ASAWoO ANR Project
Deliverable 2.3: Description and Deployment of Services

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

Abstract

This document presents the different types of physical objects we consider and proposes a taxonomy of these objects. It gives an overview of the Avatar architecture, and of the concept of service we consider in the ASAWoO project. It specifies how the functionalities provided and required by the services are described semantically, and how this semantic description is used to deployed services on physical objects and/or on cloud infrastructures.

Contents

1 Introduction

The “Web of Things” (WoT) extends the Internet of Things so that physical object can be accessed and controlled using Web standards. Objects are expected to expose logical interfaces through Web services, to describe Web contents and services using semantic Web languages and annotations, and to communicate together through standard protocols in order to provide software interoperability between objects. Although the number and types of connected objects increases quickly, the WoT is not yet a reality, as several issues must be addressed in order to seamlessly interconnect physical objects, and to make these objects accessible on the Web. Indeed, objects are heterogeneous and are rarely able to communicate with each other as they do not share the same communication protocols or the same data formats.

In project ASAWoO, we introduce a new kind of software artifact called “avatar”. Avatars provide a virtual extension to physical objects. Physical objects can therefore be coupled with avatars, to form cyber-physical objects that are compatible with WoT constraints and infrastructures. Avatars are implemented using a distributed software platform that can be fully deployed on powerful objets, or distributed over resource-constrained objects and a cloud infrastructure. This platform is designed and developed in the ASAWoO project to address the research challenges of the WoT.

2 Physical Object Constraints

Physical objects (e.g., robots, sensors, camera, mobile phones, connected-watches) can have radically different functional and physical capabilities (i.e., processing, acting, sensing and storage). Some objects are fixed and can be connected to the Internet using wired links, while others are mobile and can suffer from connectivity disruptions due to their mobility and the short communication range of their wireless interface.

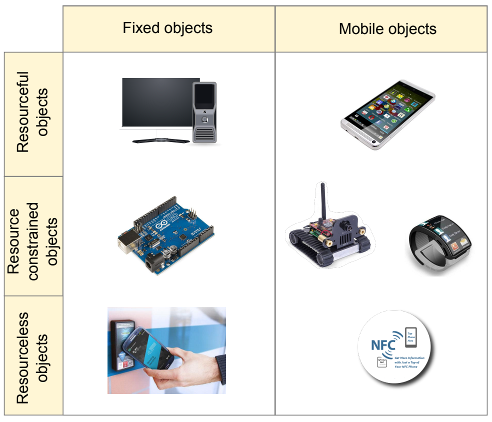

Figure 1: Examples of physical objects considered in project ASAWoO.

From our point of view, connected objects can be classified in 6 different categories, depending if they are fixed or mobile and depending on their physical capabilities. Resourceful objects embed a Web of Things (WoT) platform that hosts all the services they provide. Installation of these objects is often simple since they are standalone and do not require additional infrastructure elements to run. Resource-constrained objects cannot embed the whole WoT software platform due to restricted resources, but it is possible to link them to distant hosts that can embed missing parts of the platform. Resourceless Objects are passive objects, detected using unique identifiers such as QR codes or RFID tags. They do not have any computation, storage or memory capability. They can be extended with software deployed on the cloud or on the local network gateway.

3 Overview of the Avatar Architecture and of the WoT Infrastructure

In order to address the issues posed by the interoperability, the management, the communication of these heterogeneous devices, we have designed both an autonomous distributed platform, called Avatar, and an WoT infrastructure allowing to access and to deploy instances of the Avatar platform on the connected objects or/and on hosts forming a cloud infrastructure.

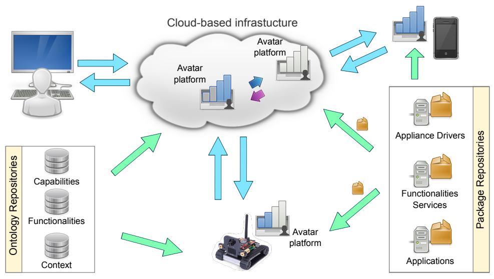

Figure 2: WoT Infrastructure designed in project ASAWoO

The WoT infrastructure illustrated in Figure 2 is composed of ontology repositories, software package repositories and runtime environments that can host avatar platforms. Ontology repositories contain RDF-based ontologies that can be used by avatar platforms to infer from their runtime context, and from the capabilities and functionalities they provide, which additional functionalities and capabilities they could offer by deploying new services locally, or by exploiting remote services provided by other avatar platforms. These ontologies are processed by a specific element, called Reasoner (see Figure 3). This reasoner relies on three other main managers of the core module, namely the functionality manager, the deployment manager and the context manager. The functionality manager is used to maintain an up-to-date list of the functionalities provided by the avatar (i.e., by the physical object). These functionalities can appear and disappear dynamically according to the service that are deployed, started and stopped by the avatar platform itself according to its execution context. For example, due to power budget limitation, a mobile object (e.g., a robot) can deactivate some functionalities that are known for their high energy consumption. The deployment manager is especially responsible for 1) obtaining semantic descriptions of services from the software package repositories, 2) providing these descriptors to the reasoner, 3) downloading from the repositories the software packages identified by the reasoner, and 4) deploying these packages locally. These packages are the service deployment units. They contain pieces of code and additional resources (configuration files, ...). These deployment units are detailed in the remainder of this document.

In our platform, the Web Service module allows to declare local services as REST Web Services. Doing so, it is possible to interact with avatars (and the physical objects) using the standard HTTP or COAP protocols, and to ensure the interoperability between the avatars/objects. This Web Service module is also detailed in the rest of this document. The three other modules shown in Figure 3, are not in the topic of this document, and therefore are not presented.

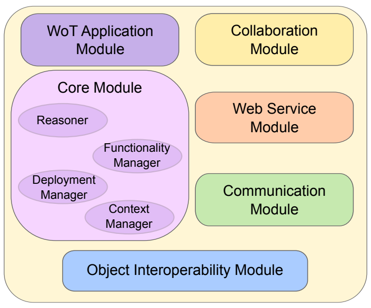

Figure 3: General Architecture of the Avatar Platform.

4 Discovery of Objects and Avatars

To advertise the services they offer, the avatars/objects are expected to announce their presence in the WoT infrastructure. We consider two types of infrastructures: a centralized infrastructure and a peer-to-peer infrastructure. In the centralized infrastructure, the avatars announce both their presence and their services using a centralized registry. Doing so, users or other avatars can discover the avatars and can interact with them through the services they offer. This centralized infrastructure is suited for objects that must be accessed through the Internet, and the objects that must be controlled and used within a private local area network. The peer-to-peer approach is instead more suited for local area networks, where devices can announce their presence thanks to discovery protocols such as UPNP, Bonjour or SLP. In this peer-to-peer approach, each avatar maintains in its own service registry. This registry contains the service provided by the avatar, and the services discovered in the network by this one. This service registry is implemented as a REST Web service, and as such it can be accessed by the users and the other avatars in order to discover the services provided by a given avatar.

5 Services

In project ASAWoO, services are pieces of code that are described by their functional and non-functional properties using RDF-based semantic notations. These semantic notations are used by the reasoner implemented in the avatar platform so as to decide which services can be deployed locally and which services are equivalents in terms of functionality. The semantic notation provides a more comprehensible and abstract description of the services than their software interface, and thus facilitate the reasoning process and the combination of services. In project ASAWoO, the services are implemented in a given programming language and offer a specific programming interface. For instance, in our Java/OSGi based implementation of the Avatar platform, the services are implemented in Java. They are described by their Java interface and registered as services in an OSGi middleware. Some services are moreover exposed as REST Web services. Indeed, some services must not be exploited directly by end-users or by other avatars, because they provide too basic functionalities. Moreover for security reasons, some services must not be exposed as REST Web services. In our Java/OSGi based implementation of the Avatar platform, services are exposed as REST Web services thanks to a dedicated OSGi service that implements the Distributed OSGi specification.

The semantic description of services can be achieved in several manners. In our Java/OSGi-based Avatar platform, developers of services can provide a description of their services using Java annotations. These annotations are used to separate the implementation of services from their description. They are also exploited at runtime to expose the functionalities offered by the avatars, and to generate a file including the semantic description of the service(s). An example of a semantic description of a temperature sensing service is given in Figure 4.

```
<owl:NamedIndividual rdf:about="&DATA;functionality.owl#ThermoSensor">
  <rdf:type rdf:resource="&DATA;functionality.owl#Functionality"/>
  <isImplementedBy rdf:resource="&DATA;functionality.owl#SimpleThermometer"/>
</owl:NamedIndividual>
```

Figure 4: Example of the RDF-based functionality description for a temperature sensor.

6 Service Deployment

Services are deployed on Avatar platforms using software packages. A software package includes pieces of code, additional resources that are needed by the services provided in this package (e.g., configuration files, database files), and a semantic description of the functional and non-functional properties of the services. A software package also includes a description file that defines the methods that must be called to start and stop the services provided in the software package. Typically, in or Java/OSGi-based implementation of the Avatar platform, the software packages are OSGi bundles. These bundles include a set of Java classes and additional files. The descriptor of a bundle defines the Java class –implementing the *BundleActivator* interface– that must be used to start and stop the bundle. This class is used to register and unregister the services provided in this bundle in the Avatar platform.

Software packages are downloaded by avatar platforms from remote package repositories, and are deployed locally. The deployment process relies on the semantic description of services. More precisely, the deployment manager implemented in the Avatar platform is designed to download from a package repository a specific file that contains the semantic descriptions of the services offered by the packages available on the repository. This manager provides this file to the *reasoner*, which in turn relies on the deployment manager to download from the remote repositories the package that includes the services it requires, and to start these services.

In our Java/OSGi-based Avatar platform implementation, we currently use an OSGi bundle repository coupled with a file that gathers the semantic descriptions of services, and that links these descriptions with the bundles.