Specification and Implementation of WoT Smart Object Architecture

Deliverable 2.2 (version 1)

Philippe Capdepuy

31 December 2014

Project ASAWoO

Adaptive Supervision of Avatar / Object Links for the Web of Objects

Grant Agreement: ANR-13-INFR-0012-04

Abstract

This document presents the architecture of smart objects. Broadly speaking, smart objects embed enough computational power to run the complete ASAWoO stack. We define the functional specifications of such smart objects, and the current implementation. Different scenarios are considered as testbeds for the architecture. Current progress and future work are detailed.

Contents

1 Introduction

This document describes the specification and current implementation of the ASAWoO smart object architecture on various smart objects.
Smart objects are defined as objects embedding enough computational power to run the entire WoT stack, providing code deployment capabilities and network connectivity. For instance these can be:

- Standard personal computers
- Servers
- Smartphones
- Embedded computers of high-end robots (generally runnning a Linux OS)
- Single-board computers found in many robotics, IoT and DIY projects (Raspberry Pi as a prototypical example)

On top of their own capabilities, smart objects also typically act as relays or WoT wrappers around various other equipments that are directly connected to them, such as sensors (smartphones GPS and IMUs, cameras, microphones, robot specific sensors) and actuators (heaters/coolers, lights, robot motors, audio systems), but also indirectly connected through various networking protocols (UPnP/DLNA, ROS, Avahi, DNS/DNS-SD, UbiWare, ModBus ...).

2 Functional Specifications

From a functional perspective, the WoT smart object architecture, should support the following:

- Instantiation of appliances and functionalities based on:
  - predefined configuration (configuration files)
  - dynamic requests (through an HTTP REST interface)
  - automatic discovery and composition
- Association of semantic aspects to functionalities through:
  - explicit definition
  - automatic discovery
  - composition/aggregation of already defined semantics (sub functionalities)
- Exposition of functionalities and their semantics:
  - exposition as HTTP REST interfaces for external access
  - direct exposition for local-machine in-avatar access
  - remote exposition for remote-machine in-avatar access
- Annoucement and discovery of avatar:
  - annoucement of Avatar functionalities/applications based on standard protocols
  - discovery of existing ASAWoO avatars and funtionalities
  - discovery of existing non-ASAWoO functionalities based on standard protocols
- Exposition of user interfaces provided as WoT applications
- Dynamic code deployment

3 Technical Specifications and Implementation

At this stage of the project, we have put the focus on implementing the core components of the architecture, providing the essential steps to configure and instantiate avatars, communicate with them locally and remotely, deploy and use basic WoT applications. Semantic contents are left aside from this document but the architecture defines the communication channels that are being used to convey these elements.

3.1 Test Scenarios

The following scenarios are used as testbeds to validate the architecture:

- Temperature regulation: regulate temperature of a room to a target temperature provided by the user based on the following devices: thermometer, heater, cooler
- Static intrusion monitoring: detect and notify a user of intrusions in a specific location based on the following avatars: motion sensor, camera, user interface
- Patrol and intrusion monitoring: patrol an area with a robot, detect intrusions based on a motion sensor, notify user with camera images, based on the following avatars: mobile robot, camera, motion sensor, user interface

Each of these scenarios can be instantiated in several configurations, from having all the components running on a single machine, to distributing them on multiple machines in an in-avatar or multi-avatar setup.

3.2 Implementation

Based on previous technical specifications deliverable-1-2, we use the Java OSGi component-based framework for the reference implementation. OSGi provides us with a framework and patterns for implementing weakly-coupled software components and handling dynamic code deployment and execution through the concept of OSGi bundles.
OSGi can run on all devices providing a Java Virtual Machine (JVM), which encompasses all the devices that are defined as Smart Objects. Java also provides us with portability to many different targets. In the test scenarios mentioned above the main hardware parts that have been used as targets are:

- Personal computers
- Raspberry Pi Single-Board-Computers
- Android smartphones

The WoT architecture follows the principles and organization defined in deliverable-1-1. The component deployment part is handled by OSGi native mechanisms. The current implementation covers the following components of the architecture:

- Interoperability Module: provides interfaces, mechanisms and specific drivers for communicating with physical appliances. Drivers are targeted at the test scenarios, more specifically: PC cameras, Raspberry Pi camera, Raspberry Pi GPIO connected to temperature sensors, motions sensors, relays for controlling heater/coolers, Android smartphone notification mechanisms, Thymio mobile robot. Definition of the appliances through configuration files with semantic placeholders.
- Local Functionality Module: registration of appliances' capabilities and functionalities, exposition for local in-avatar communication as OSGi services, exposition for remote in-avatar communication as DOSGi services.
- Collaboration Module: discovery of other avatars using ZooKepper registry mechanisms. Mechanisms for importing remote functionalities in the Functionality manager.

- Web Service Module: exposition of specific functionalities with an HTTP REST API allowing functionality invokation and semantic queries using Jetty and Java Servlets. Access to remote functionalities using HTTP client interfaces and integration in the functionality manager.
- WoT Application Module: definition of WoT application templates and implementation of examples applications
  - Functionality applications: provide new functionalities building up on existing ones, for instance: temperature regulation, static intrusion detection, patrol and intrusion detection
  - End-User applications: Web GUIs allowing to invoke and configure functionalities: display current temperature and define target, notify instrusions to user, configure patrolling and receive notifications
  - Developer applications: Web GUIs to introspect and configure avatars, list existing functionalities, invoke them, spawn and configure new appliances and functionalities;

4 Future Work

The current implementation allows for end-to-end testing of the architecture, from the spawning and configuration of avatars and appliances to end-user applications. In the next phase of the project the following points are going to be implemented:

- Filtering module: based on the semantics defining the context, the Context Manager will automatically filter exposed functionalities in order to provide only those that are relevant to the current context and that obey some user-defined configuration of privacy and security requirements.
- Automatic functionality spawning: thanks to reasoning on the available functionalities and the current context semantics, new functionalities will be automatically generated that are relevant to the user.
- Application repository and code deployment: at the moment application binaries are manually deployed and can then be dynamically instantiated. OSGi already provides some of the infrastructure for dynamically loading binaries. The next step involves defining mechanisms for declaring trusted repositories of applications, querying them semantically to obtain functionalities and then automatically obtaining and deploying the corresponding binaries to the local machine.
- Other drivers: More drivers will be implemented to extend the range of physical appliances that can be integrated in ASAWoO and to automatically wrap existing network-based functionalities.