The UniversAAL Platform

Alexander Kocian

Department of Computer Science
University of Pisa
Largo B. Pontecorvo 3
56127 Pisa

2014/2015
Table of Contents

1. Introduction
2. From Challenges to Solutions
3. OSGi component based platform
4. Middleware
5. Experimentation with the UniversAAL Platform
What is UniversAAL?\footnote{[1]}

In fact, UNIVERsal open platform and reference Specification for Ambient Assisted Living is a piece of software.

**Definition**

UniversAAL is an open-source software platform for AT where various, **heterogeneous** technical devices may be connected to a single, unified network.

**Alert**

The MS Windows and Apple MacOS platforms are only able to handle **homogeneous** technical devices.
The technical devices are either sensors or actuators or both.

- Sensors provide the system with information about the current state of the environment (so-called “contextual information”). Examples: pressure sensor, motion sensor, brightness sensor, camera, clock,…

- Actuators can be used by the system to influence the current state of the environment. Examples: heater, TV, electric window,…
The universAAL platform is called a Platform, because it is more than just a software layer that lies between operating system and the applications in a distributed computer network (aka “Middleware”)

- **Runtime Support** (Implementation of the Execution Environment)
- **Development Support** (a suite of SW tools for supporting the SW developer)
- **Community Support** (a suite of SW facilities and technical infrastructure to assist end users, service providers and developers in community-building)
The platform can logically be divided into various layers: **Middleware, Managers, Applications.**

**Figure: Layered Model**

---

Alexander Kocian (UniPisa)  The UniversAAL Platform  2014/2015
The Middleware Layer

- It needs to be available on every active node.
- Its task is to hide the distribution and heterogeneity of the nodes.
- Each communication bus (Context-Bus, Service-Bus, User-Interaction-Bus) handles a specific type of message.
The Middleware (cont’d)

- The **Context-Bus** is responsible for sharing context information, i.e. sharing knowledge that is used to dynamically adapt services from application to the user and vice versa\(^3\).

**Examples of context**

identity, location (geographical data), status (temperature, ambient illumination, noise level) and time\(^4\).

- The **Service-Bus** is responsible for sharing access to the service, i.e. sharing functionality.
- The **User-Interaction-Bus** is responsible for sharing information to active user interaction.
The Application Platform

The challenge - running applications on multiple heterogeneous devices.
The challenge - running applications on multiple heterogeneous devices.
The Application Platform

The challenge - running applications on multiple heterogeneous devices.
The challenge - running applications on multiple heterogeneous devices.
The Application Platform

The challenge - running applications on multiple heterogeneous devices.

- TV
- Radio
- Phone
- Oven
- Security camera
The challenge - running applications on multiple heterogeneous devices.
The challenge - running applications on multiple heterogeneous devices.
The challenge - running applications on multiple heterogeneous devices.
Heterogeneity of the devices

- Independent development and production of consumer items.
- Ability to exchange data depends on
  - Networking protocol (switching and routing)
  - Access protocol (synchronization, FEC)
  - Data representation (compression, encryption)
- Several application domains
- Several standards per application domain
- Several application profiles per standard
- What to do if all are relevant?
Middleware solutions

- For “AAL” components, a main protocol for networking & communication, optimally based on a single solution for data representation
- Integration of legacy components through adapters
  - Networking layer: protocol-specific gateways
  - Link and Presentation layers: component-specific wrappers
Challenges

Devices can come and go

Applications can come and go

It is not feasible to restart the platform for any change in a device/an application.

The platform and the application should auto-adapt to any change.

Alexander Kocian (UniPisa)
Challenges

Devices can come and go

- Mobile devices - smart phones, body sensors, portable audio players
From Challenges to Solutions

Challenges

Devices can come and go

- Mobile devices - smart phones, body sensors, portable audio players
- can be switched on and off
Challenges

Devices can come and go

- Mobile devices - smart phones, body sensors, portable audio players
- can be switched on and off
- can fail and be restarted
Challenges

Devices can come and go

- Mobile devices - smart phones, body sensors, portable audio players
- can be switched on and off
- can fail and be restarted

Applications can come and go
Challenges

Devices can come and go
- Mobile devices - smart phones, body sensors, portable audio players
- can be switched on and off
- can fail and be restarted

Applications can come and go
- can be installed, updated, uninstalled
Challenges

Devices can come and go
- Mobile devices - smart phones, body sensors, portable audio players
- can be switched on and off
- can fail and be restarted

Applications can come and go
- can be installed, updated, uninstalled
- can fail and be restarted
Challenges

Devices can come and go

- Mobile devices - smart phones, body sensors, portable audio players
- can be switched on and off
- can fail and be restarted

Applications can come and go

- can be installed, updated, uninstalled
- can fail and be restarted

It is **not feasible to restart** the platform for any change in a device/an application.
# Challenges

## Devices can come and go
- Mobile devices - smart phones, body sensors, portable audio players
- can be switched on and off
- can fail and be restarted

## Applications can come and go
- can be installed, updated, uninstalled
- can fail and be restarted

It is **not feasible to restart** the platform for any change in a device/an application. The platform and the application should auto-**adapt** to any change.
The Solution: Open Service Gateway initiative (OSGi)\cite{5}
OSGi component based platform

The Solution: Open Service Gateway initiative (OSGi)\cite{5}

- is a specification.
The Solution: Open Service Gateway initiative (OSGi)\textsuperscript{[5]}

- is a specification.
- The core of the spec defines a \textbf{component and service} model for Java ©.
The Solution: Open Service Gateway initiative (OSGi)[5]

- is a specification.
- The core of the spec defines a **component and service** model for Java ®.
- Components and services (i.e. Java interfaces) can be **dynamically** installed, started, stopped, updated and uninstalled **without restarting** the container.
The Solution: Open Service Gateway initiative (OSGi)\textsuperscript{[5]}

- is a specification.
- The core of the spec defines a component and service model for Java \textsuperscript{\textregistered}.
- Components and services (i.e. Java interfaces) can be dynamically installed, started, stopped, updated and uninstalled \textbf{without restarting the container}.
- OSGi has several implementations, such as Equinox, Knopflerfish OSGi or \textbf{Apache Felix}.
OSGi Bundles

- Services are packaged into bundles.
Services are packaged into bundles.

Bundles are a cohesive, self-contained units of functionality.
OSGi Bundles

- Services are packaged into bundles.
- Bundles are a cohesive, self-contained units of functionality.
- Technically, OSGi bundles are .jar files with additional meta information (images, libraries,...), stored in MANIFEST.MF file.
OSGi Bundles

- Services are packaged into bundles.
- Bundles are a cohesive, self-contained units of functionality.
- Technically, OSGi bundles are `.jar` files with additional meta information (images, libraries,...), stored in `MANIFEST.MF` file.
- Dependencies to other modules and services are explicitly defined via `MANIFEST.MF`.
OSGi Bundles

- Services are packaged into bundles.
- Bundles are a cohesive, self-contained units of functionality.
- Technically, OSGi bundles are .jar files with additional meta information (images, libraries,...), stored in MANIFEST.MF file.
- Dependencies to other modules and services are explicitly defined via MANIFEST.MF.
- Any non-OSGi runtime ignores the OSGi metadata.
OSGi Bundles

- Services are packaged into bundles.
- Bundles are a cohesive, self-contained units of functionality.
- Technically, OSGi bundles are .jar files with additional meta information (images, libraries,...), stored in MANIFEST.MF file.
- Dependencies to other modules and services are explicitly defined via MANIFEST.MF.
- Any non-OSGi runtime ignores the OSGi metadata.
- OSGi bundles have a life-cycle.
With `install <.jar>` in the OSGi runtime, the bundles are persisted in a local cache. A bundle ID is returned.

With `resolve`, bundle dependencies are resolved.

More bundles can be installed and resolved.

**Figure**: State Diagram of the Bundle life cycle
Next, start `<bundle id>`.

The bundle is now running i.e., in active state.

With `stop <bundle id>`, the bundle is still in the local bundle cache.

`uninstall <bundle id>`, to remove the bundle from the cache.

Figure: State Diagram of the Bundle life cycle
Complexity of Software

Figure: Complexity of SW$^6$
An OSGi Service is defined by a standard Java® class or interface.
A bundle can register and use OSGi services.
Another bundle can register and use OSGi services.

Figure: Pattern for service-oriented component model\[^7\]
A service is requested.

Figure: Pattern for service-oriented component model\textsuperscript{[7]}
If several services are valid for the same API, then OSGi chooses that with lowest service ID.

Figure: Pattern for service-oriented component model[7]
OSGi - a service oriented architecture

Figure: Pattern for service-oriented component model\(^7\)

- Service providers can be switched on the fly.
Interoperability Problem

- The Service Requester and all Service Providers have to agree a priori on **exactly** the same service interface.
- **Mismatch otherwise.**
Solution

Instead of directly connecting service provider with service interface, we apply reasoning using **ontology**.
Ontology in UniversAAL

**Definition**

- Ontology (from Greek: οντολογια) is the philosophical study of the nature of being.
- In computer science, an ontology is an “explicit specification of a conceptionalization” [8]. Simply, a model of the real world so that information in the model can be processed by computers.

**Purpose**

- Distribution of knowledge (Context Bus in uAAL)
- Sharing of functionalities (Service Bus in uAAL)
Two apps that share knowledge interpret info by ontology in **exact** the same way.

**Construction**

- Ontologies are made up of classes, properties, and data types.
- Every ontology has a uniform resource identifier URI.
<table>
<thead>
<tr>
<th>CLASSES</th>
<th>TYPES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROPERTIES</td>
</tr>
</tbody>
</table>
A Taste of Resource Description Framework (RDF)
A Taste of Resource Description Framework (RDF)
A Taste of Resource Description Framework (RDF)
A Taste of Resource Description Framework (RDF)

CLASSES

Light Source

PROPERTIES

TYPES
A Taste of Resource Description Framework (RDF)
A Taste of Resource Description Framework (RDF)

CLASSES

Light Source

PROPERTIES

Brightness

TYPES

Integer
A Taste of Resource Description Framework (RDF)

Diagram:
- Classes: Light Source
- Types: Integer
- Properties: Brightness

Example:
- LivingRoom
  - Brightness: 100

OSGi component based platform
RDF Statement

Definition

- An RDF statement is a **triple** (subject, predicate, object)
- All subjects of RDF statements are resources with **Unique Resource Identifier** (URI)

Example

http://ontology.universaal.org/Lighting.owl#LightSource
public class LightSource extends PhysicalThing {
    public static final String MY_URI = "http://ontology.persona.ima.igd.fhg.de/Lighting.owl#LightSource";
    public static final String PROP_AMBIENT_COVERAGE = "http://ontology.persona.ima.igd.fhg.de/Lighting.owl#ambientCoverage";
    public static final String PROP_HAS_TYPE = "http://ontology.persona.ima.igd.fhg.de/Lighting.owl#hasType";
    public static final String PROP_SOURCE_BRIGHTNESS = "http://ontology.persona.ima.igd.fhg.de/Lighting.owl#srcBrightness";
}
Non-OSGi devices

The Problem

- JVM does not exist on every device;
- OSGi-like module framework for C does not emulate Java® features (bytecode, classloading,...);
- ergo, OSGi cannot be installed on every device.

The Solution
Non-OSGi devices

The Problem

- JVM does not exist on every device;
- OSGi-like module framework for C does not emulate Java® features (bytecode, classloading,…);
- ergo, OSGi cannot be installed on every device.

The Solution

- Adapters
Non-OSGi devices (cont’d)

Sensors added as external nodes via adapters
- as other low-computational-power devices
- or devices without JVM
- or devices not supporting the inter-middleware protocols
Android™

- Operating system, Middleware, and application framework of Google®.
- Open-source
- Implementations on
  - Cellular phones
  - Netbooks
  - Tablets
  - TV sets
The UniversAAL middleware can directly be ported to Android™.
UniversAAL on any Device

Middleware

Applications

Middleware Middleware Middleware Middleware

OSGi OSGi Android™ Other

Applications
Scenario

- The client-app. makes a request.
- The Service Bus forwards the request to the server-app., and switches the requested light on.
- Real lights can be switched on/off with slight modifications.
Introductionary Example
The Lightning Example

Scenario

- The client-app. makes a request.
- The Service Bus forwards the request to the server-app., and switches the requested light on.
- Real lights can be switched on/off with slight modifications.
Preparation

1. Register at forge.universaal.org/wiki/support: RD_First_Steps

2. From the Project-tab, choose and join the groups Support and Ontologies;
Install Software

3. Apache SubVersion Client (SVN)

Figure: free SVN client at tortoisesvn.net;

4. Check-out from fully-recursive repository forge.universaal.orgsvn/support/;
Install Software (cont’d)

5. Java JDK6 (version!);
6. Eclipse (with reference to Java JDK6);
7. AAL Studio from http://depot.universAAL.org/eclipse-update
Import the Sources into Eclipse

8. Inside the Package Manager, Import: Maven: Existing Maven projects;
9. Our samples are smp.lighting.server.osgi and smp.lighting.client.osgi;
10. Keep all projects selected!
From the Package Explorer choose the two projects, and click on the hammer in AAL Studio;
Experimentation with the UniversAAL Platform

Run the Lighting Example

12 Select tab *Run:Run Configurations*;
13 Choose *Example-Lighting-LATEST_Complete*;
14 Run.


