Avatar-oriented WoT application specification

Deliverable 1.3 (version 2)

Université Lyon 1 - LIRIS

31 December 2017

Project ASAWoO

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

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

Contents

1 Introduction

In order to ease WoT application design and keep it independent from the characteristics of available things, a WoT application only deals with the functionality level. Hence, it mainly describes a hierarchy of functionalities, the end nodes of which are terminal functionalities (i.e. that have to be implemented by an object capability), and all others nodes are composed functionalities (i.e. that require sub-functionalities and query them using code modules). Some of these composed functionalities may require the capabilities of several things, and therefore, a collaboration between several avatars. The top-level functionality then corresponds to the application that the end-user wishes to use and possesses a Web GUI.

2 Packaging

The packaging process of a WoT app is inspired from existing formats from other projects such as Java Web Archive, firefox OS or COMPOSE. A WoT application is packaged in a compressed file, composed of: a Manifest file, containing the application name and description, and describing its contents; the semantic descriptions of the above-mentioned hierarchy of functionalities; the code modules corresponding to the algorithms that implement composed functionalities; the application context model, containing a semantic description of the application domain and a set of adaptation rules; a set of static files that constitute the application interface and allow end-users to execute and control the application through their Web browser by querying avatar functionalities as standard Web RESTful resources. Contents In the final version of the project, a WoT app is actually a Maven project, including all functionalities as OSGi bundles. In order to be executable in the OSGi WoT platform, it requires the following elements:

- A proper Provide-Capability[1] instruction in the configuration of the maven-bundle-plugin[2] for felix[3] in the pom.xml[4] file.
- One or many Require-Capability instructions to declare dependencies toward Asawoo functionalities
- A main Java class[5] that implements the asawoo.core.wotapp.WoTApp interface and uses felix annotations to be instantiated at startup.
- A build process that produces a directory target/classes/WEB-INF with an index.html file.

3 Existing applications

We developed several applications of this type, for toy projects, for our final demonstration, as well as a “Hello World” application to help future developers start with the ASAWoO WoT infrastructure. The ASAWoO WoT application architecture is described in Deliverable 1.3 and the list of available WoT applications and functionalities in Deliverable 5.4.

4 Marketplace

A dedicated Marketplace currently hosts our applications, but can import further ones. As it is designed to be queried by WoT platforms, it has no frontend user interface.

[1] https://gitlab.com/asawoo/applications/blob/master/wotapps/hello-world-wotapp/pom.xml#L45
[2] http://felix.apache.org/documentation/subprojects/apache-felix-maven-bundle-plugin-bnd.html
[3] http://felix.apache.org/
[4] https://gitlab.com/asawoo/applications/blob/master/wotapps/hello-world-wotapp/pom.xml#L42
[5] https://gitlab.com/asawoo/applications/blob/master/wotapps/hello-world-wotapp/src/main/java/asawoo/wotapp/helloworld/HelloWorld.java