Short Paper: A Scripting-Free Control Logic Editor for the Internet of Things

Markus Jung, Esad Hajdarevic, Wolfgang Kastner
Institute of Computer Aided Automation
Vienna University of Technology
Vienna, Austria
e-mail: {mjung,ehajdarevic,k}@auto.tuwien.ac.at

Antonio Jara
Information Systems Institute
University of Applied Sciences Western Switzerland
Sierre, Switzerland
e-mail: jara@ieee.org

***Abstract*—The Internet of Things scales the Internet to billions of embedded nodes and allows to link physical and cyber systems to form complex control systems. Current research focuses mainly on the networking and communication protocols and leaves the application layer and aspects like the engineering process and creation of control logic out of scope. Existing approaches are mainly based on using scripting languages to create control logic for the Internet of Things, which is a problem for non-technical users. This paper presents oBeliX which is a generic user interface and graphical control logic editor for the Internet of Things. The system requirements to enable a scripting-free creation of control logic are stated and a concrete system fulfilling these requirements together with a proof of concept implementation and evaluation are presented.**

***Index Terms*—Internet of Things, IPv6, graphical control logic**

I. Related work

A centralized control approach based on a RESTful runtime script container is presented in [1] where a JavaScript execution engine is used as runtime container and acts a client/server model with CoAP enabled devices.

The WoTKit [2] provides a framework to create IoT mashups. Its architecture is based on a Java Web application using the Spring Framework. For exchanging sensor data between components, a Java Messaging Service (JMS) broker is used. A graphical editor allows wiring logic modules similar to the popular Yahoo Pipe Web mashup editor.

There are several Internet of Things platforms like Paraimpu [3], Xively[1] (known as COSM or Pachube) or ThingSpeak[2]. What they have in common is a centralized cloud platform that is used to collect sensor data and information about devices.

II. System requirements

Within this section some generalized requirements are stated that need to be provided by an Internet of Things system in order to allow a generic user interface for creating distributed control logic based on graphical means without any needs for programming or scripting. In general, the different ways to create control logic for the Internet of Things can be divided into only script-based, graphical-based or hybrid-based approaches. Further, the control logic is executed on a central controller or distributed amongst the devices that are involved to fulfil the desired function. The following requirements need to be met in order to create control logic by graphical means only and to provide a decentralized execution: i) **data point centric information representation**, ii) **generic base information model**, iii) **simple application layer communication services**, iv) **group communication interaction model**, and v) **logic and virtual entities**.

[1]https://xively.com/
[2]https://www.thingspeak.com/

III. IoTSyS: A communication stack for the Internet of Things

IoTSyS is a complete system stack designed for the Internet of Things fulfilling the requirements stated in Section II. It can be deployed on constrained sensors or actuators and can also be used as integration middleware for state of the art home and building automation technologies. Details on the communication stack have been presented in [4].

The IoTSyS protocol stack is based on mature and novel Internet communication protocols and Web standards. It uses oBIX for the application layer information models which follow a **data point centric object model** and **simple application layer communication services** are based on a RESTful design paradigm offering `read`, `write`, `delete` or `invoke` protocol commands. Within IoTSyS a novel protocol binding to CoAP and message encodings to JSON and EXI have been created. In that way, the stack provides HTTP, CoAP and SOAP for message exchange. A client can freely choose as message encoding either XML, JSON, EXI or oBIX Binary. IPv6 can be used as network layer and is required by our **group communication mechanism** which cares for a peer to peer interaction model for oBIX devices. Virtual logic objects provide functionality that is used to create complex control scenarios. This can range from simple numerical functions, boolean operators to more sophisticated functions like PID controllers.

Figure 1 illustrates an overview of the complete system architecture. Existing home and building automation systems are integrated into the system stack using the IoTSyS gateway, offering oBIX communication interfaces for these devices. Beside the connectors to sensors and actuators, also virtual

978-1-4799-3459-1/14/$31.00 ©2014 IEEE

connectors to, for example, a weather data service, are integrated into the gateway. The gateway also hosts the oBeliX HTML5 user interface that can directly operate on oBIX interfaces. For the user interface and for engineering the group communication endpoints, all objects need to be hosted in the gateway. For "native" devices that use the communication stack in the field, a simple CoAP to HTTP proxy module provides a virtual representation of these objects. In this way, the complete engineering of the control logic can be performed at the gateway but process communication happens in a decentralized way since the group tables are stored on the devices. For process communication, IPv6 multicasting is used together with the CoAP binding and EXI encoding to optimize the payload size.

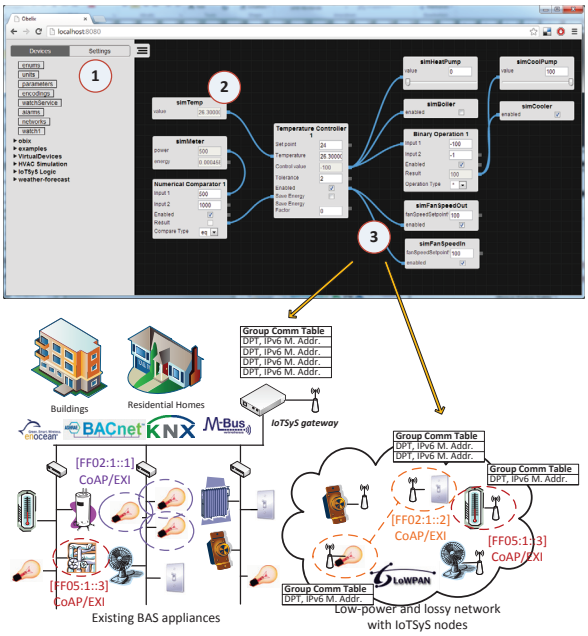

Fig. 1: IoTSyS overview with oBeliX graphical user interface

IV. oBeliX: A generic HTML5 user interface and control logic editor

oBeliX is an open-source[3] HTML5 control engineering interface based on JavaScript and CSS. It is a generic oBIX client that directly operates on the Web service interfaces provided by the gateway component. It is deployed as standalone file served by the gateway component and uses the HTTP and JSON protocol binding for oBIX. An ① **object browser** is the central entry point for a user as illustrated in Figure 1. It lists the available objects based on a query using the oBIX lobby. Since everything is an object in oBIX which may have an arbitrary number of sub objects the structure is recursively analyzed and an entry is displayed in the object browser based on the name. The user can drag an element out of the browser into the object canvas for display and to update values.

[3]http://code.google.com/p/iotsys

For each oBIX object in the object canvas an ② **object component** is displayed. Following the oBIX object model, an object consists of sub objects which are either base value types like, for example, `bool`, `int`, `real`, `str` or complex objects. An object is rendered as component that provides HTML5 input elements for all base value types. For rendering the object, a simple `get` request is performed on the object URI and the object structure is parsed dynamically. On a change of a base value property an according `put` request is performed and the object is updated at the server side.

The object component allows a simple interaction with devices and virtual objects represented through oBIX objects. For engineering the ③ **group communication relationships**, a graphical wire tool allows to group data points of different objects together. Whether a data point can participate in group communication or not is determined through a group communication object that is attached as child object to the basic value object. If such an object is present, connectors are displayed that can be used to graphically wire objects using a drag and drop mechanism. Once a connection is established a dedicated IPv6 multicast address is added to the according group communication objects.

For the evaluation of the control logic editor, a testbed equipped with heterogeneous home and building automation technologies is used and use case scenarios for **lighting control**, **alarming** and **HVAC control** have been successfully realized.

V. Conclusion

This paper presented a scripting-free control logic editor for the Internet of Things by stating the general requirements to create scripting-free control logic. Further, it was shown how a protocol stack based on IPv6, CoAP and oBIX can fulfil these requirements to setup scripting-free distributed control logic.

Acknowledgement

Authors express their acknowledgement to the consortium of the project IoT6 (www.iot6.eu). The IoT6 project is supported by funding under the Seventh Research Framework Program of the European Union, with the grant agreement FP7-ICT-2011-7-288445 and the Internet Foundation Austria (IPA).

References

[1] M. Kovatsch, M. Lanter, and S. Duquennoy, "Actinium: A RESTful Runtime Container for Scriptable Internet of Things Applications," in *Proceedings of the 3rd International Conference on the Internet of Things*, 2012.

[2] M. Blackstock and R. Lea, "IoT mashups with the WoTKit," in *Proceedings of the 3rd International Conference on the Internet of Things*, 2012.

[3] D. C. Pintus, Antonio and A. Piras, "Paraimpu: a platform for a social Web of things." in *Proceedings of the 21st ACM International Conference Companion on World Wide Web*, 2012.

[4] M. Jung and W. Kastner, "Efficient group communication based on Web services for reliable control in wireless automation," in *Proceedings of the 39th Annual Conference of the IEEE Industrial Electronics Society*, 2013.