Integration of IoT Platform and JADE Agent Infrastructure

Project Acronym	IoT4Industry
Document-Id	D.1
File name	
Version	FINAL document
Date	Start: 01 October 2014 End: 31 November 2014
Author(s)	Felix Strohmeier
QA Process	Dr. Violeta Damjanovic-Behrendt

Table of Content

IoT4Industry in a Nutshell

IoT4Industry is an exploratory project funded by the Austrian Research Promotion Agency, for the period October 2014 – September 2015. It stands for "Secure, Privacy-preserving Agents for the Industrial Internet". The core research areas of IoT4Industry relate to security, privacy and IPR/data protection requirements, translated into game-theoretic agent behavior, which is simulated in a demo factory floor environment, supporting a supply chain negotiation. A demo factory floor is implemented within our IoT-lab, which is located at Salzburg Research.

While it is still not clear which protocols and technologies will become mainstream for the Industrial Internet, it has become clear that security and privacy will feature strongly in any risk assessment concerning the adoption of practices using the Industrial Internet. Thus, in IoT4Industry, we focus on the use of policy-enacting, multi-agent systems that securely manage machines and manufacturing cells, and we build a feasibility demonstrator based on open source tools and firmware. In addition, IoT4Industry helps us to prepare for internationally recognized contributions to science and technology in the field of Industrial Internet, notably in Horizon 2020.

IoT4Industry Task 1 Description

(from the *IoT4Industry* project proposal)

D1: Integration of IoT Platform and JADE Agent Infrastructure

Goals:

1. Thingsquare/Contiki IoT platform and evaluation Kit is operational
2. JADE/WADE agent environment is operational
3. On-line presence of the integrated platforms
4. Simple communication between the platforms can be demonstrated

Description of the content

First integration of the Thingsquare/Contiki platform with JADE agents. We can receive and further communicate sensor data from the IoT platform to one agent and beyond.

Method

Hardware and software integration ranging from Thingsquare sensors (evaluation kit) to messaging between Contiki platform and JADE agent platform.

Milestones, results and deliverables

Working software is online, short technical report on the project Wiki.

Glossary

Constrained Application Protocol (CoAP) is a software protocol which allows simple electronics devices to communicate interactively over the Internet. It is particularly targeted for small low power sensors, switches, valves and similar components that need to be controlled or supervised remotely, through standard Internet networks (c.f. http://en.wikipedia.org/wiki/Constrained_Application_Protocol).

General Purpose Input/ Output (GPIO) is a generic pin on an integrated circuit whose behavior, including whether it is an input or output pin, can be controlled by the user at run time (c.f. http://en.wikipedia.org/wiki/General-purpose_input/output).

Lightweight M2M (LWM2M) is a standard for Device Management, Network Management and Application Data for the Internet of Things. This is CoAP and DTLS based standard which provides a complete system interface solution for M2M devices and services (c.f. http://www.slideshare.net/zdshelby/oma-lightweightm2-mtutorial).

1. Problem Statement

IoT4Industry investigates Industrial Internet (Industry 4.0) applications and their requirements related to privacy, security and data protection. In D.1 "*Integration of IoT Platform and JADE Agent Infrastructure"*, we summarize technical experience gained during setting up our local IoT laboratory, that is equipped with the Thingsquare/Contiki, Arduino and Raspberry Pi-based IoT platforms. In addition, we installed JADE agent framework in order to explore its usage in an IoT setting.

Document organization. After a brief description of the problem statement in *IoT4Industry*, in Section 2, we present our technical experience on integrating the agent framework and the IoT platform. In Section 3, we discuss our next steps targeting several alternative IoT platforms and agent frameworks supporting IoT.

2. Technical Experience from the Integration of the IoT platform and the Agent Framework

We identified the Thingsquare/Contiki as an IoT platform with the potential to implement the *IoT4Industry* project requirements, and to demonstrate the role of technologies such as agent systems and their Game Theoretic (GT) foundation to IoT-based supply chain negotiation. In parallel, we explored security and privacy aspects of IoT applications, and integrated Thingsquare/Contiki IoT platform and JADE agents. Our technical experience from this task is presented below.

2.1. Thingsquare / Contiki IoT Platform and its Evaluation Kit

This section summarizes our experience in installing Thingsquare/Contiki platform on the Texas Instruments Development Kit "cc2538dk". It required the following tasks to be accomplished:

- **Measurement of analogue values on the GPIO(s) (General Purpose Input/ Output)** to simulate an arbitrary sensor input. The sensor data has to be sent to a remote entity.
- **Setup of IEEE 802.15.4 communication between cc2538 node and a border router:**
    - Wireless Sensor Networks (WSN) can be extended by adding more nodes to the network. Each of the nodes can act as a router for other nodes.
    - The border router provides the connection from the WSN to IPv4/IPv6 Internet.
- **Setup of IP/HTTP communication between cc2538 and a Web server:** Application-

layer communication is required to support the agent communication (as discussed in Section 2.2). Instead of HTTP, CoAP (Constrained Application Protocol) which is based on LWM2M (Lightweight M2M) standard, will be considered in the future.

Following the instructions provided in [1], the following hardware (sensors, nodes and a router) has been purchased and connected (see Figure 1):

- cc2538 Development Kit (Texas Instruments[1]) with the following components:
  - 2x SmartRF06 Evaluation Boards,
  - 2x cc2538 Evaluation Modules;
- ENC28J60 Ethernet controller development board (Olimex[2]);
- Jumper Wires female/female;
- An Ethernet Hub with cables for packet sniffing;
- HTTP-server on remote virtual machine.

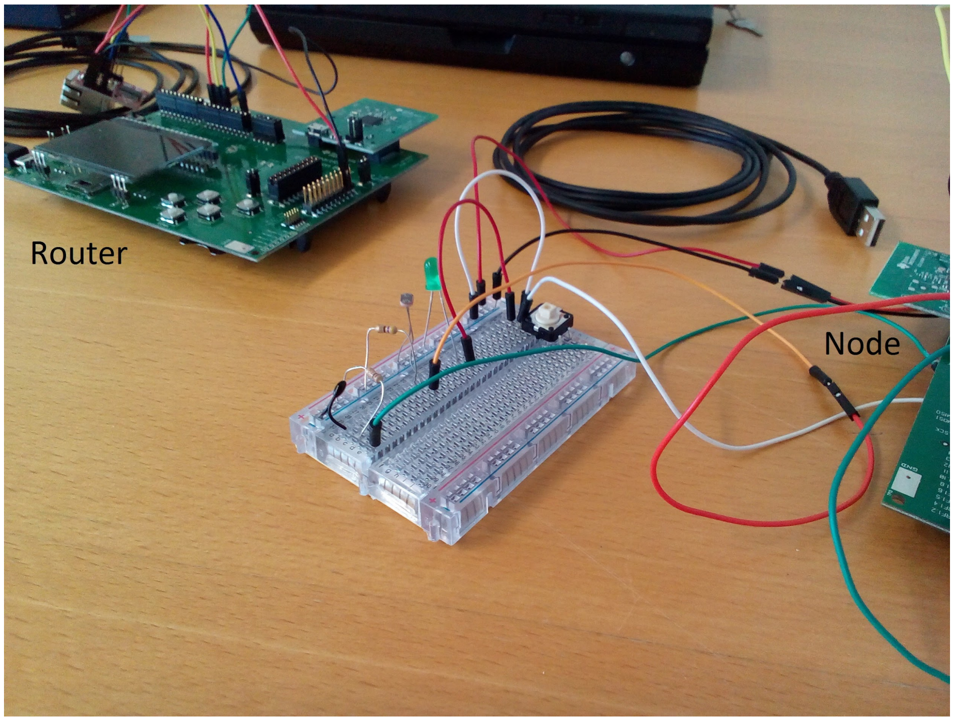

Figure 1: Hardware setup with a sensor, a node and a router

The following steps have been carried out in task T1:

1. **Testing of the modules** using the default firmware (by following the quick-start guide as

[1] Texas Instruments: http://www.ti.com/tool/cc2538dk
[2] Olimex: https://www.olimex.com/Products/Modules/Ethernet/ENC28J60-H/

presented in [2]): During this initial step, we tested the basic functionality of the hardware and communication. Test packets are sent from one module to another. Afterwards, the pre-installed firmware has been removed.

2. **Flashing the Thingsquare Firmware** using SmartRF Studio 7 [3], running one board as “node”, and the other board as “border router”.
3. **Connecting to the Thingsquare cloud** (c.f. https://demo.thsq.io/): In this step, the “node” has been connected to the publicly available Thingsqare cloud. Registering the “node” in the cloud is usually done automatically, but this did not work for the first try. The reason was that DNS-Lookup to the pre-configured DNS Server (8.8.8.8) was blocked by our firewall. Therefore the IP address from the Thingsquare demo server had to be entered manually (54.247.103.86) using the board buttons + LCD info. After registering on the platform, the “node” can be integrated into the demo environment by entering the “PIN” displayed on the LCD.
4. **Loading one of the demo programs** from the cloud (“blink”, “http-post”, “http-get”): Using the `http-post` and `http-get` examples provide data to be sent to the server. However, reading the sensor data requires additional libraries from Texas Instruments, which were not available using the cloud service. This lead us to build a Contiki firmware image from source, and installation into the flash memory of the nodes.
5. **Compiling and flashing own Contiki Image using Eclipse:** This step requires the installation of a *toolchain* as described in [4] (GNU Toolchain components include the compiler, linker, assembler, and a debugger). We tested it, but do not recommend it to the community because of many “dangerous” pit holes. We consider it to be an important step only if the features of the Eclipse IDE, such as code-autocompletion or remote debugging using `gdb`, are required.
6. **Compiling and flashing own Contiki Image using “Instant Contiki”**. Instant Contiki is a ready-to-use VM-Image (c.f. [5] for more details). The simulation steps (the Contiki network simulator, Cooja (c.f. http://www.contiki-os.org/start.html)) can be skipped in this case. We started with “Step 4: Run Contiki on hardware”. To run Contiki on our target platform, we had to replace “z1” with “cc2538dk”.
7. **Implementing “http-get”, “http-post”** based on the examples in the cloud using the GPIOs as sensor input. To cross-compile the code locally and flash it to platform, both libraries the cc2538dk_foundation_firmware [6] and the Thingsquare release of Contiki (thingsquare-mist) are required.

2.2. JADE Agent Environment

The introduction of an agent framework into the IoT context moves the communication paradigm from the client/server communication to peer-to-peer (p2p) communication between autonomous and proactive software agents. A discussion on the differences is available in [10]. In the context of *IoT4Industry*, each of the agents can therefore have its own sphere of action, such as reading and managing data from sensors or sensor platforms, retrieving information from other agents, publishing information, control actuators, etc.

JADE [7] provides a Java-based agent development framework for creating multi-agent systems by using an agent communication protocol that is standardised by the FIPA (Foundation for Intelligent Physical Agents). The agents exchange their messages using the agent communication language ACL, using communicative acts such as *inform*, *request*, *agree*, *not understood*, and *refuse*. It furthermore enables remote control of agents via a remote GUI.

The main features of JADE are as follows:

- the agent abstraction;
- a simple, powerful task execution;
- asynchronous p2p agent communication;
- publish/ subscribe discovery via a yellow pages service;
- agent communication via NAT (Network Address Translation) and firewalls including usage of dynamic IP addresses.

In addition to devices running Java SE version 5 or higher, JADE agents can also be deployed on devices running Android or J2ME-CLDC MIDP1.0. More information on the JADE platform is available in the JADE-Book [8]. An extension project to JADE is WADE [9], which brings in the "WorkflowEngineAgent", which allows for the execution of subsequent tasks within workflows. Both projects are released under the LGPL license.

2.2.1. JADE Deployment in IoT4Industry

In the following, we describe the deployment of a simple JADE Agent in *IoT4Industry* environment. JADE has been installed on several distributed machines in our IoT-lab, e.g. on a virtual server instance ("IL050"), a physical Laptop-PC ("iotlab1"), and several agents are

deployed on Raspberry Pi’s[3]. The Server IL050 which runs the main agent container, hosts the Directory Facility (DF) and the Agent Management Service (AMS), the RMA (“Remote Monitoring Agent”, including the GUI). The IL050 contains a “remote container”, and runs the agents remotely. Other agents run on the Raspberry Pi’s in the IoT-Lab.
Start-up commands of JADE platform are shown below:

```
il050 (the main container with DF and AMS):
java -cp "/data/jade/lib/jade.jar" jade.Boot -platform-id IoTAgentPlatform

iotlab1 (with RMA-GUI + 2 PingAgents):
java -cp "/home/iotlab/jade/lib/jade.jar:/home/iotlab/jade/classes" \
jade.Boot -container -host il050 –gui –agents \
"ping1:examples.PingAgent.PingAgent;ping2:examples.PingAgent.PingAgent"

Raspberry Pis (with one PingAgent):
java -cp "/home/pi/jade/lib/jade.jar:/home/pi/jade/classes" jade.Boot \
-container -host il050 –agents "ping-rasp:examples.PingAgent.PingAgent"
```

For the deployment of multiple agent platforms the following should be noted:
JADE is very dependent on the network configuration, including DNS and IP addresses, and therefore the setup of the communication only works if the forward and reverse DNS-lookup between IP addresses and hostnames is possible, and delivers the same results. To be able to communicate with remote JADE-Platforms, an MTP (Message Transfer Protocol) needs to be enabled. The default configuration of MTP does not work, because the ports are bound to the addresses retrieved by DNS-resolved address of the given hostname. In order to enable inter-platform communication, the DNS can be circumvented by passing the following parameter to the host of the Main Container for each of the communicating platforms like:

```
… -mtp "jade.mtp.http.MessageTransportProtocol(http://192.168.48.50:7778/acc)
```

Another approach to fix the DNS-resolution would be to edit the hostname resolution in the `/etc/hosts` file (used for local DNS) and use the IP-address of the network interface instead of a local address.

2.2.2. Running the “Contract-Net” Interaction Protocol with JADE

One of the interesting features of an agent framework such as JADE, is that predefined “interaction protocols”, e.g. Contract-Net are already implemented. Interaction protocols consist of a predefined sequence of communication acts, which go beyond simple publish-subscribe message protocols that are provided by other messaging systems.

[3] Raspberry Pi is a small embedded Linux PC, and an interesting platform for prototyping IoT applications. It is capable of running a Java runtime, and therefore JADE Agents.

In our IoT-lab, the Contract-Net interaction protocol is tested for the following scenario:
3D-Printing facilities can offer to print a product with different time and cost. A "call for proposals" is distributed from the `OrderingAgent` to all 3D printing agents called `PrintingFacilityAgent`, each of them providing a time and cost proposal depending on their current utilisation. Based on the proposed offers, the `OrderingAgent` can select the printing facility with a time*cost factor fitting best to its needs.

The following example sequence would include one `OrderingAgent` and four `3D-PrintingAgent`:

```
● Agent PrintingFacilityAgent1 waiting for CFP...
● Agent PrintingFacilityAgent2 waiting for CFP...
● Agent PrintingFacilityAgent3 waiting for CFP...
● Agent PrintingFacilityAgent4 waiting for CFP...
● Agent OrderingAgent trying to delegate printing-job to one out of 4
  responders.
● Agent PrintingFacilityAgent2: CFP received from OrderingAgent@AP.
● Action is printing-job
● Agent PrintingFacilityAgent2: Proposing Time: 7 Proposing Cost: 5 (35)
● Agent PrintingFacilityAgent3: CFP received from OrderingAgent@AP.
● Action is printing-job
● Agent PrintingFacilityAgent3: Proposing Time: 3 Proposing Cost: 15 (45)
● Agent PrintingFacilityAgent1: CFP received from OrderingAgent@AP.
● Action is printing-job
● Agent PrintingFacilityAgent1: Proposing Time: 20 Proposing Cost: 2 (40)
● Agent PrintingFacilityAgent4: CFP received from OrderingAgent@AP.
● Action is printing-job
● Agent PrintingFacilityAgent4: Refuse (no resources)
● Agent PrintingFacilityAgent2@AP proposed 7
● Agent PrintingFacilityAgent3@AP proposed 5
● Agent PrintingFacilityAgent1@AP proposed 3
● Agent PrintingFacilityAgent4@AP refused
● Accepting proposal 7 from responder PrintingFacilityAgent2@AP
● Agent PrintingFacilityAgent2: Proposal accepted
● Agent PrintingFacilityAgent2: Action successfully performed
● Agent PrintingFacilityAgent2@AP successfully performed the requested
  action
```

- `Agent PrintingFacilityAgent1: Proposal rejected`
- `Agent PrintingFacilityAgent3: Proposal rejected`

Due to the best time*cost value of 35 (7*5=35), `PrintingFacilityAgent2` will be selected, while offers of the other proposing agents will be rejected. In this example `PrintingFacilityAgent4` totally refused the acceptance of the proposal, while the proposals from `PrintingFacilityAgent1` and `PrintingFacilityAgent3` were rejected by the `OrderingAgent`.

3. Next Steps

Here we briefly present some alternative IoT platforms and alternative agent frameworks to be considered in an IoT industrial setting. For more information on alternative IoT platforms we refer the reader on our additional report: "*Report of Emerging IoT Platforms for the Industrial Internet"*, while more information on alternative agent frameworks is summarized in Appendix of the D.5 "*Security, Privacy and IPR/Data Protection Requirements translated into Game-Theoretic Agent Behaviour*".

3.1. Alternative IoT-Platforms

3.1.1 Arduino-based Sensor Platform

To include different types of sensor platforms, such as the Arduino-based sensor platform, an XBee-based Wireless Communication Shield has been set-up and tested. Arduino (c.f., http://arduino.cc) is an open source platform focusing on easy-to-use hardware and software. It can be equipped with an XBee-Shield (c.f., http://bit.ly/1Nr635Q) together with a WiFi module for Arduino "Roving RN-XVee", to provide sensors-agents communication via HTTP.

3.1.2 RIOT-OS

RIOT-OS (c.f., http://www.riot-os.org) provides similar functionality to Thingsquare/Contiki, and is now actively developed to support more and more platforms. In *IoT4Industry*, we consider further replacement of the Thingsquare/Contiki platform by RIOT-OS to gain more flexibility.

3.2. Alternative Agent Frameworks

3.2.1 AKKA

AKKA agents are bound to a single storage location for their lifetime, and allow mutation of that location (to a new state) to occur as a result of an action. Actions dispatched to an agent from another thread will occur in the order they were sent, potentially interleaved with actions dispatched to the same agent from other threads.

3.2.2 JIAC

JIAC (c.f. http://www.jiac.de/agent-frameworks/jiac-v/) supports the design, implementation, and deployment of software agent systems, as well as the development of the BDI (Belief - Desire - Intention) agents. JIAC can be used with a set of development, configuration and monitoring tools, such as:

- VSDT (Visual Service Design Tool) tool, which is a BPMN (Business Process Modeling Notation) editor, and multi-language transformation and workflow simulator;
- Asguard, for controlling distributed multi-agent infrastructures at runtime;
- AWE (Agent World Editor) for modeling and configuring an MAS. It also generates files for deploying the system;
- JIAC Toolipse, which is an integrated development environment containing all above mentioned tools (for more: http://www.jiac.de/development-tools/jiac-toolipse/).

Current development of JIAC and publications target industry scenarios (see: http://www.jiac.de/publications/).

References

[1] GitHub website:
https://github.com/thingsquare/hardware/blob/master/ti/cc2538dk/assembly/README.md
[2] CC2538 Development Kit Quick Start Guide. April 2013: http://www.ti.com/lit/pdf/swru347
[3] Texas Instruments, SmartRF Studio: http://www.ti.com/tool/smartrftm-studio
[4] B. Selvig. AN128 - Using GCC/GDB With CC2538. Application Reprot, SWRA553 - February 2014. http://www.ti.com/lit/an/swra443/swra443.pdf
[5] Get Started with Contiki: http://www.contiki-os.org/start.html
[6] Texas Instruments, CC2538 Foundation Firmware. http://www.ti.com/tool/cc2538-sw
[7] Telecom Italia Lab, JADE (Java Agent DEvelopment Framework). http://jade.tilab.com
[8] F. L. Bellifemine, G. Caire, D. Greenwood. “Developing Multi-Agent Systems with JADE” published by John Wiley & Sons, Ltd. 2007. http://jade.tilab.com/documentation/book/
[9] Telecom Italia Lab, WADE (Workflows and Agents Development Environment): http://jade.tilab.com/wadeproject/
[10] Jason Bloomberg, “Architecting the Internet of Things with Intelligent Agents”, http://www.devx.com/blog/agile/architecting-the-internet-of-things-with-intelligent-agents.html