BEMO-COFRA

Brazil-Europe Monitoring and Control Framework

(Project No. 288133)

D5.3.1 Initial LinkSmart-enabled environment

Published by the BEMO-COFRA Consortium

Dissemination Level: Public

European Commission
Information Society and Media

Project co-funded by the European Commission within the 7th Framework Programme
and
Conselho Nacional de Desenvolvimento Científico e Tecnológico
Objective ICT-2011-EU-Brazil

Document control page

Document file: D5.3.1_Initial_LinkSmart-enabled_environment_Final.docx
Document version: 1.0
Document owner: Peeter Kool (CNet)

Work package: WP5 – Distributed Control Logic and Enabling Features
Task: Task 5.3 LinkSmart-enabled monitoring and control infrastructure
Deliverable type: P

Document status:
☒ approved by the document owner for internal review
☐ approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of Changes made
0.1	Peeter Kool	2012-11-01	Initial ToC
0.5	Peeter Kool	2012-11-15	Initial content
0.7	Peeter Kool, Ardi Tjandra	2012-11-20	Added content
1.0	Peeter Kool	2012-12-12	Final version submitted to the European Commission

Internal review history:

Reviewed by	Date	Summary of comments
Matts Ahlsén	2012-12-12	Approved

Legal Notice

The information in this document is subject to change without notice.

The Members of the BEMO-COFRA Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the BEMO-COFRA Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Index:

1. Executive summary 4
2. Introduction 5
3. Architecture of the initial LinkSmart-enabled environment 6
3.1 LinkSmart Network Manager and addressing scheme 8
3.2 LinkSmart Event Manager 11
4. Examples of LinkSmart integration 16
4.1 Architecture of the PLC Proxy 16
4.2 Architecture of the Unity integration (Monitoring Application) 17
References 21

1. Executive summary

This document is delivered with the software deliverable D5.3.1 Initial LinkSmart-enabled environment. This deliverable documents the prototype from a LinkSmart environment perspective and includes descriptions of the typical integrations done, one based on proxies and another one where the integration is done in the device. The document includes also an overview of what a LinkSmart environment is.

2. Introduction

This document is delivered with the software *D5.3.1 Initial LinkSmart-enabled environment*.

The initial LinkSmart-enabled environment is described at an architecture level in chapter 3. This chapter also includes a small overview of the LinkSmart middleware. The following chapter 4 contains description of some of the most important integrations in to the initial LinkSmart environment, i.e. LinkSmart proxies in the prototype deliverable. The actual code documentation for these proxies can be found in deliverable *D6.1 IoT-enabled legacy devices for production monitoring*.

The work has primarily been done in Task 5.3 LinkSmart-enabled monitoring and control infrastructure.

3. Architecture of the initial LinkSmart-enabled environment

The initial LinkSmart enabled environment consist basically of the LinkSmart network, see Figure 1 below, and the different devices and their proxies.

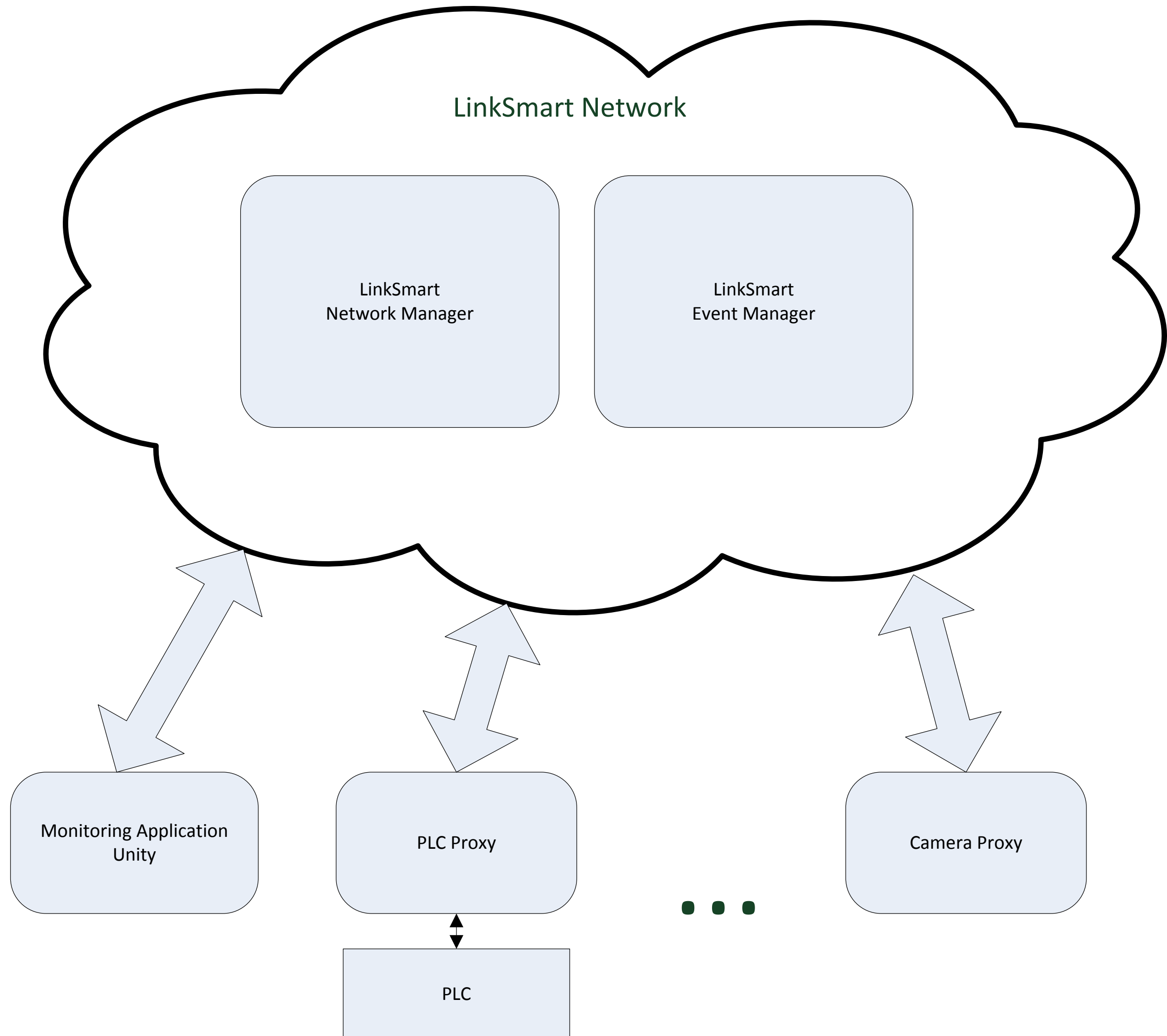

Figure 1: Architecture of the initial LinkSmart enabled environment.

The LinkSmart network is a private P2P network that provides communication services and addressing through SOAP-tunnelling (see 3.1) as well as an event manager that can be used for publishing of and subscribing to events (See section 3.1 and section 3.2 for an introduction to LinkSmart). In the first demonstrator setup we ran only a small LinkSmart network containing only one LinkSmart server node (see deployment view Figure 2). From the developers point of view it will be completely transparent if the network is extended over several nodes in different network locations in a real deployment, i.e. the proxies and applications will not change.

The initial LinkSmart environment was primarily implemented using the LinkSmart event infrastructure as the carrier of messages in-between different components. All components that are integrated register

their event service in the network manager in order to get a LinkSmart address (HID) that is then used for making subscriptions in the LinkSmart Event Manager.

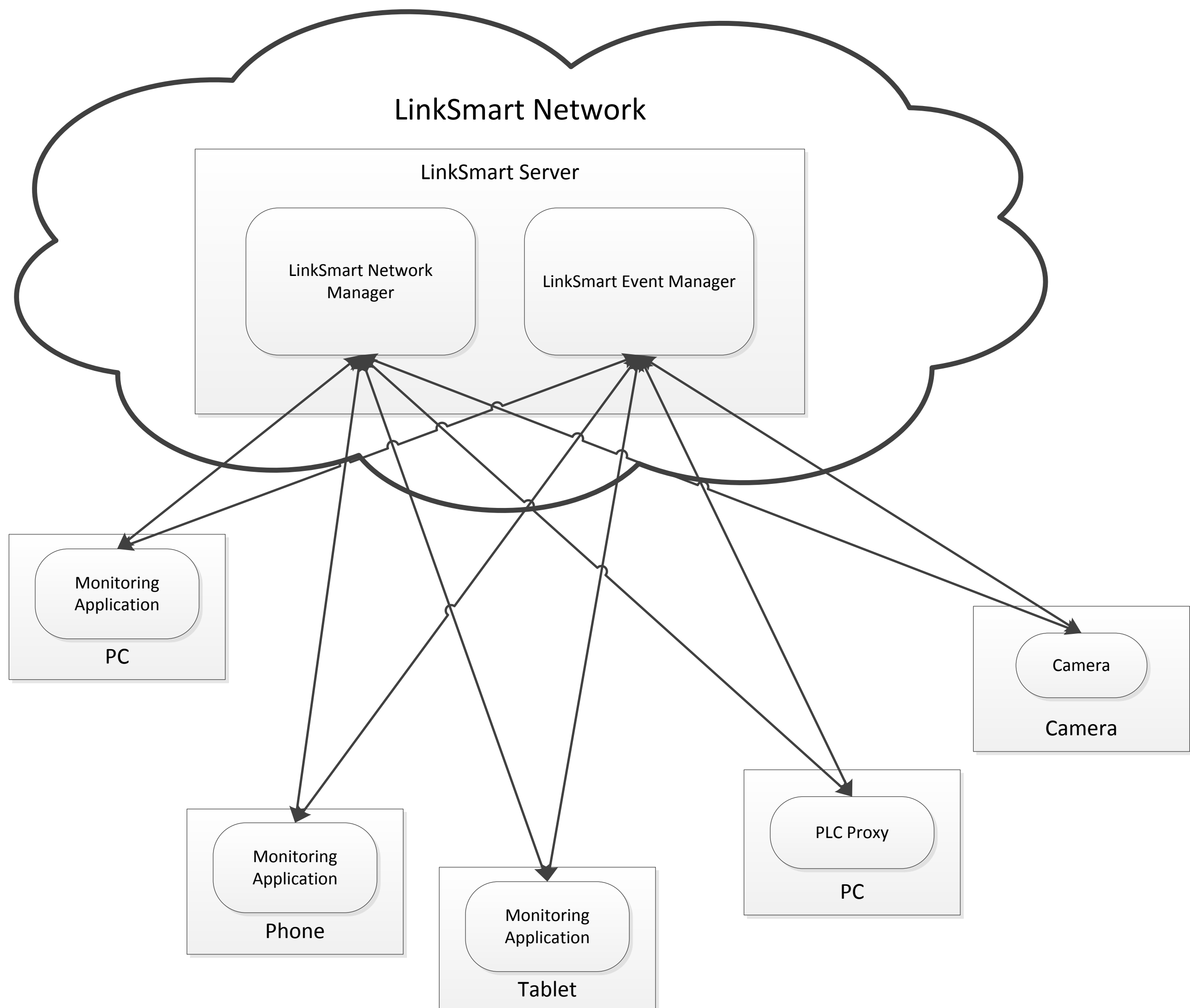

Figure 2: Deployment overview of demonstrator

In the first demonstrator there were two main strategies for adding devices/services to the LinkSmart infrastructure:

1. Creating a proxy: This was done for the devices/services that cannot be extended with LinkSmart functionality, either because of processing power or because they are closed systems. These include the PLC and Arduinos. For these devices a LinkSmart proxy is created that runs on a gateway and implements the LinkSmart services and communicates with the device, i.e., acting as the proxy for the device in the LinkSmart network.
2. Direct integration: Some of the devices, such as the camera and the Unity based monitoring application, are powerful enough to run the LinkSmart services directly on the device. This means that the devices themselves can register their services on the LinkSmart network and host the services on the Device.

Section 4.1 provides an example for a proxy based integration for the PLC proxy integration. Section 4.2 provides an example of direct integration.

The following two sections will give a short overview of LinkSmart, further reading regarding LinkSmart can be found in (LINKSMART, 2012), (LINKSMART2,2012) and (LINKSMART3,2012)

3.1 LinkSmart Network Manager and addressing scheme

The LinkSmart network manager consists of three main functions:

- P2P overlay network
- HID addressing scheme
- SOAP Tunneling

The following sections will give a short introduction to these functions

3.1.1 Building a P2P overlay network

The main objective of the Network Manager is to interconnect different LinkSmart Enabled Devices and services through the network. The main problem of this task is that most of the networks may be hidden in Local Area Networks, behind firewalls, routers and Network Addressing Translators (NATs), so it would be difficult to interconnect the different nodes.

However, the Network Manager solves this problem by building an overlay network, independently of the network addressing and protocols.

The Network Manager, relies on JXTA P2P[1] platform in order to build the overlay network. JXTA is a set of open, generalised P2P protocols enabling any connected device on the network to communicate and collaborate. Using the JXTA protocols, LinkSmart devices and services are directly connected even if they are connected in different networks separated by firewalls or NATs.

[1] http://en.wikipedia.org/wiki/JXTA

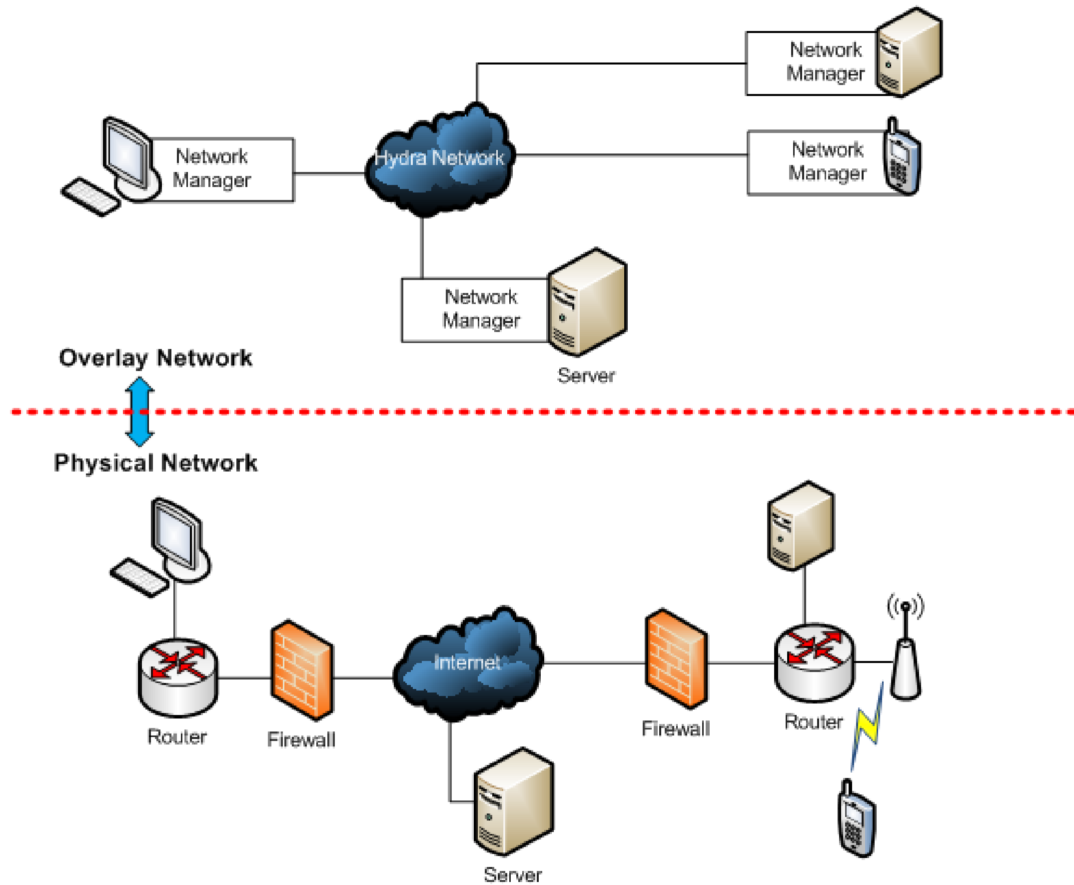

Figure 3: Overlay Network

The figure above shows an example of how the different nodes are interconnected in the LinkSmart Network.

3.1.2 The HID addressing scheme

The addressing scheme used in the LinkSmart Network is based on identifiers called HID. An HID represents a service endpoint, for instance a WebService endpoint. Each service or device has to create an HID in order to be visible inside the LinkSmart Network. For simple services the Network Manager provides one interface for registration:

```
String HID createHIDwDesc(String description, String endpoint)
```

Any application, or software component, in the system can register its services or devices in the Network Manager. A specific interface (createHIDwDesc) provides a mechanism for registering HIDs using a description and the local endpoint where the service will be placed.

The description is provided by the application or component itself, and it provides a way for identifying the service in other LinkSmart-enabled devices. In this way, an application running on another LinkSmart-enabled device is able to get all the HIDs matching a specific description through the following Network Manager interface:

```
String[] getHIDsbyDescription(String description)
```

The endpoint allows the Network Manager to know where to deliver the data received for an HID, because otherwise it would be impossible to determine which component or application is responsible of managing the resource registered with that HID.

3.1.3 SOAP Tunnelling

Thus, the Network Manager enables a way to communicate between different LinkSmart Enabled devices transparently, building an overlay network in which resources (devices, services and contents) are identified by an HID. The main objective of the SOAP tunnelling communication is for LinkSmart to provide SOAP messages exchange using the P2P transport schemes provided by the Network Manager.

In order to use P2P networking/addressing/transport schemes together with web services and UPnP we need some kind of virtualisation of endpoints that allow us to use P2P networking. For this reason, all endpoints for UPnP and web service calls are grounded in a SOAP sink (ideally locally) which repackages the SOAP message and routes it through the Network Manager, as shown in Figure 4. The Network Manager is responsible of the message transmission and finally calls the SOAP sink that performs a local SOAP call to the intended SOAP endpoint.

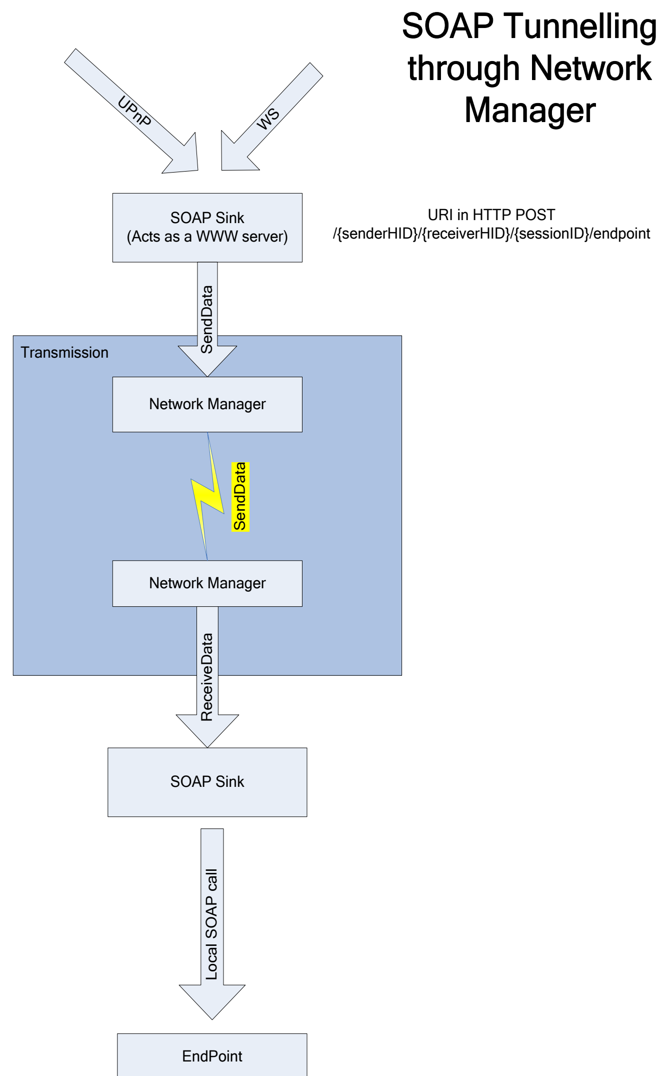

Figure 4: SOAP tunnel

The P2P networking with the SOAP tunnelling technique will facilitate event management, as well as SOA in generalin the BemoCofra architecture.

3.2 LinkSmart Event Manager

Below we describe the practical usage of events when developing application logic based on the LinkSmart event manager. Examples are based on the .Net client code of LinkSmart, but there is also a corresponding Java version.

3.2.1 Event structure

Events are a useful tool for several situations in **application** development. When working with sensors publish/subscribe based events processing is an efficient way of retrieving values, instead of polling sensor values. This way, multiple clients can receive events with the current sensor values.

Events in LinkSmart are implemented using a standard “publish/subscribe” model and the event itself is built up with a “topic” that is used for defining the event topic and an arbitrary number of key value pairs.

```
<Event>
  <Topic>MyTopic/SubTopic</Topic>
  <Part>
    <Key>ExampleKey</Key>
    <Value>ExampleValue</Value>
  </Part>
  <Part>
    <Key>ExampleKey2</Key>
    <Value>ExampleValue2</Value>
  </Part>
  <Part>
    <Key>ExampleKeyN</Key>
    <Value>ExampleValueN</Value>
  </Part>
</Event>
```

Listing 1: Event topic and events

There are two main parts when working with events to be used in applications:

- Creating events to be consumed elsewhere
- Listening to events

3.2.2 Creating events

In order to create events one only needs to contact the Event Manager. In LinkSmart projects created using the .net libraries the Event manager is available in meventmanager. Basically one sets the Topic of the event and then populates the Key/Value pairs.

```
//Create an event with two key/value pairs
global::part[] eventValueKayPairs = new global::part[2];
eventValueKayPairs[0].key = "ExampleKey";
eventValueKayPairs[0].value = "ExampleValue";

eventValueKayPairs[1].key = "ExampleKey2";
eventValueKayPairs[1].value = "ExampleValue2";
```

```
m_eventmanager.publish("ExampleTopic", eventValueKayPairs);
```

Listing 2: Example code creating and publishing an event

3.2.3 Listening to events

Listening to events require that a web service is created that receives the events and it is required that it follows a specific EventSubscriber WSDL. In the projects created with the LinkSmart.net libraries this already done and exists in the EventSubscriberService.cs.

The creation of the Web Service is done using standard .net WCF methods:

```
//Create the ws on port 8123
string address = string.Format("http://{0}:{1}/Service", "localhost", "8123");
Uri[] BaseAddresses = new Uri[]{
             new Uri(address)};
//Turn off 100-continue
System.Net.ServicePointManager.Expect100Continue = false;
//Create the event subscriber
using (ServiceHost serviceHost = new ServiceHost(typeof(Test), BaseAddresses))
{
    try
    {
        ServiceMetadataBehavior smb;
        if                                  ((smb                                =
serviceHost.Description.Behaviors.Find<ServiceMetadataBehavior>()) == null)
        {
            smb = new ServiceMetadataBehavior();
            smb.HttpGetEnabled = true;
            serviceHost.Description.Behaviors.Add(smb);
        }
serviceHost.AddServiceEndpoint(typeof(IMetadataExchange),
MetadataExchangeBindings.CreateMexHttpBinding(), address + "mex");
serviceHost.AddServiceEndpoint(typeof(EventSubscriber),                    new
BasicHttpBinding(BasicHttpSecurityMode.None), "");
        serviceHost.Open();
    }
catch (Exception e) { Console.WriteLine(e.Message); }
```

Listing 3: Creation of the Web Service for event listening

The next step is to define the function that will receive/implement the Web Service call. In this case the message handling needs to be quick, because the Event Manager will remove the subscriber if the call fails due to time out. If the processing of individual events takes a lot of time one should consider using asynchronous worker threads so that Web Service call can return immediately.

The code below shows an example implementation of the method that receives the events. This implementation only writes the event to the console.

```
#region EventSubscriber Members

//Event call back interface
notifyResponse EventSubscriber.notify(notify request)
{
    string result = request.topic + "\n========\n";
    try
    {
        foreach (part _part in request.@event)
        {
            result += _part.key + "=" + _part.value + "\n";
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
    Console.WriteLine(result);
    return new notifyResponse(true);        }
#endregion
```

Listing 4: Event reception method

The final step is to subscribe to the events that one wants to handle. This involves creating an HID for the event Web Service endpoint and registering the events that will be subscribed using the Event Manager.

```
//Create HID for Event Subscriber WS
    string myhid = m_networkmanager.createHIDwDesc("eventExample", address);

 //Listen to ExempleEvent
 m_eventmanager.subscribeWithHID("ExampleEvent", myhid);
 //One can listen to multiple events with same interface
 m_eventmanager.subscribeWithHID("ExampleEvent2", myhid);
```

It is also important to remove the subscriptions when the process ends. Otherwise one can receive multiple events for one actual event, because we might have multiple subscribers with the same end point. The code below cleans up subscriptions and HID

```
//Unsubscribe to Events

m_eventmanager.subscribeWithHID("ExampleEvent", myhid);

m_eventmanager.subscribeWithHID("ExampleEvent2", myhid);


//Remove HID

m_networkmanager.removeHID(myhid);
```

Listing 5: Subscribe/Unsubscribe to events

4. Examples of LinkSmart integration

4.1 Architecture of the PLC Proxy

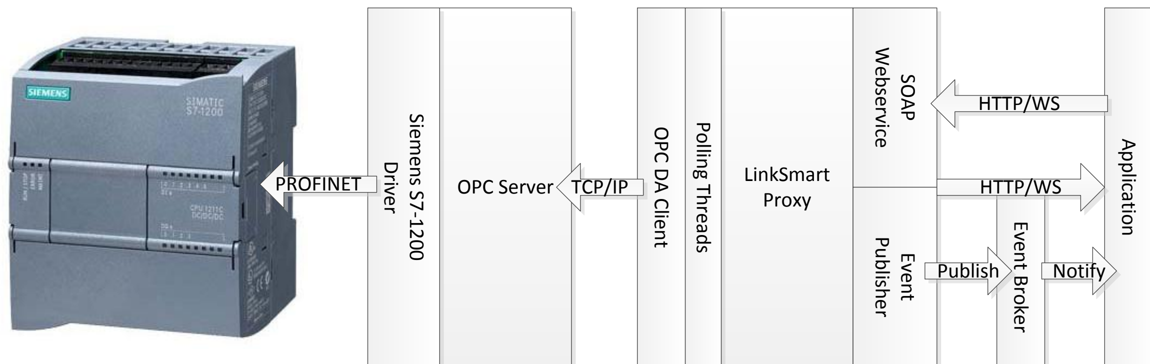

Figure 5. PLC Proxy Architecture

OPC: the term OPC stands for **O**LE (Objects Linking and Embedding) for **P**rocess **C**ontrol specific for automation devices. It was developed to ensure the communication of real-time data between control devices coming from different suppliers, in order to provide a common bridge for Windows based software applications and process control hardware.

Therefore, to enable this uniform integration between hardware and software, a connection through the OPC server must be established. Moreover, following the requirements of OPC Data Access, an **OPC Item** object, in this case a PLC variable, must be bundled into an **OPC Group** object before it can be accessed by an OPC client.

In our scenario, the OPC client is a .NET application that accesses the PLC variables through the OPC server and is mostly responsible for three things:

- Publish events in accordance to the states of PLC variables:
  The prerequisite for this task is the initialization of an event publisher within the LinkSmart network, as well as the various event objects that represent different data flows in the scenario. To keep track of the PLC variables, the application enters a thread that pools the variables and performs comparisons of their values at different timestamps (in this case, every 500 ms). When a variable's value changes, the appropriate event object is updated and subsequently published into the network.
- Modify PLC variables in accordance to incoming events:
  The application must also initialize an event subscriber and provide a delegate method that determines which tasks should be performed when a certain incoming event is received. Afterwards, the client simply waits for the events and modifies the appropriate PLC variables when a certain event arrives.

- Provide an outward mechanism to access and modify PLC variables:
  The client then utilizes Windows Communication Foundation (WCF) service to provide the other parts of the system a possibility to work with the PLC variables. These services include the setter

and getter methods for most of the variables contained in the PLC. It should be noted that the PLC variables that actually control the state of hardware (the “actuator” variables) should not be modified from outside the PLC in order to ensure the correctness of the PLC logic, and therefore no *set* methods are provided for them.

The following is the (sequence) diagram of the scenario that the client application adheres to. The client application resides the same lane/column as the PLC. Incoming arrow into the column denotes incoming events while arrows with the opposite direction denote events published by the application.

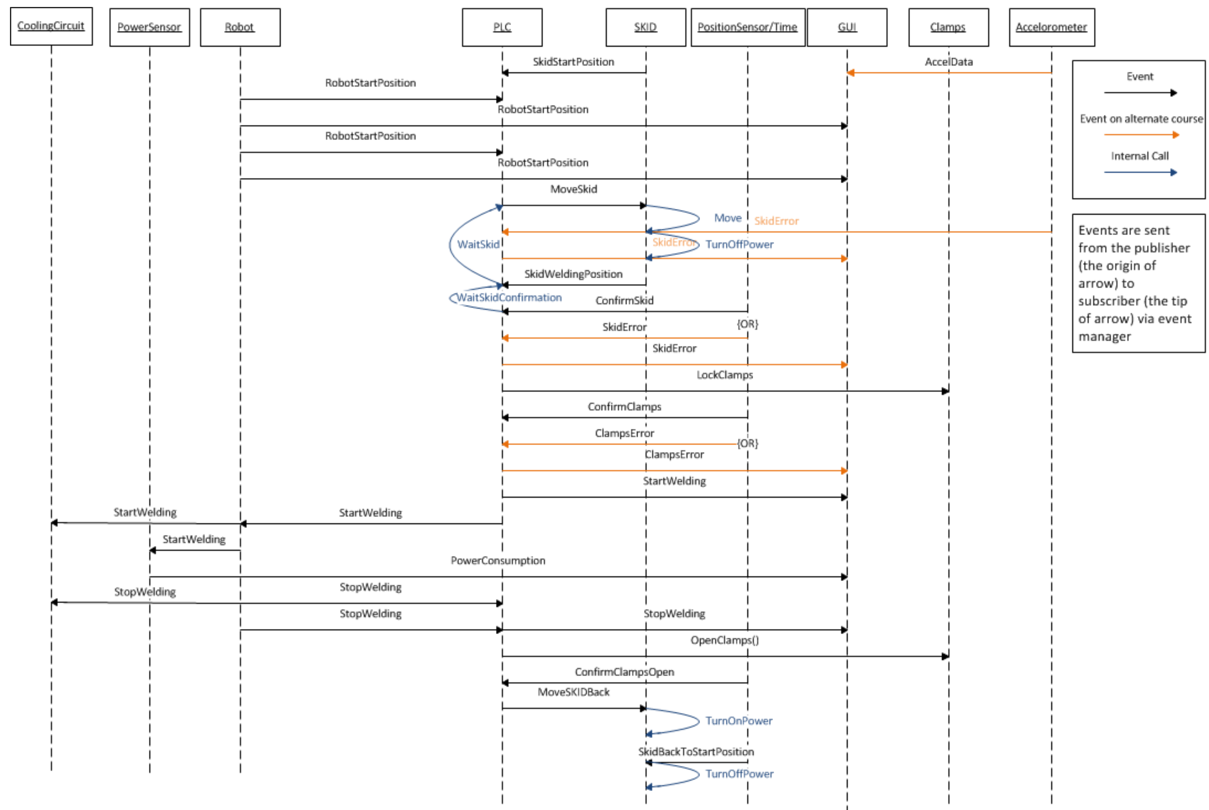

Figure 6: System Scenario

It bears repeating that to perform its tasks the client application must first establish a connection with the OPC server. Furthermore, the client also needs to register an OPC Group and an array of OPC items (PLC variables) that it wants to have access to. Finally, due to the nature of its functionalities, the client application must be active as long as the system is running.

4.2 Architecture of the Unity integration (Monitoring Application)

Unity is a cross platform development platform for games and other 3D visualisations, see (Unity,2012). Applications developed with Unity run on a range of different platforms including PC, Android and iOS. The language used for the developing in Unity is Mono which is an Open Source version of the .Net environment.

The monitoring application is developed in Unity because of the good visualisation capabilities as well as the cross platform capabilities. Because Unity runs on more powerful platforms, i.e. tablets, smartphones etc, and that it contains a full development environment we choose to integrate it with LinkSmart directly without usage of any proxy.

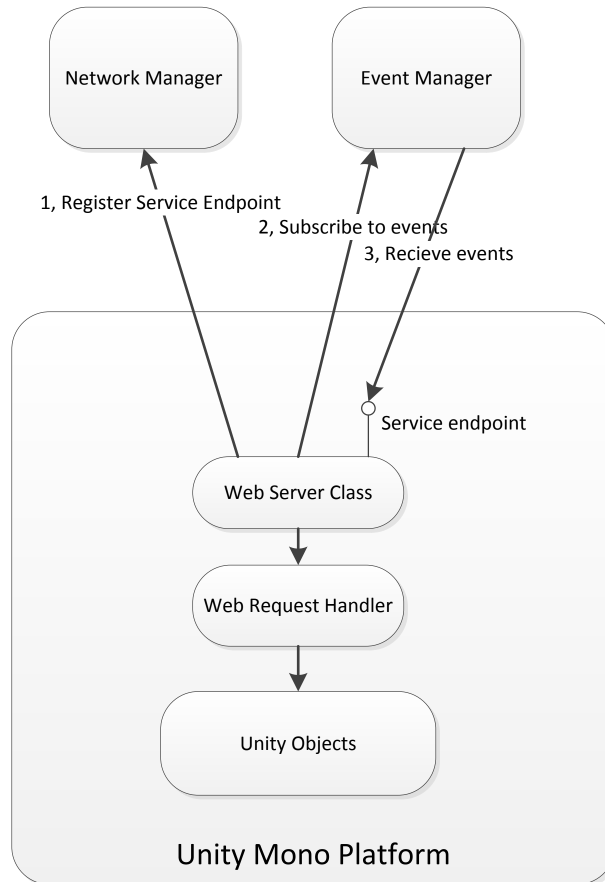

Figure 7: LinkSmart integration with monitoring application (Unity)

The basic integration of LinkSmart (See Figure 7) consists of two basic components:

- Web Server class: Implements an HTTP server that can be extended with handlers.
- Web Request Handler: That intercepts HTTP request made to the Web Server class, parses them and finally invokes the corresponding actions that map to an event.

The basic interaction steps necessary to connect to the LinkSmart network are:

1. Create a service endpoint: This involves registering the event consumer as a service endpoint in the LinkSmart Network Manager, i.e. getting an HID for the endpoint. As soon as there is an HID for the endpoint it is possible to invoke the service on the LinkSmart network using SOAP tunnelling.
2. Subscribe to events: This step uses the HID for the event consumer to subscribe to events. This means that all matching events (Depending on which Topics has been subscribed to) will be forwarded to the event consumer service endpoint.

Note that normally in most programming environments these call backs would be made using Web Service libraries as opposed to running a general Web Server and intercepting the HTTP calls, but Unity lacked the

support for this when handling parallel incoming events so we had to go down to the HTTP level and parse the incoming messages ourselves.

The following listings will illustrate (though a bit simplified) how this work.

```
public void RunHttpServer()
        {
            //Start the Web Server at port 8080
            m_endPoint = "http://*:8080/"; ;
            m_webServer = new WebServer(m_endPoint);

            //Put up a request Listener
            m_webServer.IncomingRequest += WebServer_IncomingRequest;
            m_webServer.Start();


            //Attach the endPoint to the LinkSmart network
                   NetworkManager.NetworkManagerApplicationClient nm = new
       NetworkManager.NetworkManagerApplicationClient();
            string hid = nm.createHIDwDesc(0,0,"UnityClient", m_endPoint);

            //Make a subscription , Topic is a regexp and we want to subscribe to all
events, i.e Topic=.*
            EventManager.EventManagerPortClient em = new
EventManager.EventManagerPortClient();
            em.subscribeWithHID(".*", hid);

        }
```

Listing 6: Setting up the web server and registering the endpoint

In Listing 6 above shows how the web server is started and how we register the service in the LinkSmart network . It also shows how we make a subscription for all events to the service end point. Not also that we add a request listener that will intercept the calls to the web server.

```
        public void WebServer_IncomingRequest(object sender, HttpRequestEventArgs e)
        {
            HttpListenerResponse response = e.RequestContext.Response;
            HttpListenerRequest request = e.RequestContext.Request;

            StreamReader sr = new StreamReader(request.InputStream);
            try
            {
                    XmlDocument xDoc = new XmlDocument();
                    if (!sr.EndOfStream)
                    {
                        xDoc.Load(sr);

                       //We have a request;
                        string Topic = "";
                        XmlNode xNode = xDoc.SelectSingleNode("//*[local-name()='topic']");
                        if (xNode != null)
                        {
                            Topic = xNode.InnerText;
                             Listener.Part[] parts = CreateParts(xDoc);
                             ProcessEvent(Topic, parts);
                         }
```

```
                }

            }

        catch (Exception ex)
        {
            System.Console.WriteLine("Exception When Processing HTTP Call");
            System.Console.WriteLine(ex.Message);
        }

    }

    static public Listener.Part[] CreateParts(XmlDocument xDoc)
    {
        XmlNodeList xNodes = xDoc.SelectNodes("//*[local-name()='Part']");
        Listener.Part[] res = new Listener.Part[xNodes.Count];
        for (int i = 0; i < xNodes.Count; i++)
        {
            Listener.Part p = new Listener.Part();
            XmlNode xKey = xNodes[i].SelectSingleNode("//*[local-name()='key']");
            if (xKey != null)
                p.key = xKey.InnerText;
            XmlNode xValue = xNodes[i].SelectSingleNode("//*[local-name()='value']");
            if (xValue != null)
                p.value = xValue.InnerText;

            res[i] = p;
        }
        return res;
    }
```

Listing 7: Parsing the incoming event

Listing 7 contains the actual parsing of the incoming event . The function WebServer_IncomingRequest is called by the Web Server when an HTTP call has been received. Since it is a Web Service call the payload is in SOAP format, i.e. XML, we parse it using an XML parser. The CreateParts function parses the events key value pairs. In most programming environments all this parsing would be automatically handled by the normal Web Service libraries. But as explained earlier the Mono framework in Unity lacked support for parallel invocation of the web service service.

References

(LINKSMART, 2012) http://www.hydramiddleware.eu/news.php, visited 2012-11-15.

(LINKSMART2,2012) http://sourceforge.net/projects/linksmart/, visited 2012-11-15.

(LINKSMART3,2012) D12.9 Final External Developers Workshops Teaching Materials.pdf ,visited 2012-11-15.

(Unity,2012) http://unity3d.com/unity/ , visited 2012-11-15.