The universAAL Primer

Basic

Introduction

The EU-funded project **universAAL** aims to produce an open platform that provides a standardized approach making it technically feasible and economically viable to develop AAL solutions. More information is available at http://universaal.org/.

This **Basic Primer Book** intends to give a brief and simple overview of the universAAL Execution Platform. Only the core concepts of **uAAL** are explained and these are accompanied by indicative diagrams and schemes.

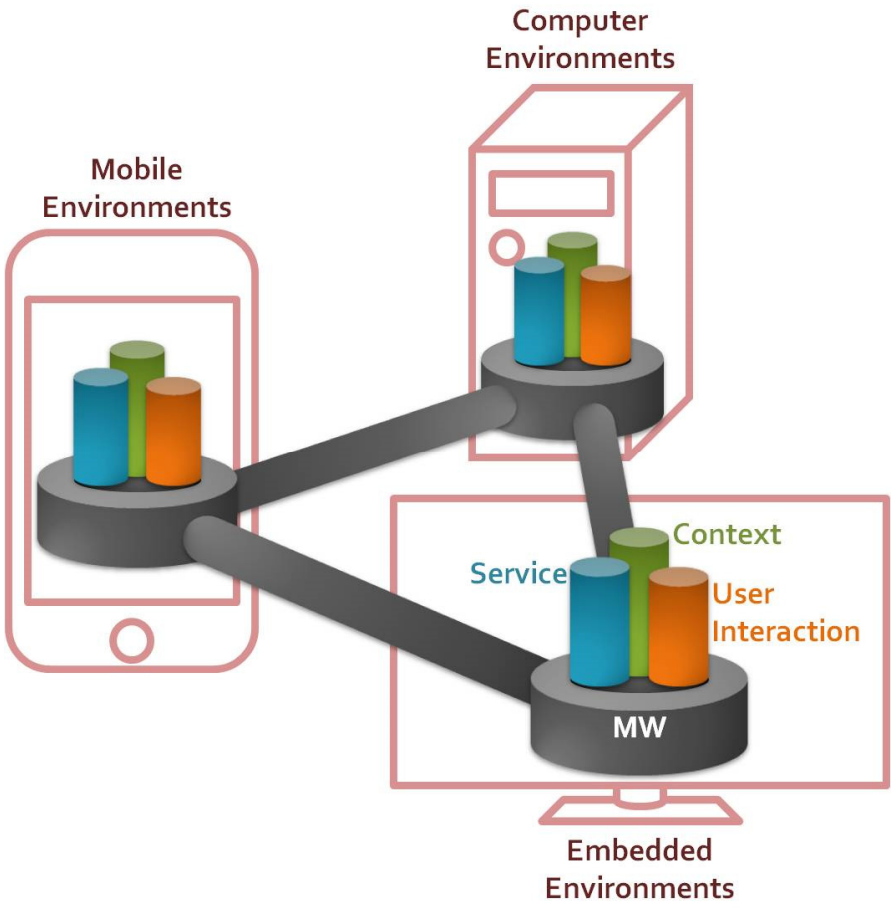
The targeted audience is anyone with some interest in the field of AAL, as the text tries to remain as less technically-dependent as possible. However some knowledge about ICT and programming is welcome, since some concepts are used without being explained that are commonly known to these domains.

Take into account that the content of this **Primer** does neither replace nor take precedence under any circumstance over the official contents developed in project **universAAL**, reported in the public deliverables available in http://universaal.org/es/about/deliverables.

The current version of this **Primer** is compatible with the release **2.0** of the **universAAL** Execution Platform. The software sources and binaries are available through http://forge.universaal.org/gf/.

Contents

Middleware



The Middleware delivers buses across multiple instances in different environments

The Middleware is the core part of uAAL platform and takes care that all uAAL nodes in a Space can cooperate one with each other. It establishes peer to peer communications between them so that they can share the different kinds of uAAL semantic communication: Context, Service and User Interaction, following the shared Ontological Model.

What the middleware is composed of

The **Container** is the part that lets the Middleware logic execute in different environments. There are different Containers so that the Middleware can run on devices with plain Java, computers or embedded systems running OSGi, or in Android smartphones, so far.

The **Peering** part is responsible for interconnecting and communicating the instances of the Middleware regardless of where they are running, using technologies such as jSLP and jGroups.

The **Communication** part is the one holding the ultimate logic of the Middleware that enables the flow of uAAL semantic information across peers, by defining specific-purposes Buses. These Buses is what applications connect to, and when they do so, they are in constant contact no matter the Device, Container or Peering technology they are running with. There is a bus for each type of communication (Context, Service and User Interaction), handling its own specific strategy, semantics, reasoning and match-making of participants.

Ontological Model

Knowledge is shared in uAAL in the form of Ontologies. It is its information model. Ontologies are a way to represent real-life information so it can be understood by computers. You can think of Ontologies as a network of concepts linked by properties. One tricky thing is that while we usually think in tree-view, Ontologies are meshes.

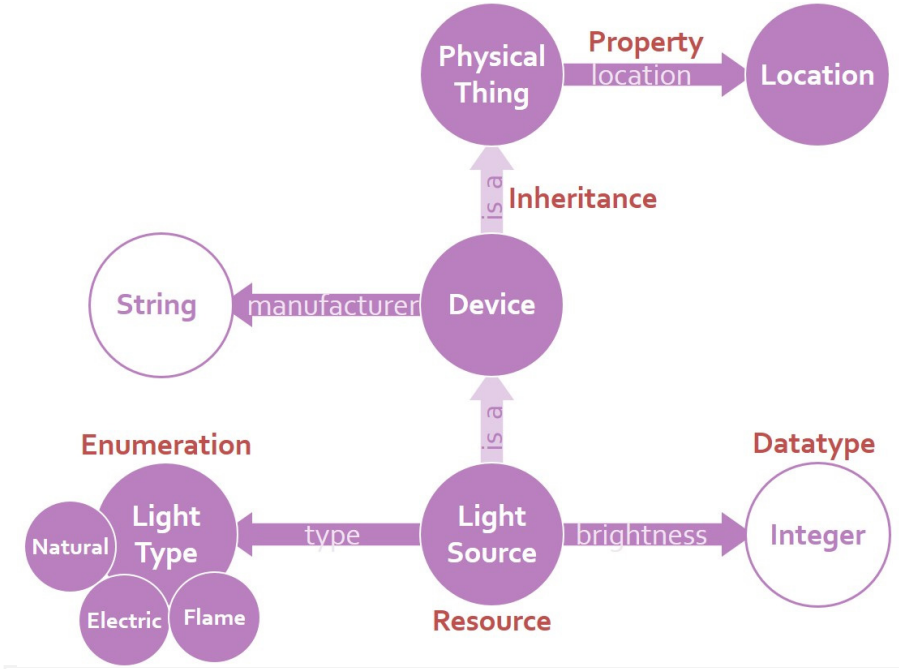
What ontologies are made of

Resources are how the concepts are represented. They are the nodes in the mesh. They are identified by a URI. They can inherit from other resources, and have properties that link to other resources or datatypes.

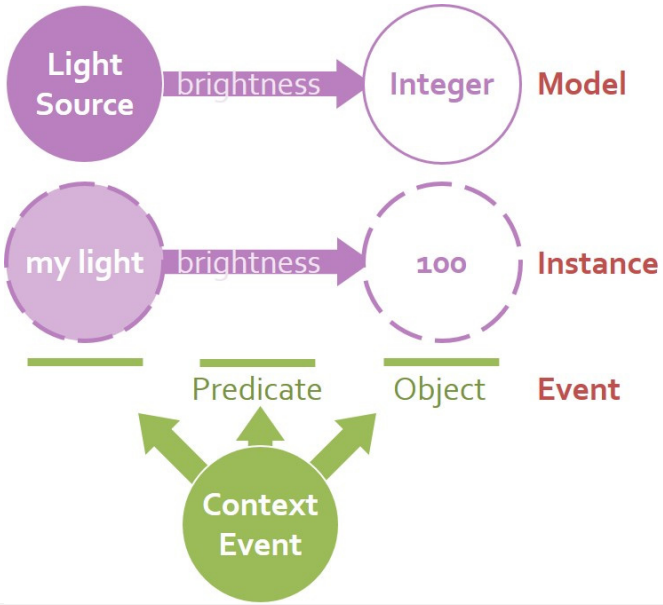
Properties are links between the concepts. They are also identified by URIs and can also inherit from other properties. They can have restrictions upon them, like cardinality.

Datatypes are the native data formats, like Boolean, Integer and so on. They are always present by default and don't have properties.

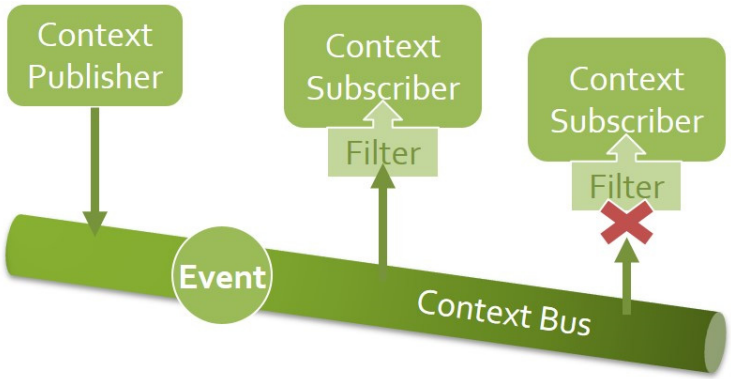
Enumerations are sets of instances of Resources, representing different specific values that a property can point to.



Ontologies link resources through properties



Context Events are built on the ontological model



Context Events are forwarded through the bus to the appropriate subscribers

Context

Context information represents the environment of the system, from physical surrounding, including users, to system information. Context communication is event-based, forwarding updates of the context. This is done in the form of Context Events, which are sent by Context Publishers, and can be consumed by Context Subscribers.

How context information is shared

Context Events are the minimal unit of context information sharing and are built on the Ontological model of uAAL. The minimal context information that can be extracted from an Ontological model is a link of two concepts through a property, modelled as a triple with subject, predicate and object. This is known as a statement. That is the structure of a Context Event, along with metadata.

Context Publishers are applications that are capable of sending Context Events. They build these events with the Ontological model and broadcast them.

Context Subscribers are any application interested in consuming Context Events. They define a filter to restrict which types of Events they are exactly interested in.

Service

Services are request/response interactions between applications. They are semantic, which means that you don´t need to call a service like "turn on a light by ID" (which you can) but you can ask to for services that "turn on all lights in a given location", without knowing them in advance. This is possible because services are described with the Ontological model.

How services are provided and called

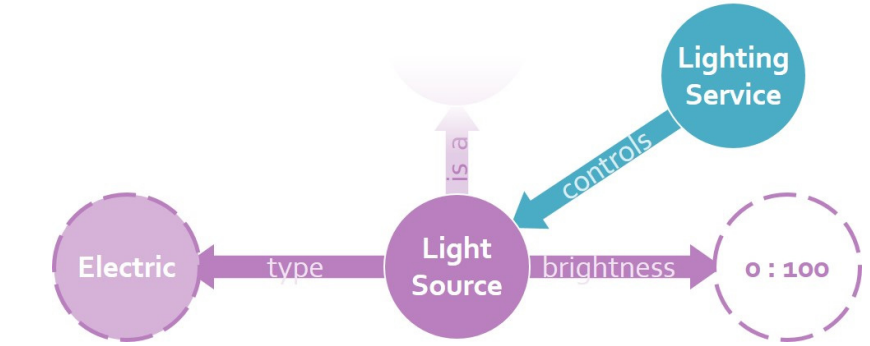
A **Service Ontology** is needed; it is the shared model between requester and provider. It works as an anchor to the Ontological model. It also allows restrictions over the original model to make services more specific.

Service Callees are those applications that provide services of certain Service Ontology. They do so by registering Service Profiles.

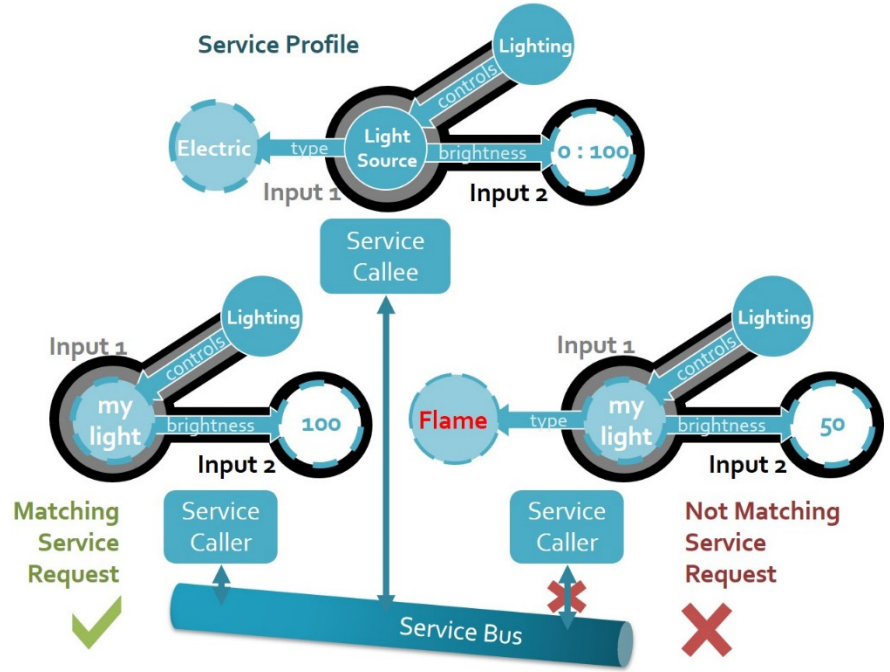
Service Profiles are the equivalent to methods. They represent the operation to perform. Starting at the Service Ontology they describe arguments as a Path to a concept on which an Effect is expected.

Service Callers are the applications that request the execution of a service. This is achieved by issuing Service Requests. The Requests are matched to registered Profiles and if they are ontologically equivalent, the Callee(s) that registered them will be called and will give an answer.

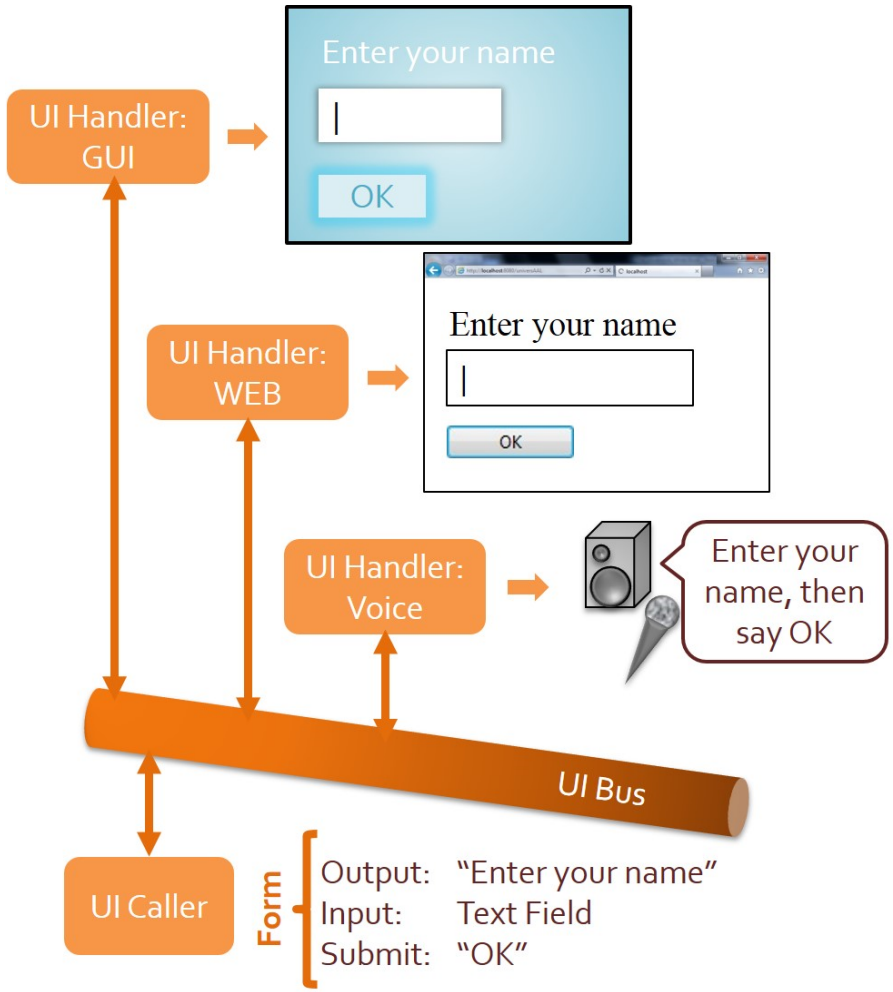
Service Requests are the counterpart of Service Profiles, built the same way but declare what the Caller wants to execute.



Service Ontologies are defined over the ontological model



Service Request´s arguments must match registered profiles to be delivered to callee



Forms can be rendered in many places and modalities, depending on the handler

User Interaction

Direct User Interaction is achieved by uAAL applications in a decoupled fashion. They do not handle how this information is presented to the user, only what is being presented (and handled in return). To do so the interaction information is abstracted by representing it with an Ontological model.

How the user interaction is handled

User Interaction Callers are the applications that want to have some kind of direct interaction with the user. They build a Form that represents exactly what they want to show to the user and what they want in return.

Forms are the ontological representation of the typical user interaction components, like textual inputs, multiple selections, buttons, and so on. Forms are created by UI Callers and sent to UI Handlers to be rendered, filled by user, and sent back to UI Callers to be processed.

User Interaction Handlers are special types of applications in charge of translating the Forms sent by UI Callers to a physical rendering that a user can interact with, such as a GUI, a sound output or Web page. Then interpret the user responses to fill in the information requested by UI Callers into the Form and send it back. There can be several UI Handlers in different locations, with different modalities, and the UI Callers are oblivious to them, thus achieving multi-modal and multi-location interaction.

Applications and Managers

A uAAL Application is the software part of an AAL Service, and is understood as a piece of software that communicates with others by making use of the uAAL Execution Platform. A Manager is an Application that is part of the platform itself and is necessary for its proper operation, or provides relevant basic services or events for other applications.

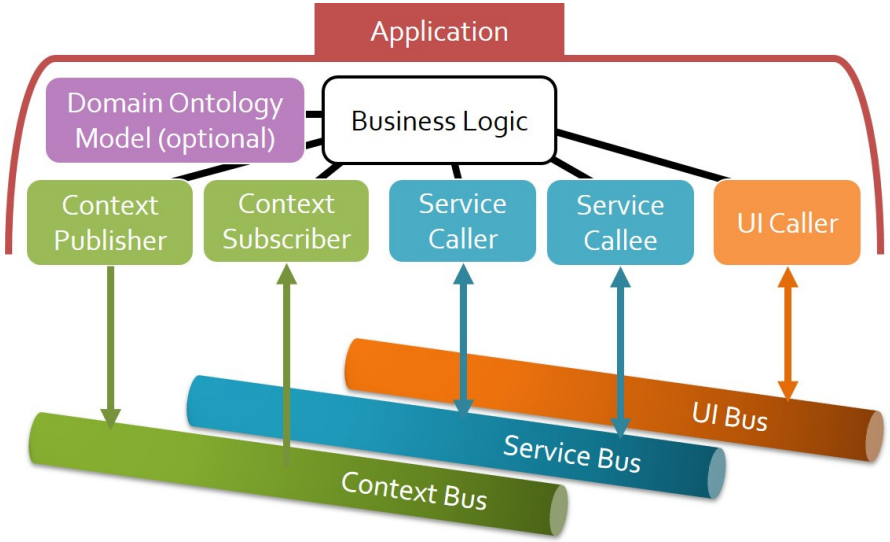
What parts an application is made of

An application, in addition to its own business logic, and regardless of its structure, needs one or more of the uAAL "wrappers" presented until now: Context Publisher, Context Subscriber, Service Caller, Service Callee and User Interaction Caller.

Each of this must be created at some point during the application execution, at which they will be connected to uAAL. When the application stops, these must be closed.

Because Ontologies are used as data model, the Application must make use of an Ontology describing its information domain. An application can define its own ontological model but it is strongly recommended that an existing implementation is reused (if it exists), for interoperability purposes.

Some Managers allow the creation of pseudo-Applications by scripting workflows or rules.



An **Application** running in the container connects to the buses using its own wrappers