It also provides the means for managing the dynamic environment where services can come and go.

Document URL:	ISBN number:
http://www.sec-bridge.eu/deliverables/...	

Table of Contents

Version History

Version[1]	Description	Date	Who
0.1	Initial TOC	7.11.2011	Peeter Kool
0.2	Agent Platform Interoperability + FIPA	14.3.2012	David Mobach
0.3	AgentScape + Dataspace	15.3.2012	Reinier Timmer
0.4	CHAP Eve added, created first e-room version	22.4.2012	Hongliang Guo, Peeter Kool
0.5	Added Network Manager section	13.10.2014	Mark Vinkovits Peeter Kool
0.6	Added Service Catalogue section	23.10.2014	Mark Vinkovits Matts Ahlsen Peeter Kool
0.8	Added On-Site Storage and Transformation Services	22.02.2015	Peeter Kool
1.0	Version ready for peer-review	24.04.2015	Peeter Kool
1.1	Addressed comments from peer review by Sander van Splunter.	29.04.2015	Peeter Kool
1.2	Addressed comments from peer-review by Antoine Pultier	30.04.2015	Peeter Kool

Contributing partners

	**CNet** Danderyd Sweden	***Matts Ahlsén*** matts.ahlsen@cnet.se ***Peeter Kool*** Peeter Kool@cnet.se
	**UNIKLU** Alpen-Adria-Universität Klagenfurt Klagenfurt, Austria	**Christian Raffelsberger** christian.raffelsberger@aau.at
	**FIT** Fraunhofer-Institut für Angewandte Informationstechnik Sankt Augustin, Germany	**Andreas Zimmermann** andreas.zimmermann@fit.fraunhofer.de **Mark Vinkovits** Mark.vinkovits@fit.fraunhofer.de
	**SAAB Group** Sweden	**Andreas Carlsson** andreas.ac.carlsson@saabgroup.com
	**Thales R&T Nederland** Delft The Netherlands	**David Mobach** david.mobach@d-cis.nl **Reinier Timmer** reinier.timmer@d-cis.nl
	**Almende** Rotterdam Netherlands	**Ludo Stellingsdorf** ludo@almende.org

List of Figures

List of Abbreviations

DLNA	Digital Living Network Alliance
EDXL	Emergency Data Exchange Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JXTA	P2P protocol https://jxta.kenai.com/
P2P	Peer to Peer network
REST	Representational State Transfer
SCPD	Service Control Protocol Description
SOAP	Simple Object Access Protocol
SSDP	Simple Service Discovery Protocol
UPnP	Universal Plug and Play
WSDL	Web Services Description Language
XSLT	Extensible Stylesheet Language Transformations

1 Introduction

This report describes the prototype deliverable D5.2 Middleware which is the result from Task 5.2 Middleware. The aim of the task was to design and develop the middleware infrastructure for BRIDGE networks, which provides an abstraction layer to the network-level infrastructure and interfaces to higher-level components and GUI-layers. The work performed in this task has taken into account the results achieved in the HYDRA project: The HYDRA middleware was extended and adapted to meet the BRIDGE requirements elicited in WP2. Note that the HYDRA middleware has been renamed to LinkSmart, and will be referred as such in the remainder of this deliverable.

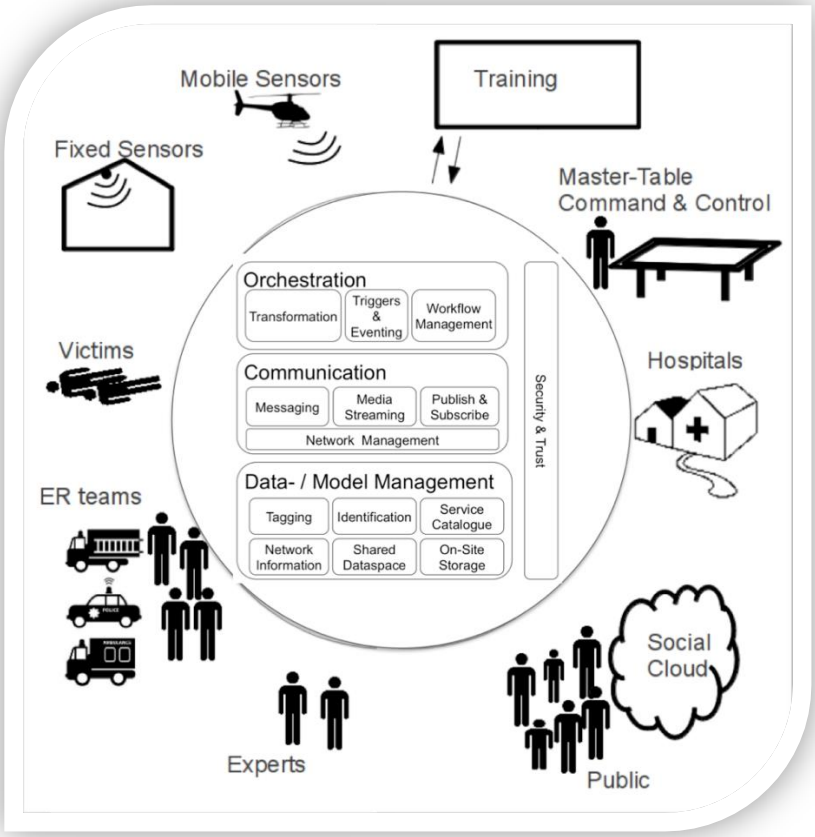

Figure 1: BRIDGE system of systems context

The main part of this deliverable deals with the extensions made to the LinkSmart components and new components that have been created to support the very dynamic environment where the BRIDGE middleware will be deployed. These components have been used in the different demonstrations providing the glue in-between the applications running on top of the BRIDGE middleware.

There is also a section, section 2.6, dedicated to the interoperability of the agent-based parts of the BRIDGE platform.

2 Overview

There are five main parts in the BRIDGE middleware which provide the virtualisation of the BRIDGE network as well as the means to provide a framework for managing a very dynamic environment:

- The Network Manager that provides the means to be able to communicate in between services.
- The Service Catalogue that provides information about the actual services available at a given time in the network. In the BRIDGE network we use two different means for searching depending on situation.
- The Shared Dataspace that provides services for storing and distributing information in the network, in addition it provides functionality for eventing.
- The On-Site Storage that provides a generic means for providing all sorts of information on the BRIDGE network on a retrieve if needed basis.
- The Transformation Component provides reusable templates in for transforming in-between formats and protocols.

The following subchapters describe these parts and illustrate their usage with examples. The components which are extensions to LinkSmart are already published on the LinkSmart source repository. Finally there is an appendix with code documentation of the most important parts.

2.1 The Network Manager

The BRIDGE Network Manager is based on LinkSmart Network Manager. In the BRIDGE project the code was refactored and new functionalities were added. Among the most important additions are:

- The Service Catalogues
- Tunneling of large objects in the BRIDGE network.
- The possibility to tunnel all HTTP based communications through the BRIDGE Network Manager, in order to provide support for REST based services.
- Modularized implementation to ease replacing and additions of new modules.

The BRIDGE network consist of a set of Network Managers that form a private secured P2P network with its own addressing space see Figure 2:

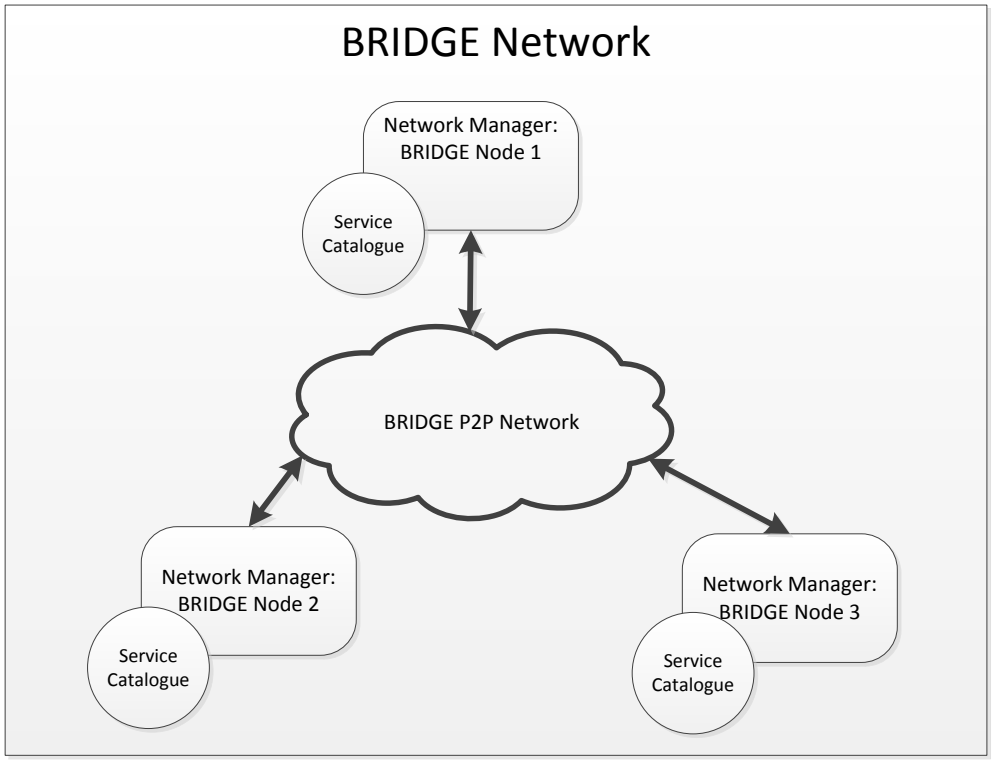

Figure 2: BRIDGE P2P Network

Below we describe how communication is done by the services and subsequently we describe the usage of the Service Catalogue. But first we look at the individual components within the Network Manager itself but in order to understand the components we start by explaining the concept of a Virtual Address.

2.1.1 Virtual Address

The Virtual Address object contains the address that each registered service has within the BRIDGE network. In the BRIDGE network each service endpoint is identified by a 32-byte long Virtual Address that has the following format: contextID-3.contextID-2.contextID-1.serviceID e.g.: 0.0.0.8248725583067352822.

The Virtual Address is used to access and invoke the service, it also keeps the IP endpoint hidden to ensure privacy and security of the service.

Virtual Address assignment is done automatically by the Network Manager when developers register their web services to it.

2.1.2 Network Manager Components

The Network Manager is a plug in based architecture where with well-defined components that have strict interfaces. These interfaces also provide the possibility for adding functionality without interfering with the rest of the code, or even completely replace one of the components. The following subchapters will provide a description of the modules and their functions. The overview of the architecture can be seen in Figure 3.

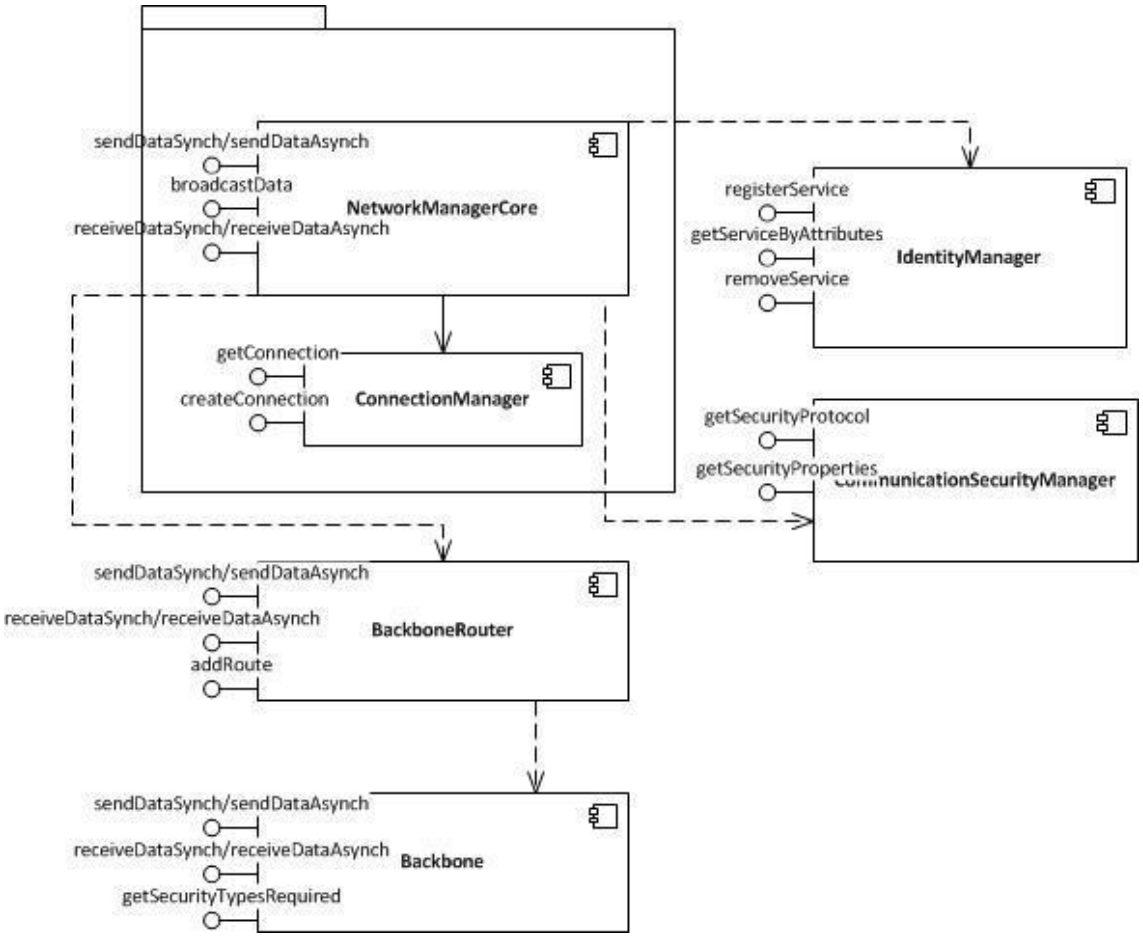

Figure 3: Architecture of the Network Manager

2.1.3 Backbone

Backbone bundles are responsible for handling the physical channel over which messages are sent. The backbone has to handle the synchronous and asynchronous nature of the communication as for the rest of the application backbone calls should always be handled asynchronously. How broadcasting and multicasting is done is also handled internally and with best effort. The Backbone holds a routing table to pair Virtual Addresses to physical Ids. The physical Ids should never leave the Backbone except for presentation purposes. The Backbone must resolve by itself how to retrieve Virtual Addresses from sent packages. The current implementation uses JXTA as the low level communication P2P implementation.

2.1.4 Backbone Router

The BackboneRouter is holding references and managing several Backbones. The BackboneRouter perform the mapping of a Virtual Address to a Backbone. The BackboneRouter offers methods to register and unregister routes. It does this also automatically by storing the Virtual Address to the Backbone its message came from. The NetworkManagerCore invokes the BackboneRouter to register route entries, i.e. a specific backbone implementation (e.g. JXTA, SOAP) mapped to an endpoint. In case the indicated backbone is not available yet, the BackboneRouter registers potential routes. Those potential routes can become active routes as soon as the BackboneRouter was able to bind the proper backbone implementation. If the backbone is unbound from the BackboneRouter the RouteEntry is removed from

the route cache. In future, an application might be able to assign more than one route entry to one Virtual Address, e.g. a REST WS endpoint could be registered in addition to a SOAP WS endpoint.

2.1.5 Network Manager Core

The Core implements the interfaces of the NetworkManager, which is the entry point to a LinkSmart network. It is the connection bundle between the different modules as it forwards requests coming to it to the destination. It has a ConnectionManager that holds references to Connections, which process data to send over the network. This includes security operations, compression, encoding, etc. The NetworkManagerCore should be kept as simple as possible and logic should be put into external modules.

2.1.6 Communication Security Manager

The CommunicationSecurityManager provides implementations of SecurityProtocol objects, which can be used for securing communication. Security for a connection is established by assigning a SecurityProtocol object to a Connection object. The CommunicationSecurityManager holds references to actual implementation bundles that implement the security scheme. This means the security can be easily exchanged even in run time.

2.1.7 Identity Manager

The IdentityManager is responsible for generating Virtual Addresses and for pairing Virtual Addresses to identities. An identity is independent from the Backbone it uses. This means that the IdentityManager should only see the application level Virtual Address identity and nothing else. A Virtual Address is in general an address and a set of attributes. What these attributes are is responsibility of the IdentityManager implementation.

2.1.8 HTTP Tunneling

An important concept in the network addressing is the usage of the HTTP tunnel provided by the Network Manager to invoke and access services on the BRIDGE network.

In the P2P overlay network of the BRIDGE network every node only communicates with its local NetworkManager. To ease the access to services of the NetworkManager, HTTP tunneling is provided allowing access between services over generic endpoints using Virtual Addresses.

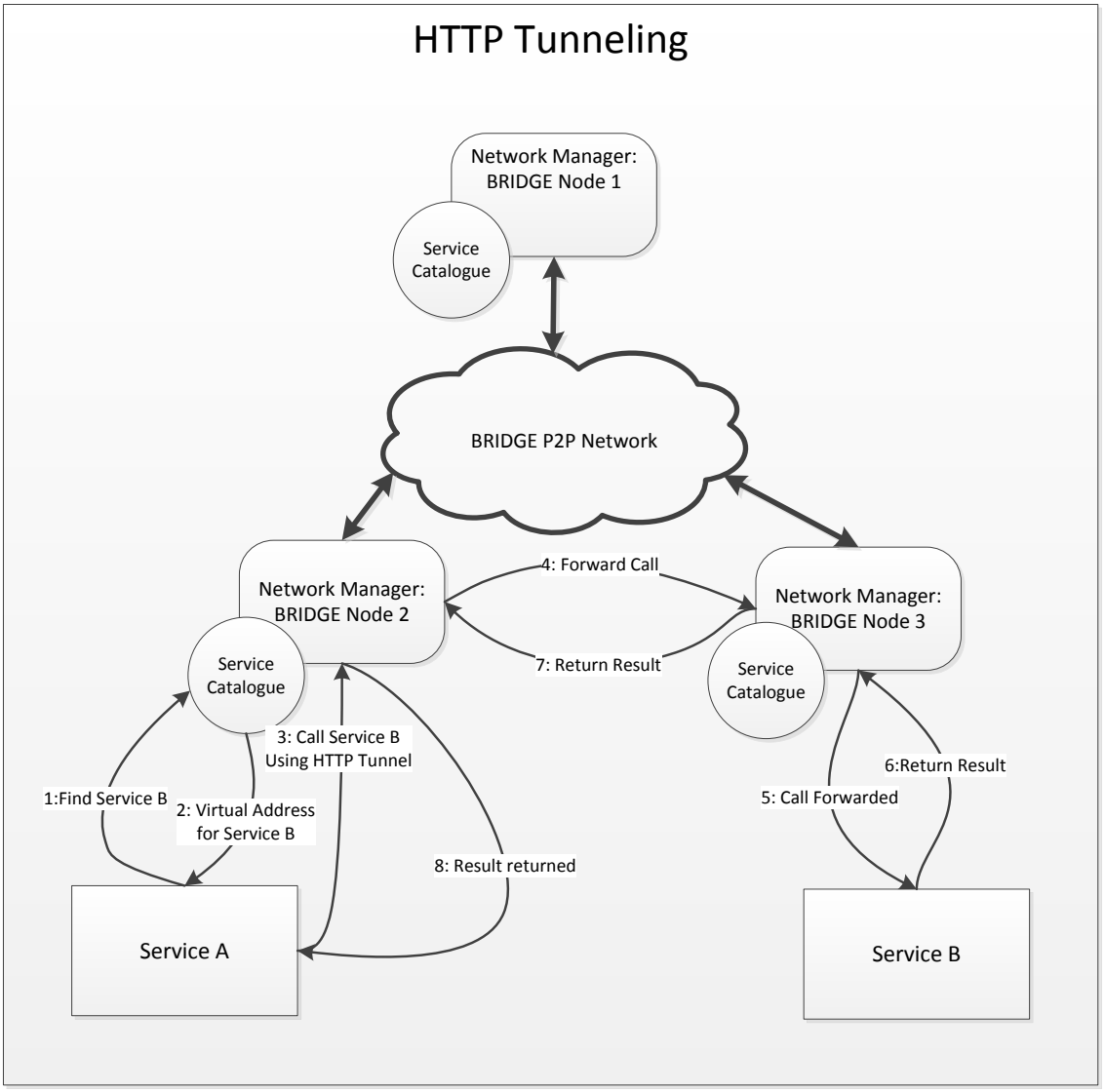

Figure 4: HTTP Tunneling example

A simple example of a service invocation is shown in Figure 4, where Service A wants to invoke Service B. The following are the basic steps that need to be followed:

1. Find the Virtual Address for Service B: This is done by asking the service catalogue for a specific service, see section 2.2.
2. If the service is found in the catalogue the Virtual Address will be returned.
3. Call Service B by creating the HTTP tunneling URL using the Virtual Address. If it is a Web Service that is invoked the endpoint for the Web Service Client is changed to the HTTP tunneling URL. If it is REST or other HTTP based call the tunneling URL can be used immediately. (The format of the Tunneling URL is explained in the following section)
4. The Network Manager forwards the call including the data to the Network Manager that has Service B using the P2P network.
5. Finally the call is forwarded to Service B supplying the data and any necessary headers.

6. Service B returns the data in the same way as usual.
7. The Network Manager returns the result to the invoking Network Manager using the P2P network.
8. Finally the result is returned to Service A.

The special code necessary to achieve this in Service A is that it needs to resolve Service B in the Service Catalogue, create the end point and change the Web Service Client endpoint. Service A only needs to register its service in the Service Catalogue. All other interactions and code is the same as if the services are addressed directly, i.e. all standard tools for creating Web Services/REST servers and clients can be used with HTTP tunneling.

Tunneling URL specification

The tunneling URL always begins with the local NetworkManager Address, port and the path /GRANDTunneling. For instance http://127.0.0.1:8082/GRANDTunneling. There are two basic ways to create the rest of the URL, one when the Virtual Address is known and one where the URL contains a Service Catalogue query.

Define recipient explicitly

The first most common case is when the URL is built using a known Virtual Address, i.e. it has been retrieved using the service catalogue. In this case the URL follows the following format:

http://NetworkManagerAddress:Port/GRANDTunneling/**senderAddress/recipientAddress**

- **senderAddress**: The Virtual Address of the sender of the message. Alternatively one can use 0 which basically means anonymous sender
- **recipientAddress**: The Virtual Address of the recipient

Define recipient by attributes

It is also possible to have the NetworkManager execute a service search over the P2P network by providing the search attributes in the URL. The generic format for this type of URL is as follows:

http://NetworkManagerAddress:Port/GRANDTunneling/**senderAddress**{/**default**}{/**1**}?**2**

- **{}** are optional parameters.
- **senderAddress**: The Virtual Address of the sender of the message. Alternatively one can use 0 which basically means anonymous sender
- **/default**: if there are multiple options the first one should be selected. If this part is missing and there are multiple options an error is returned
- **/1**: additional URL aimed for the registered service
- **2**: query of searchable attributes like description="calculator"&sid="eu.bridge.eventmanager" et c. These attributes are the ones defined in the Network Manager Service Catalogue, see 2.2.1.

2.2 Service Catalogue

In the BRIDGE middleware two types of service catalogues exist that can be used for finding services:

- The Network Manager Service Catalogue, which is synchronized in-between all the Network Manager nodes.
- The IoT Resource Catalogue that contains more metadata and annotations of the services available.

The reason for having two separate catalogues is performance, both for bandwidth as for lookup time. Secondly each satisfies two distinct usage scenarios. The main usage scenario for the Network Manager Service Catalogue is when the client knows exactly which type or instance of a service it wants to invoke, i.e. when it is known at design time what service is to be invoked. The main usage scenario for the IoT Resource Catalogue is when the client want to resolve services available in run time and invoke them dynamically, i.e. it will use the metadata and service description to decide which service to invoke.

2.2.1 The Network Manager Service Catalogue

This service catalogue has always to be used when invoking services since all service invocations require that Virtual Address of the invoked service is known. The catalogue is a simple attribute based description of the service itself that can be searched. There are only two attributes that are mandatory and one additional that is standardized but not mandatory:

- **DESCRIPTION:** The description of the service as a string, for instance SharedDataspaceThales:WebService
- **SID:** The service identity, for instance urn:http:ws:BRIDGE:Middleware:SharedDataSpace:1, this attribute describes which interface/service is implemented. Several instances of services with the same SID can coexist in the BRIDGE network.
- **PID(Optional):** Is a BRIDGE network unique id for this service. Useful for accessing a specific instance of a service.

Apart from these attributes one can define any other suitable attributes when registering the service, see Listing 1 below where HOST_NAME and START_TIME are added.

```
//Connect to the Service Catalogue
ServiceCatalogue.NetworkManager sc = new ServiceCatalogue.NetworkManager();
//Using the local network manager
sc.Url = "http://localhost:9090/cxf/services/NetworkManager";

//Using the local network manager
ServiceCatalogue.Part[] parts = new ServiceCatalogue.Part[5];
ServiceCatalogue.Part p = new ServiceCatalogue.Part();
//Create the DESCRIPTION (Mandatory key)
p.key = "DESCRIPTION";
p.value = "S2D2SDevice:StaticWS";
parts[0] = p;

//Create the SID (Mandatory key), Service ID
p = new ServiceCatalogue.Part();
p.key = "SID";
p.value = "urn:http:ws:BRIDGE:Middleware:SharedDataSpace:1";
parts[1] = p;

//Create the PID (Optional key), Persistent ID. Needs to be a unique name on
the BRIDGE network.
p = new ServiceCatalogue.Part();
p.key = "PID";
p.value = "my unique id";
parts[2] = p;

//Examples of additional keys
p = new ServiceCatalogue.Part();
p.key = "HOST_NAME";
p.value = Environment.MachineName;
parts[3] = p;

p = new ServiceCatalogue.Part();
p.key = "START_TIME";
p.value = DateTime.Now.ToString(); ;
parts[4] = p;

//Make the registration
ServiceCatalogue.Registration rid = sc.registerService(parts, m_wsendpoint,
"eu.linksmart.network.grand.impl.GrandMessageHandlerImpl");

HID = rid.virtualAddressAsString;
System.Console.WriteLine("Virtual Address:" + HID);
```

Listing 1: Example of service registration in C#.

The main usage of the Network Manager Service Catalogue is to provide a simple and efficient way of finding services when the caller knows what service it wants to find and then invoke. It is important to note that the Network Manager Service Catalogue responds always based on the services currently available and available to reach. If the BRIDGE network node becomes detached from the other nodes it will only contain the locally accessible nodes. But as soon it reaches other nodes they will synchronise so that all services on the other BRIDGE nodes are available. This provides the flexibility for applications to determine what services they can call at a given moment making it possible to perform graceful degradation.

2.2.2 IoT Resource Catalogue

The IoT Resource Catalogue provides the means to store more elaborate metadata regarding the services compared to the meta-date stored in the Network Manager Service Catalogue. The information in the IoT Resource Catalogue is not synchronised in-between BRIDGE network nodes and the IoT Resource Catalogue has to be found using the Network Manager Service Catalogue.

The IoT Resource Catalogue uses service descriptions that are expressed in an extended version of SCPD (Service Control Protocol Description), which is the standard for service descriptions in DLNA/UPnP. An example of the SCPD description is shown below in Listing 2.

The reason for using the extended SCPD format is that it is well defined, used for service discovery, and that it is possible to describe services independently of their implementation. These properties enable to describe REST based services which do not really have any established formal description language.

```
<?xml version="1.0" encoding="utf-8"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <device>
    <deviceType>urn:schemas-upnp-org:IoTdevice:SharedDataSpace:1</deviceType>
    <gateway xmlns="IoT">AIRBUS</gateway>
    <status xmlns="IoT">web service initiated</status>
    <wsendpoint
xmlns="IoT">http://192.168.9.96:8081/S2D2SDevice/S2D2SService</wsendpoint>
    <virtualAddress xmlns="IoT">128.5151.99292.22222</virtualAddress>
    <networkmanager xmlns="IoT" />
    <friendlyName>S2D2SDevice</friendlyName>
    <manufacturer>BRIDGE Integration Meeting</manufacturer>
    <manufacturerURL>http://wwwcnet.se</manufacturerURL>
    <modelDescription>Proxy for S2D2s</modelDescription>
    <modelName>S2D2s</modelName>
    <modelNumber>1</modelNumber>
    <UDN>uuid:caae981e-cf1f-4cf5-bcc7-6849b45144b2</UDN>
    <serviceList>
      <service>
        <serviceType>urn:schemas-upnp-org:service:shareddataspace:1</serviceType>
        <serviceId>urn:upnp-org:serviceId:shareddataspace</serviceId>
        <scpd xmlns="urn:schemas-upnp-org:service-1-0">
          <specVersion>
            <major>1</major>
            <minor>0</minor>
          </specVersion>
          <actionList>
            <action IoTannotation="">
              <name>ListSubscriptions</name>
              <argumentList>
                <argument>
                  <name>subscriptions</name>
                  <direction>out</direction>
                  <retval />
                  <relatedStateVariable>Subscriptions</relatedStateVariable>
                </argument>
              </argumentList>
            </action>
          </actionList>
          <serviceStateTable>
            <stateVariable sendEvents="no">
              <name>Subscriptions</name>
              <dataType>string</dataType>
            </stateVariable>
          </serviceStateTable>
        </scpd>
      </service>
    </serviceList>
  </device>
</root>
```

Listing 2: Example of a service description in SCPD

There are two ways to register an IoTResource, i.e. service, with the IoT Resource Catalogue:

- UPnP Discovery using SSDP and SCPD
- SELF Registration

If an IoTResource supports the UPnP Protocol the IoTResource will register automatically with the IoT Resource Catalogue. If a service is integrated into the BRIDGE network using the developer tools this information will be created mostly automatically and the service will be discovered dynamically by UPnP as well.

However, it also possible to manually register the service in the IoT Resource Catalogue by using the RegisterResource action of the catalogue service of the IoTResourceCatalogue.

IoTResource Query Language

The IoTResource Catalogue provides a query language for finding IoTResources and their services. This query language is based on the XPath language[2] for querying XML documents. The queried XML documents are IoTResource Description files which are based on the SCPD (Service Control Protocol Description) from the UPnP-standard. The IoTResource Catalogue takes an XPath expression and applies it to the SCPD document of the IoTResources. The IoTResources that matches the XPath expression are then returned.

IoTResource retrieval

The XPath querying can be used directly in the REST URL:

http://<catalogueendpoint>/<xpathexpression

for example,

http://<catalogueendpoint>//UPnP: serviceType [.='urn:schemas-upnp-org:service:shareddataspace:1']

returns all IoTResources that of the catalogue at that support the service "urn:schemas-upnp-org:service:shareddataspace:1"

http://192.168.9.15:44441/*	List all available resources
http://192.168.9.15:44441//UPnP:serviceType [.=' *urn:schemas-upnp-org:service:shareddataspace:1'*]	List all shared dataspace services known to the catalogue
http://192.168.9.15:44441//upnp:device[upnp:manufacturer='CNet']	List all resources from manufacturer CNet
http://192.168.9.15:44441//upnp:device[upnp:manufacturer='CNet'][UPnP:serviceType [.=' *urn:schemas-upnp-*	List all resources from manufacturer CNet that are

[2] http://www.w3schools.com/xpath/

org:service:shareddataspace:1']	shared data space services

All the methods are available in the WebService interface as well. The full documentation for the IoT Resource Catalogue service interface is in section 3.2.

2.3 The Shared Dataspace

The main purpose of the Shared Dataspace is to provide common storage within the BRIDGE network and also to provide a publish/subscribe based service for distribution of data as well as events. The actual implementation of the shared dataspace in BRIDGE is called S2D2S.

The S2D2S distributed dataspace has a relatively simple interface for publishing and retrieving data. The dataspace offers a mechanism to persist and forward data using a variety of infrastructures.

Depending on how the dataspace is configured, there may be one or more entry points to the dataspace. In the BRIDGE middleware, each S2D2S entry point is represented as a LinkSmart virtual device and offers access to the publish/subscribe API of S2D2S. Each cluster or subnet may have a dedicated entry point device to provide access to users of LinkSmart on the same network. However, internally these entry points could all be connected over the LinkSmart infrastructure.

From a user perspective there is not much difference between accessing one entry point or the other: the API will look exactly the same. However, the distribution of data may vary between clusters, so that actual data that can be seen could differ between entry points.

In order to ensure interoperability between different users of the dataspace (i.e. the agent platforms and other supporting systems) S2D2S offers a REST API based on JSON-RPC. This API is accessed via LinkSmart using a binding to a virtual S2D2S device. The REST API supports the following operations:

Method	**Description**	**Parameters**
Publish	Publish content to a specific topic	• **String** topic: topic to publish data • **String** contentType: type of data to publish • **String** payLoad: data to publish in S2D2S • **String** metadata: Additional data describing the payload • **String** persist: If true the data will be stored, if false the data will not be query able. False is used when one only wants to use publish/subscribe. • **String** itemId: If supplied this will replace the stored value of the instance with itemId.

subscribe	Subscribe to a specific topic.	• **String** topic: topic to subscribe • **String** url: callback URL address for notifications • **String** filter: Filter on specific attributes or content.
unsubscribe	Unsubscribe a specific subscription	• **String** subscriptionid
Query	Query a topic for specific content	• **String** topic: the topic to search for data • **String** filter: Filter on specific attributes or content.
list_subscriptions	Return a list of all active subscriptions.	•
removeByQuery	Remove specific content from a specific topic.	• **String** topic: the topic to search for data to be deleted • **String** filter: Filter on specific attributes or content for the deletion.

Table 1: S2D2S operations

These operations can all be accessed via JSON-RPC requests. The operations should be specified in the "method" fields and the parameters should be placed in the "params" map:

subscribe: registers an endpoint that will be notified when data changes

```
{
  "method": "subscribe",
  "params": {
    "topic": "App.Bridge.Some.Topic"
    "url": "http://linksmart.address/callback"
  }
 }
```

The **publish** method writes data into a topic. If there are any clients that used the **subscribe** operation for that particular topic, a notification of the new data is pushed to the URL of each subscribed user. Other users may use the **query** operation to inspect the topics periodically. This will enable users to search through all data (if it was persisted and has not been removed yet).

2.3.1 The Shared Dataspace proxy

A proxy was developed to make the S2D2S part of the BRIDGE network. The proxy enabled the registration of S2D2S in the service catalogue as well as the managing of the creation of HTTP tunneling URLs, see Figure 5.

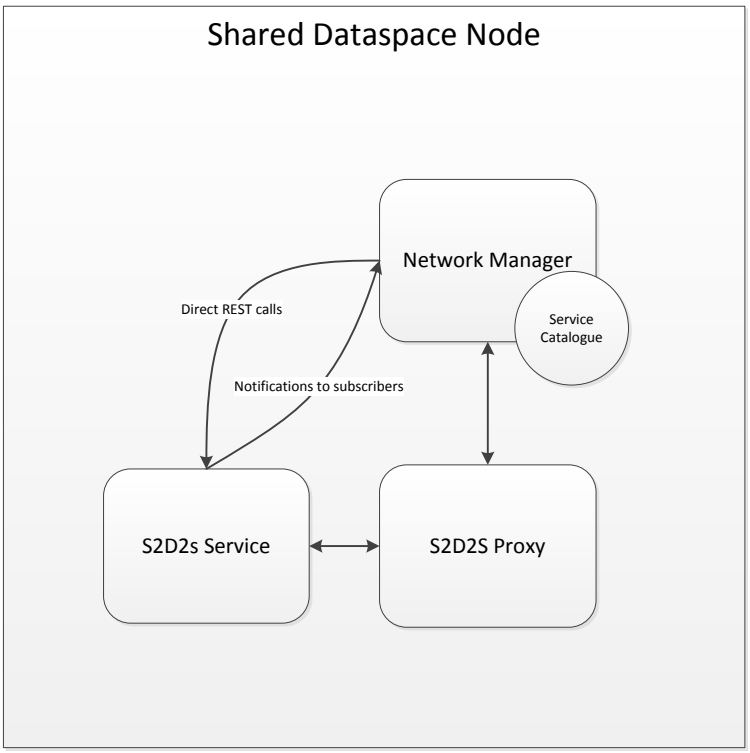

Figure 5: The shared dataspace proxy

The shared dataspace proxy called S2D2S Proxy registers the services in the Service Catalogues, both its own services as well as the direct REST interface to the S2D2s. This makes it possible for clients directly to find the REST service in the service catalogue and to invoke it using HTTP tunneling, thus bypassing the proxy.

The important part of the proxy is to automate the usage of the BRIDGE P2P network tunneling. Any client making a subscription provides its virtual address to the proxy that then creates the callback URL pointing to correct BRIDGE network tunnel endpoint. This means that all notifications to the subscribers go directly to the Network Manager for tunneling without needing to pass the proxy, see Figure 5.

For the actual documentation of the Shared Data Space proxy class see section 3.3.

2.4 On-Site Storage Service

The On-Site Storage Server provides a generic way of publishing data as links in the BRIDGE network, i.e. this enables the publication of larger pieces of data, such as images, without putting additional load on the BRIDGE network. Data will not be transmitted before somebody actually requests it.

The On-Site Storage Service uses the BRIDGE link format to express links.

2.4.1 BRIDGE Link Format

The BRIDGE link format is intended to be used when distributing links to be used for access to data that could be retrieved via the BRIDGE network.

The format is designed using a very simple schema. This allows a client to have multiple link instances to the same target data resource but in possibly different resolutions or formats (depending on target data type).

```
<xs:schema xmlns:bridge="urn:bridge:link" attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace="urn:bridge:link" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Links">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="Link">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Url" type="xs:string" />
              <xs:element name="Size" type="xs:unsignedInt" />
              <xs:element name="Mimetype" type="xs:string" />
            </xs:sequence>
            <xs:attribute name="description" type="xs:string" use="optional" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 6: BRIDGE Link schema

One Link consists of a *Url* that can be used to retrieve the data, *Size* in bytes of the data, *Mimetype* of the target data and an optional description attribute. This makes it possible to create a link that contains multiple resolutions and data formats and it is up to the consumer to decide if and what to retrieve from the source.

```
<bridge:Links xmlns:bridge="urn:bridge:link">
  <bridge:Link description="Full resulotion">
    <bridge:Url>http://127.0.0.1:8082/GRANDTunnel/0/122388338/0/x033282t7</bridge:Url>
    <bridge:Size>58937392</bridge:Size>
    <bridge:Mimetype>image/png</bridge:Mimetype>
  </bridge:Link>
  <bridge:Link description="Medium resulotion">
    <bridge:Url>http://127.0.0.1:8082/GRANDTunnel/0/122388338/0/x033232t7</bridge:Url>
    <bridge:Size>50883</bridge:Size>
    <bridge:Mimetype>image/png</bridge:Mimetype>
  </bridge:Link>
  <bridge:Link description="Small resolution">
    <bridge:Url>http://127.0.0.1:8082/GRANDTunnel/0/122388338/0/x033232t7</bridge:Url>
    <bridge:Size>3322</bridge:Size>
    <bridge:Mimetype>image/jpg</bridge:Mimetype>
  </bridge:Link>
</bridge:Links>
```

Figure 7: BRIDGE Link instance

Note that links expressed in the BRIDGE Link format cab either be expressed as standard URLs or that can be expressed using the Virtual Address Scheme, in Figure 7 above the links use the Virtual Address based HTTP Tunneling format described in section 2.1.8.

2.4.2 Usage of On-Site Storage

The On-Site Storage component can be used and configured in many ways. All uses involve services that have data that needs to be published on the BRIDGE network.

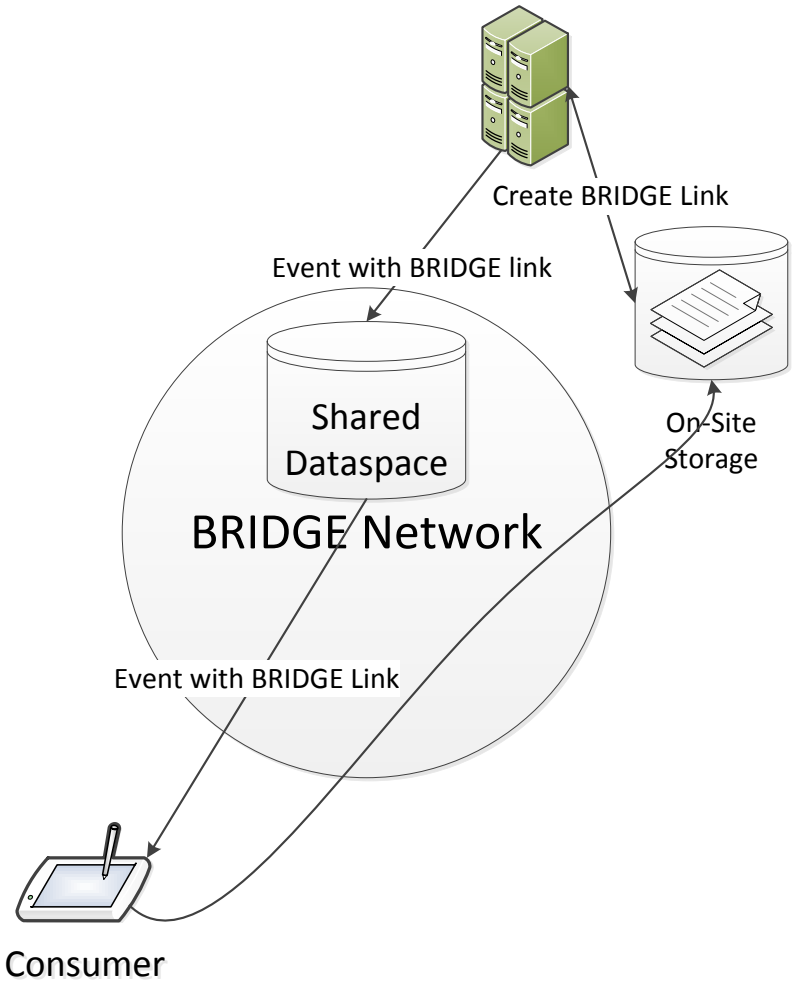

Figure 8: Example of BRIDGE On-Site Storage deployment

Figure 8 above is an example of the use of the On-Site Storage. In this case a service has a large piece of data that it wants publish on the network. Note that the service itself does not have its own HTTP based web server, instead it asks the On-Site storage to host the data. The On-Site storage stores the data and returns the BRIDGE Links that can be used to retrieve the data. The BRIDGE Links are then used as part of messages sent to be shared on the network. Finally a consumer of the information can decide to retrieve the data using the BRIDGE Link. This data will then be retrieved from the On-Site Storage component.

Note that the On-Site Storage component should be running in the same P2P node as the service that wants to publish the information if one wants to keep the communication overhead low. So the On-Site storage component is not a singular component in the network, instead all services that want to use the functionality should create their own instance locally.

The On-Storage component can also act passively and be used for registering an existing WWW based server and manage the registrations in the Service Catalogue as well as creating BRIDGE Link formatted links when the service needs them.

The third deployment option is that the On-Site Storage can be used to manage a complete directory structure, like a web server, making all of the information available for retrieval on the BRIDGE network. In this case the consumers would only need to use the service catalogue to find the appropriate endpoint and then by adding paths to the HTTP tunneling URL

The full documentation of the On-Site storage service is available in section 3.4.

2.5 Transformation Services

The Transformation Services within BRIDGE are used for transforming in-between different data formats and protocols. Since there is a wide array of different needs for transformations depending on usage scenarios the Transformation Services is not a singular service, rather it is a toolbox with different tools that can be applied depending on the needs and situation.

In BRIDGE two main types of Transformation Services were developed:

- Format Transformation: This is a set of templates for creating transformations to a specific target format.
- Protocol Transformation: A set of classes that can be used when developing tools and integrating systems into the BRIDGE platform.

The following subsections will outline the functionality of these libraries with examples.

2.5.1 Format Transformation

Format Transformation is a set of templates based on the W3C standard XSLT (Extensible Stylesheet Language Transformations). The templates are meant to be used by developers when they need to transform data into a specific format, for instance EDXL-RM. These templates provide the basic bolts for creating the correct output format and also include extension functions to the XSLT language for BRIDGE specific parts.

An example of a template is shown below in Listing 3.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt" exclude-result-prefixes="msxsl"
>

  <xsl:output method="xml" indent="yes"/>
  <xsl:param name="urlEnd" select="'?Description=Asa:Webserver'"/>
  <xsl:param name="messageGuid" select="'3F2504E0-4F89-41D3-9A0C-0305E82C3301'"/>
  <xsl:param name="webPath" select="'/StaticImage/'"/>
  <xsl:param name="urlStem" select="'http://127.0.0.1:8082/GRANDunneling/0/'"/>
  <xsl:param name="fileName" select="'image2.jpg'"/>
  <xsl:param name="timeNow" select="'2014-06-18T22:23:12.573Z'"/>
  <xsl:template match="/">
    <EDXLDistribution xmlns='urn:oasis:names:tc:emergency:EDXL:DE:1.0'>
      <distributionID>
        <xsl:value-of select='$messageGuid'/>
      </distributionID>
      <senderID>ASA@bridgeproject.eu</senderID>
      <dateTimeSent>
        <xsl:value-of select='$timeNow'/>
      </dateTimeSent>
      <distributionStatus>Exercise</distributionStatus>
      <distributionType>Request</distributionType>
      <combinedConfidentiality>UNCLASSIFIED AND NOT SENSITIVE</combinedConfidentiality>
      <language>EN</language>
      <contentObject>
```

```
      <contentDescription>MEXL-UAVStaticImageUpdate</contentDescription>
      <contentKeyword>
        <valueListUrn>http://icnet.mitre.org/ValueLists/ContentKeywords</valueListUrn>
        <value>MEXL-UAVStaticImageUpdate</value>
      </contentKeyword>
      <xmlContent>
        <embeddedXMLContent>
          <UAVStaticImageUpdate xmlns='urn:BRIDGE:ASA'>
            <ImagePosition xmlns=''>
                <xsl:variable name ='firstCoord' select='substring(.//PositionContent,2)' />
              <gml:Point xmlns:gml='http://www.opengis.net/gml'>
                <gml:pos>
                  <xsl:value-of select='translate(substring-before($firstCoord,"]"),","," ")'/>
                </gml:pos>
              </gml:Point>
            </ImagePosition>
            <SituationObservation xmlns=''>
              <ObservationText>
                <xsl:value-of select='.//Description'/>
              </ObservationText>
              <TimeStamp>
                <xsl:value-of select='.//TimeStampContent'/>
              </TimeStamp>
              <bridge:Links xmlns:bridge="urn:bridge:link">
                <bridge:Link description="UAV Image">
                  <bridge:Url>
                    <xsl:value-of select="concat($urlStem,$webPath,$fileName,$urlEnd)"/>
                  </bridge:Url>
                  <bridge:Size>58937392</bridge:Size>
                  <bridge:Mimetype>image/png</bridge:Mimetype>
                </bridge:Link>
              </bridge:Links>
            </SituationObservation>
          </UAVStaticImageUpdate>
        </embeddedXMLContent>
      </xmlContent>
    </contentObject>
  </EDXLDistribution>
</xsl:template>
</xsl:stylesheet>
```

Listing 3: Example of a Transformation template

This template is actually used when the UAV transforms its internal UAV static image format into an EDXL-DE based message. The different parts of the transformation are:

- First there is the parameter definition with default values, these values are easily interchangeable.
- Secondly there is an EDXL-DE envelope created with the correct format and namespaces containing all necessary information to create a valid message.
- Thirdly there is an XML content object which is UAV specific.
- Finally inside it there is a BRIDGE Link created that will point to the actual image.

These templates can be reused in different applications, but they will always be specific in the sense that they will reflect the structure of the incoming format. However if the incoming format is well defined and used by many components the template is completely reusable.

2.5.2 Protocol Transformation

As protocol transformation is hard to make completely generic within a middleware environment, we instead choose to develop modules that can be used for this transformation. These modules are used by the developer at development time. The developer tools provide support for selecting and integrating these

protocol transformations, the developer tools are explained in deliverable D8.2. Of course the tools are extendable and can be adapted with new Protocol Transformation components.

2.6 Agent Interoperability

Part of the BRIDGE platform functionality will be provided by software agent-platforms. These agent-platforms are provided by various partners and have specific internal architectures. To ensure that agents on different platforms are able to communicate with each other, and also to ensure that in the future other agent platforms will be able to join the BRIDGE platform, a standard method of information exchange is required.

As input for the agent interoperability the standards created by the Foundation for Intelligent Physical Agents (http://www.fipa.org/) were used as a baseline. The FIPA Specifications that are related to enabling agent-agent communication across platform boundaries are:

1. **ACL Message Structure (SC00061G)**
   Defines a set of message fields that allow flexible exchange of messages between agents.
2. **Agent Message Transport Service (SC00067F)**
   Provides a reference model for an agent Message Transport Service and a specification for message transport information (as an envelope around the FIPA ACL Message).
3. **Message Transport Protocol for HTTP (SC00084F)**
   Specifies the transport of messages between platforms over HTTP.
4. **FIPA Communicative Act Library (CAL) Specification (SC00037J)**
   Defines a library of standard communicative acts that can be used by agents.
5. **ACL Message Representation in String Specification (SC00070I)**
   Defines the syntax of the FIPA ACL in string form.

Using ACL Message Structure (SC00061G) the decision was connect the different agent platforms using JSON-RPC as the underlying protocol. A basic API and message structure have been agreed upon to exchange information. The API allows for an agent-platform to request a (JSON-RPC) communication endpoint at another platform, based on a specified endpoint type. Each created endpoint then implements a simple message passing interface to receive messages and replies to messages. This approach allows for agent-platforms to choose whether to expose a single endpoint (and return this endpoint upon each incoming creation request, and internally route/deliver the message to the correct agent), or for example to create a new agent for each endpoint creation request (and allow for messages to be delivered directly to the agent from the outside). The interface is specified below:

create: Returns the URL of a JSON-RPC endpoint

```
{
   "method": "create",
   "params": {
      "type": <integer indicating endpoint type>,
      "url":  <supply reverse JSON-RPC contact URL here>
   }
}
```

postMsg / postReply: Returns status indicating successful reception of message

```
{
  "method": "postMsg/postReply",
    "params": {
    "message": {
      "header": {
        "msg_type": <integer indicating message type>
      }
      "body": { <message specific content> }
    }
  }
 }
```

3 Software Components

3.1 Network Manager Service Catalogue API

getService

eu.linksmart.network.VirtualAddress **getService**()
throws java.rmi.RemoteException
Retrieves VirtualAddress of NetworkManagerCore instance.

Returns:
VirtualAddress of NetworkManagerCore instance

Throws:
java.rmi.RemoteException

registerService

eu.linksmart.network.Registration **registerService**(eu.linksmart.utils.Part[] *attributes*,
java.lang.String *endpoint*,
java.lang.String *backboneName*)
throws java.rmi.RemoteException
Creates a VirtualAddress for a particular service.

Parameters:
attributes - Attributes such as PID (which should be unique) or description
endpoint - Backbone specific endpoint for the service, e.g. URL or JXTA id
backboneName - Class name of the Backbone from which this service is reachable

Returns:
VirtualAddress instance.

Throws:
java.rmi.RemoteException

removeService

boolean **removeService**(eu.linksmart.network.VirtualAddress *virtualAddress*)
throws java.rmi.RemoteException
Removes a particular service from internal memory

Parameters:
virtualAddress - for particular service.

Returns:
TRUE if operation succeeded and FALSE if not.

Throws:
java.rmi.RemoteException

getAvailableBackbones

java.lang.String[] **getAvailableBackbones**()
throws java.rmi.RemoteException
Provides the list of names of communication channels or backbones the NetworkManager supports. This information can be used by a service to decide which channel to register over.

Returns:
Class names of the connected Backbones

Throws:
java.rmi.RemoteException

getServiceByAttributes

eu.linksmart.network.Registration[] **getServiceByAttributes**(
eu.linksmart.utils.Part[] *attributes*)
throws java.rmi.RemoteException

Simplest method to get services which match given attributes. If one or more services with these particular attributes are found, an array containing the registrations of services is returned. If a service does not contain all attributes that were used for the search (e.g. the service does not contain an attribute "description"), but the values of the other required attributes match, the service will be returned. If it has all attributes, but not all values of these attributes match the required values, the service is not returned. Method will wait default timeout to discover a remote set of registrations.

Parameters:
attributes - The attributes the service is supposed to have

Returns:
The services as registration objects, containing virtual addresses and attributes

Throws:
java.rmi.RemoteException

getServiceByAttributes

eu.linksmart.network.Registration[] **getServiceByAttributes**(
eu.linksmart.utils.Part[] *attributes*,
long *timeOut*,
boolean *returnFirst*,
boolean *isStrictRequest*)
throws java.rmi.RemoteException

Method to get services with given attributes. Requires additional parameters to control in detail how to search and what services to search for.

Parameters:
attributes - The attributes the service should have
timeOut - Time to wait for discovery responses
returnFirst - If true, method will stop searching when one service is found. If more than one service is found at the same time, the other services will be returned as well.
isStrictRequest -
true - only services will be discovered which possess all attributes
false - attribute types which a service does not have are ignored as long there is at least one matching attribute

Returns:
The services with matching attributes as registration objects. Even if returnFirst is set true more than one registration of services may be available.

Throws:
java.rmi.RemoteException

getServiceByPID

eu.linksmart.network.Registration **getServiceByPID**(java.lang.String *PID*)
java.lang.IllegalArgumentException,
java.rmi.RemoteException

Gets the VirtualAddress for the available service with a given PID.

Parameters:
PID - The persistent identifier of the service.

Returns:
The Registration object, null if no VirtualAddress exists for the given PID.

Throws:
java.rmi.RemoteException
java.lang.IllegalArgumentException

getServiceByDescription

eu.linksmart.network.Registration[] **getServiceByDescription**(java.lang.String *description*)
java.rmi.RemoteException

Gets the Registration for the available service(s) with a given description.

Parameters:
description - The required service description.

Returns:
The services with matching descriptions as registration objects

Throws:
java.rmi.RemoteException

getServiceByQuery

eu.linksmart.network.Registration[] **getServiceByQuery**(java.lang.String *query*)
java.rmi.RemoteException

Gets the registration objects for the locally available services matching the passed query. Remote services cannot be tested against the query containing other attributes than 'description'. This method should only be used by advanced developers.

Parameters:
query - The formulated query.

Returns:
The matching services as registration objects

Throws:
java.rmi.RemoteException

sendData

eu.linksmart.network.NMResponse **sendData**(eu.linksmart.network.VirtualAddress *sender*,
eu.linksmart.network.VirtualAddress *receiver*,

byte[] *data*,
boolean synch)

throws java.rmi.RemoteException

Send data from one LinkSmart node to another node.

Parameters:
sender - The virtual address of the sender
receiver - The virtual address of the receiver
data - The data to be sent
synch - boolean indicating whether method call should be synchronous or asynchronous

Returns:
Response instance

Throws:
java.rmi.RemoteException

3.2 IoT::IoTResourceCatalogue Class Reference

The **IoT** Resource Catalogue manages all knowledge regarding IoTResources that have been discovered and are active in the network. The **IoT** Resource Catalogue knows about IoTResources from a network perspective but does not handle the locations or context of the IoTResources.

Public Member Functions

- void **ResolveIoTResources** (System.String gateway, System.String discovermanagertype)
  Initiates a resolve process for unresolved IoTResources of a certain type on a specific gateway.
- void **DiscoverIoTResources** (System.String gateway)
  Initiates a discovery process on a specific gateway. The discovery will be done for all types of IoTResources.
- string **ProcessErrorMessage** (string IoTResourceId, string theMessage)
  Process an errormessage from a specifc device and returns the result.
- string **GetDeviceXML** (string IoTResourceId, string idtype)
  Gives an XML description of a device in SCPD (Service Control Point Document) format.
- string **GetIoTIoTResources** (string gateway)
  Gives a list of XML descriptions for all IoTResources at a gateway.
- string **GetIoTIoTResourcesFromXpath** (string xpath)
  Gives a list of XML descriptions for all IoTResources based on a XPath selection.
- string **GetIoTIoTResourcesEndpoints** (string gateway)
  Gives a list of web service endpoints for all IoTResources at a gateway.
- string **GetIoTIoTResourcesFromType** (string gateway, string devicetype)
  *Gives a list of XML descriptions for all **IoT** IoTResources at a gateway based on the device type.*
- string **InvokeIoTService** (string IoTResourceId, string idtype, string method, string arguments)
  *Allows invocation of any method offered in the general **IoT** service of a device.*
- string **InvokeService** (string IoTResourceId, string idtype, string serviceid, string method, string arguments)
  Allows invocation of any method offered in any service of a device.
- string **InvokeServiceXPath** (string xpath, string serviceid, string method, string arguments)

Allows invocation of any method offered in any service on a set of IoTResources selected by an Xpath expression.

- string **AddDevice** (string devicedescription)
  Allows manual adding of IoTResources to the network that cannot be discovered using the default discovery protocol.
- void **RemoveDevice** (string IoTResourceId, string idtype)
  *Removes a device from the **IoT** Resource Catalogue and stops the device.*
- string **GetIoTURLsFromXpath** (string xpath, string VirtualAddressType, string sender, string callerNMSoapTunelUriURL)
  *Returns a list of **IoT** encoded urls for the IoTResources that match xpath.*
- bool **IsRegistered** (string VirtualAddress)
  *Tells if a device with a given **IoT** ID is registered with the catalogue.*
- string **GetWSEndpoint** (string IoTResourceId, string idtype)
  Returns the web service endpoint for a given device.
- string **GetIoTWSEndpoint** (string IoTResourceId, string idtype)
  *Returns the endpoint for the generic **IoT** web service a given device.*
- string **GetWSDL** (string IoTResourceId, string idtype)
  Returns the WSDL description of a given device.
- void **StartIoTResources** (string xpath)
  Starts IoTResources that match a given xpath expression. The expression is applied to the SCPD XML of the device.
- void **StopIoTResources** (string xpath)
  Stops IoTResources that match a given xpath expression. The expression is applied to the SCPD XML of the device.
- string **GetVirtualAddress** (string application, string devicelocalid)
  Returns the VirtualAddress for a device based on the local application id assinged to IoTResources.
- string **GetVirtualAddresssFromXPath** (string application, string xpath, string VirtualAddressType)
  Returns the VirtualAddress for a device based on an xpath description which is applied to the SCPD devoce model.

Detailed Description

The **IoT** Resource Catalogue manages all knowledge regarding IoTResources that have been discovered and are active in the network. The **IoT** Resource Catalogue knows about IoTResources from a network perspective but does not handle the locations or context of the IoTResources.

Member Function Documentation

void IoT::IoTResourceCatalogue::ResolveIoTResources (System.String *gateway*, System.String *discovermanagertype*) `[inline]`

Initiates a resolve process for unresolved IoTResources of a certain type on a specific gateway.

Parameters:

gateway The gateway for physical IoTResources
discovermanagertype The discovery manager which should resolve the IoTResources, for instance BluetoothDiscoverymanager,

void IoT::IoTResourceCatalogue::DiscoverIoTResources (System.String *gateway*) `[inline]`

Initiates a discovery process on a specific gateway. The discovery will be done for all types of IoTResources.

Parameters:

gateway The gateway for physical IoTResources, where to do a discovery process

string IoT::IoTResourceCatalogue::ProcessErrorMessage (string *IoTResourceId*, string *theMessage*) `[inline]`

Process an errormessage from a specific device and returns the result.

Parameters:

IoTResourceId The unique Virtual Address for the device
theMessage The error message

string IoT::IoTResourceCatalogue::GetDeviceXML (string *IoTResourceId*, string *idtype*) `[inline]`

Gives an XML description of a device in SCPD (Service Control Point Document) format.

Parameters:

IoTResourceId The an id for the IoTResource
idtype The type of identifier used, values could be UDN, FriendlyName, or Virtual Address

Returns:

A string with an SPCD XML for the device

string IoT::IoTResourceCatalogue::GetIoTIoTResources (string *gateway*) `[inline]`

Gives a list of XML descriptions for all IoTResources at a gateway.

Parameters:

gateway The name of the gateway

Returns:

A string with SPCD XML:s for all IoTResources at gateway

string IoT::IoTResourceCatalogue::GetIoTIoTResourcesFromXpath (string *xpath*) `[inline]`

Gives a list of XML descriptions for all IoTResources based on a XPath selection.

Parameters:

xpath An XPath expression that will be applied to the device XML as a selection filter. IoTResources that match the Xpath expression, will be selected.

Returns:

A string with SPCD XML:s for all IoTResources at gateway

string IoT::IoTResourceCatalogue::GetIoTIoTResourcesEndpoints (string *gateway*) `[inline]`

Gives a list of web service endpoints for all IoTResources at a gateway.

Parameters:

gateway The name of the gateway,if empty it will return IoTResources for all gateways

Returns:

An XML string with pairs of device id:s and their web service endpoints

string IoT::IoTResourceCatalogue::GetIoTIoTResourcesFromType (string *gateway*, string *devicetype*) `[inline]`

Gives a list of XML descriptions for all **IoT** IoTResources at a gateway based on the device type.

Parameters:

gateway The name of the gateway,if empty it will return IoTResources for all gateways
devicetype A device URN

Returns:

A string with SPCD XML:s for all IoTResources that match the device type

string IoT::IoTResourceCatalogue::InvokeIoTService (string *IoTResourceId*, string *idtype*, string *method*, string *arguments*) `[inline]`

Allows invocation of any method offered in the general **IoT** service of a device.

Parameters:

IoTResourceId The id for the device
idtype The type of identfier used, values could be UDN, FriendlyName, or Virtual Address
method The method to invoke
arguments Arguments to use following the format: par1=12;par2=mystring;par3=45

Returns:

A string with the result of the invocation

string IoT::IoTResourceCatalogue::InvokeService (string *IoTResourceId*, string *idtype*, string *serviceid*, string *method*, string *arguments*) `[inline]`

Allows invocation of any method offered in any service of a device.

Parameters:

IoTResourceId The id for the device
idtype The type of identifier used, values could be UDN, FriendlyName, or Virtual Address
serviceid The serviceid following the format "urn:upnp-org:serviceId:weatherservice:thermometer:1"
method The method to invoke
arguments Arguments to use following the format: par1=12;par2=mystring;par3=45

Returns:

A string with the result of the invocation

string IoT::IoTResourceCatalogue::InvokeServiceXPath (string *xpath*, string *serviceid*, string *method*, string *arguments*) [inline]

Allows invocation of any method offered in any service on a set of IoTResources selected by an Xpath expression.

Parameters:

xpath An xpath expression to select IoTResources for which the method invocation should be done
serviceid The serviceid following the format "urn:upnp-org:serviceId:weatherservice:thermometer:1"
method The method to invoke
arguments Arguments to use following the format: par1=12;par2=mystring;par3=45

Returns:

A string with the result of the invocation
A string with the result of the invocation

string IoT::IoTResourceCatalogue::AddDevice (string *devicedescription*) [inline]

Allows manual adding of IoTResources to the network that cannot be discovered using the default discovery protocol.

Parameters:

devicedescription An SPCD description of the device to be added

void IoT::IoTResourceCatalogue::RemoveDevice (string *IoTResourceId*, string *idtype*) [inline]

Removes a device from the **IoT** Resource Catalogue and stops the device.

Parameters:

IoTResourceId The id for the device
idtype The type of identifier used, values could be UDN, FriendlyName, or Virtual Address

string IoT::IoTResourceCatalogue::GetIoTURLsFromXpath (string *xpath*, string *VirtualAddressType*, string *sender*, string *callerNMSoapTunelUriURL*) [inline]

Returns a list of **IoT** encoded urls for the IoTResources that match xpath.

Parameters:

xpath Valid xpath expression
VirtualAddressType The type ofVirtualAddressused
sender TheVirtualAddressof the sender, normally an empty string
callerNMSoapTunelUriURL The url for the callers SOAP tunnel, if null is sent in `http://localhost:8082/GRANDTunneling/` is used

bool IoT::IoTResourceCatalogue::IsRegistered (string *VirtualAddress*) [inline]

Tells if a device with a given **IoT** ID is registered with the catalogue.

Parameters:

Virtual Address The id for the device

Returns:

True if a device with the Virtual Address is registered otherwise false

string IoT::IoTResourceCatalogue::GetWSEndpoint (string *IoTResourceId*, string *idtype*) [inline]

Returns the web service endpoint for a given device.

Parameters:

IoTResourceId The id for the device
idtype The type of identifier used, values could be UDN, FriendlyName, or Virtual Address

string IoT::IoTResourceCatalogue::GetIoTWSEndpoint (string *IoTResourceId*, string *idtype*) [inline]

Returns the endpoint for the generic **IoT** web service a given device.

Parameters:

IoTResourceId The id for the device
idtype The type of identifier used, values could be UDN, FriendlyName, or Virtual Address

string IoT::IoTResourceCatalogue::GetWSDL (string *IoTResourceId*, string *idtype*) [inline]

Returns the WSDL description of a given device.

Parameters:

IoTResourceId The id for the device
idtype The type of identifier used, values could be UDN, FriendlyName, or Virtual Address

void IoT::IoTResourceCatalogue::StartIoTResources (string *xpath*) [inline]

Starts IoTResources that match a given xpath expression. The expression is applied to the SCPD XML of the device.

Parameters:

xpath A valid Xpath expression

void IoT::IoTResourceCatalogue::StopIoTResources (string *xpath*) [inline]

Stops IoTResources that match a given xpath expression. The expression is applied to the SCPD XML of the device.

Parameters:

xpath A valid Xpath expression

string IoT::IoTResourceCatalogue::GetVirtualAddress (string *application*, string *devicelocalid*) [inline]

Returns the Virtual Address for a device based on the local application id assigned to IoTResources.

Parameters:

application The application were the device resides
devicelocalid The local id for the device within the application for instance MyDiscoBall

string IoT::IoTResourceCatalogue::GetVirtualAddresssFromXPath (string *application*, string *xpath*, string *VirtualAddressType*) `[inline]`

Returns the Virtual Address for a device based on an xpath description which is applied to the SCPD device model.

Parameters:

application The application were the device resides
xpath a valid xpath expression

3.3 Shared Dataspace Class Reference

This class implements the proxy for the shared dataspace.

Public Member Functions

- void **publish** (System.String topic, System.String contentType, System.String metadata, System.String payLoad, System.String persist, System.String itemId)
  Publishes on the specified topic.
- void **query** (System.String topic, System.String filter, out System.String dataItem)
  Queries the specified topic.
- void **subscribe** (System.String topic, System.String filter, System.String callBack)
  Subscribes the specified topic.
- void **unSubscribe** (System.String subscriptionId)
  Unsubscribe.
- void **Remove** (System.String Topic, System.String ItemId)
  Removes the specified data item.
- System.String **ListSubscriptions** ()
  Lists all the subscriptions.
- System.String **subscribeWithServiceFilter** (System.String topic, System.String filter, System.String Filter)
  Subscribes to the topic with a filter.

Detailed Description

This class implements the proxy for the shared dataspace.

Member Function Documentation

void SharedDateSpaceNS::SharedDataspace::publish (System.String *topic*, System.String *contentType*, System.String *metadata*, System.String *payLoad*, System.String *persist*, System.String *itemId*) `[inline]`

Publishes on the specified topic.

Parameters:

topic The topic.
contentType Type of the content.
metadata The metadata.
payLoad The pay load.
persist The persist.
itemId The item identifier.

void SharedDateSpaceNS::SharedDataspace::query (System.String *topic*, System.String *filter*, out System.String *dataItem*) `[inline]`

Queries the specified topic.

Parameters:

topic The topic.
filter The filter.
dataItem The data item.

void SharedDateSpaceNS::SharedDataspace::subscribe (System.String *topic*, System.String *filter*, System.String *callBack*) `[inline]`

Subscribes the specified topic.

Parameters:

topic The topic.
filter The filter.
callBack The call back.

void SharedDateSpaceNS::SharedDataspace::unSubscribe (System.String *subscriptionId*) `[inline]`

Unsubscribe.

Parameters:

subscriptionId The subscription identifier.

void SharedDateSpaceNS::SharedDataspace::Remove (System.String *Topic*, System.String *ItemId*) `[inline]`

Removes the specified data item.

Parameters:

Topic The topic.
ItemId The item identifier.

System.String SharedDateSpaceNS::SharedDataspace::ListSubscriptions () `[inline]`

Lists all the subscriptions.

Returns:

System.String SharedDateSpaceNS::SharedDataspace::subscribeWithServiceFilter (System.String *topic*, System.String *filter*, System.String *Filter*) `[inline]`

Subscribes to the topic with a filter.

Parameters:

topic The topic.
filter The filter.
Filter The filter.

Returns:

3.4 OnSiteStorage Class Reference

3.4.1 OnSiteStorage::FileInfo Class Reference

Public Member Functions

- **FileInfo** ()
- **FileInfo** (string fileLocation, string size, string mimetype, string description)
  *Initializes a new instance of the **FileInfo** class.*

Properties

- string **Filelocation** `[get, set]`
  Gets or sets the filelocation.
- string **Mimetype** `[get, set]`
  Gets or sets the mimetype.
- string **Size** `[get, set]`
  Gets or sets the size.
- string **Description** `[get, set]`
  Gets or sets the description.

Constructor & Destructor Documentation

OnSiteStorage::FileInfo::FileInfo () `[inline]`

OnSiteStorage::FileInfo::FileInfo (string *fileLocation*, string *size*, string *mimetype*, string *description*) `[inline]`

Initializes a new instance of the **FileInfo** class.

Parameters:

fileLocation The file location.
size The size.
mimetype The mimetype.
description The description.

Property Documentation

string OnSiteStorage::FileInfo::Filelocation `[get, set]`

Gets or sets the filelocation.

The filelocation.

string OnSiteStorage::FileInfo::Mimetype `[get, set]`

Gets or sets the mimetype.

The mimetype.

string OnSiteStorage::FileInfo::Size `[get, set]`

Gets or sets the size.

The size.

string OnSiteStorage::FileInfo::Description `[get, set]`

Gets or sets the description.

The description.

3.4.2 OnSiteStorage::OnSiteStorage Class Reference

This is the interface for the On-Site Storage Manager.

Public Member Functions

- bool **ConnectToNetwork** (string SID, string description, out string VirtualAddress)
  Connects the On-Site storage to the network and registers in the Service Catalogue.
- bool **DeConnectFromNetwork** ()
  Removes the On-Site storage from the network and deregisters from Service Catalogue.
- bool **CreateNetworkRegistration** (string SID, string description, string endpoint, out string VirtualAddress)
  Creates the network registration for an existing HTTP based service.
- bool **DeleteNetworkRegistration** (string VirtualAddress)
  Deletes the network registration in the service catalogue.
- bool **StartPublishing** (int port)
  Starts the internal HTTP server in On-Site storage.
- bool **StopPublishing** ()
  Stops the publishing.

Private Member Functions

- XmlDocument **CreateBRIDGELink** (**FileInfo**[] dataItem)
  Creates the BRIDGE link.
- string **PublishData** (**FileInfo**[] dataItems, out XmlDocument BRIDGELink)
  Publishes the data.
- bool **UnPublishData** (string dataId)
  Removes published data.
- **FileInfo**[] **ListAllPublishedData** ()
  Lists all data published in the On-Site Storage.

Detailed Description

This is the interface for the On-Site Storage Manager.

Member Function Documentation

bool OnSiteStorage::OnSiteStorage::ConnectToNetwork (string *SID*, string *description*, out string *VirtualAddress*) `[inline]`

Connects the On-Site storage to the network and registers in the Service Catalogue.

Parameters:

SID The sid.
description The description.
VirtualAddress The virtual address for started service.

Returns:

true if successfully registered

bool OnSiteStorage::OnSiteStorage::DeConnectFromNetwork () `[inline]`

Removes the On-Site storage from the network and deregisters from Service Catalogue.

Returns:

bool OnSiteStorage::OnSiteStorage::CreateNetworkRegistration (string *SID*, string *description*, string *endpoint*, out string *VirtualAddress*) `[inline]`

Creates the network registration for an existing HTTP based service.

Parameters:

SID The sid.
description The description.
endpoint The endpoint of the HTTP service.
VirtualAddress The virtual address.

Returns:

true if successfully registered

bool OnSiteStorage::OnSiteStorage::DeleteNetworkRegistration (string *VirtualAddress*) `[inline]`

Deletes the network registration in the service catalogue.

Parameters:

VirtualAddress The virtual address.

Returns:

true if successfully deregistered

bool OnSiteStorage::OnSiteStorage::StartPublishing (int *port*) `[inline]`

Starts the internal HTTP server in On-Site storage.

Parameters:

port The port to be used

Returns:

TRUE if successful

bool OnSiteStorage::OnSiteStorage::StopPublishing () `[inline]`

Stops the publishing.

Returns:

TRUE if successful;

XmlDocument OnSiteStorage::OnSiteStorage::CreateBRIDGELink (FileInfo[] *dataItem*) `[inline, private]`

Creates the BRIDGE link.

Parameters:

dataItems The data items to part of the BRIDGE link

Returns:

The BRIDGE link in an XmlDocument

string OnSiteStorage::OnSiteStorage::PublishData (FileInfo[] *dataItems*, out XmlDocument *BRIDGELink*) `[inline, private]`

Publishes the data.

Parameters:

dataItems The data items.
BRIDGELink The created bridge link.

Returns:

The ID of the published data

bool OnSiteStorage::OnSiteStorage::UnPublishData (string *dataId*) `[inline, private]`

Removes published data.

Parameters:

dataId The data item identifier.

Returns:

TRUE if successful

FileInfo [] OnSiteStorage::OnSiteStorage::ListAllPublishedData () `[inline, private]`

Lists all data published in the On-Site Storage.

Returns:

All currently published data items