OBOX – The Orchestration BOX for Raspberry Pi

Ondřej Severa[1]

1 Introduction

With the recent grow of the computational power on small embedded devices it is easier for developers to use advanced techniques and software packages which requires more power and memory than the specific optimized code for target platforms. Thus there are new possibilities of multi-platform solutions for embedded devices.

One of them is Java Virtual Machine (JVM). It runs precompiled code written in Java language. Nowadays such a virtual machine can run on top of the ARM processors which are very popular in small scale computers or embedded devices.

Cooperation between devices is built on top of the information exchange ie. between multiple devices in the factory network. Nowadays one of the favorite technique is Service Oriented Architecture (SOA) which uses Web Services as a multi-platform information exchange solution.

2 OBOX - Overview

The main goal of the orchestration box is to have a small embedded device which can be connected to the local network in the factory floor and will be able to orchestrate (control) all the processes using Web Services. The Raspberry PI is very suitable hardware for this task. It is full scale computer based on Linux OS. It have enough computational power and memory to run Java applications. It has one Ethernet port which allows connection of the board to the factory network. The main program is written in Java and executed using JVM for ARM processors. Described BPEL engine uses DPWS for discovery of all devices on the network and advanced XML Parser to process the BPEL file and execute given BPEL Activities.

2.1 DPWS Stack

The key part of the OBOX application is DPWS Stack (DPWS). It is framework which is capable to discovery new devices on the network and also communicate with them. There is only one known DPWS stack for Java with active development. It is called JMEDS - Java Multi Edition DPWS Stack (JMEDS). It allows developer to easily add or remove new Web Services, discover new devices, etc. It is compliant with several WS-* specifications.

2.2 BPEL Engine

The BPEL (Business process execution language) runtime engine is implemented according to the part of the WS-BPEL specification (BPEL). The engine is capable of execution of defined activities. It can be i.e. control of the conveyor, interaction with human operator,

[1] student of the postgraduate study program Applied science and Informatics, field Cybernetics, e-mail: osevera@kky.zcu.cz

confirmation of the new order, etc. It is composed from following modules:

- **XML Parser** - Parses the input BPEL file and creates hierarchy of instances according to the source file
- **BPEL Engine Core** - After initialization where all the variables are assigned and all the necessary Web Services are added it executes all the activities supplied by source BPEL file. One can find basic description in following paper (BPEL for Java).
- **DPWS Stack** - It is used for handling the Web Service communication. BPEL Core can introduce new Web Services, also all the call for the external WS. DPWS Stack is responsible for translation of the program data to SOAP(Simple Object Access Protocol) messages which are used in Web service communication.

Acknowledgement

This work was supported by E-SCOP project, the sub-program of Artemis call - project No. 332946 and by University of West Bohemia - SGS-2014-054

References

[BPEL] OASIS. *Web Services Business Process Execution Language Version 2.0.*(2007) URL: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

[BPEL for Java] ActiveVOS. *BPEL for Java developers.*(2013) URL: http://www.activevos.com/content/ start_here/technology/bpel_for_java_developers.pdf

[JMEDS] WS4D. *JMEDS - Java Multi Edition DPWS Stack*(2011) URL: http://ws4d.e-technik.uni-rostock.de/wp-content/uploads/2011/05/StackOverview.pdf

[DPWS] OASIS. *Devices Profile for Web Services (DPWS)*(2009) URL: http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01