Deliverable reference:	Date:	Responsible partner:
D08.2	11 May 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:

Integrated and Quality Assured BRIDGE platform

Editor(s):	Approved by:
Peeter Kool	PCC
	Classification: Public

Abstract / Executive summary:

This deliverable describes the prototype D08.2 Integrated and Quality Assured BRIDGE platform. The main part of the deliverable describes some typical integration cases in detail, outlining how the BRIDGE platform can be reused and extended. There is also a section that describes which tools and software components are available from the project and how to use them.

Note that this is not the final version of the deliverable but it will be extended and submitted before the end of the project. The main addition that will be added is the example of creating and extending the BRIDGE platform with protocol transformations, see section 2.3. Finally, the Open Source section 4 will be updated with additional components.

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

ISBN number:
-

Table of Contents

Version History

Version[1]	Description	Date	Who
0.1	Initial TOC	7.11.2011	Peeter Kool
0.2	ASA Integration Case	09.06.2014	Peeter Kool, Aslak Eide, Antoine Pultier Mark Vinkovits
0.3	IoTResource Builder, initial content	10.10.2014	Matts Ahlsén Peeter Kool
0.4	BRIDGE WISE Integration	31.3.2015	Andreas Carlsson, Peeter Kool
0.5	Added initial content for Open Source section	15.4.2015	Peeter Kool, Ludo Stellingwerff
0.6	Edited for final draft version to be submitted before the review	11.05.2015	Matts Ahlsén 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
	SINTEF Strindveien 4 7034 Trondheim Norway	**Antoine Pultier** Antoine.Pultier@sintef.no **Aslak Eide** Arnor.Solberg@sintef.no
	SAAB Group Sweden	**Andreas Carlsson** andreas.ac.carlsson@saabgroup.com
	Almende Rotterdam Netherlands	**Ludo Stellingwerff** 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
USDL	Unified Service Description Language
WSDL	Web Services Description Language
XSLT	Extensible Stylesheet Language Transformations

1 Introduction

This report describes the prototype deliverable D08.2 Integrated and Quality Assured BRIDGE platform which is the result from Task 8.2 Integration of System Components.

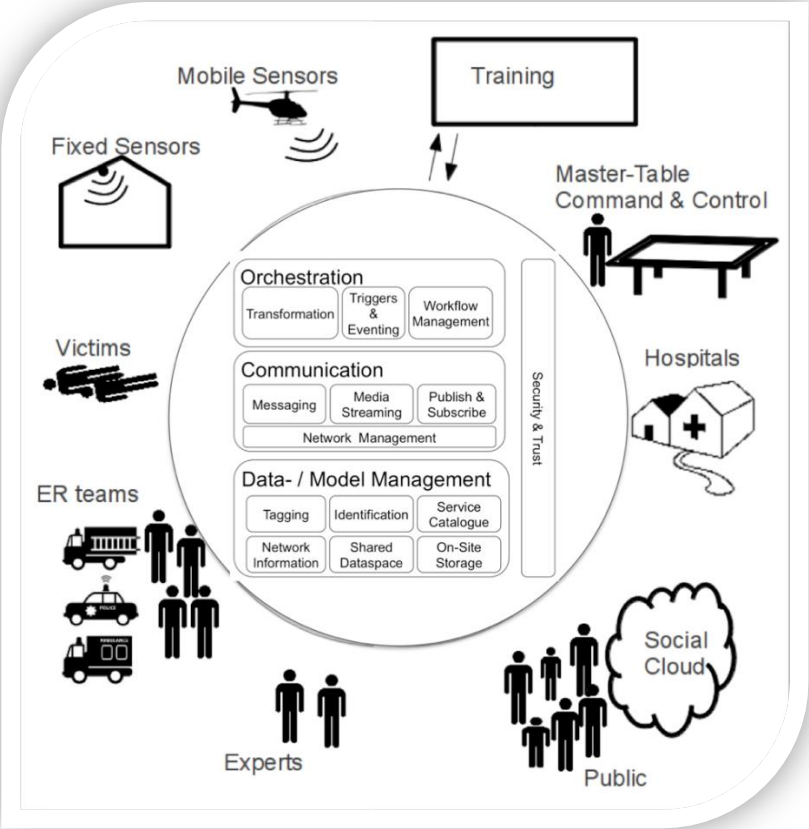

Figure 1: BRIDGE system of systems context

The BRIDGE system of systems is a very complex platform with many interacting components. The components themselves and client systems interfaces are primarily described in other deliverables:

- Middleware in D05.2, together with D05.3
- Agent based components including QoS in D07.5A and D07.4
- Connection to command post (the Master) and mobile devices in D06.3
- The overall system architecture and service organization is described in deliverables D04.2 and D04.3

The main part of the deliverable describes some typical integration cases in detail, outlining how the BRIDGE platform can be reused and extended. There is also a section that describes which tools and software components are available from the project and how to use them.

Finally there is also a section, section 4, dedicated to describe the parts of the BRIDGE platform which have been published as open source in different public repositories. This section contain links where to find the source code as well as other developer resources.

Note that this is not the final version of the deliverable but it will be extended and submitted before the end of the project. The main addition that will be added is the example of creating and extending BRIDGE platform with protocol transformations, see section 2.3. Additionally the Open Source section 4 will be updated with additional components.

2 Integrations

In order to describe some typical cases of integration with the BRIDGE platform we have selected four cases which are based on actual integrations made within the project. The four cases are:

- Publishing information and events on the BRIDGE platform
- Consuming information and events on the BRIDGE platform
- Creating a protocol transformation
- Enabling a service in the BRIDGE platform

The following subsections will describe these types of integrations.

2.1 Publishing information and events

This case is based on the integration of the Advanced Situational Awareness (ASA) concept case in to the BRIDGE platform. The main goals for this integration is to link the information generated by the ASA systems to be published on the BRIDGE network. The basic architecture is shown in Figure 2.

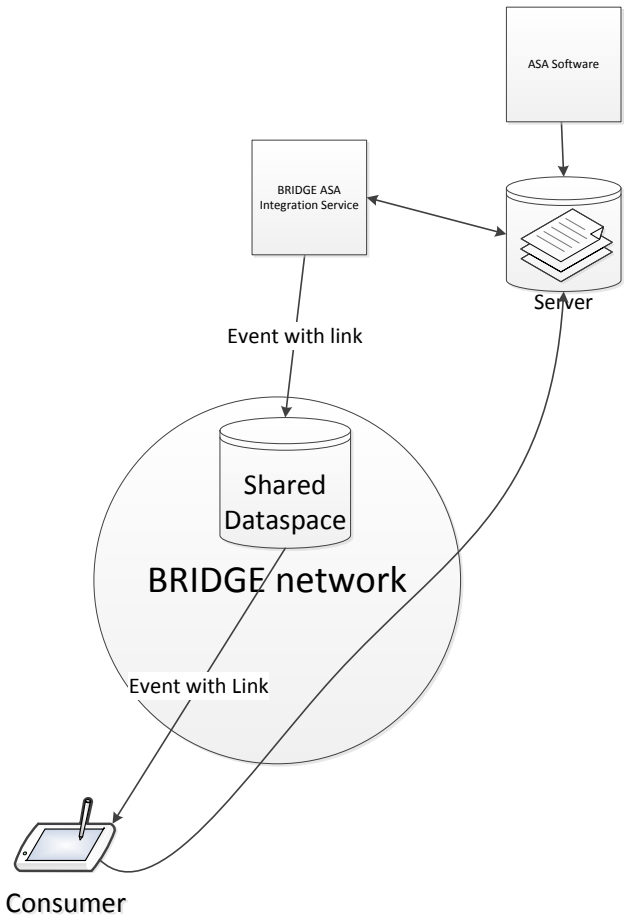

Figure 2: Architecture of the ASA Integration

The output from the ASA system is stored in a file system that is part of a web server.

The main components in the architecture are:

- ASA Software: All the programs that produce information, i.e. Hexacopter, Expert System and Modelling.
- WWW Server: A webserver provided by ASA to distribute information.
- BRIDGE ASA Integration Service: Will publish the available information in the ASA WWW Server to the BRIDGE network.

The integration with the BRIDGE platform is done using the file system at the ASA server as connection, i.e. all information transfer is using the file system as media.

An important concept when the ASA software publishes information is Metadata files. These are XML files which contain extra information that describes the content files/streams that are published on the ASA WWW Server. This is the information that the BRIDGE ASA Integration Service will use to create the events that will be sent to the shared dataspace.

Since we do not want to send the actual information payload with the event the events will contain links that can be used to retrieve the information when needed. The link format is described in BRIDGE deliverable D05.2[1].

2.1.1 The BRIDGE ASA Integration Service

The BRIDGE ASA Integration Service (BAIS) was created as the glue in-between the BRIDGE network and the ASA system. The implementation of BAIS is based on BRIDGE middleware components.

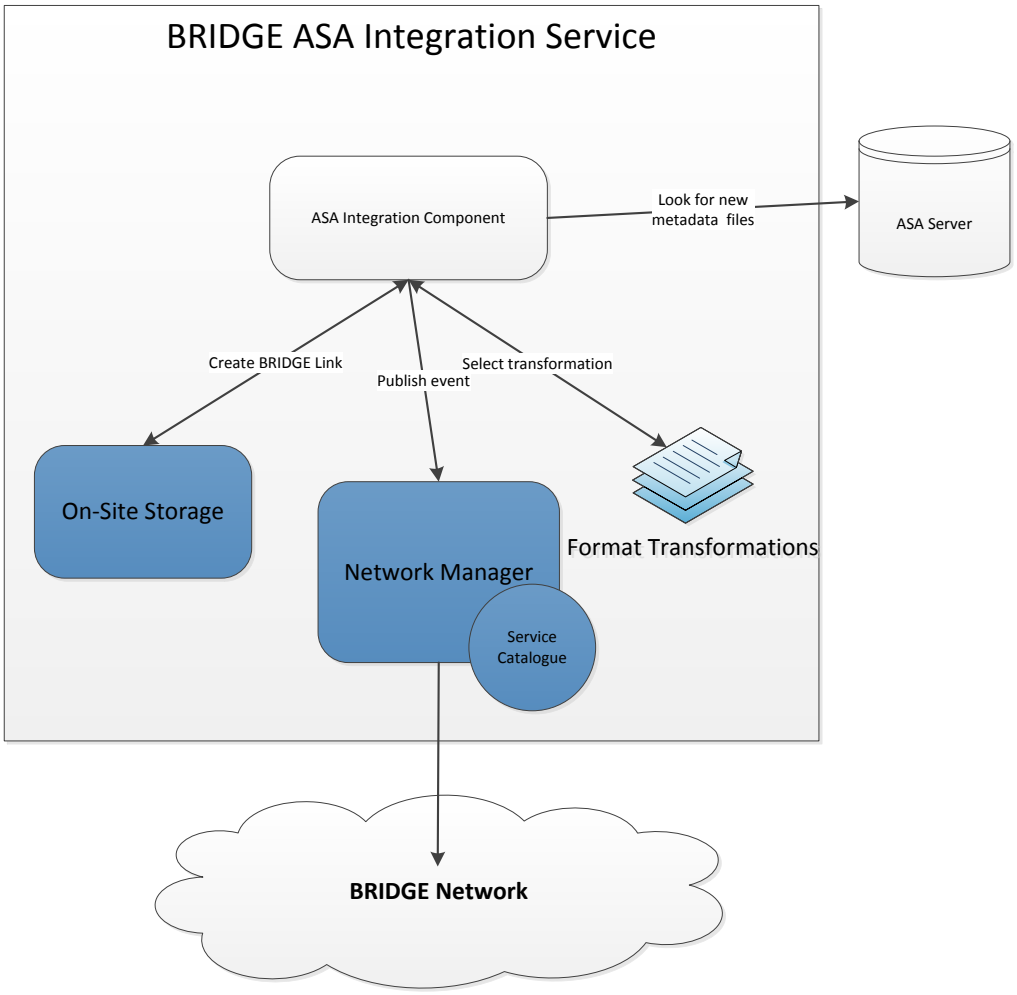

Figure 3: The BRIDGE ASA Integration Service Architecture

In Figure 3 the components in the BAIS which are blue colored are BRIDGE Middleware components. The architecture of the BAIS contains the following components:

- ASA Integration Component: Contains all the integration specific bespoke functionality. I.e. it contains the logic necessary to access the ASA server and retrieve information. It will then use the BRIDGE Middleware components to enable the ASA data to be published and consumed in the BRIDGE network.
- On-Site Storage: In this integration the On-Site storage is used for creating the BRIDGE links that enables access to information such as images etc. over the BRIDGE network.
- Format Transformations: Provide transformations from the ASA internal formats to BRIDGE network compatible formats, i.e. EDXL based formats.
- Network Manager and Service Catalogue: Enables communication with the BRIDGE network.

In the BRIDGE ASA Integration Service a number of integrations were implemented for the different types of information that ASA to be published and made available on the BRIDGE network:

- Advise: Output from the expert system
- Modelling data: Outputs from modelling, i.e. plume dispersion images et c.
- UAV Resource data: i.e. Position, status.
- UAV Static Images: Images captured by the UAV operator.
- UAV Video feeds: The feed from the video and IR cameras on the UAV.

We have selected to highlight two of these integrations in the next subsections to illustrate the how these integrations were implemented. The two different cases represent the typical patterns used in the ASA integration.

Modelling data

This is the output from the modelling, for instance toxic plume dispersion over time. Each output contains files in jpg format of different resolutions to enable the consumer of the information to select resolution depending on need, for instance bandwidth constraints or display device resolution.

```
<ASAMetaData>
    <Description>Plume modelling output 30 minutes with wind change to SW 3 m/s </Description>
    <TimeStamp>2014-05-22T23:02:01.122Z</TimeStamp>
        <!-- In seconds (optional) -->
    <ModelTimeAfterIncident>1800</ModelTimeAfterIncident>
        <!--GPS coordinates For UAV images and video in [x,y(,z)]... format-->
    <PositionContent>[18.9827839, 69.7031209][19.9827839, 69.7031209]</PositionContent>
        <!-- Upper left and lower right -->
</ASAMetaData>
```

Listing 1: Metadata file for modelling data

Listing 1 shows an example of the Metadata file that accompanies modelling images. This file contains additional information that is used for creating the BRIDGE event that is published. The Metadata fields are:

- Description: Plain text describing the contents being published.
- TimeStamp: At what time was this model created.
- ModelTimeAfterIncident. Optional field that contains the number of seconds past from the incident the contents depict.
- PostionContent: Upper left and lower right geo coordinates for the model. This is useful to overlay the images on a map.

When the ASA Integration Service discovers this Metadata file it creates the BRIDGE link using the On-Site Storage service and the finally selects an appropriate Format Transformation.

```
<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="'/CloudModels/'"/>
  <xsl:param name="urlStem" select="'http://127.0.0.1:8082/SOAPTunneling/0/0/'"/>
  <xsl:param name="fileName" select="'Test'"/>
  <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-CloudModelUpdate</contentDescription>
        <contentKeyword>
          <valueListUrn>http://icnet.mitre.org/ValueLists/ContentKeywords</valueListUrn>
          <value>MEXL-CloudModelUpdate</value>
        </contentKeyword>
        <xmlContent>
          <embeddedXMLContent>
            <CloudModelUpdate xmlns='urn:BRIDGE:ASA'>
              <CloudRectangle xmlns=''>
                <TopLeft xmlns=''>
                  <xsl:variable name ='firstCoord' select='substring-
before(substring(.//PositionContent,2),"[")' />
                <gml:Point xmlns:gml='http://www.opengis.net/gml'>
                  <gml:pos>
                    <xsl:value-of select='translate(substring-before($firstCoord,"]"),","," 
")'/>
                  </gml:pos>
                </gml:Point>
                </TopLeft>
                <LowerRight xmlns=''>
                  <xsl:variable name ='secondCoord' select='substring-
after(substring(.//PositionContent,2),"[")' />
                  <gml:Point xmlns:gml='http://www.opengis.net/gml'>
                    <gml:pos>
                      <xsl:value-of select='translate(substring-before($secondCoord,"]"),","," 
")'/>
                    </gml:pos>
                  </gml:Point>
                </LowerRight>
              </CloudRectangle>
              <SituationObservation xmlns=''>
                <ObservationText>
                  <xsl:value-of select='.//Description'/>
                </ObservationText>
                <ModelTimeAfterIncident>
                  <xsl:value-of select='.//ModelTimeAfterIncident'/>
                </ModelTimeAfterIncident>
                <TimeStamp>
                  <xsl:value-of select='.//TimeStamp'/>
                </TimeStamp>
                <xsl:copy-of select='bridge:Links' xmlns:bridge="urn:bridge:link"/>
              </SituationObservation>
            </CloudModelUpdate>
          </embeddedXMLContent>
        </xmlContent>
      </contentObject>
```

```
    </EDXLDistribution>
  </xsl:template>
</xsl:stylesheet>
```

Listing 2: Modelling data transformation.

Listing 2 shows the XSLT transformation that transforms the Modelling data with the BRIDGE links into an EDXL based BRIDGE format that can be consumed on the BRIDGE network. It can be noted that the transformation also does transformation of the actual data values, in this case coordinates are transformed into GML based format.

```
<EDXLDistribution xmlns="urn:oasis:names:tc:emergency:EDXL:DE:1.0">
  <distributionID>3F2504E0-4F89-41D3-9A0C-0305E82C3301</distributionID>
  <senderID>ASA@bridgeproject.eu</senderID>
  <dateTimeSent>2015-04-01T22:23:12.573Z</dateTimeSent>
  <distributionStatus>Exercise</distributionStatus>
  <distributionType>Request</distributionType>
  <combinedConfidentiality>UNCLASSIFIED AND NOT SENSITIVE</combinedConfidentiality>
  <language>EN</language>
  <contentObject>
    <contentDescription>MEXL-CloudModelUpdate</contentDescription>
    <contentKeyword>
      <valueListUrn>http://icnet.mitre.org/ValueLists/ContentKeywords</valueListUrn>
      <value>MEXL-CloudModelUpdate</value>
    </contentKeyword>
    <xmlContent>
      <embeddedXMLContent>
        <CloudModelUpdate xmlns="urn:BRIDGE:ASA">
          <CloudRectangle xmlns="">
            <TopLeft>
              <gml:Point xmlns:gml="http://www.opengis.net/gml">
                <gml:pos>18.9827839  69.7031209</gml:pos>
              </gml:Point>
            </TopLeft>
            <LowerRight>
              <gml:Point xmlns:gml="http://www.opengis.net/gml">
                <gml:pos>19.9827839  69.7031209</gml:pos>
              </gml:Point>
            </LowerRight>
          </CloudRectangle>
          <SituationObservation xmlns="">
            <ObservationText>Plume modelling output 30 minutes with wind change to  SW 3 m/s
</ObservationText>
            <ModelTimeAfterIncident>1800</ModelTimeAfterIncident>
            <TimeStamp>2014-05-22T23:02:01.122Z</TimeStamp>
            <bridge:Links xmlns:bridge="urn:bridge:link">
              <bridge:Link description="Full resolution">

<bridge:Url>http://127.0.0.1:8082/GrandTunneling/0/0/CloudModels/Testhigh.jpg?Description=Asa:We
bserver</bridge:Url>
                <bridge:Size>58937392</bridge:Size>
                <bridge:Mimetype>image/png</bridge:Mimetype>
              </bridge:Link>
              <bridge:Link description="Medium resolution">

<bridge:Url>http://127.0.0.1:8082/GrandTunneling/0/0/CloudModels/Testmiddle.jpg?Description=Asa:
Webserver</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/GrandTunneling/0/0/CloudModels/Testlow.jpg?Description=Asa:Web
server</bridge:Url>
                <bridge:Size>3322</bridge:Size>
```

```
            <bridge:Mimetype>image/jpg</bridge:Mimetype>
          </bridge:Link>
        </bridge:Links>
      </SituationObservation>
    </CloudModelUpdate>
  </embeddedXMLContent>
 </xmlContent>
</contentObject>
</EDXLDistribution>
```

Listing 3: Example of the final event format for Modelling data

Listing 3 shows the final result of the transformation wrapped in EDXL format. This data is then forwarded to the BRIDGE network as an event using the Network Manager and the Service Catalogue.

UAV Position

The UAV position and status is continuously updated every 3-5 seconds when the UAV is active. Since this is very lightweight information only a Metadata file will be created and no other data files.

```
<ASAMetaData>
  <!-- Description is Resource ID, i.e. the UAV identity-->
  <Description>UAV1</Description>
  <TimeStamp>2014-05-22T23:02:01.122Z</TimeStamp>
  <!--GPS coordinates For [x,y(,z)]... format-->
  <PositionContent>[18.9827839, 69.7031209, 120]</PositionContent>
</ASAMetaData>
```

Listing 4: UAV Position Metadata

Listing 4 shows an example of the Metadata file that accompanies the UAV position. This file contains additional information that is used for creating the BRIDGE event that is published. The Metadata fields are:

- Description: Resource ID, i.e. the UAV identity.
- TimeStamp: At what time the UAV position was recorded.
- PostionContent: The UAV position including height.

When the ASA Integration Service discovers this Metadata file it will immediately select the appropriate Format Transformation. Since there is no additional data associated with the Metadata there is no need to invoke the On-Site storage because all of the information will be contained in the event that is published on the BRIDGE network.

```
<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="messageGuid" select="'3F2504E0-4F89-41D3-9A0C-0305E82C3301'"/>
  <xsl:param name="fileName" select="'Test'"/>
  <xsl:param name="timeNow" select="'2014-06-18T22:23:12.573Z'"/>
  <xsl:template match="/">
```

```
    <ReportResourceDeploymentStatus xmlns="urn:oasis:names:tc:emergency:EDXL:RM:1.0:msg"
xmlns:rm="urn:oasis:names:tc:emergency:EDXL:RM:1.0" xmlns:geo-
oasis="urn:oasis:names:tc:emergency:EDXL:HAVE:1.0:geo-oasis"
xmlns:xal="urn:oasis:names:tc:ciq:xal:3" xmlns:xnl="urn:oasis:names:tc:ciq:xnl:3"
xmlns:xpil="urn:oasis:names:tc:ciq:xpil:3" xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:emergency:EDXL:RM:1.0:msg EDXL-
RMReportResourceDeploymentStatus.xsd">
      <MessageContentType xmlns="">ReportResourceDeploymentStatus</MessageContentType>
      <MessageID xmlns="">
        <xsl:value-of select='$messageGuid'/>
      </MessageID>
      <SentDateTime xmlns="">
        <xsl:value-of select='$timeNow'/>
      </SentDateTime>
      <OriginatingMessageID xmlns="">001</OriginatingMessageID>
      <ContactInformation xmlns="">
        <ContactRole>Sender</ContactRole>
      </ContactInformation>
      <ResourceInformation xmlns="">
        <ResourceInfoElementID>1</ResourceInfoElementID>
        <Resource>
          <ResourceID>0ec33f9b-b8f1-44e2-99de-0a8dde04a68b</ResourceID>
          <Name>UAV</Name>
          <TypeStructure>
            <rm:Value>UAV</rm:Value>
            <rm:ValueListURN>urn:x-hazard:vocab:resourceTypes</rm:ValueListURN>
          </TypeStructure>
          <ResourceStatus>
            <DeploymentStatus>
              <rm:Value>Unknown</rm:Value>
              <rm:ValueListURN>urn:x-hazard:vocab:deploymentStatusTypes</rm:ValueListURN>
            </DeploymentStatus>
          </ResourceStatus>
        </Resource>
        <ScheduleInformation>
          <ScheduleType>Current</ScheduleType>
          <Location>
            <rm:TargetArea>
              <xsl:variable name ='firstCoord' select='substring-
before(substring(.//PositionContent,2),"]")' />
              <gml:Point>
                <gml:pos>
                  <xsl:value-of select='translate($firstCoord,","," ")'/>
                </gml:pos>
              </gml:Point>
            </rm:TargetArea>
          </Location>
        </ScheduleInformation>
      </ResourceInformation>
    </ReportResourceDeploymentStatus>
  </xsl:template>
</xsl:stylesheet>
```

Listing 5: The UAV position transformation

Listing 5 shows the transformation that is used for creating the BRIDGE event from UAV position Metadata. In this case it creates an EDXL-RM formatted message.

```
<ReportResourceDeploymentStatus xsi:schemaLocation="urn:oasis:names:tc:emergency:EDXL:RM:1.0:msg
EDXL-RMReportResourceDeploymentStatus.xsd" xmlns="urn:oasis:names:tc:emergency:EDXL:RM:1.0:msg"
xmlns:rm="urn:oasis:names:tc:emergency:EDXL:RM:1.0" xmlns:geo-
oasis="urn:oasis:names:tc:emergency:EDXL:HAVE:1.0:geo-oasis"
xmlns:xal="urn:oasis:names:tc:ciq:xal:3" xmlns:xnl="urn:oasis:names:tc:ciq:xnl:3"
```

```
xmlns:xpil="urn:oasis:names:tc:ciq:xpil:3" xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <MessageContentType xmlns="">ReportResourceDeploymentStatus</MessageContentType>
  <MessageID xmlns="">3F2504E0-4F89-41D3-9A0C-0305E82C3301</MessageID>
  <SentDateTime xmlns="">2015-04-11T22:23:12.573Z</SentDateTime>
  <OriginatingMessageID xmlns="">001</OriginatingMessageID>
  <ContactInformation xmlns="">
    <ContactRole>Sender</ContactRole>
  </ContactInformation>
  <ResourceInformation xmlns="">
    <ResourceInfoElementID>1</ResourceInfoElementID>
    <Resource>
      <ResourceID>0ec33f9b-b8f1-44e2-99de-0a8dde04a68b</ResourceID>
      <Name>UAV</Name>
      <TypeStructure>
        <rm:Value>UAV</rm:Value>
        <rm:ValueListURN>urn:x-hazard:vocab:resourceTypes</rm:ValueListURN>
      </TypeStructure>
      <ResourceStatus>
        <DeploymentStatus>
          <rm:Value>Unknown</rm:Value>
          <rm:ValueListURN>urn:x-hazard:vocab:deploymentStatusTypes</rm:ValueListURN>
        </DeploymentStatus>
      </ResourceStatus>
    </Resource>
    <ScheduleInformation>
      <ScheduleType>Current</ScheduleType>
      <Location>
        <rm:TargetArea>
          <gml:Point>
            <gml:pos>18.9827839  69.7031209  120</gml:pos>
          </gml:Point>
        </rm:TargetArea>
      </Location>
    </ScheduleInformation>
  </ResourceInformation>
</ReportResourceDeploymentStatus>
```

Listing 6: The EDXL-RM message that is created from UAV position Metadata

Listing 6 shows the final result of the transformation wrapped in EDXL-RM format. This data is then forwarded to the BRIDGE network as an event using the Network Manager and the Service Catalogue.

2.2 Consuming information and events

This case is based on the actual integration of the WISE system in the BRIDGE platform. The basic functionality of the integration is to allow WISE to eavesdrop on events and data being processed on the BRIDGE network.

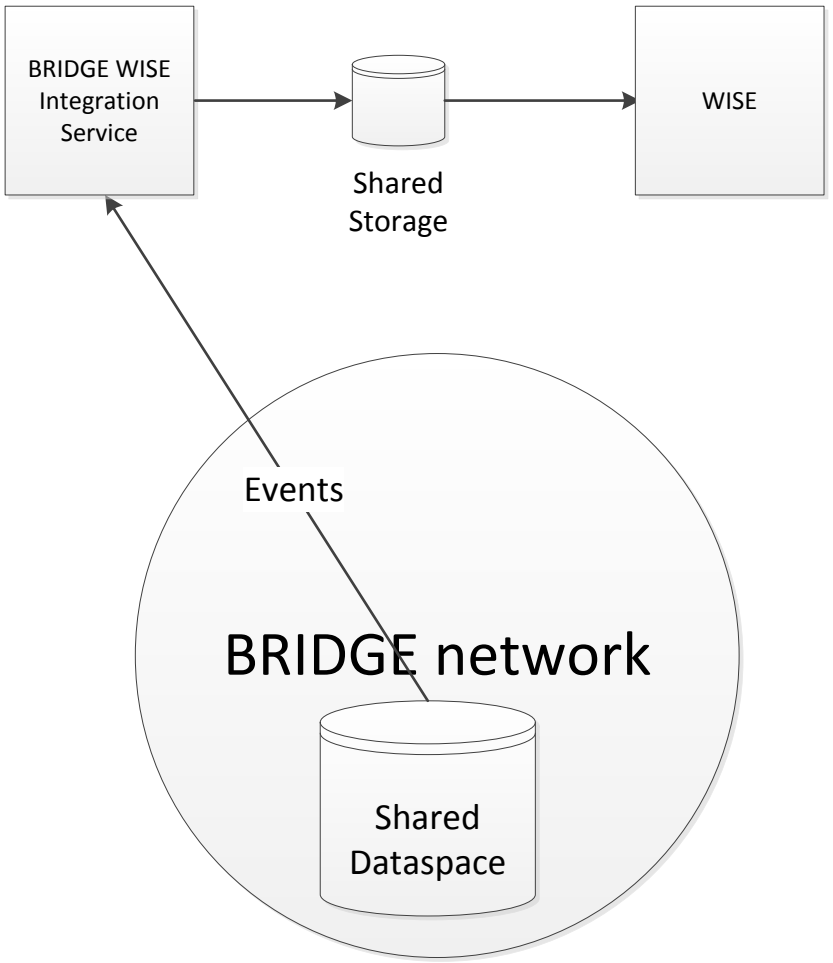

Figure 4: WISE integration architecture

Figure 4 shows the WISE integration architecture with the connection to the WISE system. The method of communicating with WISE is through a shared storage, i.e. the BRIDGE WISE Integration Service writes events to the shared storage and the WISE system reads the events. The WISE system has no need to interact with the BRIDGE network except for receiving events so the integration is unidirectional.

2.2.1 The BRIDGE WISE Integration Service

The BBRIDGE WISE Integration Service acts as the glue for the WISE system to part of the BRIDGE network. The service itself is built out both BRIDGE Middleware components and BRIDGE software classes.

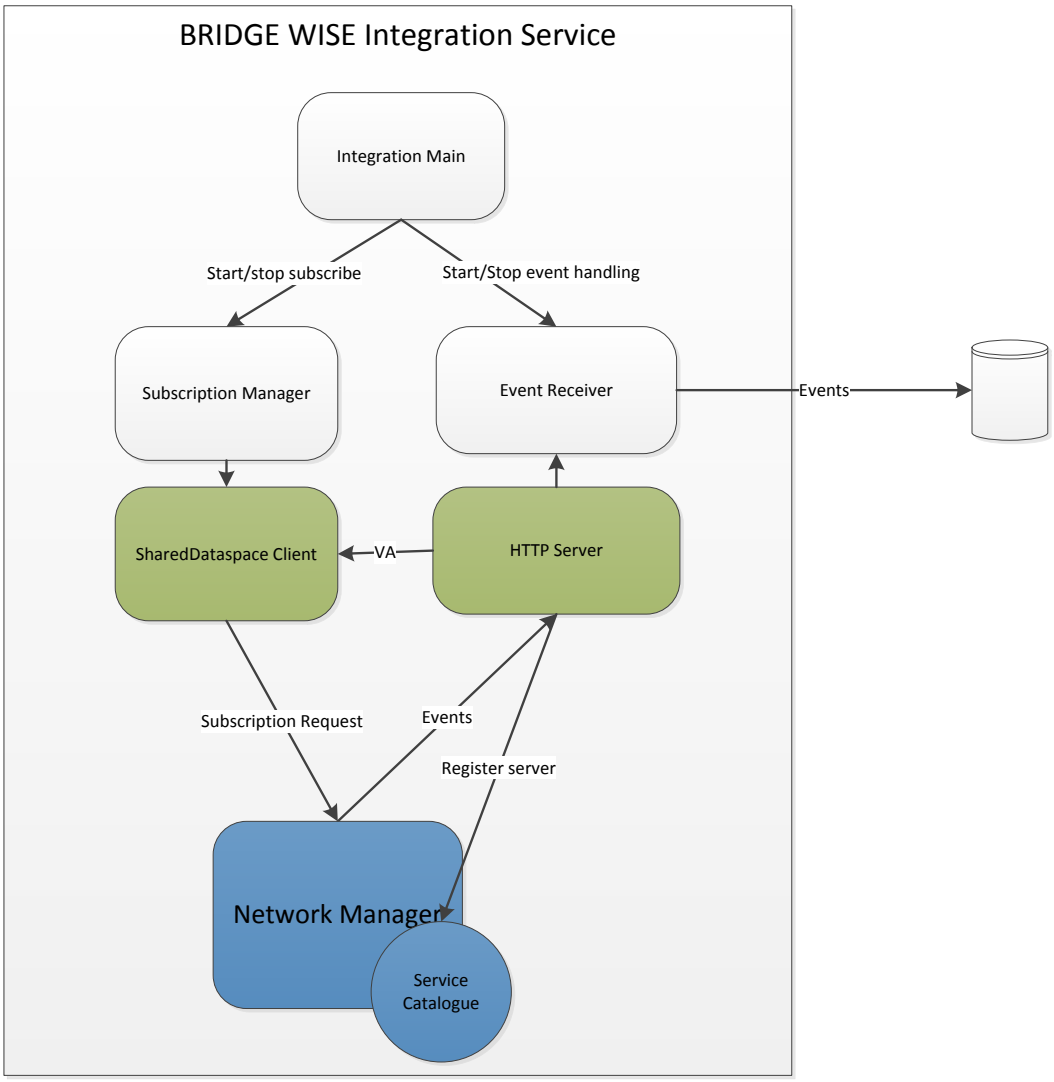

Figure 5: BRIDGE WISE Integration Service architecture

Figure 5 shows the different components of the BRIDGE WISE Integration Services. The blue coloured components are part of the BRIDGE Middleware services, the green coloured components are part of the BRIDGE class library and the white components are bespoke to the integration.

The difference between the BRIDGE Middleware Services and the BRIDGE class library is that the Middleware Services run as separate components whilst the BRIDGE Middleware classes are used as class library, i.e. they become part of the executable. The BRIDGE Middleware classes are reusable components that can be used by any service or component.

The components in Figure 5 are

- Integration Main: Starts and stop the subscriptions and events listening.
- Subscription Manager: Takes care of which subscriptions should be made.

- Event Receiver: Manages the received events and forwards them to shared storage. It also controls the HTTP Server that is the entry point for events.
- Shared Dataspace Client: Exposes the shared dataspace services in the BRIDGE network, hiding the communication complexity when interacting with the shared dataspace.
- HTTP Server: Is a generic class that implements an HTTP server which in this case is used to receive the events from the BRIDGE network.
- Network Manager and Service Catalogue: Enables communication with the BRIDGE network.

In the following subsections we will describe how to create an event listener and to manage subscriptions using the BRIDGE classes using both descriptions and actual code listings.

Setting up an event listener

The HTTP Server class provides the necessary functionality to create and run a BRIDGE network enabled event listener. Some BRIDGE Middleware services use this class in their own implementation, such as the On-Site Storage Service.

The API provided by the HTTP Server class is simple and straightforward with the following public methods and properties:

- **HTTPServer** (string listenerPrefix, bool registerInServiceCatalogue, Parts[] Registration) Initializes a new instance of the **HTTPServer** class.
- event EventHandler< **HttpRequestEventArgs** > **IncomingRequest** = null
  Callback function for incoming HTTP requests.
- virtual void **Start** ():
  Starts this instance.
- virtual void **Stop** ()
  Stops this instance.
- **State RunState** `[get]`
  Gets the state of the HTTP Server.
- **VirtualAddress virtualAddress** `[get]`
  Gets the virtual address.
- virtual Uri **Url** `[get, set]`
  Gets the URL for the HTTP server.

Using the HTTPServer class involves the following steps:

- Define a callback for incoming HTTP requests, this is where the actual processing of the request is done.
- Create the HTTPServer providing which endpoint it will listen to, for instance http://127.0.0.1:88/, if it should be registered in the service catalogue and finally the service catalogue information.

- Start the HTTP server.

The following listing provides an example of the necessary code implementing the described steps.

```
ServiceCatalogue.VirtualAddress _VirtualAddress = null;
HTTPServer listener = null;

private bool SetupEventListenerAndRegisterToBRIDGENetwork()
{
    ServiceCatalogue.Part[] parts = new ServiceCatalogue.Part[2];

    ServiceCatalogue.Part a = new ServiceCatalogue.Part();
    a.key = "DESCRIPTION";
    a.value = "TEST_BRIDGE_listener";
    parts[0] = a;

    a = new NetworkManager.NetworkManager20WebReference.Part();
    a.key = "SID";
    a.value = "urn:BRIDGE:event-listener";
    parts[1] = a;
    //Create the HTTPServer
    listener = new HTTPServer("http://127.0.0.1:88/, true, parts);
    _VirtualAddress = listener.virtualAddress;
    //Set up the Callback for HTTP requests
    listener.IncomingRequest += (WebServer_IncomingRequest);
    listener.Start();
}

/// <summary>
/// Handles the IncomingRequest event of the HTTPServer control.
/// </summary>
/// <param name="sender">The source of the event.</param>
/// <param name="e">The <see cref="HttpRequestEventArgs"/> instance containing
the event data.</param>
protected void WebServer_IncomingRequest(object sender, HttpRequestEventArgs e)
{
    HttpListenerResponse response = e.RequestContext.Response;
    HttpListenerRequest request = e.RequestContext.Request;
    //Create an StreamReader for the Payload
    var streamReader = new StreamReader(request.InputStream);
    try
    {
        if (!streamReader.EndOfStream)
            {
            //Process the request and payload
            }
        //Return OK to the caller
        string responseString = String.Empty;
        byte[] buffer = Encoding.UTF8.GetBytes(responseString);
        response.StatusCode = (int)HttpStatusCode.OK;
        response.StatusDescription = "OK";
        response.ContentLength64 = buffer.Length;
        response.ContentEncoding = Encoding.UTF8;
```

```
            response.OutputStream.Write(buffer, 0, buffer.Length);
            response.OutputStream.Close();
            response.Close();
            return;
        }
        catch (Exception ex)
        {
            System.Console.WriteLine("error:" + ex.Message);
        }
}
```

Listing 7: Setting up an event listener in C# with HTTPServer

In Listing 7 the method SetupEventListenerAndRegisterToBRIDGENetwork creates the HTTPServer instance using the adding the DESCRIPTION=TEST_BRIDGE_listener and SID= urn:BRIDGE:event-listener to the Service Catalogue. The method WebServer_IncomingRequest is the place where HTTP calls will be processed.

Creating a subscription using the SharedDataspace Client class.

The SharedDataspace Client class encapsulates the functionality of the Shared Dataspace Service and extends it to include the functionality of finding the Shared Dataspace Service in the BRIDGE network using the Service Catalogue.

The API for the Shared Dataspace has the following functions:

- **SharedDataspaceClient** (Part[] SharedDataspaceServiceCatalogueQuery)
  *Initializes a new instance of the **SharedDataspaceClient** class and it will connect to the Shared Dataspace Service that will match the **SharedDataspaceServiceCatalogueQuery** in the Service Catalogue.*
- bool **publish** (System.String topic, System.String contentType, System.String metadata, System.String payLoad, string persist, string itemId)
  Publishes the specified payload on the topic.
- string **query** (System.String topic, System.String filter)
  Queries the specified topic.
- void **subscribe** (string topic, string filter, string VirtualAddress)
  Subscribes to the specified topic.
- string **subscribeWithServiceFilter** (string topic, string filter, string VirtualAddress)
  Subscribes to the Topic the with service filter.
- void **Remove** (System.String Topic, System.String ItemId)
  Removes the specified item in the topic.
- void **UnSubscribe** (string subscriptionId)
  Unsubscribe.
- string **ListSubscriptions** ()
  Lists all the subscriptions in the shared data space.

The only real difference in-between the Shared Dataspace Service and the SharedDataspace Client class is the constructor. The constructor takes a Service Catalogue query as a parameter. This query will be used to determine which of the Shared Dataspace Services on the BRIDGE network that will be invoked when using the Shared Dataspace Client.

The following listing will show how the Shared Dataspace Client can be used in order to make a subscription to a specific topics.

```
public void SetUpMySubScription(VirtualAddress VA)
{
    //Create Query
    Part[] parts = new Part[1];
    Part a = new Part();
    a.key = "DESCRIPTION";
    a.value = "S2D2sCNet:StaticWS";
    parts[0] = a;

    SharedDataspaceClient sdc = new SharedDataspaceClient(parts);

    //Subscribe
    sdc.subscribe("App.global.ResourceStatus", "", VA.asString);
}
```

Listing 8: Creating a subscription in C# with SharedDataspace Client

In Listing 8 describes how an subscription is created in the Shared Dataspace Service with the description "S2D2sCNet:StaticWS" for the topic "App.global.ResourceStatus". Note that the Virtual Address of the event listener should be provided as parameter.

2.3 Creating a protocol transformation

In the final version of the deliverable this will show how to extend the BRIDGE platform with a new protocol transformation. The example will show how the platform is extended to support CONTINUA based standards and formats http://www.continuaalliance.org/ enabling different medical devices to be connected to the BRIDGE platform.

2.4 Enabling a service in the BRIDGE platform

There are three main ways of enabling a service in the BRIDGE platform, i.e. making a service available and accessible on the BRIDGE network:

- Creating a proxy that communicates with service using the IoTResource Builder.
- Incorporating the code generated by the IoTResource Builder into the service.
- Enabling the service using the available BRIDGE Middleware services.

The two first options involving the IoTResource Builder are the easiest way to enable the service in the BRIDGE network. The code stubs generated already contain the code for registering the service in the Service Catalogue and it has automatically produced the service description that will be part of the IoTResource Catalogue. The usage of the tools is described in section 3.1.

Therefore this section will deal with what is necessary for the third option where the service is enabled by using the BRIDGE Middleware services. The integration with the BRIDGE system and network can be done at different depths depending on ambition level and need. The simple one where the service is discoverable and can be invoked using the BRIDGE network but the caller must now what the service and its API beforehand. The fully enabled one where the

service provides additional metadata describing the service, its methods etc. that makes it possible to dynamically determine what the service does and its API.

The two following sub section will detail how a service can be BRIDGE network enabled by using the BRIDGE Middleware services. In order to understand the process please look at the deliverable D05.2 that describes the inner working of the BRIDGE network.

2.4.1 Enabling a service on the BRIDGE network

In order to register a service on the BRIDGE network the service endpoints must be known. The endpoint is the URL where the service can be accessed by a client application. The service can have multiple endpoints, for example in order to make it available using different protocols, for instance one for REST and one for Web Service protocols. Each of the services endpoints needs to be registered individually in the Network Manager Service Catalogue

The process is repeated for each endpoint according to these steps:

- Create the meta description for the endpoint. This is a list of key value pairs with properties. Two values are compulsory in all registrations: DESCRIPTION that describes the endpoint, for instance S2D2sCNet:REST; SID The service identity, for instance urn:http:ws:BRIDGE:Middleware:SharedDataSpace:1, this attribute describes which interface/service is implemented.
- Make a registration call to the local BRIDGE Network Manager to register the service endpoint. Note that the supplied endpoint must be accessible for the Network Manager to reach, otherwise the service can never be called.

```
//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 9: Example of registering a service endpoint in C#

Listing 9 shows the steps in code where m_wsendpoint contains the endpoint for the service. The SID (service identity) can be common for many services in the BRIDGE network when they implement the same service interface (API). When this registration is done the service and its endpoint is available to the whole BRIDGE network.

2.4.2 Registering Service Metadata in the IoTResource Catalogue

Using the IoTResource Builder services automatically register to the IoT Resource Catalogue by using the standard UPnP discovery mechanism. This section will describe how to register the Service Metadata using BRIDGE middleware service invocations in code.

The actual call to register the services is simple but some extra functionality must be implemented in order for the service to be properly registered. This requires a small understanding on how the SCPD works in relation with UPnP and how the IoTResource Catalogue deals with service descriptions.

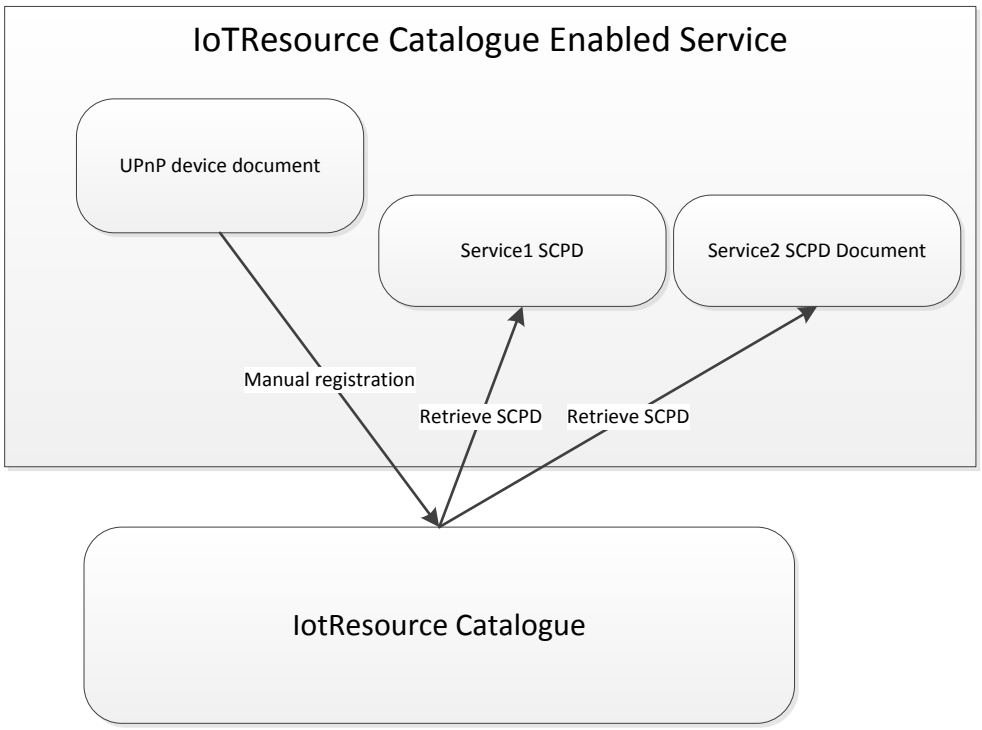

Figure 6: Manual IoTResource Catalogue registration

Figure 6 shows how the IoTResource Catalogue finds services manually, the steps are as follows:

- First the service makes a manual registration of the services it provides, this registration contain links to SCPD files decribing each of the individual services.
- The IoTResource Catalogue retrieves the description of the services and adds them to the catalogue.

This means that the service must be able to publish the SCPD files for HTTP based retrieval. This can be done by using any HTTP-based Web Server that can be accessed from the IoTResource Catalogue.

The UPnP Device document that is provided for the manual registration is a standard UPnP device document.

```
<?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:IoTresource:testservice:1</deviceType>
    <friendlyName>TestService</friendlyName>
    <manufacturer>BRIDGE</manufacturer>
    <modelName>Test</modelName>
    <modelNumber>1</modelNumber>
    <UDN>uuid:15696574-1a4d-42e0-8907-bee3e110e2f1</UDN>
    <serviceList>
      <service>
        <serviceType>urn:schemas-upnp-org:service:testservice:1</serviceType>
        <serviceId>urn:serviceId:testservice:1</serviceId>
        <SCPDURL>http://127:0.0.1:7237/serviceId-testservice-1_scpd.xml</SCPDURL>
      </service>
    </serviceList>
  </device>
</root>
```

Listing 10: Example of XML document used to register service

Listing 10 shows a simple example device document that is used to register a service. In this example only one service is provided “urn:serviceId:testservice:1”, but mor eservices can be added to the serviceList. The most important parts that need to be eneterd correctly in the document are:

- friendlyName: This is the name that service will have in different service browser
- serviceId: Should have the same content as the SID used to register in the Network Manager Service Catalogue
- SCPDURL: This is the link to the SCPD document describing the service, this URL must be accesible for the IoTResource Catalogue.

The SCPD document for the service needs to be manually created using an XML editor, the actual syntax and also links to some tools can be found at the UPnP forum web site http://www.upnp.org/. A very simple example of an SCPD document is shown in Listing 11.

```
<?xml version="1.0" encoding="utf-8"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <actionList>
    <action>
      <name>TestMethod</name>
      <argumentList>
        <argument>
          <name>testInput</name>
          <direction>in</direction>
```

```
          <relatedStateVariable>Test</relatedStateVariable>
        </argument>
        <argument>
          <name>testResponse</name>
          <direction>out</direction>
          <retval />
          <relatedStateVariable>Result</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
  </actionList>
  <serviceStateTable>
    <stateVariable sendEvents="no">
      <name>Test</name>
      <dataType>string</dataType>
    </stateVariable>
    <stateVariable sendEvents="no">
      <name>Result</name>
      <dataType>boolean</dataType>
    </stateVariable>
  </serviceStateTable>
</scpd>
```

Listing 11: Example SCPD

The example SCPD can be extended with the service annotations described in section 3.1.1 to add further meta data to the service.

Finally we show the code necessary to register the service in the IoTResource Catalogue using the BRIDGE Middleware API.

```
public void RegisterService()
{
    XmlDocument xDeviceDocument = new XmlDocument();
    xDeviceDocument.Load("mydeviceDocument.xml");

    IoTResourceCatalogue.ApplicationDeviceManager
              IoTResCat = new IoTResourceCatalogue.ApplicationDeviceManager();

    IoTResCat.AddDevice(xDeviceDocument.InnerXml /*xml as string*/);

}
```

Listing 12: Manual Service Registration in the IoTResource Catalogue

As shown in Listing 12 the actual call to make the registration is simple but the complexity lies in the creation of the SCPD files and to make sure that the description matches the actual implementation. Therefore we recommend the usage of the IoTResource Builder tool when creating or enabling services on the BRIDGE network because the SCPD and registration information will be created by the tool.

3 Development Tools

3.1 IoTResource Builder and service annotations

The BRIDGE middleware provides access to the set of ICT resources in an emergency system context, such as different sensing devices, data repositories, social media streams, UAVs etc. The middleware service layer provides client applications uniform access to all such resources (below referred to as IoTResources, which is the LinkSmart resource concept).

The IoTResource Builder allows a developer to define IoTResources and automatically generate the necessary IoTResource code stubs. Services can then be built using these IoTResources. The IoTResources will also automatically register themselves in both the Network Manager Service Catalogue as well as the IoTResource Catalogue.

Complete description, tutorials and download of the IoTResource Builder are available at: http://www.iotworldservices.com/wiki/iotworldserviceswiki/iot-resource-builder/iotresource-builder/

3.1.1 Service annotations

In order to facilitate the use of BRIDGE services both in run-time and in design time, the platform supports the annotation of both services and resources. Annotations in this context means the possibility to associate various semantic descriptions to IoTResources via their service access points.

The annotations can be made searchable for developers as an aid in service development. They can also be used to facilitate the resource discovery processes, and service matching for potential application clients. The annotations are included in the service definitions, which are used as input to the code generation process, which creates program stubs for IoTResources

There are different levels of service annotations.

- Service Summary, a description of the overall function of a service, including references to standards or other external sources.
- Service Actions. Each service has one or more actions (operation /methods). Each action implements some sensing or actuation function. Annotations include action purpose, and arguments and results.
- Property Level (state variables). The arguments and results, the state variables, can also be described in more detail, including their value sets and references to standards.
- Effect annotations. Actions can also be annotated with a list of possible effects they might have in the applications context, or more specifically on other state variables. As an example, turning off a fan might cause a temperature raise, and perhaps also a decrease in energy consumption.

There are numerous approaches to service description frameworks. The service description tool does not impose the use of any specific service annotation standard, but rather encourages the referencing to domain specific standards, controlled vocabularies (or ontologies), in the annotations of the service semantics, e.g., emergency messaging data set standards like EDXL.

From a structural and syntactical view, the service description is based on the UPnP[2] device descriptions and the SCPD format, and USDL[3].

3.1.2 Use of The resource builder

The IoTResource Builder allows you to define your IoTResources and automatically generate the necessary IoTResource code stubs. Services can then be built using these IoTResources. The following sections give examples of how a service can be described using the Resource Builder tool.

Service Summary

This service will mapped to an IoTResource which monitors in-door air quality, using a CO_2 Sensing device in conference room. We start by providing the overall description, the Service Summary.

[2] http://www.upnp.org/

[3] Unified Service Description Language, http://linked-usdl.org/

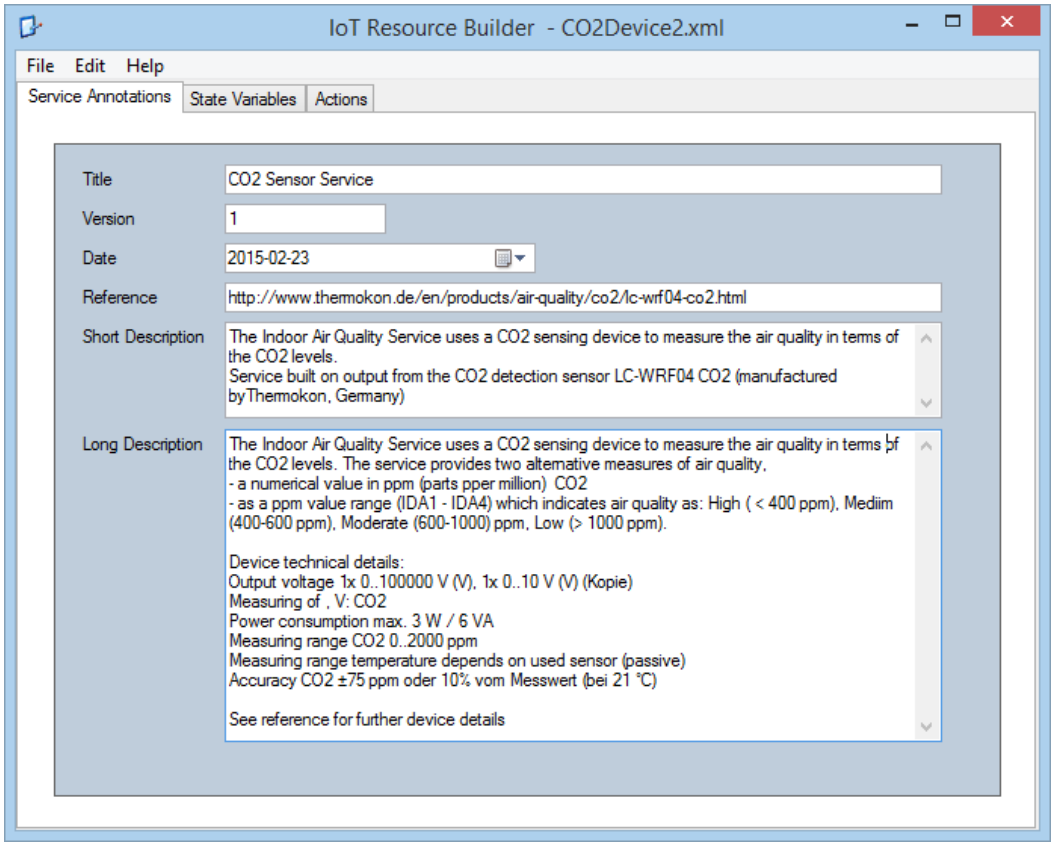

Figure 7: Service Summary description using the IoTResource Builder

The annotations are encoded in an XML vocabulary which will be associated to the IoTResource in the code generation process (see below).

The Service Summary description is shown in its corresponding XML encoding below.

```
<serviceAnnotations xmlns="IoT">
      <name>Indoor Air Quality Service</name>
      <version>1</version>
      <date>2015-02-23</date>
      <shortDescription>

          The Indoor Air Quality Service uses a CO2 sensing device to measure the
          air quality in terms of the CO2 levels. Service built on output from the
          CO2 detection sensor LC-WRF04 CO2 (manufactured byThermokon, Germany)
      </shortDescription>
      <longDescription>

          The Indoor Air Quality Service uses a CO2 sensing device to measure the air
          quality in temrs of the CO2 levels. The service provides two alternative
          measures of air quality,
          - a numerical value in ppm (parts pper million)  CO2
```

```
            - as a ppm value range (IDA1 - IDA4) which indicates air quality as: High (
            &amp;lt; 400 ppm), Mediim (400-600 ppm), Moderate (600-1000) ppm, Low
            (&amp;gt; 1000 ppm).

            Device technical details:
            Output voltage 1x 0..100000 V (V), 1x 0..10 V (V) (Kopie)
            Measuring of , V: CO2
            Power consumption max. 3 W / 6 VA
            Measuring range CO2 0..2000 ppm
            Measuring range temperature depends on used sensor (passive)
            Accuracy CO2 ±75 ppm oder 10% vom Messwert (bei 21 °C)
            See reference for further device details
        </longDescription>
          <referenceUrl>
            http://www.thermokon.de/en/products/air-quality/co2/lc-wrf04-co2.html
        </referenceUrl>
    </serviceAnnotations>
```

Figure 8: Service Summary XML

Actions

Each action (similar to operations/methods) of a service may also have their own annotations specified. This example shows an action for reporting the air quality in ppm (parts per million) CO_2 based on a standard for indoor air quality (IDA[4]).

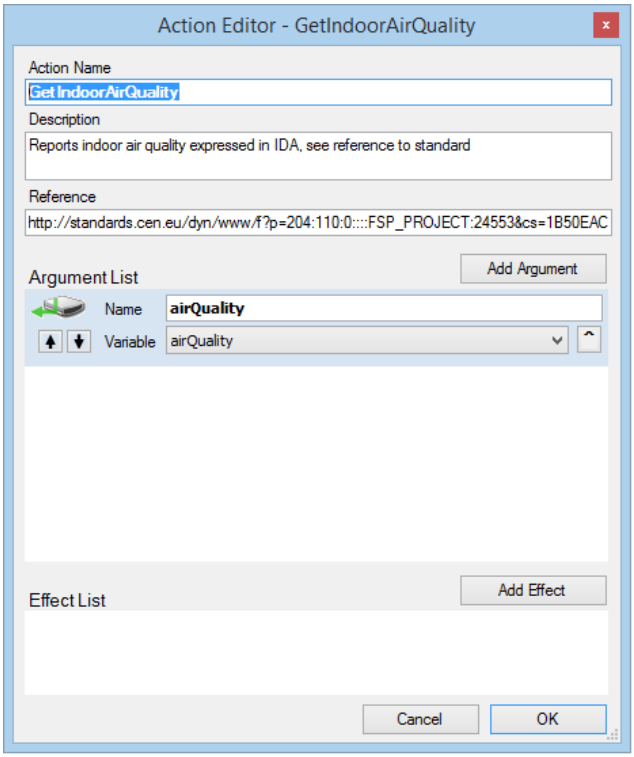

Figure 9: Action annotation

[4] http://www.aafeurope.com/en/155/en13779-standard

The corresponding XML follows. It also contains two additional actions for the service.

```
<actionmetadata xmlns="IoT">
     <actionList>
          <action name="GetIndoorAirQuality">
             <description>
                  Reports indoor air quality expressed in IDA, see reference to
                  standard.
             </description>
             <referenceUrl>

                  http://standards.cen.eu/dyn/www/f?p=204:110:0:::FSP_PROJECT:24553&amp
                  ;cs=1B50EAC84642115F35A7D9F005762E46B
             </referenceUrl>
                <effects>

                   No effects reported
                </effects>
          </action>
          <action name="GetCO2Level">
             <description>
                  Reports indoor air quality expressed as level of CO2 in ppm
             </description>
              <referenceUrl></referenceUrl>
             <effects>
                <effect>
                   <stateVariable></stateVariable>
                   <description>text explaining possible effect</description>
                   <referenceUrl></referenceUrl>
                </effect>
             </effects>
          </action>
          <action name="TurnOffCO2Sensor">
             <description>
                  Turns off the CO2 Sensor Device and returns the current CO2Level
             </description>
             <referenceUrl></referenceUrl>
             <effects>
                <effect>
                   <stateVariable></stateVariable>
                   <description>
                      Device is turned off. Last measurement accessible in log.
                   </description>
                   <referenceUrl></referenceUrl>
                </effect>
             </effects>
          </action>
     </actionList>

</actionmetadata>
```

Figure 10: Action annotation XML

As mentioned above, it also possible to describe any additional effects an action might have. Note that these “effects” are not to be seen as hard dependencies between actions/ state variables maintained by the service run-time, but rather as a way to document possible effects in the application context.

In the example above (TurnOffCO2Sensor), the “effect” simply states that at (power) turn off, the last measured value is available as an IoTObservation from logged data.

Properties (State Variables)

The inputs/outputs of a service are represented by state variables associated with each of the actions. The Air quality action above reports measurements to be interpreted according to a standard for in-door air quality using intervals of ppm ranges.

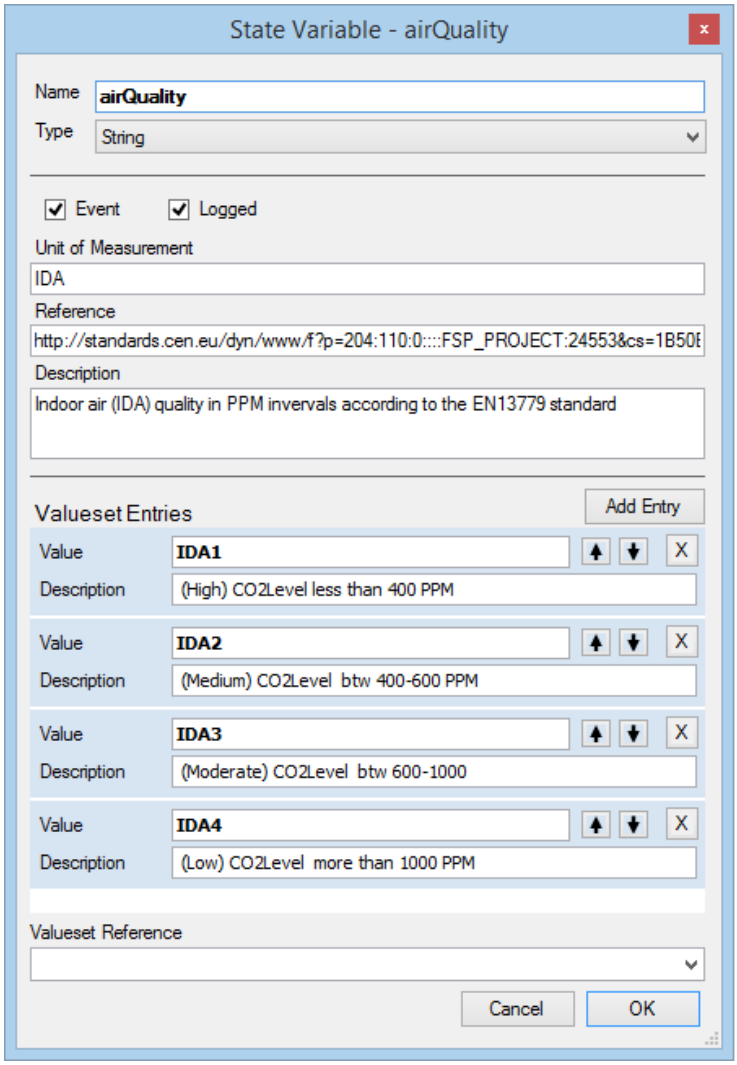

Figure 11: State variable annotation

The corresponding XML encoding for this State variable annotation is shown below.

```
<statevariablemetadata xmlns="IoT">
      <statevariableList>
         <statevariable name="airQuality">
              <IoTEvent>true</IoTEvent>
               <!–checkbox Event: Generates IoT event when state changes -->
              <IoTStored>true</IoTStored>
              <!-- checkbox Logged:Store state changes automatically using storage
              manager -->
              <IoTUoM>IDA</IoTUoM>
```

```
            <!-- Valueset? - Unit of Measurement C, cm, kg....etc -->
            <description>

            Indoor air (IDA) quality in PPM invervals according to the EN13779
            standard
            </description>
            <referenceUrl>

            http://standards.cen.eu/dyn/www/f?p=204:110:0::::FSP_PROJECT:24553&amp;cs=
            1B50EAC84642115F35A7D9F005762E46B
            </referenceUrl>
            <valueset>
               <entry>
                  <value>IDA4</value>
                  <description>(Low) CO2Level  more than 1000 PPM</description>
               </entry>
               <entry>
                  <value>IDA3</value>
                  <description>(Moderate) CO2Level  btw 600-1000</description>
               </entry>
               <entry>
                  <value>IDA2</value>
                  <description>(Medium) CO2Level  btw 400-600 PPM</description>
               </entry>
               <entry>
                  <value>IDA1</value>
                  <description>(High) CO2Level less than 400 PPM</description>
               </entry>
            </valueset>
            <vsReferenceUrl></vsReferenceUrl>
         </statevariable>

         Additional variables here….
      </statevariableList>
   </statevariablemetadata>
```

Figure 12: Annotations for State Variables

3.1.3 Components in the IoTResource Builder

The IoTResource Builder is built on two separate components, see Figure 13:

- The IoTResource Builder GUI
- The IoT Code Generator

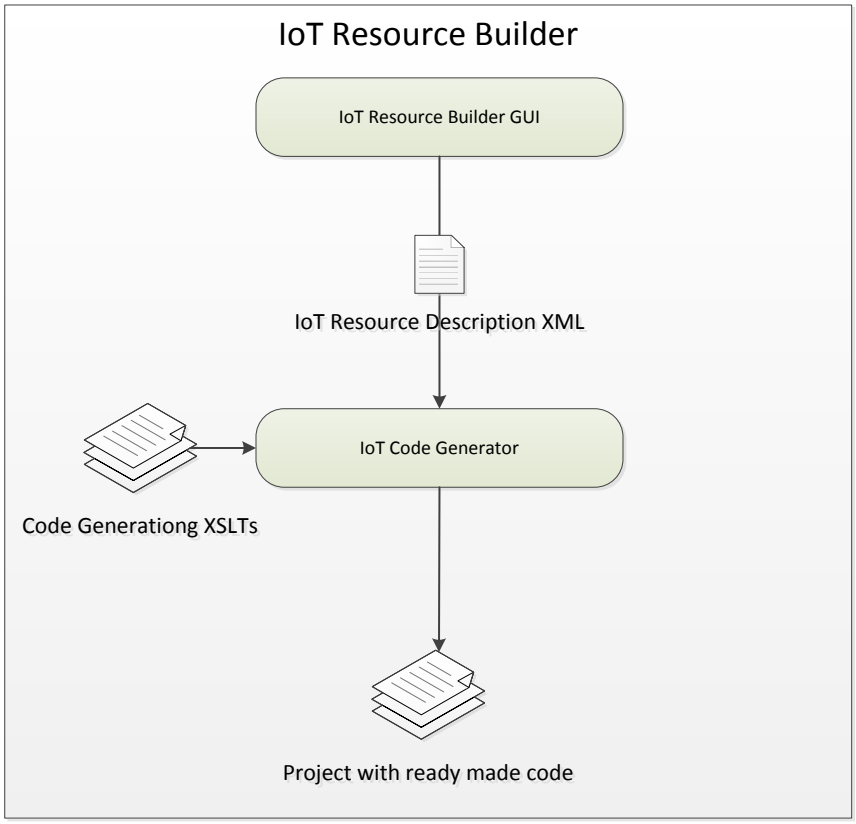

Figure 13: IoTResource Builder Components

The IoTResource Builder GUI creates an IoTResource Description XML which is then sent to the IoT Code Generator to create the code. The reason for this division is to make the IoT Code Generator reusable for other tools, for instance the GUI could be replaced by a completely web based interface but still using the same code generation.

The IoTResource Description XML is based on the UPnP device XML but with some small differences. Firstly the IoTResources service description (SCPD) is in lined in the Device XML. Secondly there is an envelope which carries some code generation meta data, see Figure 14.

```
<device>
  <deviceType>urn:schemas-upnp-org:IoTresource:CO2Sensor:1</deviceType>
  <friendlyName>CO2Sensor</friendlyName>
  <manufacturer>LinkSmart Open Source</manufacturer>
   <manufacturerURL>

       http://www.iotworldservices.com/wiki/iotworldserviceswiki/
   </manufacturerURL>
  <modelDescription>
       CO2Sensor UPnP Device Using Auto-Generated UPnP Stack
   </modelDescription>
  <modelName>CO2Sensor Device</modelName>
  <modelNumber>X1</modelNumber>
```

```
<productCode>CO2Sensor-X1</productCode>
<serviceList>
  <service>
    <serviceName>CO2SensorProject</serviceName>
    <serviceType>urn:schemas-upnp-org:service:CO2Sensor::1</serviceType>
    <serviceId>urn:upnp-org:serviceId:CO2Sensor</serviceId>
    <SCPD>
      <specVersion xmlns="urn:schemas-upnp-org:service-1-0">
        <major>1</major>
        <minor>0</minor>
      </specVersion>
      <actionList xmlns="urn:schemas-upnp-org:service-1-0">
        <action>
          <name>GetCO2Level</name>
          <argumentList>
            <argument>
              <name>CO2Level</name>
              <direction>out</direction>
              <retval />
              <relatedStateVariable>CO2Level</relatedStateVariable>
            </argument>
          </argumentList>
        </action>
        <action>
          <name>GetIndoorAirQuality</name>
          <argumentList>
            <argument>
              <name>airQuality</name>
              <direction>out</direction>
              <retval />
              <relatedStateVariable>airQuality</relatedStateVariable>
            </argument>
          </argumentList>
        </action>
        <action>
          <name>TurnOffCO2Sensor</name>
          <argumentList>
            <argument>
              <name>CO2Level</name>
              <direction>in</direction>
              <relatedStateVariable>CO2Level</relatedStateVariable>
            </argument>
          </argumentList>
        </action>
      </actionList>
      <serviceStateTable xmlns="urn:schemas-upnp-org:service-1-0">
        <stateVariable sendEvents="no">
          <name>_IoTActionMetaData_</name>
          <dataType>string</dataType>
          <defaultValue>
            <?xml version="1.0" encoding="utf-8"?>
                  <actionmetadata xmlns="IoT">
                      <actionList>
```

Actions Annotations

```
                      </actionList>
```

```
                </actionmetadata>
            </defaultValue>
            </stateVariable>
            <stateVariable sendEvents="no">
              <name>CO2Level</name>
              <dataType>i2</dataType>
            </stateVariable>
            <stateVariable sendEvents="no">
              <name>_IoTResourceMetaData_</name>
              <dataType>string</dataType>
              <defaultValue>
                <serviceAnnotations>
                  Service Summary
                    annotation
                </serviceAnnotations>
              </defaultValue>
            </stateVariable>
            <stateVariable sendEvents="no">
              <name>_IoTStateVariableMetaData_</name>
              <dataType>string</dataType>
              <defaultValue>
                <?xml version="1.0" encoding="utf-8"?>
                <statevariablemetadata xmlns="IoT">
                  <statevariableList>
                    State Variables
                      annotations
                  </statevariable>
                </statevariableList>
              </statevariablemetadata>
            </defaultValue>
            </stateVariable>
            <stateVariable sendEvents="no">
              <name>airQuality</name>
              <dataType>string</dataType>
            </stateVariable>
          </serviceStateTable>
        </SCPD>
      </service>
    </serviceList>
  </device>
 </root>
</upnp>
</DeviceInfo>
```

Figure 14: Example of an IoTResource Description XML. Coloured rectangles represent the annotation sections.

There are four specific tags in the Environment section that controls the code generation:

- *CodeNameSpace*: The namespace used for the code generated, usage depends on target language.
- *ProjectName*: The name used for the resulting code project.
- *ClassName*: Class name stem used for the generated classes for the IoTResource.
- *IoTResourceType*: Decides which type IoTResource code is generated, current possible values are IoTDevice, IoTService and IoTThing.

The actual code generation is performed by using XSLT transformations using the IoTResource Description XML as input. The set of XSLT transformation[5] create the output files that are part of the resulting development project solution.

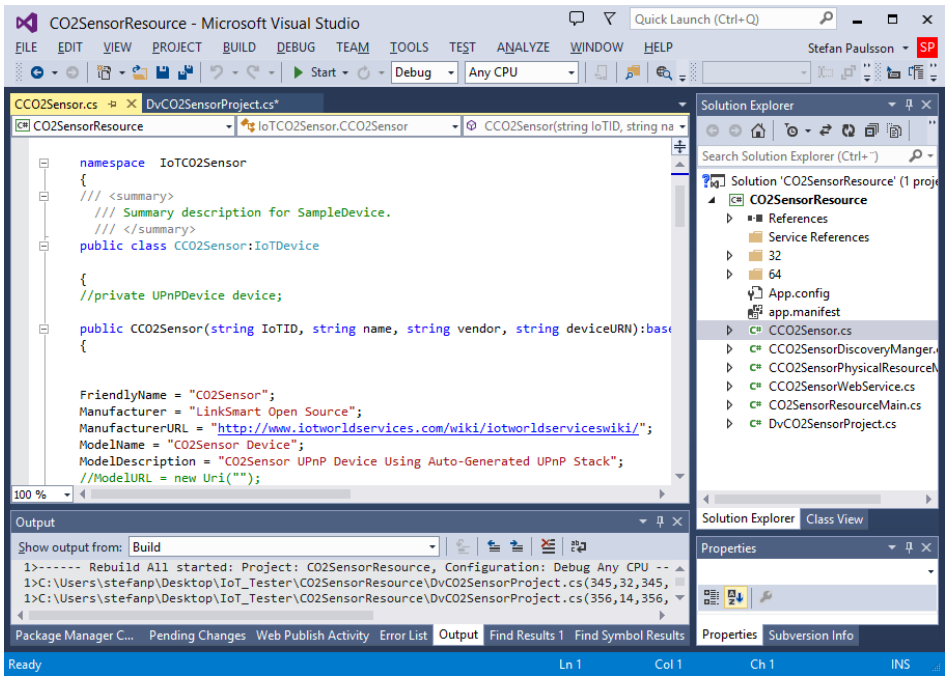

Figure 15: Development environment generated

Initially C# Visual Studio projects are supported as target environment. However, with the use of XSLT this can be easily extended to support other languages such as java, swagger.

[5] See LinkSmart code repository https://linksmart.eu/redmine/

3.2 IoTResource Catalogue

The IoTResource Catalogue discovers and keeps track of available IoTResources in the network and their service descriptions. It provides a REST and Web Service based interface to select and retrieve data about the IoTResources and their services.

As an example see Figure 16 below that shows which IoTResources have been discovered on the gateway ”KURSAAL”, which handles several physical gateways (KURSAAL, ELO2,CLEMONS) and which IoT Services they offer. IoTResources are discovered and managed by the IoT Resource Catalogue.

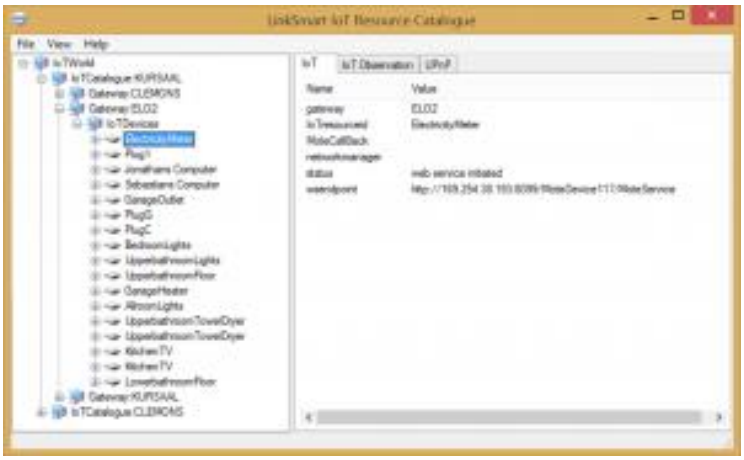

Figure 16: IoT Resource Catalogue

3.2.1 Catalogue Services and Actions

The IoTResource Catalogue offers a number of services which can be listed using the following REST-expression:

http://<catalogueendpoint>/services.

If you type this into a browser the result will be:

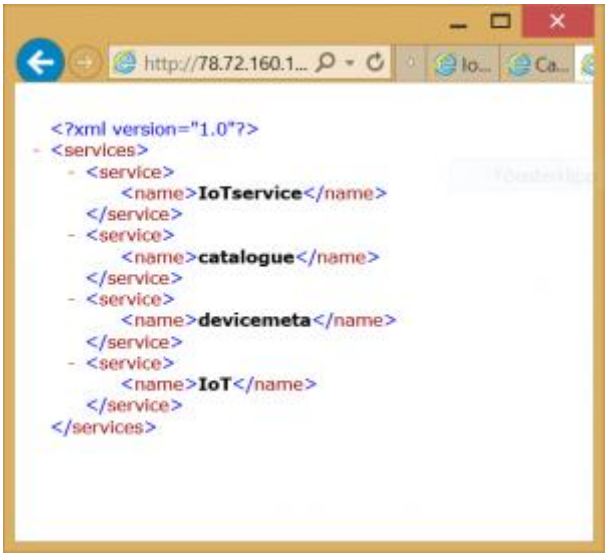

Figure 17: Catalogue Services

Each service provides a number of actions that can be performed on the IoTResource. The *catalogue* service provides the main functionality of the IoTResource Catalogue. You can list all actions provided by a service with the following REST-expression:

http://<catalogueendpoint>/services/actions

The returned XML specifies the action and the arguments needed to call it:

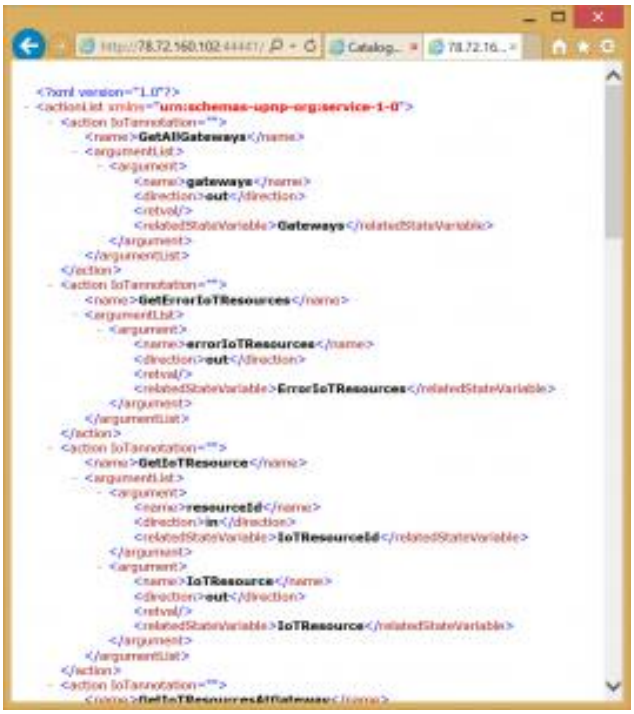

Figure 18: Result of Resource Catalogue Query

Below is a short explanation of all available actions:

GetAllGateways
Returns all gateways known by the catalogue

GetErrorIoTResources
Returns all IoTResources that are in an error state, for instance that have disappeared from the network without telling about it

GetIoTResource
Argument: resourceId
Returns the SCPD for a specified IoT Resource

GetIoTResourcesAtGateway
Argument: gateway ID
Returns the SCPD file for all IoT Resources at a specified gateway

GetIoTResourcesEndpoints
Returns the IotResourceId, FriendlyName and the localendpoint for all IoTResources known by the catalogue

GetIoTResourcesEndpointsFromXpath
Argument: Xpath expression
Returns the IotResourceId, FriendlyName and the localendpoint for all IoTResources known by the catalogue that matches the xpath description

GetIoTResourcesFromXpath
Argument: Xpath expression
Returns the SCPD file for all IoTResources known by the catalogue that matches the xpath description

RegisterResource
Register an IoTResource directly not using UPnPDiscovery.

GetManualIoTResources
Returns all IoTResources that has registered themselves and not through UPnP

GetNumberOfIoTResources
Returns the number of IoTResources, UPnPDevices, ErrorResources

RemoveErrorIoTResources
Instructs the catalogue to release and forget about the IoTResources that are currently in the error list

ReScan
Instructs the catalogue to issue a new M-SEARCH command to find new IoTResources in the network

ReStartCatalogue
Instructs the catalogue to forget about all IoTResources and ErrorResources and issue a ReScan command

3.3 IoTResource Catalogue Browser.

IoT Resource Catalogue Browser provides a user interface to look and interact with IoT resources in the network node. Basically it provides a user interface to the IoT Resource Catalogue. The IoT Resource Catalogue Browser can be used for looking at the service descriptions and also to invoke actions in the service (If the service supports this)

Complete description examples and downloads of the IoT Resource Catalogue Browsers are available at: http://www.iotworldservices.com/wiki/iotworldserviceswiki/iotresource-catalogue-browsers/.

When you double click on the executable it browser will first discover the IoTResource Catalogue in your local network. If you click on the catalogue name in the tree, you will see three tabs to the right. The first tab shows you the number of IoTResources this catalogue has discovered.

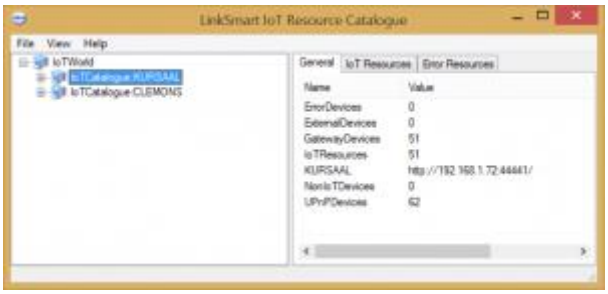

Figure 19: Initial Windows IoT Resource browser window

The second tab shows the IoTResourceIds and the endpoints to the different IoTResources. In case there are IoTResources which are in some error state and therefore cannot be accessed, they will be listed in the third tab.

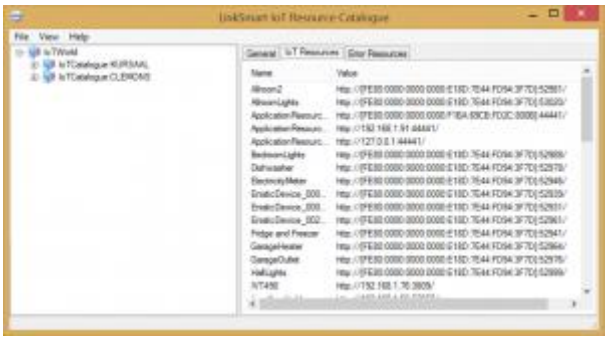

Figure 20: Second Tab with IoT Resources end points

You can now expand the tree on the left. The gateway nodes correspond to different hardware gateways (normally computers) in your network which hosts the IoTResources. If you click on one IoTResource, you will see three tabs to the right. The first tab (IoT) lists the state variables/properties that are specific for LinkSmart.

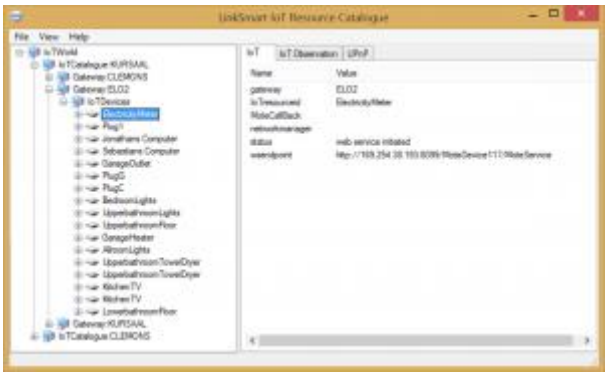

Figure 21: Expanded view

4 Components in Open Source Public Repositories

This section will be extended in the final version since many components are still in the process of being made available.

4.1 LinkSmart

Many of the LinkSmart extensions made in BRIDGE are already incorporated in the Open Source release available at https://linksmart.eu/redmine/projects/linksmart-opensource. Amongst the BRIDGE Developed components these are the most important ones made available:

- 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.
- IoTResource Tools

The list will be extended with the components that are still either under consideration to be included or already in the process of being prepared for the Open Source release.

4.2 EVE

For the agent based parts of the BRIDGE platform many developed components have been published in the EVE open source project: http://eve.almende.com/.
The Major EVE parts that have been created or extended within the BRIDGE project include:

- The XMPP transport layer
- The capability to run Eve agents effectively on Android mobile devices
- A push/pull combining "monitor" design pattern for robust communication
- A factor 100 latency decrease on generic Eve calls, part of a scalability effort for large scale resource simulation
- An innovative gossip based agent event publication design
- A further development of the CAPE personal agent model