SEVENTH FRAMEWORK PROGRAMME

SEVENTH FRAMEWORK PROGRAMME: PRIORITY 7.1B
LARGE SCALE INTEGRATING PROJECT (IP)

IP project number 247950
Project coordinator: Joe Gorman
Strategic Objective: 7.1.b

Project duration: February 2010 – February 2014
Project Coordinator Organisation: SINTEF, Norway
website: www.universaal.org

Universal Open Architecture and Platform for Ambient Assisted Living

Document Type: “Deliverable:” Item Appearing in ”List of Deliverables in DoW with delivery date shown in bold “Supplementary Report” As “Deliverable”, but delivery date *not* shown in bold. These documents are formally internal to the consortium, but can be delivered on request.	X	Project Deliverable with independent sub-parts. *Each sub-part forms a coherent whole in its own right, and has been edited and reviewed independently. The sub-parts are integrated in this document, to form the deliverable as a whole.*
		Project Deliverable (single document, no sub-parts).
		Sub-part of a Project Deliverable.

Document Identification			
Deliverable ID:	**D3.1-A (also providing D3.2-A, D3.3-A)**	Deliver-able title:	Tools for design and development (including *Transformation and generation tools*, and *Conformance tools*)
Release number/date:	V1.0 08.03.2011		
Checked and released by:	Sergio Guillén/ITACA		

Key Information from “Description of Work” (from the Contract)	
Deliverable Description	*D3.1: An integrated development environment providing UML design editors and traditional code editors adapted to niversal architecture and processes* *D3.2: A set of metamodels, transformation scripts and code generation scripts to be used in the tools from T3.1* *D3.3: A set of plug-ins and configurations that can be loaded into the tools from T3.1 to validate design and execute tests required for niversal conformance*
Dissemination Level	PU=Public
Deliverable Type	P = Prototype
Original due date (month number/date)	Month 9 / 31.Oct.2010

Authorship& Reviewer Information	
Editor (person/ partner):	Erlend Stav and Ståle Walderhaug / SINTEF
Partners contributing	SINTEF, CERTH, CNR-ISTI, Fh-IDG, FZI, ITACA-UPV
Reviewed by (person/ partner)	Juan Pablo Lázaro Ramos / TSB, Miran Mosmondor / ENT

Release History

Release number	Date issued	Milestone *	eRoom version	Release description /changes made
0.1	31.05.2010	PCOS proposed	V5	Initial version containing the structure an outline of content
0.2	24.06.2010	PCOS revised	V6	Changed according to the comments from the reviewer
0.3	11.10.2010	Intermediate proposed	V7	Updated "Structure of the Deliverable" to reflect that Part I is now a wiki, and to include links to software
0.4	04.01.2011	Intermediate revised	V8	Revised based on comments from intermediate review
0.5	29.01.2011		V9	Initial version of merge with D3.2-A and D3.3-A
0.6	02.02.2011	External proposed	V12	Added description of part IV and updated other descriptions
0.7	22.02.2011	External revised	V13	Updated figure and some small details in introduction and links based on comments from reviewers
0.8	04.03.2011	External approved	V14	Accepted all changes, turned off track changes, and updated status
1.0	08.03.2011	Released	V15	Technical Manager Release

* The project uses a multi-stage internal review and release process, with defined milestones. Milestone names include abbreviations/terms as follows:

PCOS	= "Planned Content and Structure" (describes planned contents of different sections)
Intermediate:	Document is approximately 50% complete – review checkpoint
External	For release to commission and reviewers;
proposed:	Document authors submit for internal review
revised:	Document authors produce new version in response to internal reviewer comments
approved:	Internal project reviewers accept the document
released:	Project Technical Manager/Coordinator release to Commission Services

universAAL Consortium

universAAL (Contract No. 247950) is an Large Scale Integrating Project *(IP)* within the 7th Framework Programme, Priority 7.1.b (ICT & Ageing). The consortium members are:

STIFTELSEN SINTEF (SINTEF, Project Coordinator) Contact persons: Joe Gorman Email: joe.gorman@sintef.no	**UNIVERSIDAD POLITECNICA DE VALENCIA** **(ITACA, Technical manager)** Contact person: Sergio Guillén Email: sguillen@itaca.upv.es
AUSTRIAN INSTITUTE OF TECHNOLOGY (AIT) Contact person: Sten Hanke Email: sten.hanke@ait.ac.at	**CONSIGLIO NAZIONALE DELLE RICERCHE (CNR-ISTI)** Contact person: Francesco Furfari Email: francesco.furfari@isti.cnr.it
CENTRE FOR RESEARCH AND TECHNOLOGY GREECE (CERTH) Contact person: Nicos Maglaveras Email: nicmag@med.auth.gr	**FRAUNHOFER-GESELLSCHAFT ZUR FOERDERUNG DER ANGEWANDTEN FORSCHUNG E.V (Fh-IGD)** Contact person: Saied Tazari Email: saied.tazari@igd.fraunhofer.de
ERICSSON NIKOLA TESLA (ENT) Contact person: Ivan Benc Email: ivan.benc@ericsson.com	**IBM ISRAEL – SCIENCE AND TECHNOLOGY LTD. (IBM)** Contact person: Yardena Peres Email: peres@il.ibm.com
FORSCHUNGSZENTRUM INFORMATIK AN DER UNIVERSITAET KARLSRUHE (FZI) Contact person: Andreas Schmidt Email: Andreas.Schmidt@fzi.de	**PHILIPS ELECTRONICS NEDERLAND B.V. (PHILIPS)** Contact person: Milan Petkovic Email: milan.petkovic@philips.com
IMPLEMENTAL SYSTEMS SL (IMPLEMENTAL) Contact person: Jordi Valles Email: jordi.valles@implementalsystems.com	**REGION SYDDANMARK (RSD)** Contact person: Casper Dahl Marcussen Email: cma@medcom.dk
PROSYST SOFTWARE GmbH (PROSYST) Contact person: Kai Hackbarth Email: k.hackbarth@prosyst.com	**TECHNICSHE UNIVERSITATET WIEN (TUW)** Contact person: Roman Obermeisser Email: romano@vmars.tuwien.ac.at
TSB SOLUCIONES TECNOLOGICAS (TSB) Contact person: Juan-Pablo Lázaro-Ramos Email: jplazaro@tsbtecnologias.es	**VDE VERBAND DER ELEKTROTECHNIK ELEKTRONIK INFORMATIONTECHNIK EV (DKE)** Contact person: Henriette Boos Email: henriette.boos@vde.com
UNIVERSIDAD POLITECNICA DE MADRID (UPM) Contact person: cvera@lst.tfo.upm.es Email: cvera@lst.tfo.upm.es	

1 About this deliverable

This deliverable provides tool support that facilitate software development base on the universAAL platform and reuse of universAAL components. The tools will enable developers to easily develop universAAL compliant AAL applications and reuse existing universAAL platform services that are shared within the developer community. This deliverable is primarily a software deliverable, but it also contains brief tool documentation for the tool users and tool design for tool developers (as described in the section *structure of the deliverable*).

2 Relationship to other universAAL deliverables

In addition to the content originally intended for D3.1-A, this deliverable also provides the results of task 3.2 and 3.3 that were originally intended for the separate deliverables D3.2-A and D3.3-A. During the initial work of these tasks, it was discovered the results from the tasks would be improved and better integrated if they were combined within a common structure and with shared introductions. Future versions of D3.1 will continue to provide the combined results from these tasks. Figure 1 gives an overview of the universAAL tools, show which task each tool is the result of. Tools within the border are reported in D3.1-A. Task 3.1 and 3.2 tools are included in all parts of this deliverable, while Task 3.3 tools are currently in an earlier stage of development and are primarily included in Part II.

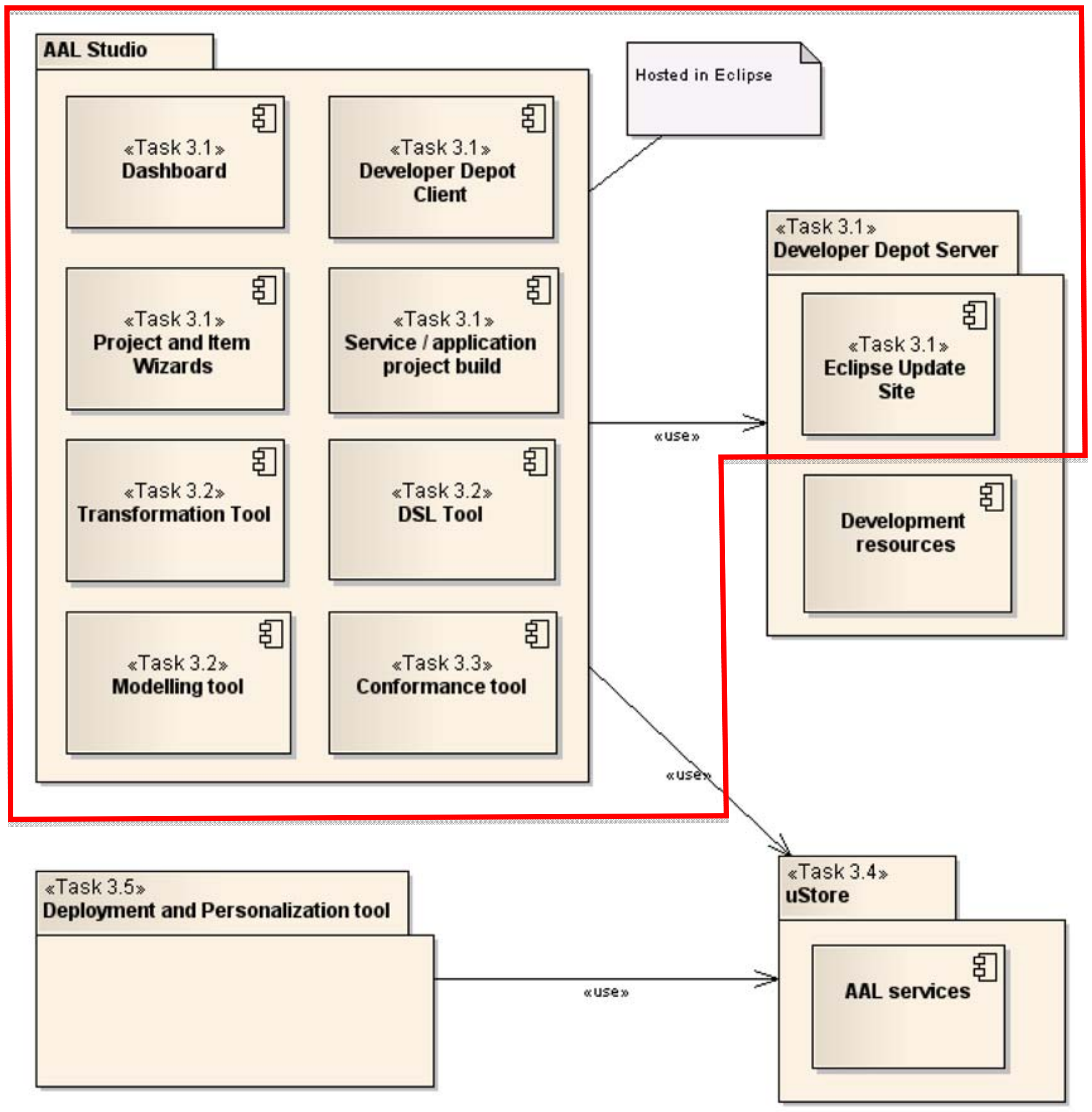

AAL Studio
«Task 3.1» Dashboard
«Task 3.1» Developer Depot Client
«Task 3.1» Project and Item Wizards
«Task 3.1» Service / application project build
«Task 3.2» Transformation Tool
«Task 3.2» DSL Tool
«Task 3.2» Modelling tool
«Task 3.3» Conformance tool
Hosted in Eclipse
«Task 3.1» Developer Depot Server
«Task 3.1» Eclipse Update Site
Development resources
«use»
«use»
«Task 3.4» uStore
AAL services
«Task 3.5» Deployment and Personalization tool
«use»

Figure 1 Overview of the universAAL tools

The architectural descriptions provided in the different versions of D1.3 define the reference architecture and the concrete architecture for the universAAL platform. These architectural descriptions will in upcoming versions include the top level tools described in this deliverable (including AAL Studio and Developer Depot), while the subdivision of the top-level tools are covered here. Currently, tool use cases and architectural overview are provided in Part II of this deliverable.

The runtime platform produced in deliverable D2.1 and D2.2 is a foundation for most of the tools in this deliverable. The tools are designed to support the developers in creating software based on this runtime platform.

Evaluation of the tools in this deliverable will be covered by deliverables D5.1 and D5.2.

3 Structure of the Deliverable

This deliverable is split into several independent sub-parts. Each of these was developed as a standalone item that is useful in its own right. The deliverable production process (authorship, editing, internal review) was applied independently to the various sub-parts. Where necessary, coordination activities at WP level were carried out to ensure consistency between the different sub-parts. The structure is as follows:

This intro: This document only provides this short overview of the full structure of the deliverable, with a short description of what is contained in each of the main parts.

Part I: Overview for tool users. This part gives an overview of the tools and the developer depot. It first provides a short architectural overview showing how the all the tools of WP3 (including uStore and Developer Depot) are combined in a tool-chain, and an overview of how to install the Eclipse-based tools. Each tool is then described, including its main functionality, installation guide, and release history. The main target audience for this part is users of the tools and depot – i.e. universAAL developers according to the stakeholder in D1.1-A. The part is provided as a wiki, available at: http://a1gforge.igd.fraunhofer.de/mediawiki/index.php?title=uaaltools:UniversAAL_D3.1

Part II: Report on the development work. This document reports on the background and process for the tool selection and development. It describes use cases and requirements for the tools, and gives an overview of the architecture for the tools. Further, brief design information is provided for tools we develop or do extensive modification or configuration of. The main target audience for this part are the tool developers.

Part III: The software. The software itself is the main part of this deliverable. The binary version of the developer tools is available as an Eclipse update site at:

http://a1gforge.igd.fraunhofer.de/eclipse-update/

The source code is available from:

http://a1gforge.igd.fraunhofer.de/gf/project/uaaltools/

Part IV: UML model. The architecture and design of the universAAL platform is recorded in a UML model that is shared across work packages and tasks. The model has been designed based on the ARCADE framework[1] [2], and the model directly reflects the ARCADE views and sub-models in its organization into UML packages. The architecture and design parts of the model for D3.1 are found in the following packages of the UML model:

[1] http://www.arcade-framework.org/

[2] The universAAL D1.1 deliverables gives an overview of the overall approach for how ARCADE is used in the project.

- Context view: Under “universAAL Platform use cases / Tools use cases”
- Requirements view: Under “Requirements Model / WP3 / 3.1”
- Component view: In the “Tools” packages under each of the main sub-packages of the view

The UML model is available from project Subversion repository at:

http://a1gforge.igd.fraunhofer.de/svn/uaalmodel

A periodically updated HTML documentation generated from the model gives is available at:

http://a1gforge.igd.fraunhofer.de/model/

Of the parts of this deliverable, only this “Intro” and “Part II” are delivered in the form of documents.

SEVENTH FRAMEWORK PROGRAMME: PRIORITY 7.1B LARGE SCALE INTEGRATING PROJECT (IP)	IP project number 247950 Project coordinator: Joe Gorman Strategic Objective: 7.1.b Project duration: February 2010 – February 2014 Project Coordinator Organisation: SINTEF, Norway website: www.universaal.org Universal Open Architecture and Platform for Ambient Assisted Living

Document Type:		
"Deliverable:" Item Appearing in "List of Deliverables in DoW with delivery date shown in bold "Supplementary Report" As "Deliverable", but delivery date *not* shown in bold. These documents are formally internal to the consortium, but can be delivered on request.		Project Deliverable with independent sub-parts. *Each sub-part forms a coherent whole in its own right, and has been edited and reviewed independently. The sub-parts are integrated in this document, to form the deliverable as a whole.*
		Project Deliverable (single document, no sub-parts).
	X	Sub-part of a Project Deliverable.

Document Identification			
Deliverable ID:	**D3.1-A (also providing D3.2-A, D3.3-A)**	Part title:	Part I – Overview for Tool Users
Release number/date:	V1.0 08.03.2011		
Checked and released by:	Sergio Guillén/ITACA		

Key Information from "Description of Work" (from the Contract)	
Deliverable Description	*D3.1: An integrated development environment providing UML design editors and traditional code editors adapted to universAAL architecture and processes* *D3.2: A set of metamodels, transformation scripts and code generation scripts to be used in the tools from T3.1* *D3.3: A set of plug-ins and configurations that can be loaded into the tools from T3.1 to validate design and execute tests required for universAAL conformance*
Dissemination Level	PU=Public
Deliverable Type	P = Prototype
Original due date (month number/date)	Month 9 / 31.Oct.2010

Authorship& Reviewer Information	
Editor (person/ partner):	Erlend Stav and Ståle Walderhaug / SINTEF
Partners contributing	SINTEF, CERTH, CNR-ISTI, Fh-IDG, FZI, ITACA-UPV
Reviewed by (person/ partner)	Juan Pablo Lazaro Ramos / TSB, Miran Mosmondor / ENT

NOTE TO THE READER

"This part is provided as a wiki, available at: http://a1gforge.igd.fraunhofer.de/mediawiki/index.php?title=uaaltools:UniversAAL_D3.1.

For the convenience of the reader, the main Part I content is made available in this document generated from the wiki, but please note that overview and formatting is degraded in the transfer from wiki to printable document."

Table of Contents

Table of Contents

1 uaaltools:Overview of the universAAL toolchain

1.1 Toolchain / architecture

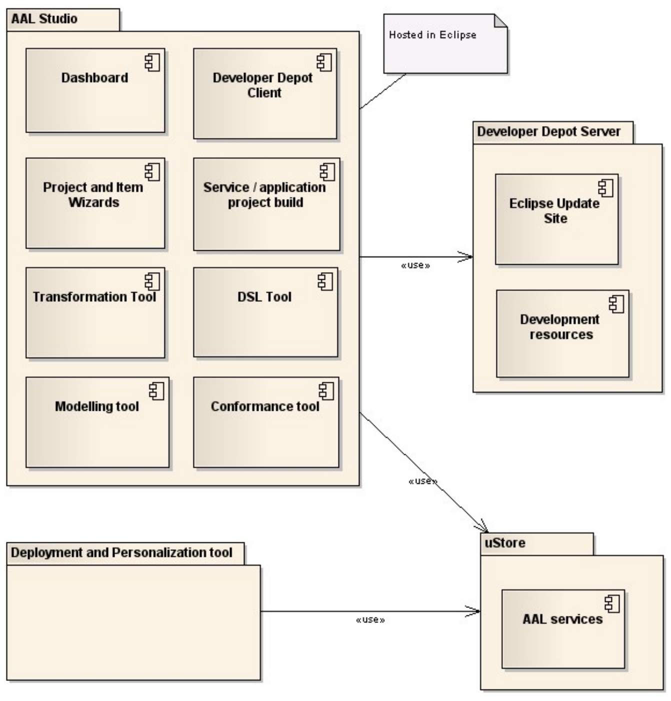
The universAAL toolchain include a set of tools for developers, service providers, deployers, and end-users of AAL services.

All resources a **developer** need to get started developing AAL applications and plug-ins for the universAAL platform will be made available in the Developer Depot. Through the Depot the developer can find execution platform, development tools, reference architectures, and guidelines which simplify and streamlines the development. The client side part of the developer tools are organized in the AAL Studio: a set of development tools hosted in an Eclipse-based IDE that provides support for different parts of the development process, and that gives the developer easy access to the resources of the Developer Depot.

The uStore gives the **end user** (elderly users or their care providers) a simple way to find and acquire AAL services. An AAL Service can include not only AAL application (software) and hardware, but also human resources. By acquiring an AAL service, required software (applications and device drivers) will be deployed to the user?s hardware, access will be provided to required remote software services, and agreements will be made with (local) **services providers** to reserve required human resources both for deployment and use of the service. If necessary, new hardware will be ordered and installed in the end user?s residence. While in some cases the end user or their care providers can download and install the services themselves, in more complex cases (typically involving hardware installation) a professional **deployer** will be involved in the installation (either guiding the installation remotely or at the installation site). The uStore also enables the developers and service providers to offer their combined service.

To personalize the deployment to the needs and the environment of the end user, the deployer or end user will use the Deployment and Personalization tool.

All the tools mentioned above, are currently in **early development**. The figure below provides an overview of the tools being developed in universAAL, grouping them into an Eclipse-hosted AAL studio, the Developer Depot, the uStore, and the Deployment and Personalization Tool.



Please note that the figure above is an initial overview, and that new tools may be added later while other tools may be modified or dropped as the development progresses. The current list is our best guess for the first iteration of the tools. Also, more detailed descriptions will be provided later of what the inputs and outputs of each tool are.

For the initial release of the tools, AAL Studio plugins have reached a maturity where the tools provide useful functionality to the developer: a Project and Item Wizard allows quick setup of a development project that uses the universAAL middleware, while the Service/Application Project Build tool simplifies building of projects. These tools access resources on the Developer Depot as part of the build process. On the server side, an initial Developer Depot Server has been set up based on a Nexus Repository, and this server provides Maven access to the libraries of the initial execution platform. The other tool is the Eclipse Update Site which packages the tools that are plugins for AAL Studio, making it easy for the developer to install these tools into the Eclipse environment.

At a later stage we will also add a description of aspects of the development process that are not covered by the tools and what the developer will have to take care of more manually.

2 uaaltools:Developer Depot Server

2.1 Role and benefits of this tool

The Developer Depot Server will host resources needed for a developer to get started developing using universAAL. This section provides a basic understanding for the Developer Depot Server actually planned as part of universAAL tool chain. The Depot aims at providing a powerful archive for developers embracing the universAAL platform. With respect to the current project phase, the role of depot is to provide the common repository functionalities such as upload, download, search, removal of deployment units. The developer will interact with the Depot either accessing the server directly or by using tools in the AAL Studio that interact with the Depot.

We choose to adopt Sonatype Maven Nexus Repository [1] open-source edition as initial Developer Depot Server implementation, it provides some essential features described at the following URLs: [2] and [3]

In order to setup an efficient resource repository without the adoption of a repository manager (like Nexus), almost two services should be configured:

1. artifact upload service, e.g. SSH or SFTP server
2. artifact download service, e.g. HTTP server

The configuration of these services not only increases the knowledge requirements for the system administrator, but also the setup and housekeeping cost of the repository itself. The upload artifact service raises some critical security issues not completely solved with the adoption of SSH or SFTP secure servers, moreover some crucial features have to be provided. As an example: managing the permissions, cluster the permissions into some defined roles, add or remove users from role, authenticate users against an external server (e.g LDAP). All these aspects can be avoided with the adoption of a repository manager.

Developer Depot Server is mainly addressed to AAL application developers, and will provide resources needed to get stated with development of AAL applications.

2.2 Overview of functionality

The Developer Depot Server provides hosting for the universAAL middleware and other useful resources for AAL application developers. More generally an AAL application is composed of several resources bundled together as artifacts (e.g. maven artifacts). Developer Depot functionality includes:

- artifact management: upload, download, removal, search
- repository management: create and set up multiple repository instances
- ease the integration with HTTP standard interface or Maven tool

2.3 Installation guide

This section provides information for application developers in order to start interacting with Developer Depot Server. The Depot can be accessed by browsing the following URL: [4].

2.3.1 Prerequisites and Dependencies

Actually some approaches can be adopted in order to interact with the Depot server:

1. Nexus Web interface: No prerequisites to use the web-based interface beyond a javascript-enabled browser

2. Developer Depot Client: please refer to the following URL for more details about the installation procedure and the prerequisites: [5]
3. Maven tool: Maven can be used in order to download or upload artifacts. Maven tool needs to be installed and properly configured, please refer to the official page at the following URL: [6].

2.3.2 Installation Procedure

N/A

2.4 Administration guide

This section provides more details about Nexus installation guide and about common repository operations. For more information about Nexus usage, please refer to the official Nexus documentation [7]:

2.4.1 Depot Server Prerequisites and Dependencies

Nexus Open Source distribution requires the Java Runtime Environment (JRE) compatible with Java 5 or higher. JRE package can be chose between one of the available implementations. The following list reports some of the most popular JDK releases:

- Sun [8]
- IBM [9]
- Kaffe [10]

2.4.2 Depot Server installation procedure

Nexus Open Source edition can be download at the following URL: [11]. The quickest approach for Nexus installation is to download the Nexus bundle version composed of the Nexus web application and an open source up-and-running web server: Jetty [12]. Once downloaded, perform the following steps:

1. unarchive the zip or tar.gz bundle into a preselected folder:

```
$unzip nexus-oss-webapp-1.x.y-bundle.zip
```

or

```
$tar xvzf nexus-oss-webapp-1.x.y-bundle.tgz
```

then start-up Nexus by running the architecture dependent lunch script. The list of scripts is located at the following path:

```
<nexus_home>/bin/jsw
```

where it is possible to find the following directory structures:

```
aix-ppc-32/          linux-ppc-64/         solaris-sparc-32/
aix-ppc-64/          linux-x86-32/         solaris-sparc-64/
hpux-parisc-32/      linux-x86-64/         solaris-x86-32/
hpux-parisc-64/      macosx-universal-32/  windows-x86-32/
```

Once located the correct architecture directory name, type the following two commands in order to assign the script execution permission and to start the bundled web server:

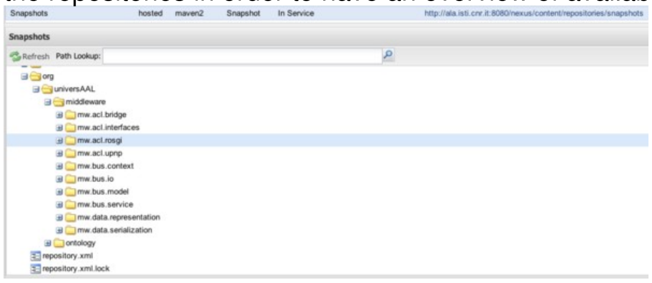
```
$chmod -R a+x bin
$./bin/jsw/<name_of_architecture_directory>/nexus start
```

It is however possible to download the WAR Nexus bundled version from the following URL: [13]. The WAR archive has been successfully tested on Tomcat, Jetty, Glassfish and Resin servlet containers. The installation procedure depends on the selected container.

2.4.3 Common repository operations

- browsing the repository

As previously described Nexus manages several artifact repositories, it becomes crucial to be able to browse the repositories in order to have an overview of available artifacts. Figure 2 depicts the repository view.



Snapshots	hosted	maven2	Snapshot	In Service	http://ala.isti.cnr.it:8080/nexus/content/repositories/snapshots

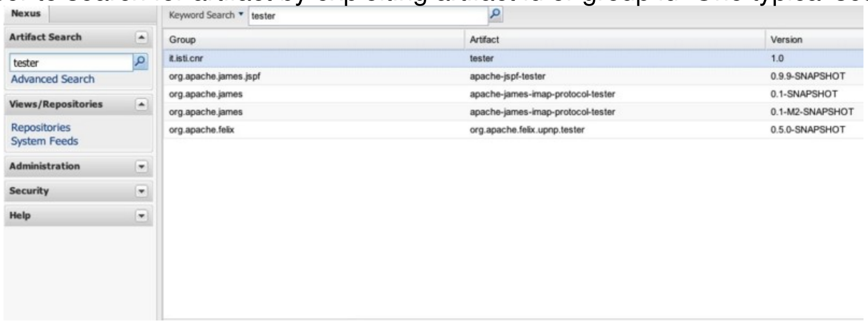
Snapshots

Refresh Path Lookup:

```
org
  universAAL
    middleware
      mw.acl.bridge
      mw.acl.interfaces
      mw.acl.rosgi
      mw.acl.upnp
      mw.bus.context
      mw.bus.io
      mw.bus.model
      mw.bus.service
      mw.data.representation
      mw.data.serialization
    ontology
repository.xml
repository.xml.lock
```

- searching for artifacts

The search feature allows user to search for artifact by exploiting artifact id or group id. One typical search result is depicted in figure 3.



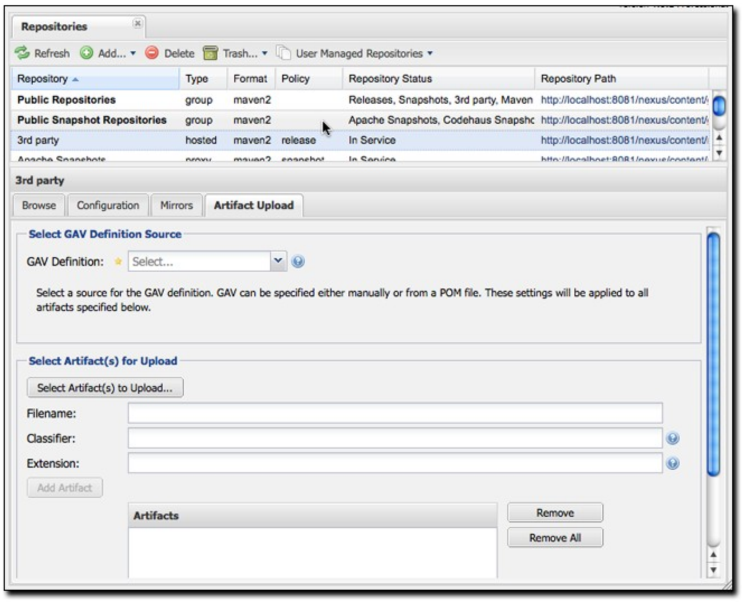
Keyword Search: tester

Artifact Search: tester

Group	Artifact	Version
it.isti.cnr	tester	1.0
org.apache.james.jspf	apache-jspf-tester	0.9.9-SNAPSHOT
org.apache.james	apache-james-imap-protocol-tester	0.1-SNAPSHOT
org.apache.james	apache-james-imap-protocol-tester	0.1-M2-SNAPSHOT
org.apache.felix	org.apache.felix.upnp.tester	0.5.0-SNAPSHOT

- uploading artifacts

Nexus allows users to upload custom artifacts through Web user interface. The upload can be accomplished with the POM file or the GAV coordinated (artifact GroupID, artifactID or versionID), the first approach only requires the artifact POM file, while the second one requires user to specify all the artifact coordinates. Figure 4 reports an example of artifact upload routine.



Repository	Type	Format	Policy	Repository Status	Repository Path
Public Repositories	group	maven2		Releases, Snapshots, 3rd party, Maven	http://localhost:8081/nexus/content/
Public Snapshot Repositories	group	maven2		Apache Snapshots, Codehaus Snapshc	http://localhost:8081/nexus/content/
3rd party	hosted	maven2	release	In Service	http://localhost:8081/nexus/content/

GAV Definition: Select...

Select a source for the GAV definition. GAV can be specified either manually or from a POM file. These settings will be applied to all artifacts specified below.

Filename:

Classifier:

Extension:

2.5 Release history

2.5.1 Release for D3.1-A

New features implemented	Developer Depot overview, brief description of functionalities and user guide
Limitations and problems fixed	This is the first release, and thus no limitations and problems from previous releases are addressed.
Known remaining limitations and problems	Evaluate Depot efficiency and maintenance activities
Features not yet implemented	Actually the standard Maven repository features have been provided. Next depot releases will evaluate the adoption of CI supporting tools in order to provide brother functionalities.

3 uaaltools:Eclipse Update Site

3.1 Role of this tool

3.1.1 Benefits

Update sites are the standard way of packaging and providing extensions to the Eclipse environment, and is well known by developers using Eclipse.

Using update sites will be the preferred way of downloading and installing the AAL Studio tools (i.e., the Eclipse based developer tools of universAAL).

3.1.2 Relation to reference architecture

N/A

3.1.3 Relation to Eclipse

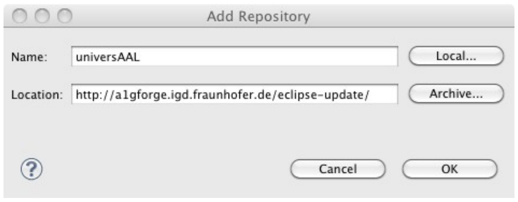
The Eclipse platform provides built-in support for installing and updating software from update sites.

3.2 Overview of functionality

The Eclipse Update Site for universAAL packages the Eclipse based tools of universAAL for easy downloading and installation from Eclipse. The update site can be accessed from the built-in functionality of Eclipse.

The update site is available at: http://a1gforge.igd.fraunhofer.de/eclipse-update/

To install the universAAL tools, add the address of the update site to the list of update sites in Eclipse. In version 3.6 of Eclipse, this can be done by first selecting "Install New Software..." under the "Help" menu. In the dialog box that appears, select "Add..." at the top right, and fill in the dialog box that appears.


Add Repository

Name: universAAL

Location: http://a1gforge.igd.fraunhofer.de/eclipse-update/

Future versions of the universAAL update site may be co-located with the Developer Depot.

3.3 Installation guide

The necessary software to install from update sites are already included in Eclipse.

Eclipse itself can be downloaded from: http://www.eclipse.org/downloads/

3.3.1 Prerequisites and dependencies

N/A

3.3.2 Tool installation procedure

N/A

3.4 Release history

3.4.1 Release for D3.1-A

New features implemented	Initial version of update site
Limitations and problems fixed	This is the first release, and thus no limitations and problems from previous releases are addressed.
Known remaining limitations and problems	None
Features not yet implemented	The update site will be updated to include the latest version of all tools for each release

4 uaaltools:AAL Studio overview and installation

4.1 Role and benefits of this tool

The AAL Studio provides an integrated development environment based on Eclipse for building applications and components using the universAAL execution platform. The AAL Studio will make it easier to get started with the development, and will make some of the development tasks more efficient. Also, it will give easy access to the resource needed by the developer.

The client side developer tools created by universAAL are implemented as Eclipse plug-ins and provide integration with other AAL Studio tools. Thus, while development of universAAL compliant applications and components do not require a specific development Java development environment, use of the Eclipse-based AAL Studio is recommended because it gives access to using the provided tools.

4.2 Overview of functionality

The functionality of the AAL Studio is provided by the individual tools that are installed within it, including wizards for creating projects, build tools for simplifying building and launching of applications, and modeling and transformation tools for making the development more efficient.

For further details of the functionality, see the description on the wiki page for each tool.

4.3 Installation guide

4.3.1 Prerequisites and dependencies

The AAL Studio is based on Eclipse, so the primary prerequisite for installing it is to have a compatible Eclipse version installed. The version we currently recommend is Eclipse 3.5 Modeling Tools, which can be downloaded from http://www.eclipse.org/downloads/packages/release/galileo/sr2.

After installing Eclipse, you will also need some prerequisite Eclipse plugins before installing the universAAL tools. These are listed in detail under each of the tools. A quick summary of what is needed to install the full set of tools:

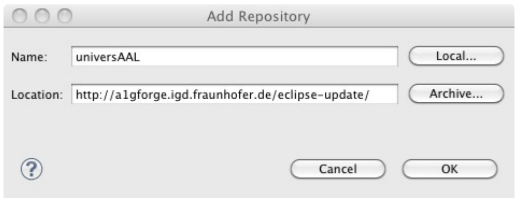
- m2Eclipse maven support from: http://m2eclipse.sonatype.org/sites/m2e/0.10.2.20100623-1649/
  This tool is required by both the wizard and build tools of the AAL Studio
- Pax Runner for Eclipse: http://www.ops4j.org/pax/eclipse/update
  This tool is required by the build tool of the AAL Studio in order to support launching of the application.

4.3.2 Tool installation procedure

The Eclipse Update Site for universAAL packages the AAL Studio tools for easy downloading and installation from Eclipse. The update site can be accessed from the built-in functionality of Eclipse. The update site is available at: http://a1gforge.igd.fraunhofer.de/eclipse-update/

Note that we are intending to move to Eclipse 3.6 Modeling Tools at a lager stage.

To install the universAAL tools, add the address of the update site to the list of update sites in Eclipse. In version 3.6 of Eclipse, this can be done by first selecting "Install New Software..." under the "Help" menu. In the dialog box that appears, select "Add..." at the top right, and fill in the dialog box that appears. Installation for Eclipse 3.5 is similar.



Name: universAAL

Location: http://a1gforge.igd.fraunhofer.de/eclipse-update/

4.4 Release History

The release history is described under the wiki page for each individual AAL Studio tool.

5 uaaltools:Project and Item Wizards

5.1 Role and benefits of this tool

This AAL Studio tool is intended to be used by developers of services and platform components. It makes it easy to crate new universAAL-compliant projects by providing a skeleton project with all the files you need and initial content to make the project work in universAAL. The item wizards generate new files required or optional to a universAAL project with the proper formatting and template or initial content.

It reduces the time of development since without this tool a developer would need to give the project or files the appropriate format to be universAAL-compliant, with the risk of missing some requirement. By using the wizards it is assured to have well-formatted files and project structures.

The usage of this tool is not mandatory but recommended. The wizard currently provide setup and configuration options which are useful for typical development projects. Future versions may add further options that also simplify more advanced setup and configuration.

5.2 Overview of functionality

The main access to Project and Item Wizards is through the File/New command in Eclipse. Then the proper wizard must be selected. An Eclipse command is available to start the wizard from any menu or other plugin.

These plug-ins provide wizards for creating universAAL resources in the Eclipse workspace. They can be either items or new projects. With ?items? we refer to new files or other resources inside an existing project in the workspace, such a new XML file that could be necessary for a universAAL project. These new files would be generated with some initial content in order to be properly formatted, as a template, or configured as defined in the parameters passed in the wizard.

On the other hand, the Project wizard would generate a blank new universAAL compliant project in the workspace, ready to develop universAAL applications or components upon. It could be initially configured as described during the wizard, that is, preset configuration parameters, storage location, initial files? The wizard will take care of generating the project hierarchy and structure, and all the mandatory files for the project to be universAAL compliant, as well as any additional file that could be defined by the user in the wizard.

This "New Project" plugin does always the same: creating a new project in the workspace. Other future wizards will create specific items on existing projects and therefore their commands could also be placed in specific contextual menus.

5.3 Installation guide

5.3.1 Prerequisites and dependencies

No dependencies on other universAAL plugins. However it has dependencies on Maven plugin for Eclipse, which is properly notified, requested and installed during installation procedure. However if it is the plug-in *source code* what is being imported into Eclipse Plug-in Development, instead of the working plug-in itself, the maven plug-in will have to be manually installed to make it available for the code to compile.

5.3.2 Tool installation procedure

To install the tool, install it as usual with the "Install new software..." menu for plugins of Eclipse. Only required extra intallation is Maven plugin, as stated above, which is automatic.

To import and compile the source code, the Maven plugin must be downloaded manually. To do so, go to "Install new software..." and introduce the Maven download site: http://m2eclipse.sonatype.org/sites/m2e. Then select to download the Maven Integration for Eclipse (at least version 0.9.0)

5.4 Release history

5.4.1 Release for D3.1-A

New features implemented	Creat a new PERSONA-compliant blank template project. Set the initial description of the project. Select the initial premade wrapper classes included. Defines command for invoking the wizard.
Limitations and problems fixed	This is the first release, and thus no limitations and problems from previous releases are addressed.
Known remaining limitations and problems	No problems detected or reported until now.
Features not yet implemented	Wizards for creating new Items (project-specific classes and files) in selected projects, folders or packages

6 uaaltools:Dashboard

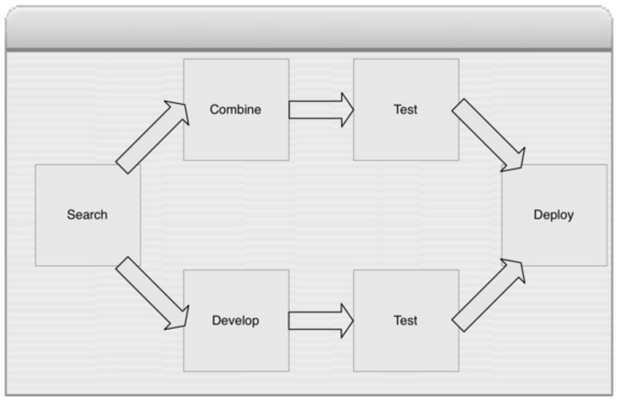
6.1 Role and benefits of this tool

The AAL Studio Dashboard is developed to streamline the process of developing universAAL applications, services and components. The underlying idea is to give developers the correct paths through the development process.

Using the universAAL dashboard ensures that developers have access to all the relevant components through an interface with which they are familiar.

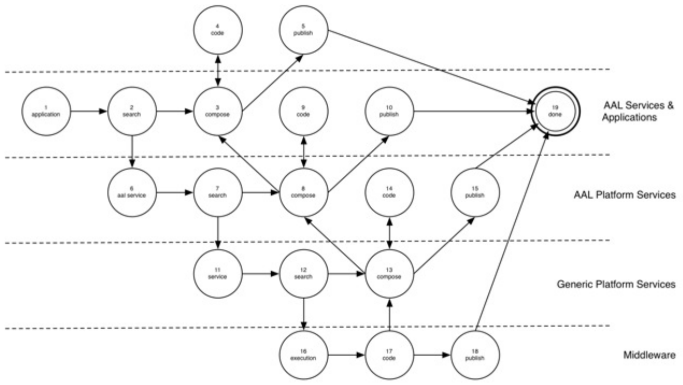
6.2 Overview of functionality

The universAAL dashbord is provided as a view in Ecplise. The figure below depicts a sketch example of the universAAL dashboard.



The figure above demonstrates how a developer will see the different steps in the process of developing universAAL components. Each box corresponds to a task that must be carried out, and the arrows correspond to the transitions between these tasks.

The dashboard give process support for the developers. The dashboard itself is modelled as a finite state machine, where each state corresponds to a task, and traditions corresponds to the completion of a task. The figure below depicts the state diagram for the initial version of the dashboard.



For each of the four layers several universAAL components are made available:

- AAL Services & Applications
  - ♦ Utilises existing use-cases from the developer depot
  - ♦ Allows for integration of existing components from both the developer depot and uStore
- AAL Platform Services
  - ♦ Utilises existing services on this layer
  - ♦ Allows for integration of existing components from both the developer depot and uStore
- Generic Platform Services
  - ♦ Utilises existing services on this layer
  - ♦ Allows for integration of existing components from both the developer depot and uStore
- Middleware
  - ♦ Allows for the development of new middleware componets

All of the above development tasks allows for publishing of finished services, applications and components to either developer depot, uStore, or both.

6.3 Installation guide

6.3.1 Prerequisites and dependencies

To fully utilize this tool, all other AAL Studio plug-ins should be installed. An updated list of prerequisites will be provided with the first binary release of this tool.

6.3.2 Tool installation procedure

A binary release of this tool has not yet been released. In the future the tool will be installable from the universAAL Eclipse update site.

6.4 Release history

6.4.1 Release for D3.1-A

New features implemented	Design of state machine
Limitations and problems fixed	This is the first release, and thus no limitations and problems from previous releases are addressed.
Known remaining limitations and problems	Not yet packaged as Eclipse feature for binary release in Eclipse update site. Whether or not to allow the state machine to have other starting states in other layers.
Features not yet implemented	Planned for next release: Mock-up of the dashboard, including functional components for a limited sub-set. Packaging as Eclipse feature and initial binary release.

7 application project build

7.1 Role and benefits of this tool

This AAL Studio tool is intended for universAAL service developers. It automates the process of building a compatible universAAL project, in order to be integrated to uStore, by utilizing Maven arctifacts and Maven build mechanism.

Project dependencies are resolved automatically in order to reduce incompatibility issues, since projects are build using universAAL compatible libraries. It also provides automatic uploading of implemented artifacts to local and remote repositories.

This tool is highly recommended, since manual build and upload can lead to incompatible to universAAL services.

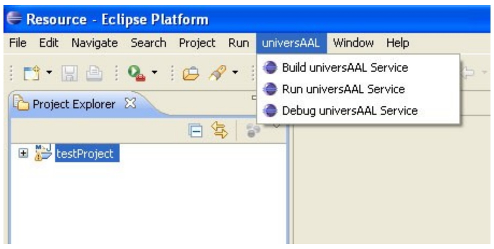
7.2 Overview of functionality

This plugin provides a safe way for building and running a universAAL service/application within Eclipse workspace.

Having based on the prototype skeleton projects by the Project and Items Wizards Tool, the developer implements his application according to universAAL standards. This tool is responsible for building the whole developed Eclipse project using the appropriate libraries and dependencies for making a new universAAL compatible service/application executable. By building such projects, the resulting jar files are deployed to the local or universAAL Maven repositories, in order to be published or reused by other projects/providers. Uploading to remote repositories (uStore) is not implemented yet.

Finally, this tool also has the capability to run or debug a new developed universAAL project. This is done by automatically creating an Eclipse launch configuration file that starts up all the universAAL middleware bundles that needed in order to run the new service/application.

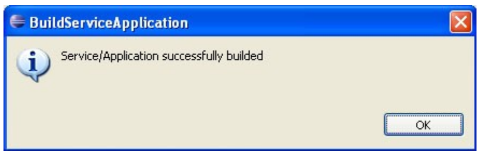
Service/application developer should select the project that wants to build in the Eclipse workspace in order that the universAAL build, run and debug options become accessible, as in the following figure:



Then, the following output should appear after a successful build in the Eclipse console:

```
[INFO]
[INFO] --- maven-bundle-plugin:2.1.0:bundle (default-bundle) @ testProject ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.578s
[INFO] Finished at: Tue Dec 21 15:55:36 EET 2010
[INFO] Final Memory: 22M/104M
[INFO] ------------------------------------------------------------------------
```

along with the following pop-up window


BuildServiceApplication

Service/Application successfully builded

Then, the created jar is uploaded automatically to the local repository along with a pom file containing all project dependencies.

When running or debugging a service/application, the Pax-runner plugin will start all the necessary middleware bundles in order to run/debug the service/application. The developer can view all the started bundles by typing

```
ps
```

in the Eclipse console window. The output of this command should look like:

```
START LEVEL 8
   ID   State         Level  Name
[   0] [Starting   ] [    0] System Bundle (1.4.0)
[   1] [Installed  ] [    8] Apache Felix Bundle Repository (1.4.2)
[   2] [Installed  ] [    6] Unnamed - testProject:testProject:bundle:1.0 (1.0.0)
[   3] [Installed  ] [    5] The PERSONA Context Ontology (0.2.0.SNAPSHOT)
[   4] [Installed  ] [    4] PERSONA Turtle serializer (0.3.0.SNAPSHOT)
[   5] [Installed  ] [    4] PERSONA middleware (0.3.0.SNAPSHOT)
[   6] [Installed  ] [    3] OPS4J Pax ConfMan - Properties Loader (0.2.2)
[   7] [Installed  ] [    3] ACL UPnP (0.2.0.SNAPSHOT)
[   8] [Installed  ] [    3] OPS4J Pax Logging - Service (1.4)
[   9] [Installed  ] [    3] Soda-Pop as OSGi Bundle (0.3.0.SNAPSHOT)
[  10] [Active     ] [    2] OPS4J Pax Logging - API (1.4)
[  11] [Active     ] [    2] wrap_mvn_jp.go.ipa_jgcl_1.0 (0)
[  12] [Active     ] [    2] wrap_mvn_java3d_vecmath_1.3.1 (0)
[  13] [Active     ] [    2] wrap_mvn_org.bouncycastle_jce.jdk13_144 (0)
[  14] [Active     ] [    2] ACL Interfaces (0.3.0.SNAPSHOT)
[  15] [Active     ] [    2] wrap_mvn_java3d_j3d-core_1.3.1 (0)
[  16] [Active     ] [    2] Apache Felix Configuration Admin Service (1.2.8)
[  17] [Starting   ] [    2] Apache Felix UPnP Base Driver (0.8.0)
[  18] [Resolved   ] [    2] wrap_mvn_org.osgi_osgi_R4_compendium_1.0 (0)
[  19] [Installed  ] [    2] Apache Felix Log Service (0.9.0.SNAPSHOT)
[  20] [Active     ] [    1] Apache Felix Shell Service (1.0.2)
[  21] [Active     ] [    1] Apache Felix Shell TUI (1.0.2)
```

where the bundle with ID 2 is the bundle that has been just uploaded. In order to start running it, the developer should enter the following command:

```
start 6
```

where 6 is the default bundle level for new developed project.

7.3 Installation guide

7.3.1 Prerequisites and dependencies

This tools has been developed and tested using Eclipse 3.5.0. Moreover, the m2eclipse plugin is needed in order to run the plugin, and more specifically the 0.10.2.20100623-1649 version (update site: http://m2eclipse.sonatype.org/sites/m2e/0.10.2.20100623-1649/). For running the middleware bundles, the Pax Runner plugin has been utilized (update site: http://www.ops4j.org/pax/eclipse/update).

7.3.2 Tool installation procedure

The installation of this plugin is performed through the "Install new Software" option of the "Help" menu of Eclipse IDE. The universAAL eclipse update site should then be inserted (http://a1gforge.igd.fraunhofer.de/eclipse-update) in order to select the Service/Application build plugin.

7.4 Release history

7.4.1 Release for D3.1-A

New features implemented	Building of a new Maven project. Uploading of the resulting artifact to local Maven repository. Automatic creation of Eclipse launch configuration file for running Persona services. Run/debug of created Persona services.
Limitations and problems fixed	This is the first release, and thus no limitations and problems from previous releases are addressed.
Known remaining limitations and problems	Run/Debug of the launch configuration is not compatible with Eclipse 3.6 Helios release due to some incompatibility with Pax Runner.
Features not yet implemented	Uploading of artifacts to the universAAL remote repository (uStore)

8 uaaltools:Developer Depot Client

8.1 NOTE: Tool in inception phase

This tool is currently in its inception phase. The descriptions below is currently only placeholders for structure and some initial keywords for content, and we do not currently commit to provide this tool or any specific functionality. Further description may be provided in the next release of the AAL studio.

8.2 Role and benefits of this tool

The main idea for this tool is to make it easier for the developer to access and use resources from the Developer Depot Server directly from the AAL Studio environment.

The tool will not be mandatory, but will simplify tasks of the developer, and make information more readily available for inclusion in the development work compared to acquiring the same resources directly from the Developer Depot Server by other means.

8.3 Overview of functionality

The developer depot client will be an AAL Studio tool that makes the functionality and resources of the universAAL Developer Depot available from Eclipse. Possible functionality include assistance for finding and reusing resources associated with typical use-cases in the domain, and utilizing the reference architecture as a resource in the development.

8.4 Installation guide

8.4.1 Prerequisites and dependencies

Eclipse version
Maven plug-in
Subversion plug-in?

8.4.2 Tool installation procedure

Install of the tool will be through the universAAL Eclipse update site The tool may need to be configured with the developers user name and password to the Developer Depot Server.

8.5 Release History

8.5.1 Release for D3.1-A

New features implemented	No implementation is currently available
Limitations and problems fixed	N/A
Known remaining limitations and problems	N/A
Features not yet implemented	N/A

9 uaaltools:Domain Specific Language Tools

9.1 Role and benefits of this tool

Designing software for the AAL domain is a daunting task. There is a plethora of stakeholders, systems, platforms, concerns and processes must be understood and dealt with. Using architecture modeling and ontologies are useful mechanisms in handling the complexity as it raises the level of abstraction and defines a common concept vocabulary. When it comes to actual software development, the use of domain specific (modeling) languages DS(M)L has an increasing popularity in the software engineering community. A DS(M)L makes it easier for the designer/developer to consistently express software system designs using a standardized domain specific modeling elements.

This section describes the relationship between the elements of the domain specific language(s) in universAAL and the initial requirements and usescases. The design rationale behind the DSL is important in order to be able to correctly apply the DSL in design and development.

The main idea behind domain languages is to provide a (definition) language that incorporate domain specific concepts. This language enables domain experts, architects and developers to more clearly define the target system's characteristics. At the same time, the implicit (domain) information in the language can be utilized by model and text transformation tools to automatize certain steps in the development process

Domain languages (and domain language engineering) is a well-known discipline that are supported by many different tools. However, when it comes to modeling tools that support domain languages there are mainly two approaches that are used:

1. Meta language frameworks such as MetaEdit from MetaCase, MOF based tools used to design metamodels, and Eclipse-based (EMF/GMF) tools that have a metamodel in EMF and editor tool based on GMF
2. UML (which is a MOF language) based tools extended with UML Profiles (also supported by Eclipse). Almost all UML tools support UML profiles.

Approach 1 restricts the existing (meta) language (model) by defining a subset that have its own rules and structures. Approach 2 extends the existing UML language (a MOF-based language) by adding a new "layer" that can be applied and un-applied to a UML model. All features of UML is preserved in the new "language".

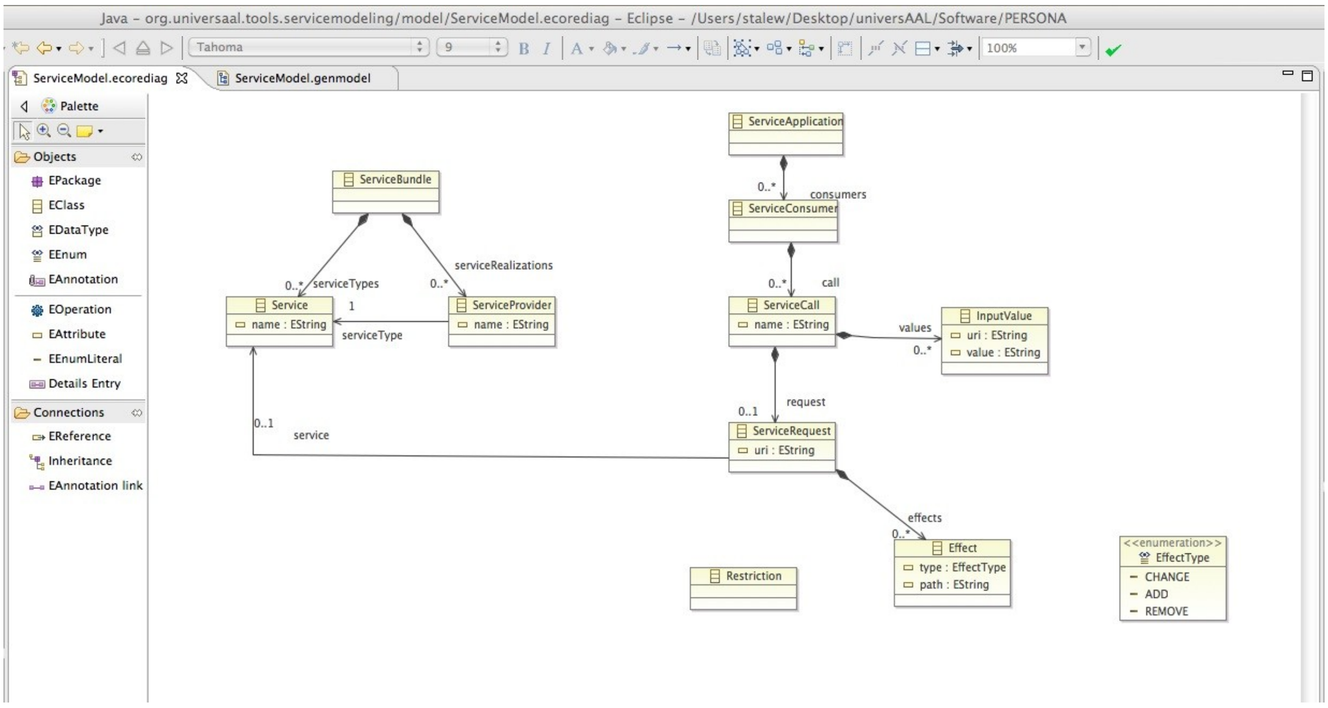
The AAL Studio from universAAL will apply a combination of the two approaches to realize the desired functionality. The approaches have different strengths and weaknesses that must be considered in each tool module.

9.2 Overview of functionality

The purpose of using DSLs in AAL Studio is to assist the development and (re)use of AAL Services. The DSL elements represent AAL Specific constructs in a way that makes it easier for developers to select, modify and compose services and service components. In this way, the development process will be improved both with respect to resources/time required and indirectly service quality as a more precise design language should make more precise designs. The latter is to be evaluated in universAAL WP5.

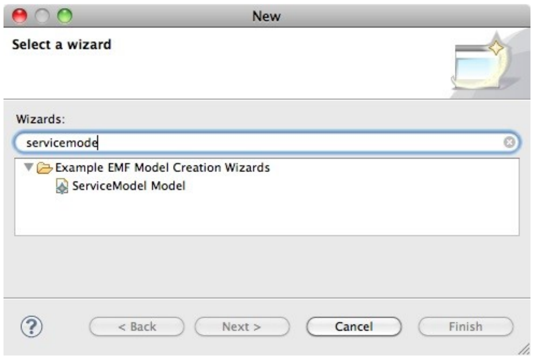
The current status is that no complete DSL has yet been defined. However, a simple DSL for Service Provider and Service Consumer is drafted. The process of creating DSLs for the AAL Studio is described in Part II of this deliverable.

This diagram shows the metamodel for the Service DSL in AAL Studio.

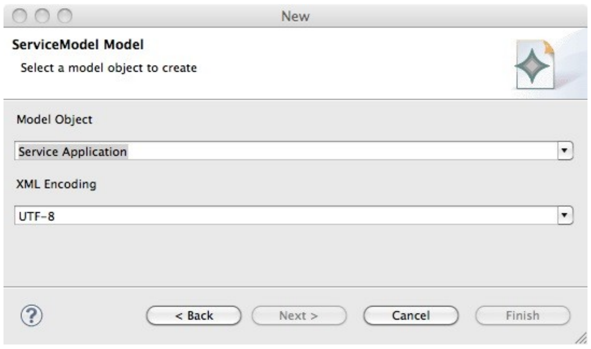


The metamodel defines main classes, their attributes and relationships. A service design in AAL Studio must adhere to these structures.

When a developer wants to create a new service design (application(consumer) or servicebundle (provider)) he/she will use the Eclipse->new file dialog as shown below



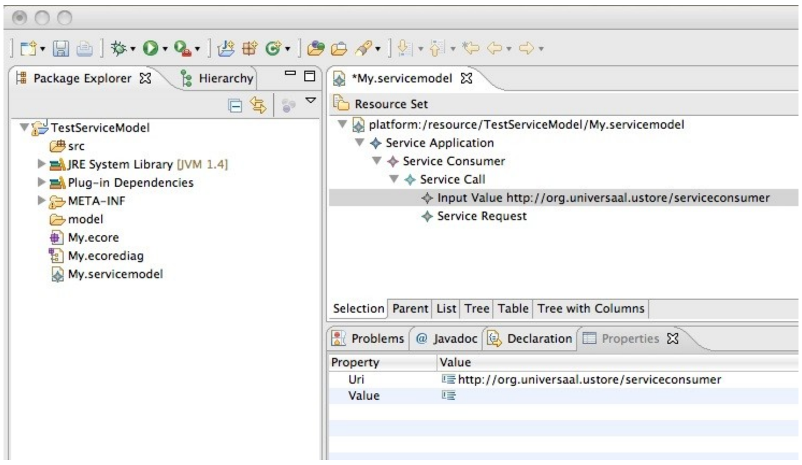
The final step is to select the main class (application or servicebundle) as shown below



Model Object: Service Application

XML Encoding: UTF-8

Depending on which main class the developer selects, the design can be created from context menus in Eclipse.The diagram below shows a tree-structure for a service consumer based on the Service DSL in AAL Studio.



Property	Value
Uri	http://org.universaal.ustore/serviceconsumer
Value	

9.3 Installation guide

9.3.1 Prerequisites and dependencies

The current version of the plugin has been tested on Eclipse version 3.5 and 3.6 The plugin requires EMF and GMF plugins. It is recommended to install the Eclipse Modeling Tools that comes with EMF/GMF preconfigured.

9.3.2 Tool installation procedure

The DSL Tool can be installed from the universAAL Eclipse update site. [Details about the update site here]

9.4 Release History

9.4.1 Release for D3.1-A (also providing D3.2-A, D3.3-A)

New features implemented	Run plugin as application Create a new service model from service metamodel. Select main class Add subclasses and set properties
Limitations and problems fixed	This is the first release, and thus no limitations and problems from previous releases are addressed.
Known remaining limitations and problems	No problems detected or reported until now.
Features not yet implemented	Model checking and export/transformation to java

10 uaaltools:Modelling Tool

10.1 Role and benefits of this tool

The AAL Studio Modeling Tool allows developers to design AAL services using UML. The Modeling Tool provides a set of UML profiles that extends the generic UML language with AAL and service specific language elements. These extensions are provided in the form of stereotypes (a meta-type name), tagged values (attributes on the stereotypes) and constraints (rules, pre/post conditions and invariants, about the elements and their relationships).

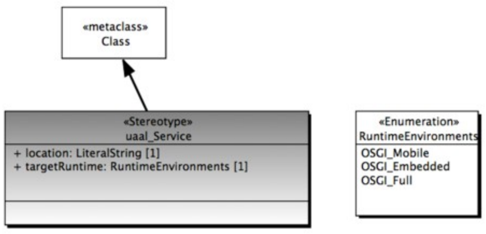
Using a UML based design language makes it possible to create service designs that:

1. Use a standardized domain vocabulary
2. Adhere to best practice domain design
3. Can be transferred to other UML tools (along with the required UML profiles=
4. Is easily communicated with other developers/designers that understand UML

The role of the Modeling Tool in AAL Studio is provide AAL service designers with a UML based editor that assists in creating correct, complete and confine services designs that will work on the uAAL Platform.

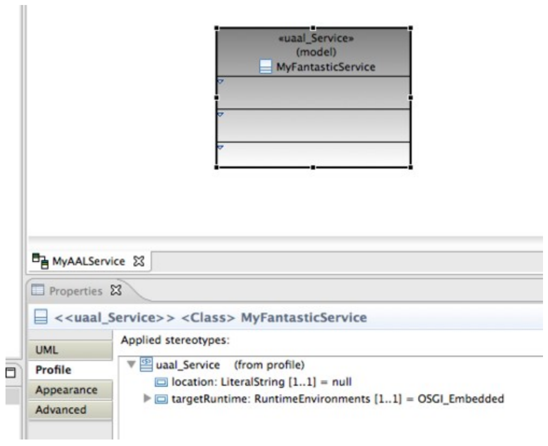
10.2 Overview of functionality

The profile allows for stereotyping service designs with universAAL specific tags. The stereotypes, tagged values and constraints are defined in a UML profile model as shown in the figure below.



The diagram shows a stereotype that extends the UML:Class meta-class. The stereotype has 2 attributes that will appear as "Tagged Values". The tagged value "RuntimeEnvironment" is of type Enumeration RuntimeEnvironments. This enumeration is defined in the profile and contains three enumeration literals for OSGI. In modeling-time, the modelere can set this variable as a meta-information element to the class that is stereotyped uaal_Service.

Applying the profile modelling-time is straight forward. Once the profile is loaded into the model, stereotypes can be assigned to the model elements. The diagram below shows a class MyFantasticService that has been stereotyped uaal_Service. In the properties window one can see and set the tagged values.



<<uaal_Service>> <Class> MyFantasticService

Applied stereotypes:

uaal_Service (from profile)
location: LiteralString [1..1] = null
targetRuntime: RuntimeEnvironments [1..1] = OSGI_Embedded

10.3 Installation guide

10.3.1 Prerequisites and dependencies

This plugin requires Eclipse Helios 3.6 with Eclipse Modelling Tools. Although 3.5 can be extended with Papyrus plugins the namespaces have changed when integrating Papyrus into the Eclipse 3.6 release.

10.3.2 Tool installation procedure

The Modelling Tool can be installed from the universAAL Eclipse update site. [Details about the update site here]

10.4 Release History

10.4.1 Release for D3.2-A

New features implemented	• Added simple stereotype for uaal_Service. • Added enumeration for RuntimeEnvironment
Limitations and problems fixed	This is the first release, and thus no limitations and problems from previous releases are addressed.
Known remaining limitations and problems	No constraints added
Features not yet implemented	Will start building a language that will result in the new profile.

11 uaaltools:Transformations in universAAL

11.1 Role and benefits of this tool

The universAAL AAL Studio provides many features that utilize transformations. Either as a model-to-model (m2m) or model-to-text (m2t) functions. Transformations will be made available wherever it can simplify and / or improve the process of creating software. These transformation mechanisms are applied for a number of functions in the AAL Studio.

The role of the transformation tool is to simplify and speed up the development of standard development artefacts such as WSDL, configuration files and code skeletons. Using transformations, the resulting artefact will adhere to a code convention defined in the transformation script which again will improve the interoperability of these artefacts on the target platform. Furthermore, maintenance of tedious text artefacts such as WSDL files can be improved using a graphical modeling editor, compared to a pure text editor.

The transformation tool can be used explicitly by the designer (invoked from the tool's menu) or run as a background process as a part for a development process. The GMF dashboard concept is a good example of a developer tool where most transformations are hidden for the developer, but have an important role in the process.

11.2 Overview of functionality

The transformation tool is provided as a command plugin where the developer can transform models into more specific models or text.

11.2.1 Model-to-model transformations (M2M) in AAL Studio

When M2M is applied in the AAL Studio, its application and relationship to the UAAL requirements will be described here in the following structure:

Transformation tool name	Tranformation Command
Transformation purpose	This version will load and execute a pre-programmed transformation script on a user-selected uml file
Input artefacts	A valid UML file with extension .uml
Output artefacts	Model file in XMI/eCore/UML format. Depends on the script.
Script	No M2M script available in this version
Tutorials	N/A

11.2.2 Model-to-text transformations (M2T) in AAL Studio

When M2T is applied in the AAL Studio, its application and relationship to the UAAL requirements will be described here in the following structure:

Transformation tool name	Transformation command
Transformation purpose	To generate text output from a design (UML) model
Input artefacts	

	This version will load and execute a pre-programmed transformation script on a user-selected uml file
Output artefacts	This version will list all the classes in the UML file in the console (only as a demonstration)
Script	A MOFScript that list the classes of a UML model file
Tutorials	Modelbased website [1]

11.3 Installation guide

11.3.1 Prerequisites and dependencies

Eclipse version 3.6 or later.

11.3.2 Tool installation procedure

Install from Eclipse update site

Include info on how to install upgrades and on uninstall if relevant

11.4 Release History

11.4.1 Release for D3.1-A

New features implemented	Added simple class listing for UML models
Limitations and problems fixed	Current script is for demonstration purposes
Known remaining limitations and problems	Current script is for demonstration purposes
Features not yet implemented	Model-to-model transformation not provided Harmonization and integration with UAAL development process (dashboard)

SEVENTH FRAMEWORK PROGRAMME: PRIORITY 7.1B LARGE SCALE INTEGRATING PROJECT (IP)	IP project number 247950 Project coordinator: Joe Gorman Strategic Objective: 7.1.b	Project duration: February 2010 – February 2014 Project Coordinator Organisation: SINTEF, Norway website: www.universaal.org
	Universal Open Architecture and Platform for Ambient Assisted Living	

Document Type: "Deliverable:" Item Appearing in "List of Deliverables in DoW with delivery date shown in bold "Supplementary Report" As "Deliverable", but delivery date *not* shown in bold. These documents are formally internal to the consortium, but can be delivered on request.		Project Deliverable with independent sub-parts. *Each sub-part forms a coherent whole in its own right, and has been edited and reviewed independently. The sub-parts are integrated in this document, to form the deliverable as a whole.*
		Project Deliverable (single document, no sub-parts).
	X	Sub-part of a Project Deliverable.

Document Identification			
Deliverable ID:	**D3.1-A (also providing D3.2-A, D3.3-A)**	Part title:	Part II – Report on the development work
Release number/date:		V1.0 08.03.2011	
Checked and released by:		Sergio Guillén/ITACA	

Key Information from "Description of Work" (from the Contract)	
Deliverable Description	*D3.1: An integrated development environment providing UML design editors and traditional code editors adapted to 1niversal architecture and processes* *D3.2: A set of metamodels, transformation scripts and code generation scripts to be used in the tools from T3.1* *D3.3: A set of plug-ins and configurations that can be loaded into the tools from T3.1 to validate design and execute tests required for 1niversal conformance*
Dissemination Level	PU=Public
Deliverable Type	P = Prototype
Original due date (month number/date)	Month 9 / 31.Oct.2010

Authorship& Reviewer Information	
Editor (person/ partner):	Erlend Stav and Ståle Walderhaug / SINTEF
Partners contributing	SINTEF, CERTH, CNR-ISTI, Fh-IDG, FZI, ITACA-UPV
Reviewed by (person/ partner)	Juan Pablo Lazaro Ramos / TSB, Miran Mosmondor / ENT

Release History

Release number	Date issued	Milestone*	eRoom version	Release description /changes made
0.1	31.05.2010	PCOS proposed	V1	Initial version containing the structure an outline of content
0.2	24.06.2010	PCOS revised	V2	Changed according to the comments from the reviewer
0.3	11.10.2010	Intermediate proposed	V6	Added initial partner contributions including tool use cases, project and item wizard, Eclipse, Update Site, and Dashboard
0.4	04.01.2011	Intermediate revised	V7	Added TODO comments for updates that are needed before the final version of the document based on comments from reviewer
0.5	20.01.2011		V9	Initial version of merge with D3.2-A
0.6	03.02.2011	External proposed	V17	Merged content from D3.3-A, added chapter on architecture, added executive summary, update intro and several updates and additions to chapter 4 and 5
0.7	01.03.2011	External revised	V24	Updated chapter 2 with new versions of diagrams and updated formatting. Several minor updates based on review comment
0.8	04.03.2011	External approved	V25	Accepted all changes, turned off track changes, and updated status
1.0	08.03.2011	Released	V26	Technical Manager release

* The project uses a multi-stage internal review and release process, with defined milestones. Milestone names include abbreviations/terms as follows:

• PCOS	”Planned Content and Structure” (describes planned contents of different sections)
• Intermediate:	Document is approximately 50% complete – review checkpoint
• External	For release to commission and reviewers;
• proposed:	Document authors submit for internal review
• revised:	Document authors produce new version in response to internal reviewer comments
• approved:	Internal project reviewers accept the document
• released:	Project Technical Manager/Coordinator release to Commission Services

Table of Contents

Table of Figures

Executive summary

The D3.1 deliverable provides tool support that facilitate software development base on the universAAL platform and reuse of universAAL components. The goal is that these tools will enable developers to easily develop universAAL compliant AAL applications and reuse existing universAAL platform services that are shared within the developer community.

This part of the multi-part deliverable describes how the tools to be developed are being selected and developed, and provides brief design descriptions. A set of use cases for the tools have been created, focusing on the tasks of the application developer role. An architectural overview shows an initial view of how the set of tools will be used as part of the full lifecycle of AAL applications, ranging from request for a service, via collection of requirements, development, and publishing, to acquisition and deployment.

Brief design information is provided for each of the tools that has so far been identified and selected to be part of the universAAL tool-chain. Existing open source tools are reused and extended as far as possible. The maturity of these tools and their descriptions range from early inception phase to initial working prototype that provide useful functionality for the developer.

The client side developer tools are combined in an integrated development environment called AAL Studio. The AAL Studio is based on the Eclipse framework, and tools that will be integrated in the studio are developed as Eclipse plugins. For this initial release of the deliverable, two such plugins have reached a maturity where the tools provide useful functionality to the developer: a Project and Item Wizard allows quick setup of a development project that uses the universAAL middleware, while the Service/Application Project Build tool simplifies building of projects. These tools access resources on the Developer Depot as part of the build process.

On the server side, two tools have been identified. The Developer Depot Server will host resources needed for a developer to get started developing using universAAL. The Nexus Repository has been selected as the basis for the developer depot. The other tool is the Eclipse Update Site which packages the tools that are plugins for AAL Studio, making it easy for the developer to install these tools into the Eclipse environment.

In addition to the tools mentioned above, the deliverable also describes initial ideas and design for a set of other tools that has not yet reached the same maturity. Development of all the tools will continue in the remaining three releases of this deliverable.

1 About this Document

1.1 Relationship to other sub-parts of this deliverable

Part I of this deliverable is provided as a wiki, and is written with tool users as the main target audience. Part I is required reading for fully understanding Part II, as the information in Part I is not repeated in this document.

The software implementation of the tools constitutes Part III of this deliverable.

The architectural work in universAAL is based on the ARCADE framework[1] [2]. Part IV of this deliverable provides the UML model. The table below shows which parts of the model that describe the D3.1 design, and shows which chapter in this document that contain extracts (e.g. diagrams) from the model.

Chapter in this document	**ARCADE view**	**Folder under the ARCADE view in the model where the diagrams are located**
Chapter 2 - Tool use cases	Context view	universAAL platform use cases / Tools use cases
Chapter 3 - Architectural overview	Component view	System Decomposition Model / Tools System Collaboration Model / Tools
Chapter 4 - Server Side Tools	Requirements view	Requirements Model / WP3 / 3.1 / Depot / Server
Chapter 5 - AAL Studio Tools	Component view	Sub-folder for the tools under: System Decomposition Model / Tools

1.2 Relationship to other versions of this Part

This version is the initial release of this part. The tools and their descriptions are at different stages of development, ranging from tools in early inception phase that are only described with brief summary for possible functionality, to tools with initial working implementations/configurations that provide useful functionality to the developer. In the latter category are the Developer Depot Server, the Project and Item Wizard, and the Service/Application Project Build tool.

Future versions of this part will update and complete the descriptions of the tools. In some cases, new tools may also be added, while other tools can be modified or even be removed from the tool chain as our understanding of the needs of the developers increase. This section of later versions of this part will give a summary of such changes.

1.3 Work process for this deliverable

In the work on this deliverable we have used a combination of workshops, individual work by the partners, and frequent telephone conferences and online chats for discussions and coordination. With

[1] http://www.arcade-framework.org/

[2] The universAAL D1.1 deliverables gives an overview of the overall approach for how ARCADE is used in the project.

the starting point of the task and deliverable descriptions provided in the Description of Work, background knowledge of the contributors, and state-of-the-art in available tools and literature, we have developed UML models including a set of use-cases and an architectural overview for the tools. Through this work, a set of tools has been identified that would provide a tool-chain to assist the developer in the development of AAL applications. For the identified tools we have selected to reuse and extend existing open source tools when possible.

Both the UML models and the source code for the tools we develop and extend are managed in a Subversion version control repository, enabling the contributors to work on their contributions in parallel.

In a collaborative work with WP5, an investigation of the developer needs have also been initiated. A questionnaire have been designed and sent to a number of developers in the AAL domain. The results from this investigation will be reported on and provide input to the tools in the next version(s) of this deliverable.

2 Tool use cases

The main focus of this chapter is the use cases that have been identified for developers of AAL applications, and these are described in detail in the second sub-chapter. The tasks identified in these use cases are candidates for tool support, in order to simplify and streamline the development process for the application developer.

The initial sub-chapter of this chapter presents the top-level use cases that also include some other main actors in the life cycle for AAL application, including their inception, development, provision, and deployment. The Architectural overview in Chapter 3 briefly follows up on the top-level use-cases by presenting sequence diagrams for the main interactions among the stakeholders and tools, but beyond this further elaboration of the use cases for other actors than the developer are considered to be outside the scope for this deliverable.

The description provided here is done in terms of UML use cases. The descriptions are extracted from the UML model provided as Part IV of this deliverable. The model provided in Part IV includes more detailed information about the use cases.

This is a work in progress, hence the use cases with their relationships and boundaries are subject to change in proceeding versions of this deliverable. The maturity and the level of detail of the model elements and packages vary in the initial versions.

2.1 Tool-chain use cases

The overall tool-chain is designed to support the use cases shown in Figure 1. A typical sequence will be initiated by the assisted person (or a person representing the assisted person) sending a request for a new AAL service. A developer discovering this request may interact with the assisted person to elicit further requirements for the service. As part of development of an application for the service, the developer will perform tasks like managing the project and searching and reusing parts of existing software. After testing and verifying the application, the developer can publish the application. To provide the actual service to the end user, the developer will make an agreement with an application provider (in simple cases the developer can also take this role). After further testing, the provider publishes a service using the application (including a service level agreement for the assisted person). Finally, the assisted person can order and purchase the published service.

The boundaries shown for the use cases in Figure 1 give an indication on which of the main tools – uStore, Developer Depot, or AAL Studio – will support the primary realization of the use case. Some of the use cases are currently not put within any of the boundaries. In these cases, it is still to be determined how these use cases will be supported. Note also that for some use cases, e.g. Publish and Test and verify, it is likely that different kind of support will be given by different tools. This will be elaborated in later versions of this deliverable.

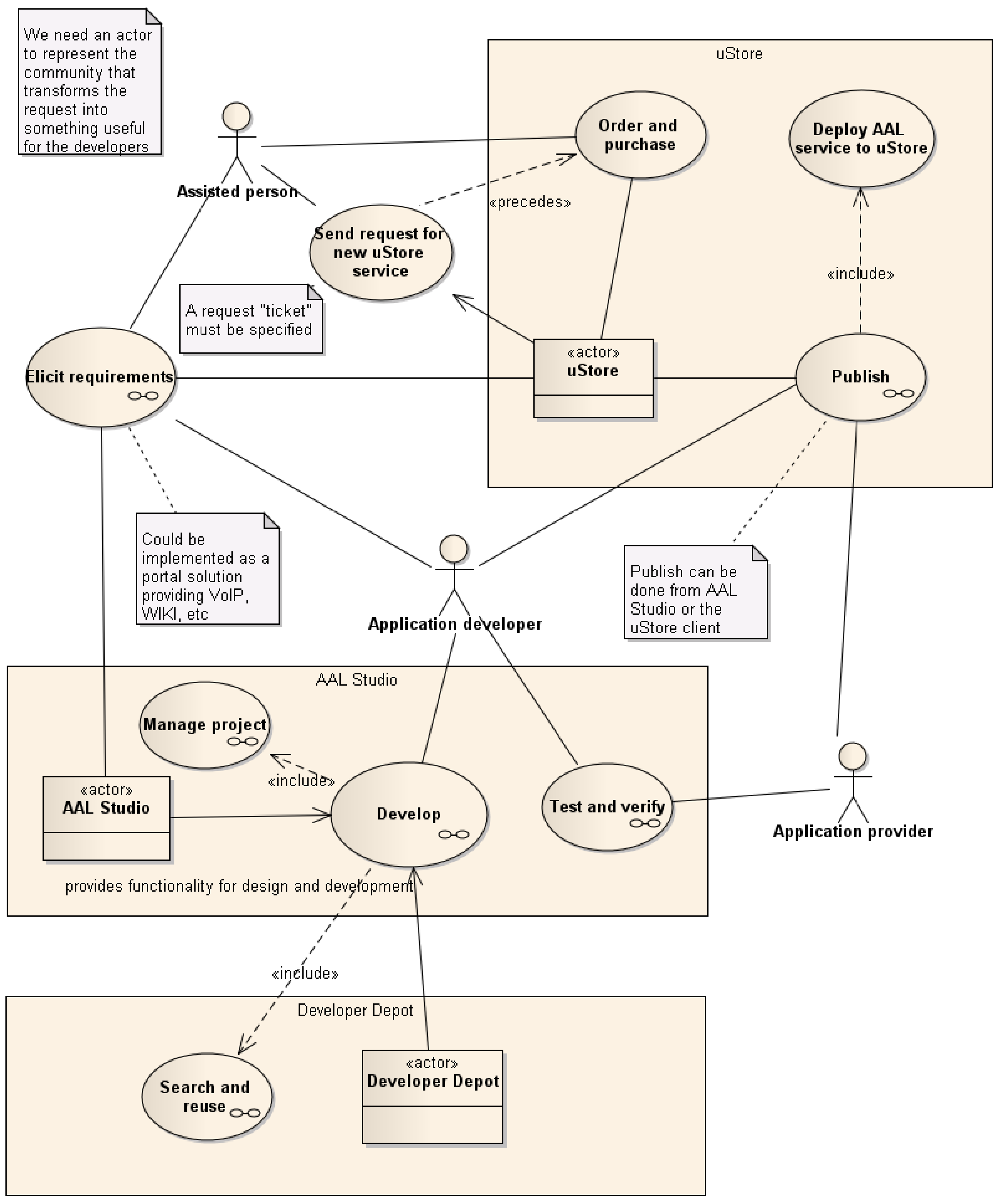

We need an actor to represent the community that transforms the request into something useful for the developers
Assisted person
uStore
Order and purchase
Deploy AAL service to uStore
«precedes»
Send request for new uStore service
«include»
A request "ticket" must be specified
«actor» uStore
Publish
Elicit requirements
Could be implemented as a portal solution providing VoIP, WIKI, etc
Application developer
Publish can be done from AAL Studio or the uStore client
AAL Studio
Manage project
«include»
«actor» AAL Studio
Develop
Test and verify
Application provider
provides functionality for design and development
«include»
Developer Depot
Search and reuse
«actor» Developer Depot

Figure 1 Overall tool-chain use cases

2.2 Tools developer use cases

The use cases for the developer are shown in Figure 2. In addition to the use-cases from the tool-chain overview, the installation of the tools has been added as a top-level use case in this diagram. Each of the use cases is described in further detail in the following sub-sections.

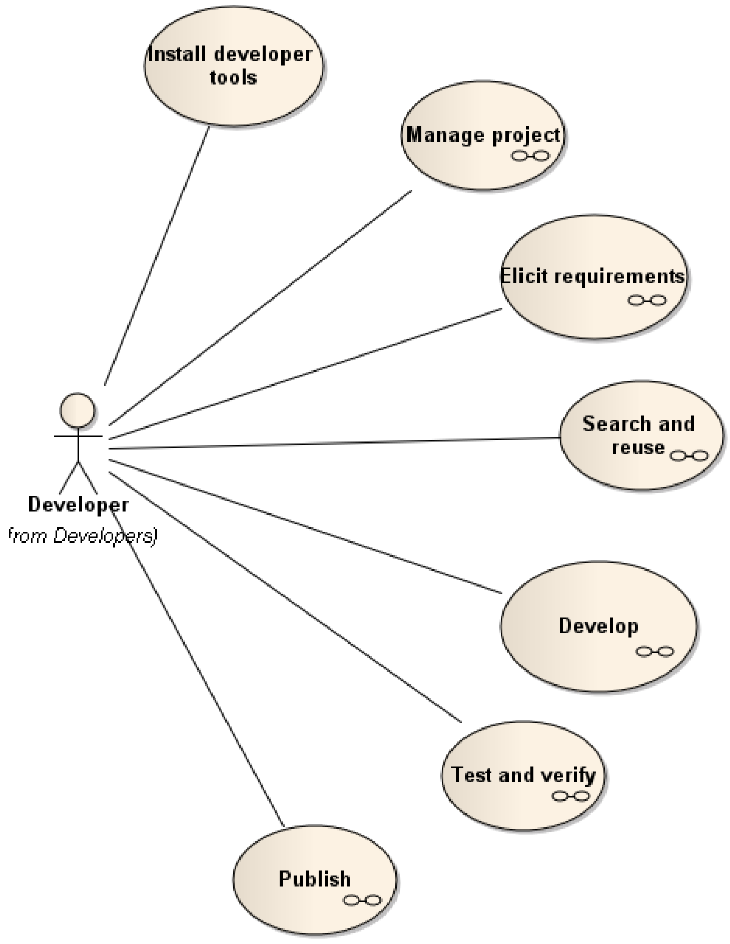


Figure 2 Overall developer use cases

2.2.1 Use Case: Install developer tools

Installing the tools needed for the development on the developer's computer. The AAL Studio tools are installed using an Eclipse update site.

2.2.2 Use Case: Test and verify

Once a service is developed it must be tested thoroughly using external resources such as test suites and protocols.

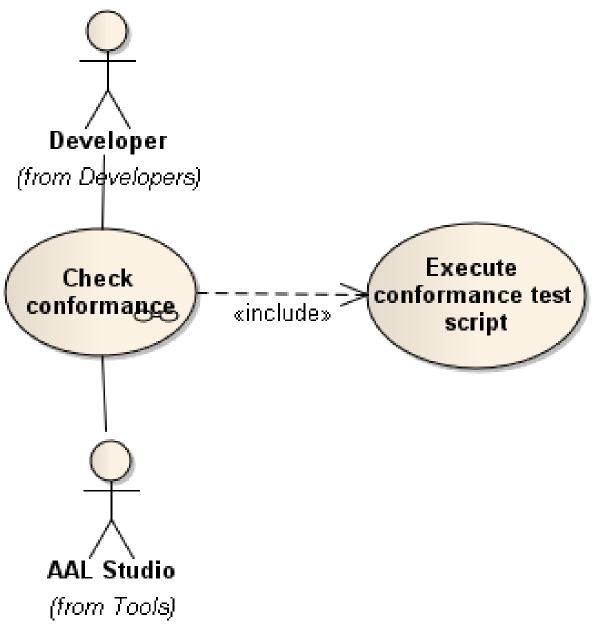


Figure 3 Test and verify use case

Run a script that checks the design against a set of conformance requirements. This can be a model-checker that runs on an OCL-engine or a text-based checker. Check conformance includes executing the conformance test from the AAL Studio. Can be triggered manually or as part of the deployment procedure.

2.2.3 *Use Case: Manage project*

Overall use case for managing the projects in AAL Studio.

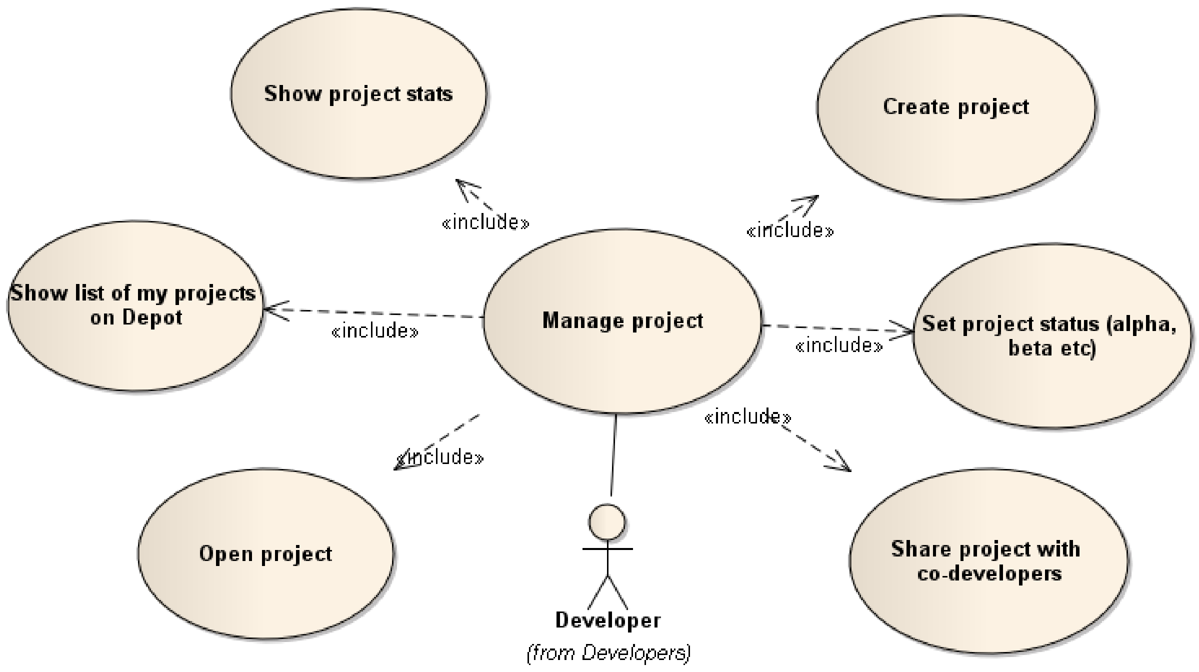


Figure 4 The Manage Project use case

Use Case: Set project status (alpha, beta etc)

Set the status of the project. This information can be used by developer depot / uStore to track the progress of a certain service development process. In cases where the project is responding to a service request, the status should include quantitative information.

Use Case: Share project with co-developers

A local project can be shared to one or more external developers through the developer depot. Access control and version control settings must be configured

Use Case: Show list of my projects on Depot

Request a list of projects from the Developer Deport where the developer has a role

Use Case: Open project

Open the project locally or from developer depot (will be downloaded)

Use Case: Show project stats

Display detailed information about the project.

If other projects are using the results from this project, this information should be made available to the project in the project stats menu

- Use Case: Show number of new comments/reviews: Get a list of external comments / requests for your project
- Use Case: Show related service requests: Get a list of service requests from Developer depot and uStore that are related to the project.

Use Case: Create project

Start a new AAL Service/Application project

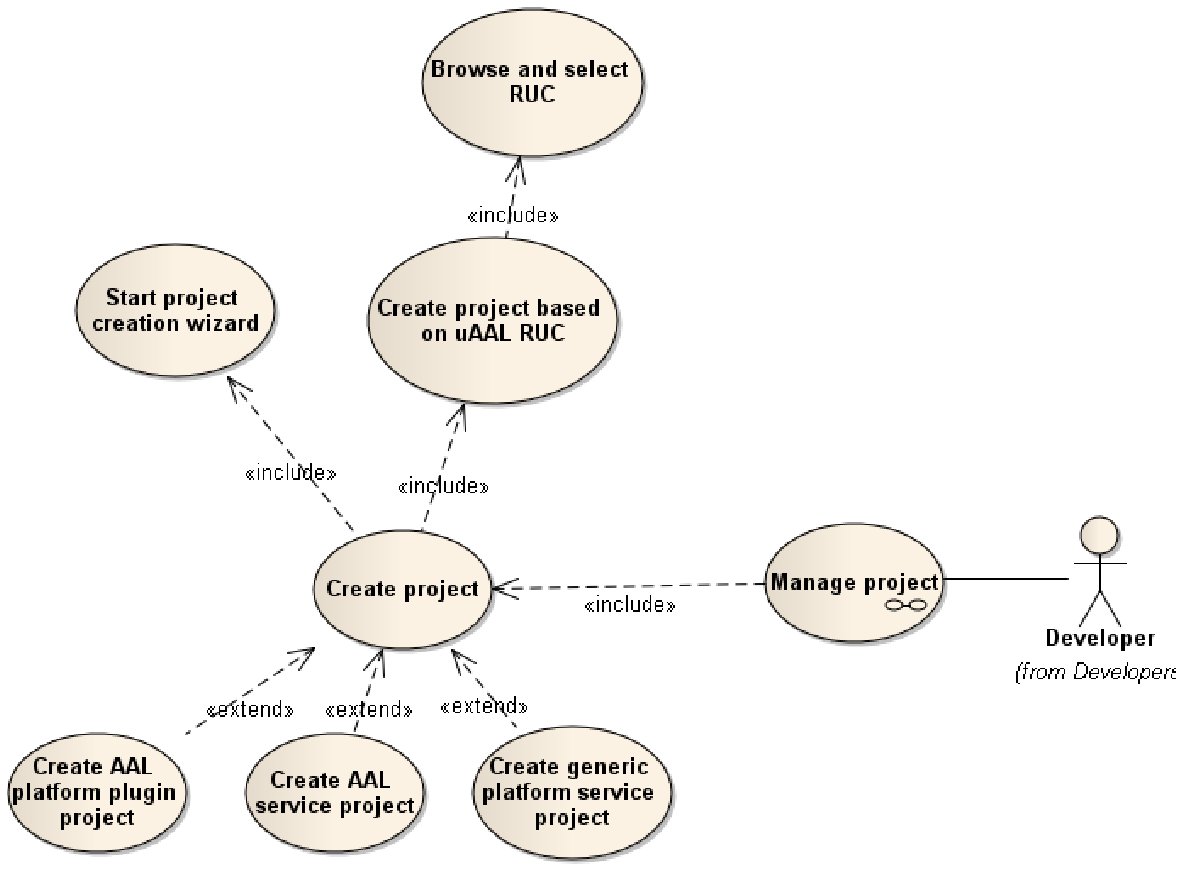

Browse and select RUC
«include»
Start project creation wizard
Create project based on uAAL RUC
«include»
«include»
Create project
«include»
Manage project
Developer
(from Developers
«extend»
«extend»
«extend»
Create AAL platform plugin project
Create AAL service project
Create generic platform service project

Figure 5 The create project use case

- Use Case: Create AAL platform plugin project: Load the template for AAL platform plugin project. All the required resources are configured
- Use Case: Create AAL service project: Load the project template and settings for a new AAL service

- Use Case: Create generic platform service project: Load the project template and settings for a new generic platform service project.

<u>Use Case: Start project creation wizard</u>

A wizard with step-by-step selection/configuration of a new project.

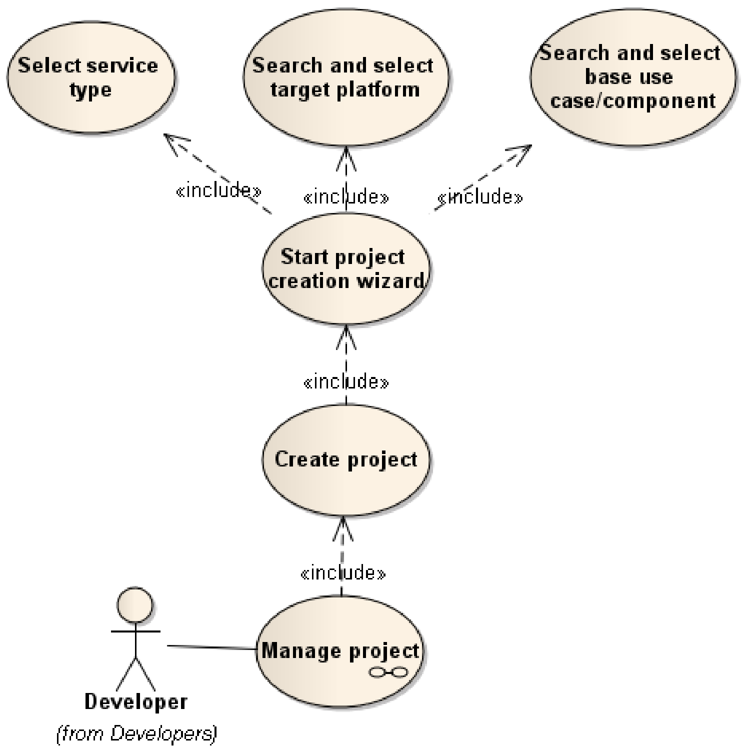


Figure 6 Project creation wizard use case

- Use Case: Select service type: Select new platform plugin, aal service or generic platform service.
- Use Case: Search and select base use case/component: The user may connect to the AAL ontology and initiate a new project based on the domain knowledge therein.
- Use Case: Search and select target platform: Final step of the wizard will be to select the target platform. This will configure the project to load the necessary libraries and dependencies.

<u>Use Case: Create project based on uAAL RUC</u>

Create a new project based on Reference Use Cases in universAAL.

Browse online repository and select the desired Reference use Case. This will automatically load the required libraries.

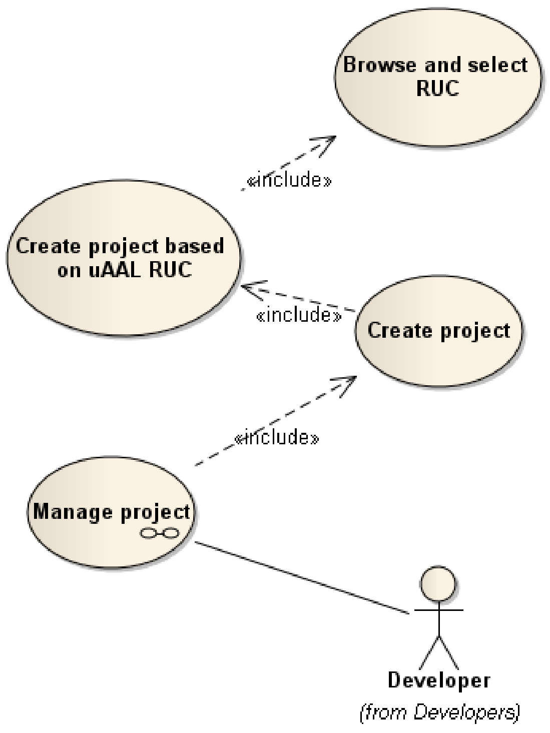


Figure 7 Create project based on uAAL RUC

2.2.4 Use Case: Elicit requirements

The developer(s) need to elicit detailed requirements from the end user about the service request. This can be done using a tool or an adhoc solution using telephone.

A full requirements process includes feedback review, structured documentation and sharing/QA of these requirements.

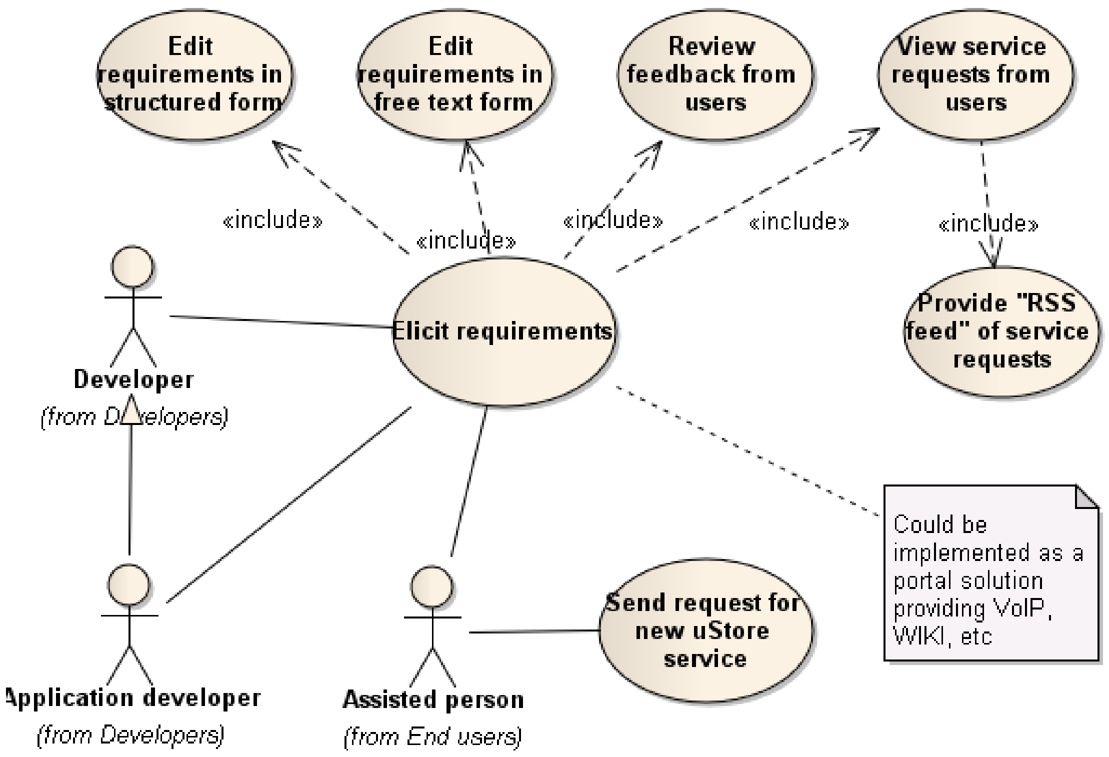

Edit requirements in structured form
Edit requirements in free text form
Review feedback from users
View service requests from users
«include»
«include»
«include»
«include»
«include»
Provide "RSS feed" of service requests
Developer
(from Developers)
Elicit requirements
Could be implemented as a portal solution providing VoIP, WIKI, etc
Application developer
(from Developers)
Assisted person
(from End users)
Send request for new uStore service

Figure 8 Elicit requirements use case

Use Case: View service requests from users

Share the uStore and Developer depot service requests as a feed

Use Case: Review feedback from users

Provide the feedback from the users in a markup language. The tool should allow editing and tagging of the feedback.

Use Case: Edit requirements in free text form

Requirements for service requests should be possible to edit in free text format.

Use Case: Edit requirements in structured form

Add requirements for a service in a structured format (e.g. checkbox selections)

Use Case: Send request for new uStore service

The end user (e.g. assisted person) issues a request for a new service. The assumption is that a search in uStore did not return any service that matched the search criteria

Must contain information about the AAL Space (user profile/preferences and environment) that is automatically inserted in the request. This information must be made available to the developers that decides to implement a service.

2.2.5 Use Case: Search and reuse

Search for reusable components and service on Developer Depot/uStore.

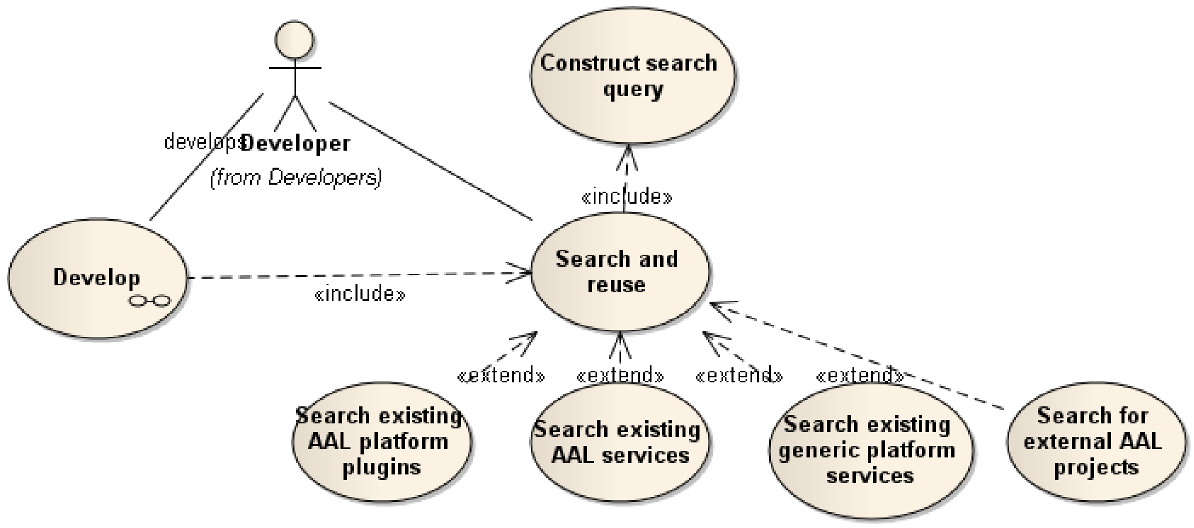

Construct search query
develops
Developer
(from Developers)
«include»
Develop
Search and reuse
«include»
«extend»
«extend»
«extend»
«extend»
Search existing AAL platform plugins
Search existing AAL services
Search existing generic platform services
Search for external AAL projects

Figure 9 Search and reuse use case

Use Case: Construct search query

Specify a search query in a form /tool

Use Case: Search existing AAL platform plugins

AAL platform service as defined in D1.3

Use Case: Search existing AAL services

Execute the search on uStore

Use Case: Search existing generic platform services

Execute the search on Developer Depot

Use Case: Search for external AAL projects

Execute the search on an external index (e.g. Bing, Google)

2.2.6 Use Case: Develop

The Developer uses AAL Studio to develop a new AAL Service. AAL Studio provides a set of mechanisms to ease the development, testing and deployment processes.

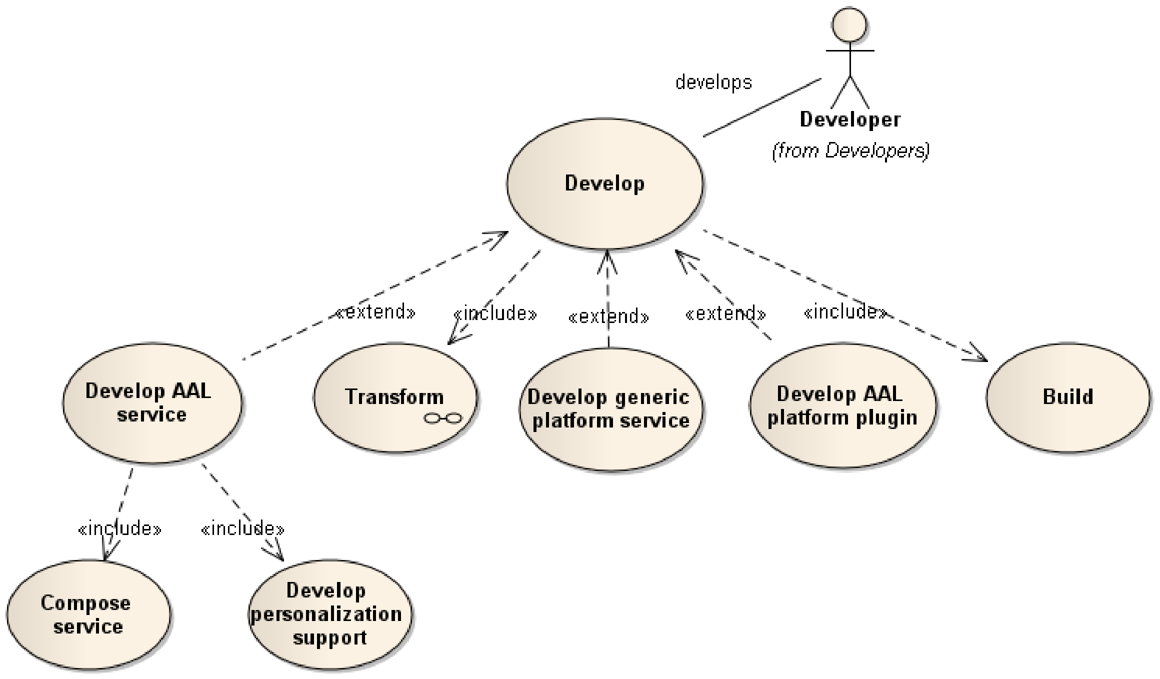


Figure 10 Develop use case

Use Case: Build

Build a deployable version of the current artefact under development

Use Case: Develop AAL platform plugin

The activity of developing a AAL Platform plugin. This is done in a specific project template that has the necessary libraries and settings preconfigured

Compose a new service from existing services

Use Case: Develop generic platform service

The activity of developing a generic platform service. This is done in a specific project template that has the necessary libraries and settings preconfigured

Use Case: Develop AAL service

The activity of developing a AAL Service. This is done in a specific project template that has the necessary libraries and settings preconfigured

Compose a new AAL Service from existing components and services***Use Case: Develop personalization support***

Add personalization support to an AAL Service.

Use Case: Compose Service

Compose a new AAL Service from existing components and services

2.2.7 Use Case: Transform

Transform one development artefact into another. Transformations can be from model to model, or model to text (e.g. generate source code from a UML model).

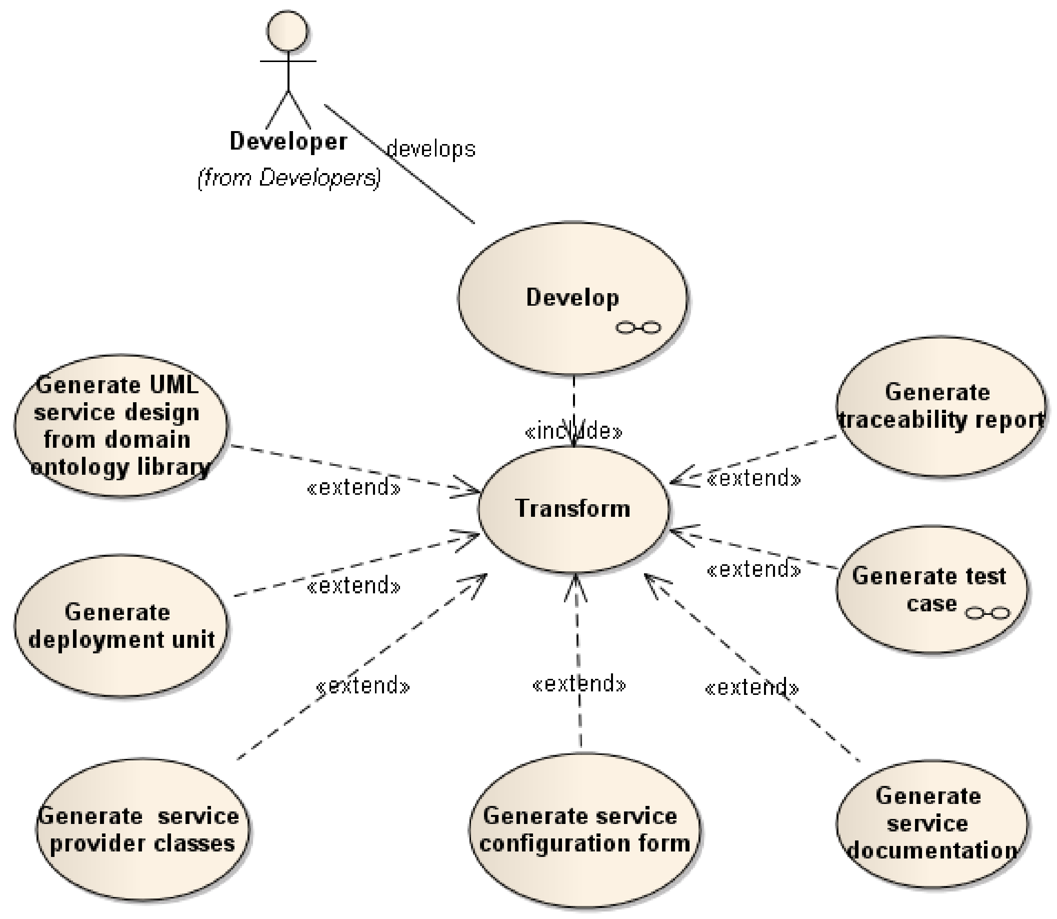

Developer
(from Developers)
develops
Develop
«include»
Transform
Generate UML service design from domain ontology library
«extend»
Generate deployment unit
«extend»
Generate service provider classes
«extend»
Generate service configuration form
«extend»
Generate service documentation
«extend»
Generate test case
«extend»
Generate traceability report
«extend»

Figure 11 Transform use case

Use Case: Generate service provider classes

Generate required (mandatory) java classes for a specific service type. The type and service profile are decided from the ontology connection/selection

Use Case: Generate deployment unit

Generate deployment files (bundle files) for a developed service. The service must be tested and certain properties should be configured (conformance). The user must select type of deployment, target, etc.

Use Case: Generate service configuration form

Generate a simple interface (web-form) for setting service configuration options (simliar to personalization tool settings). This form can be used directly or through the personalization tool

Use Case: Generate service documentation

Generate documentation for the service project. The documentation should have two levels, one for end user and one for developer/IT people

Use Case: Generate test case

A feature that generates test cases for certain components and aspects of the design. Works closely with the conformance testing tool

Use Case: Generate traceability report

Based on the tracelinks in the design, code and documentation, a report traceability is generated. The report may have different levels of detail, from simple dependencies to more advanced analysis mechanisms such as coverage analysis, orphan analysis and change impact analysis.

<u>Use Case: Generate UML service design from domain ontology library</u>
Generate UML representation for the service/types in the ontology. This allows the designer to work through UML to generate universAAL AAL services.

2.2.8 Use Case: Publish

Developer can share the artefacts with others. Eg. share a software service with others on developer depot and/or uStore

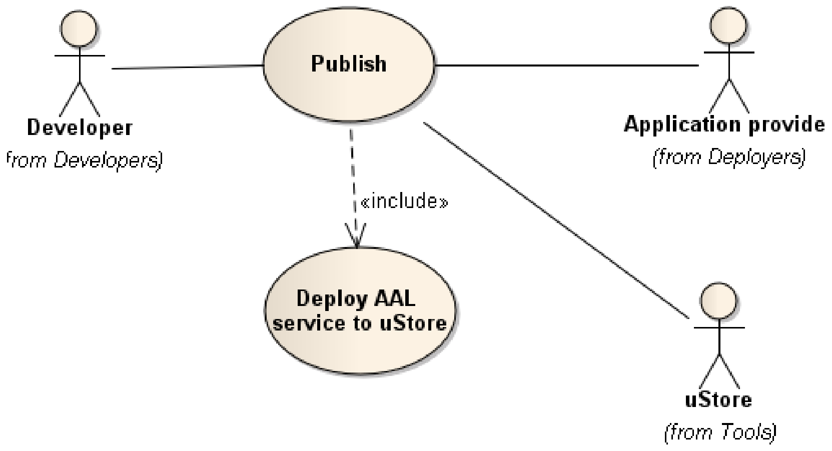


Figure 12 Publish use case

<u>Use Case: Deploy AAL service to uStore</u>
Deploy the AAL service to uStore. This requires a complete and validated deployment script.

3 Architectural overview

This chapter gives a brief architectural overview of the top-level tools of the universAAL tool-chain, their connections and their decomposition into more fine-grained tools. The information in this chapter will be integrated into and detailed in upcoming versions of the main architectural description for universAAL, which is provided in D1.3. The versions of the D1.3 deliverable that has been released prior to this document (D3.1-A, Part II) have focused mainly on the runtime platform, and have not covered how to the tools will fit into the architectural picture.

Figure 11 gives an overview of the top-level tools of universAAL and their connections, including how they are connected to the runtime Execution Environment produced in WP2.

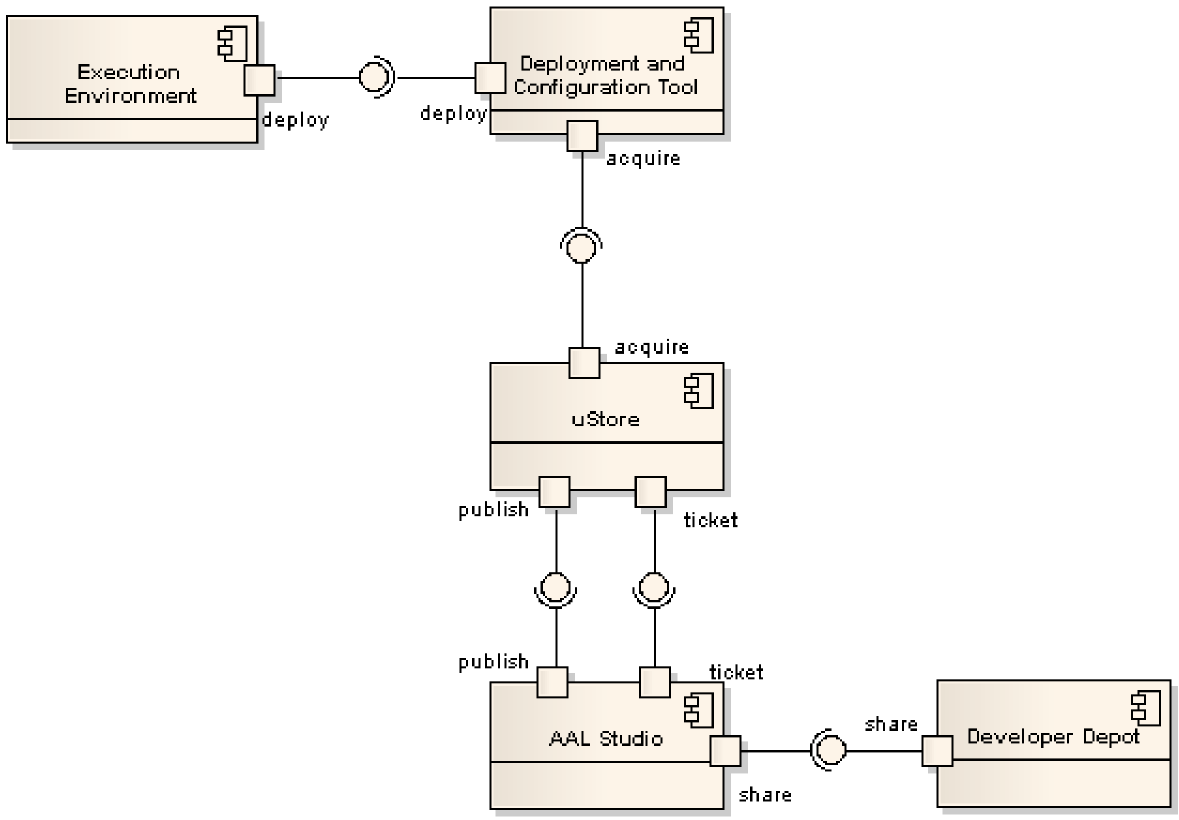


Figure 13 Overview of the top-level universAAL tools and their connections

Figure 12 shows a further decomposition of the main tools, along with which task from DoW they are the product of. The *AAL Studio* and tools contained within it are described in Chapter 5, while the *Developer Depot Server* and the *Eclipse Update Site* are described in Chapter 4. The *Development resources* shown under the *Developer Depot Server* will include the binary release of the runtime platform, architectural descriptions, tutorials and other material that are mainly produced in other work packages, and a more detailed overview will be provided in a later version of this deliverable. The *uStore* and the *Deployment and Personalization tool* are described in deliverables D3.4 and D3.5 respectively.

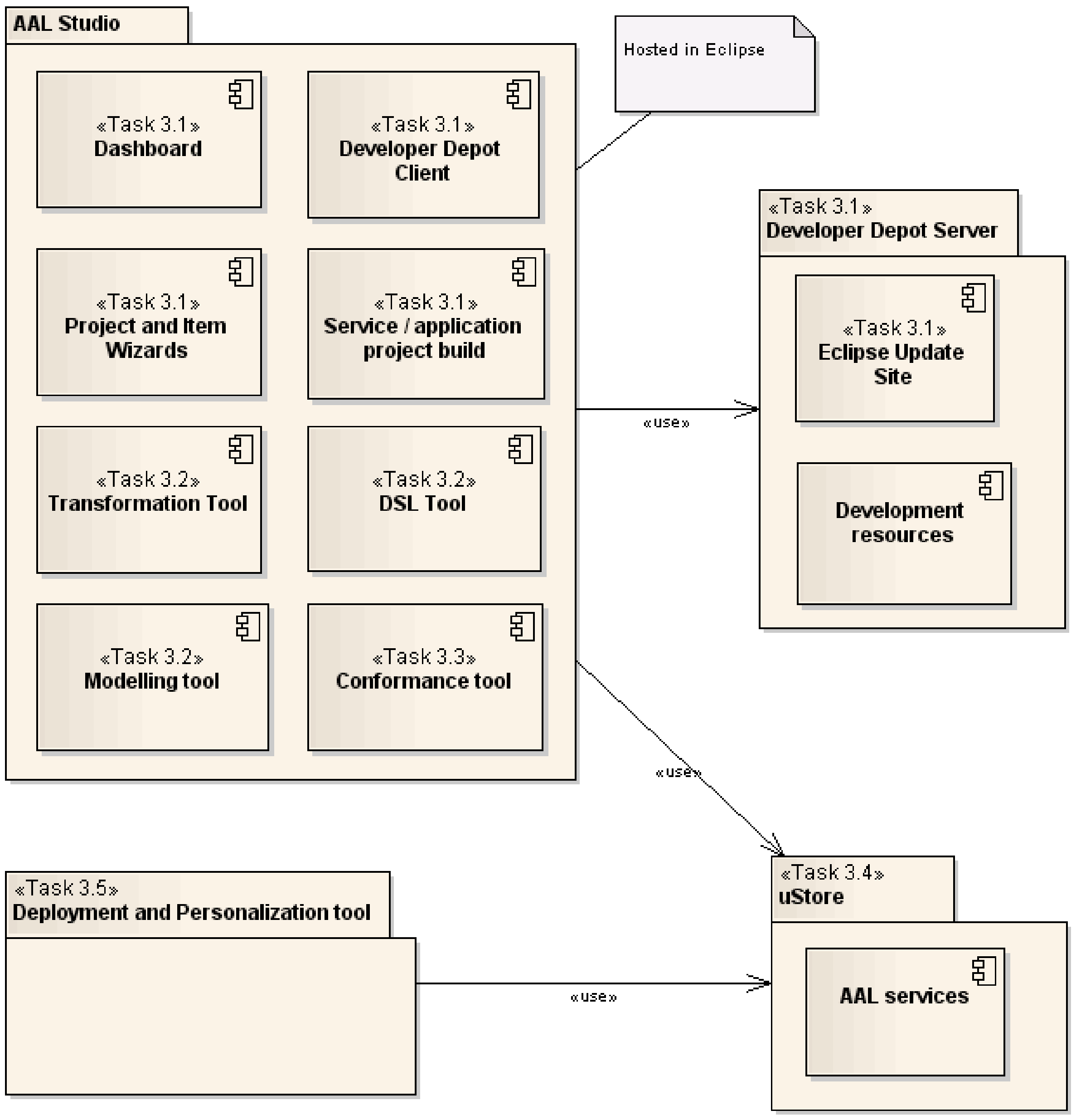

AAL Studio
«Task 3.1» Dashboard
«Task 3.1» Developer Depot Client
«Task 3.1» Project and Item Wizards
«Task 3.1» Service / application project build
«Task 3.2» Transformation Tool
«Task 3.2» DSL Tool
«Task 3.2» Modelling tool
«Task 3.3» Conformance tool
Hosted in Eclipse
«Task 3.1» Developer Depot Server
«Task 3.1» Eclipse Update Site
Development resources
«use»
«use»
«Task 3.5» Deployment and Personalization tool
«use»
«Task 3.4» uStore
AAL services

Figure 14 Decomposition of the top level tools

Based on the Overall tool-chain use cases presented in Figure 1, the interaction of the actors and tools have been further analysed and documented in two sequence diagrams. The diagrams in Figure 13 and Figure 14 give a brief overview of the interaction between the actors and the tools through the lifecycle of an application, starting with a request for a service from an assisted person, including how the AAL Studio and Developer depot are used by the Application developer to develop the application, to how the application is offered as part of a service in uStore, and to its installation and configuration in the execution environment of the assisted person.

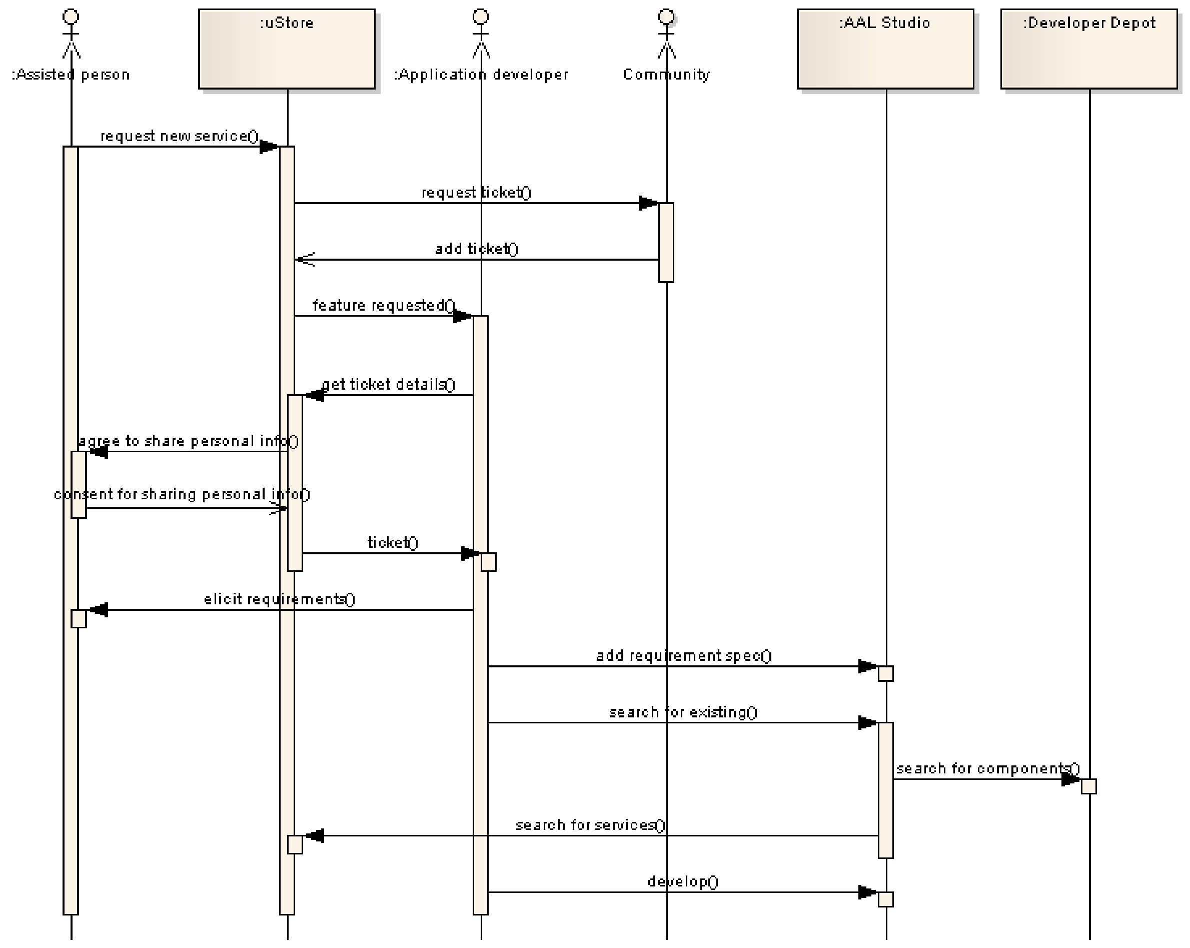


Figure 15 Sequence diagram from user request to development

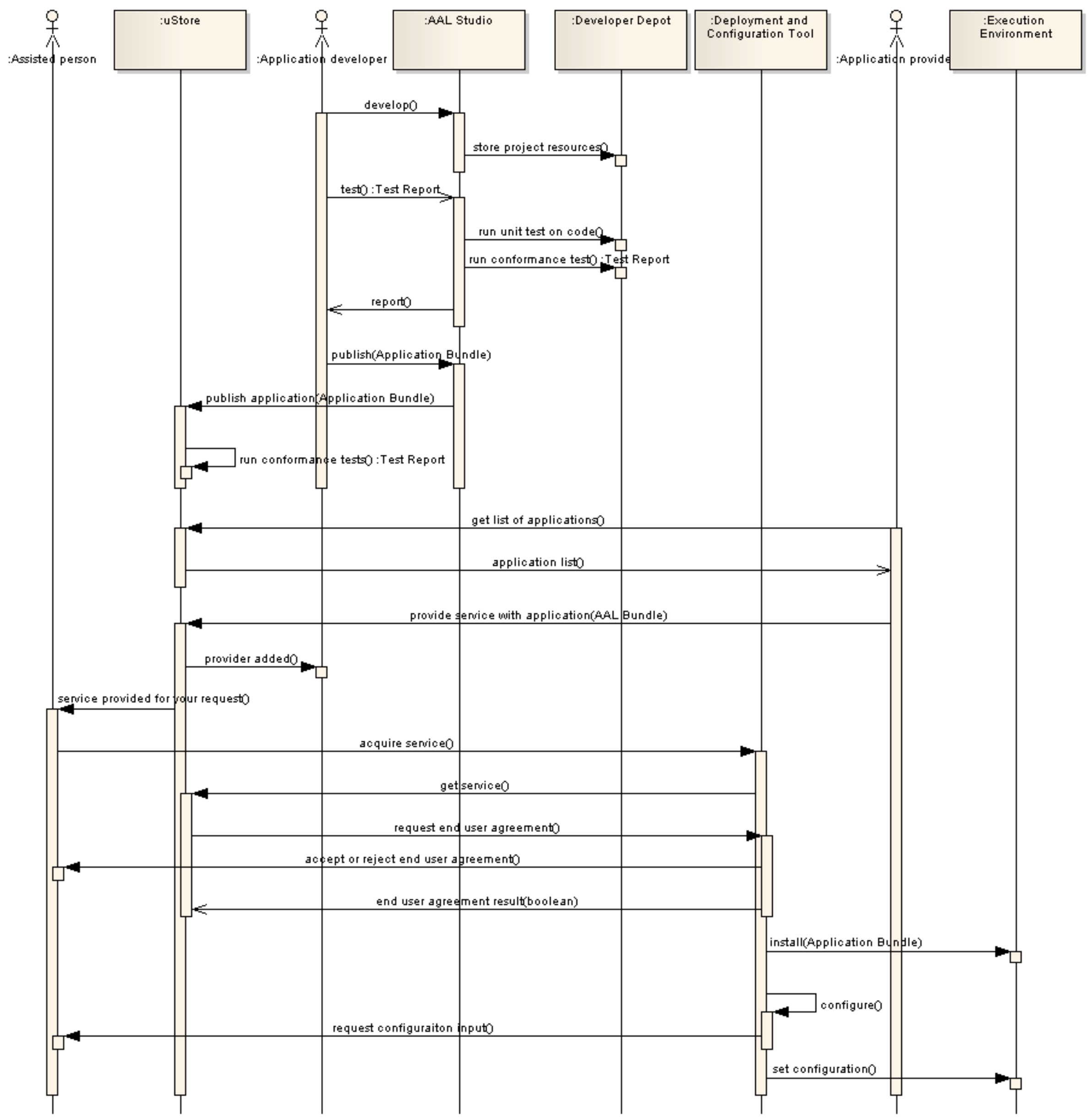

:Assisted person
:uStore
:Application developer
:AAL Studio
:Developer Depot
:Deployment and Configuration Tool
:Application provider
:Execution Environment
develop()
store project resources()
test() :Test Report
run unit test on code()
run conformance test() :Test Report
report()
publish(Application Bundle)
publish application(Application Bundle)
run conformance tests() :Test Report
get list of applications()
application list()
provide service with application(AAL Bundle)
provider added()
service provided for your request()
acquire service()
get service()
request end user agreement()
accept or reject end user agreement()
end user agreement result(boolean)
install(Application Bundle)
configure()
request configuraiton input()
set configuration()

Figure 16 Sequence diagram from development to deployment and configuration

4 Server Side Tools

This chapter will provide a high-level design for the tools which we will develop ourselves or which we will do major modifications or configurations of.

4.1 Developer Depot Server

In order to provide support for AAL application development, the Developer Depot Server has been introduced. The depot aims at enhance the development process by providing to developers resources and code samples supporting the software reuse.

Figure 12 (page 21) depicts the relationship between Developer Depot Server and AAL Studio suite. The Depot Server will be mainly addressed to two stakeholders:

- Depot Client in order to let the depot features be accessible directly within the AAL Studio suite
- AAL application developers in order to perform administrative tasks (e.g. set up repositories, configure permissions and roles, create and manage user groups).

The adoption of Depot Server into universAAL tool chain is expected to introduce the following benefits:

1. Simplify development process: AAL application developers only need the depot server URL in order to fetch the universAAL artifacts or third-party ones. Moreover the partner software contributions will be safe from accidental removals and the project results will be available to the European community for years in a standard way also afterword the project conclusion.
2. Decrease bandwidth usage: the depot manages company specific artifacts (hosted by universAAL repositories) or proxy for existing one (hosted by e.g. Maven Central Repository). In both of the cases the depot will adopt some caching mechanisms in order to reduce the download time for the most accessed artifacts. This is a more crucial aspect especially for large projects where the number of dependencies becomes non negligible and lot of people actively provide contributions.
3. Increase collaboration: developers can look for existing code into the universAAL repositories, reusing or extending them being aware of new artifact releases.
4. Managing different artifact releases: the depot provides several repository instances in order to distinguish between unstable and stable artifact versions.

4.1.1 Depot Server Requirements

This section summarizes the Depot Server requirements consolidated during the initial project phase. Two distinct requirement sets have been identified: *must-have*[3] and *should-have*. For the purpose of the current deliverable version and the current project phase, only requirements to the first set have been provided through the current Depot Server installation. Next deliverable releases will consider if and how to assign a priority to the should-have requirements.

Must-have requirements

- Depot Server must provide the following essential artifact operations: download, removal and upload.

[3] *must-have* requirements features that cannot be renounced.

- Users must be able to search for artifacts. The search must be performed by exploiting GAV[4] artifact metadata.
- Depot Server must be designed in order to correctly manage artifact developed with different programming languages. Actually only maven artifacts will be considered as valid deployment units.
- The Depot Server must provide standard HTTP management interface.
- Depot Server must provide command line tools.
- Depot Server must perform user profiling by defining: permissions, roles (permission's group) and user groups.
- Depot Server must provide tools for administration purposes.
- Depot Server must enable to create multiple repository instances.
- Depot Server architecture must support plug-in extension.

As previously introduced also a set of *should-have* requirements have been identified. Such requirements will be analysed and discussed for the next project phase.

Should-have requirements

- Depot Server should provide collaboration tools able to let developers post feedbacks and comments about one or more artifacts. The depot should publish a dashboard for every artifact where useful information should be easily gathered.
- Depot Server should perform specific actions during the artifact build phases. Several build phases should be defined (e.g. staging, pre-production, production) and for every one of them a set of specific actions should be defined.
- Depot Server should share user credentials with uStore in order to ease the interaction between both of the servers.
- Depot Server should provide mechanisms to perform scheduled back-up routines.

4.1.2 Developer Depot Tool

The expected benefits for Developer Depot drives the initial implementation. Figure 15 depicts the building blocks composing the server. In order to meet all the *must-have* requirements two building blocks are needed:

- o Nexus Repository: provides artifact management functionalities such as download, search, removal and provides user authentication mechanisms. Such functionalities can be exploited through standard HTTP interface or through maven tool.
- o CI framework: provides continuous integration functionalities such as artifact build, unit test execution, report generation, collaborative tools.

[4] Maven artifact can be identified by means of three coordinates: Group ID, Artifact ID and Version ID.

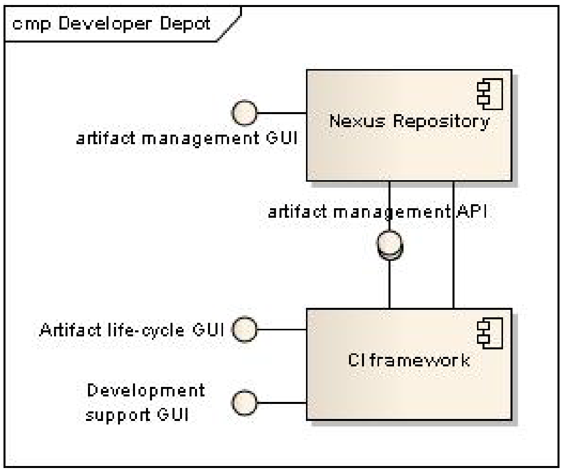


Figure 17 Developer Depot

Both of the building blocks expose multiple interfaces that can be directly accessed from Developer Client or AAL application developers. For the initial project phase only the Nexus Repository building block has been installed and properly configured, providing the following features:

- Hosting Repositories: it's possible to upload artifacts using the Nexus Web interface or deploy artifacts to the hosted repositories using Maven. Nexus will also create the standard Nexus index for all of the hosted repositories, which will allow tools like m2eclipse to rapidly locate software artifacts for Eclipse IDE.
- Proxy Remote Repositories: Nexus enables to proxy remote maven repositories.
- Repository Groups: Nexus allows to cluster multiple repositories into one single URL.
- Fine-grained Security Model: Nexus adopts the notion of privileges and roles. A privilege defines the capability for a developer to perform one specific operation (such as upload, download, removal), while roles cluster multiple privileges together. Nexus administrator is free to assign a role to a developer or more granularly a specific subset of privileges.
- LDAP Integration: Nexus allows to synchronize users against LDAP server. All the LDAP group and LDAP users will be imported inside Nexus repository.
- Artifact Search: Nexus provides to developers a smart artifact search engine, which allows to search for a number of coordinates, e.g. GAV coordinates. The search results show ready-to-use information like version number, download link, repository location and XML code snippet ready to be merged into project the POM file.
- REST Services and Nexus plugins: the Nexus functionalities are exposed through REST services, who's API are available to developers to implement custom plugins. This chance keeps Nexus choice solid against new DDS requirements.

4.2 Eclipse Update Site

Eclipse provides a built-in facility for managing and installing plug-ins to the platform from update sites. An update site contains a set of plug-ins grouped in features. From the Eclipse environment, the IDE user can add update sites to download plug-ins from, and can choose which features from the site to download and install.

An Eclipse Update Site is simply a structured collection of files made available for download over http on a regular web site. Each client side universAAL developer tool is packaged as Eclipse features, and will be provided from as part of the universAAL Eclipse update site. For more details about how Eclipse Update Sites are created we refer to the Eclipse web site:

http://www.eclipse.org/

5 AAL Studio Tools

This chapter presents a set of development tools that are combined in an integrated development environment we call AAL Studio. The chapter starts with a sub-chapter description of the selection of Eclipse as the framework on which the AAL Studio is built. Following this are sub-chapters for each of the currently identified AAL Studio tools. Each of these sub-chapters gives an overview of the design of the tool, including how the tool is integrated in the Eclipse environment.

5.1 Selection of Eclipse as framework for AAL Studio

Already in the universAAL "Description of Work", Eclipse was identified as the platform on which the (client side) development tools will be based.

Eclipse has a strong position as a Java IDE for several reasons. It is:

- a mature development platform
- open source with a large and active community and backing from major commercial actors
- widely adopted by developers and researchers
- very extensible, with a large number of existing extensions

For universAAL, it is furthermore a good match because:

- WP2 has selected OSGi and Android as a basis for the runtime platform, and Eclipse is the preferred environment for development for these
- several relevant extensions exist, e.g. for modelling and transformation support, that will be a good foundation for universAAL tools
- it provides tool-support which will assist us in developing the developer tools envisioned for universAAL
- tools from the Open Health Tools project, with which universAAL plan to integrate, are based on the Eclipse platform

While other good open source Java IDE's exist, with NetBeans as the main example, we consider Eclipse to be a superior match for the project, and thus have not invested further time and resources in doing a more formal evaluation of the alternatives.

5.2 Project and Item Wizards

A wizard in Eclipse is composed by several pages where the developer fills in a series of form fields. Once all fields are validated (contain well-structured information) the developer can move to the next page, and whenever the wizard has enough information to finalize, the developer can click the "Finish" button to get the wizard do what is supposed to do.

5.2.1 Relation to Eclipse

The current implementation of the Project Wizard uses the following Eclipse extensions:

Eclipse extension point	Description of provided realization(s)
org.eclipse.ui.newWizards	This allows the wizard to insert itself among the Eclipse "File->New" collection of wizards. It specifies also its own category, to differentiate from Eclipse own wizards, or other plug-ins.
org.eclipse.ui.commands	Here is specified the category and identifier of the command that will trigger the wizard from any other menu or plug-in. See details below.
org.eclipse.ui.handlers	Used in conjunction with the above extension to trigger the start of the wizard when the command is issued.
org.eclispe.ui.menus	By default this plug-in adds a contribution to this extension so that the wizard is started from a specific menu in the menu bar.

The main access to Project and Item Wizards is through the File/New command in Eclipse. Then the proper wizard must be selected, but the wizard can also be invoked by using the appropriate command. It can be placed in the proper menu designed for all the tools in the menu bar or started from the dashboard, or any other plug-in tool. The information required for this is shown here:

- Category ID: org.universaal.tools.newwizard.plugin.command
- Command ID: org.universaal.tools.newwizard.plugin.command.startNewWizard

5.2.2 Design

The Project wizard is launched whenever the developer wants to create a new Project. As a wizard, it leads the developer through a series of steps or pages where he introduces data that is used in the end to generate the project in the workspace.

The data that is gathered from the user is:

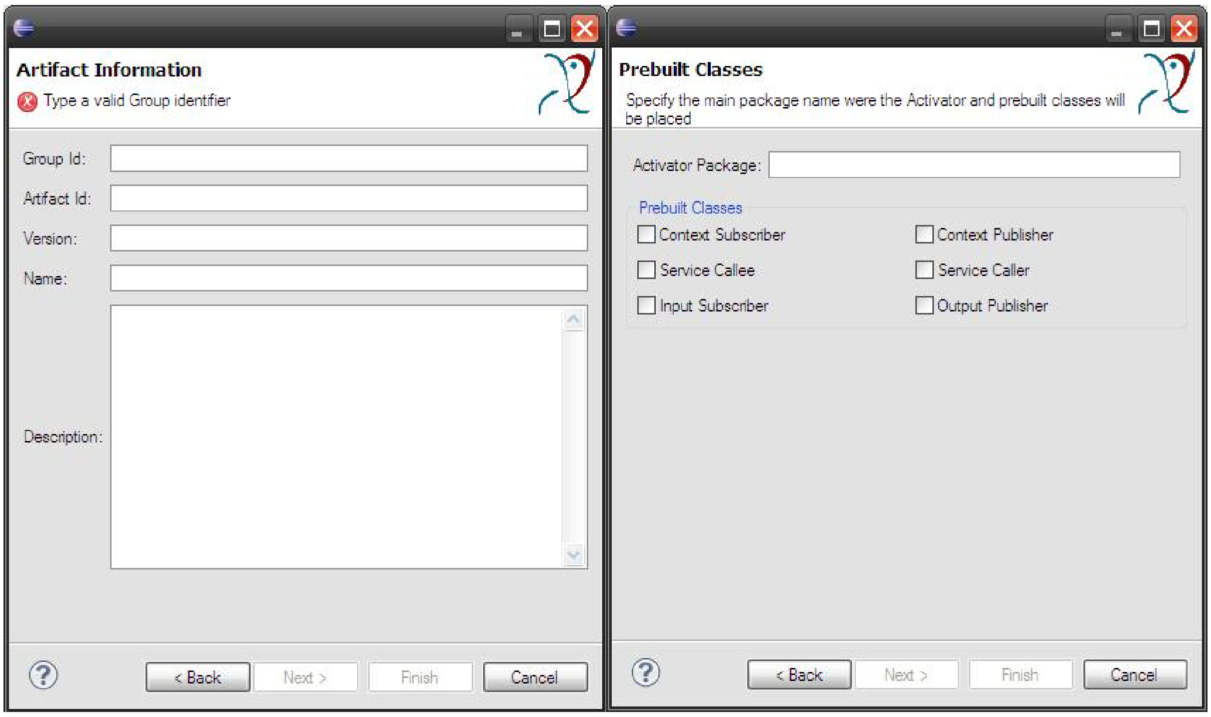
Maven data:

- Group Id
- Artifact Id
- Version
- Project Name
- Description

Project files data:

- Main package name
- Initial base classes

The forms to fill the data are presented in the following layout for each page:



Artifact Information

Type a valid Group identifier

Group Id:	
Artifact Id:	
Version:	
Name:	
Description:	

Prebuilt Classes

Specify the main package name were the Activator and prebuilt classes will be placed

Activator Package:

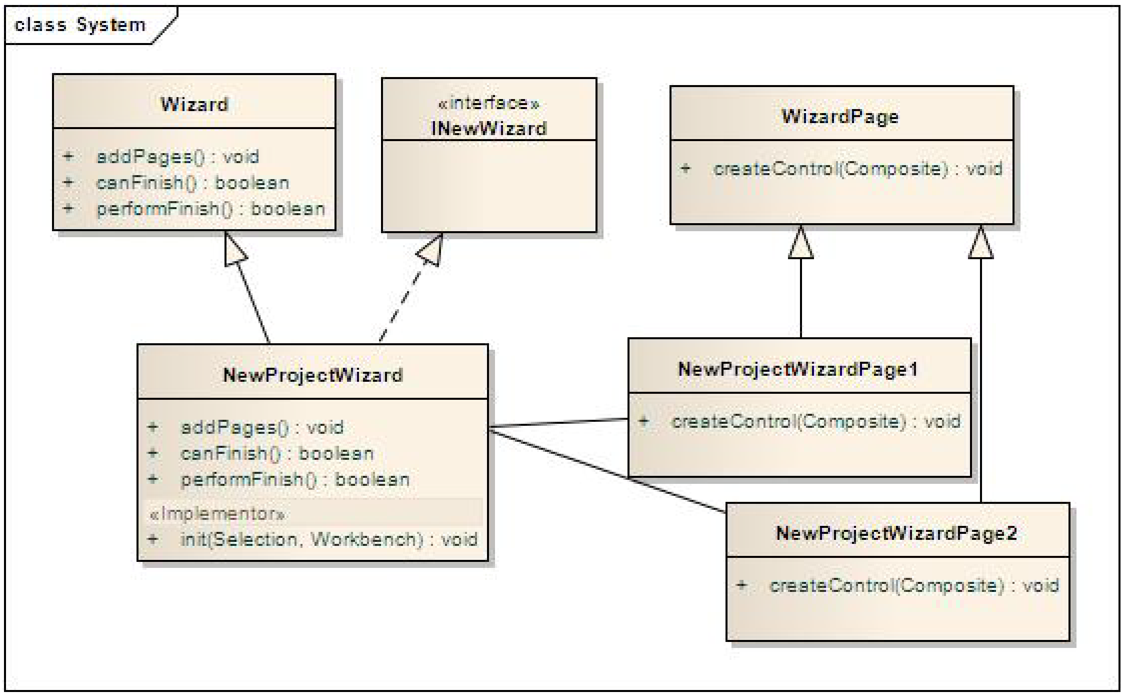
Prebuilt Classes

☐ Context Subscriber	☐ Context Publisher
☐ Service Callee	☐ Service Caller
☐ Input Subscriber	☐ Output Publisher

Once the data has been filled, the project is generated with the following outputs:

- A new Maven Project in the workspace, with the given Name, with:
    - A pom.xml file according to the Maven data provided and following a default configuration ready to be used with universAAL
    - All default folders and structures for a maven project
    - An initial package under the source folder with the given name with:
        - All base uaal-compliant classes indicated in the wizard

The code of the plug-in, apart from the required metadata that describes the plug-in itself, is composed by the classes shown in the following diagram:



Where *Wizard*, *INewWizard* and *WizardPage* are classes from the Eclipse Plug-in Development that are extended or implemented by the actual classes of the plug-in:

NewProjectPage1: Defines the forms and layout of the first page of the wizard.

NewProjectPage2: Defines the forms and layout of the second page of the wizard.

NewProjectWizard: The class where the creation of the project is performed. It adds the two pages to the wizard and when the process is finished it analyses the inputs to generate a blank Maven project with the specified properties (for which it uses the Maven plug-in) and then creates the package and the template classes specified.

There are two additional pages: *Messages*, which is used for internationalization, and *NewProjectCommandHandler*, which simply handles the command to start the wizard from another plug-in or menu. These both classes are accessory and don´t take part in the creation of projects.

5.3 Dashboard

5.3.1 Relation to Eclipse

The table below shows the currently identified extension point that will be used by the Dashboard.

Eclipse extension point	Description of provided realization(s)
org.eclipse.ui.views	A view that presents the dashboard to the developer, and allows the user to invoke steps of the development process in the correct order.
org.eclipse.ui.commands	The dashboard will not provide commands on its own, but rather it will use the commands defined by other plugins (e.g. the Project and Item Wizards) to trigger the functionality provided by these tools.

5.3.2 Design

A dashboard in Eclipse is a specific kind of view. The dashboard allows the developer to invoke the specific actions for the difference steps that is routinely used in the development of an universAAL application; all gathered in one place. Figure 16 depicts a simple sketch example of a dashboard in eclipse. The squares in the window represent the different tasks that a developer must complete in order to construct a viable application. The sketch in Figure 16 exemplifies the process by giving the initial task of searching for an existing set of services that will satisfy the goal of the application in question. Given that such a set of services exists the next task is to combine them into an application, test the application and deploy it to the *uStore*. Assuming that one or more services do not exist, the developer must first develop them, test them and then deploy them.

Figure 18 Sketch of a dashboard

In general, the tasks in a dashboard corresponds to states of the development process, and thus also the development environment, here eclipse. The arrows correspond to state-transitions in this process. Figure 17 describes the state diagram for the dashboard.

The state diagram corresponds to the four-layered architecture as described in deliverable 1.3 [1]. For convenience the layers and their names have been marked in Figure 17. The initial state of the system is the state where the developer wishes to develop a new application (state 1). The final stet is the return state following the publishing, or deployment state (state 19).

It is worth noticing that this initial approach assumes that the developer wishes to develop a full AAL application. However, situations will arise where some developer might wish to only develop a specific service on one of the underlying layers. The current design of the states in the dashboard does not support this approach. This choice is based on the assumptions that 3rd party hardware suppliers most likely supply components on the middleware layer. They will presumably develop these components following their own in-house approach and make them available through either the *developers' depot* or *uStore*. Further, services on the *generic platform layer* is assume to be supplied as an integrated part of the universAAL platform or be developed with a specific application in mind; thus, this development will be achieved through the initial attempt to make an application. Finally, components on the *AAL Platform Services* layer are also assumed connected to the development of an application. Thus, similar to the second layer, they will be covered when a developer initiates a new application.

However, the ambition of the universAAL dashboard is, if applicable, to support development on all layers. Thus, future versions of the dashboard will include initial states that will allow a developer to enter all four layers, without working in a top-down fashion.

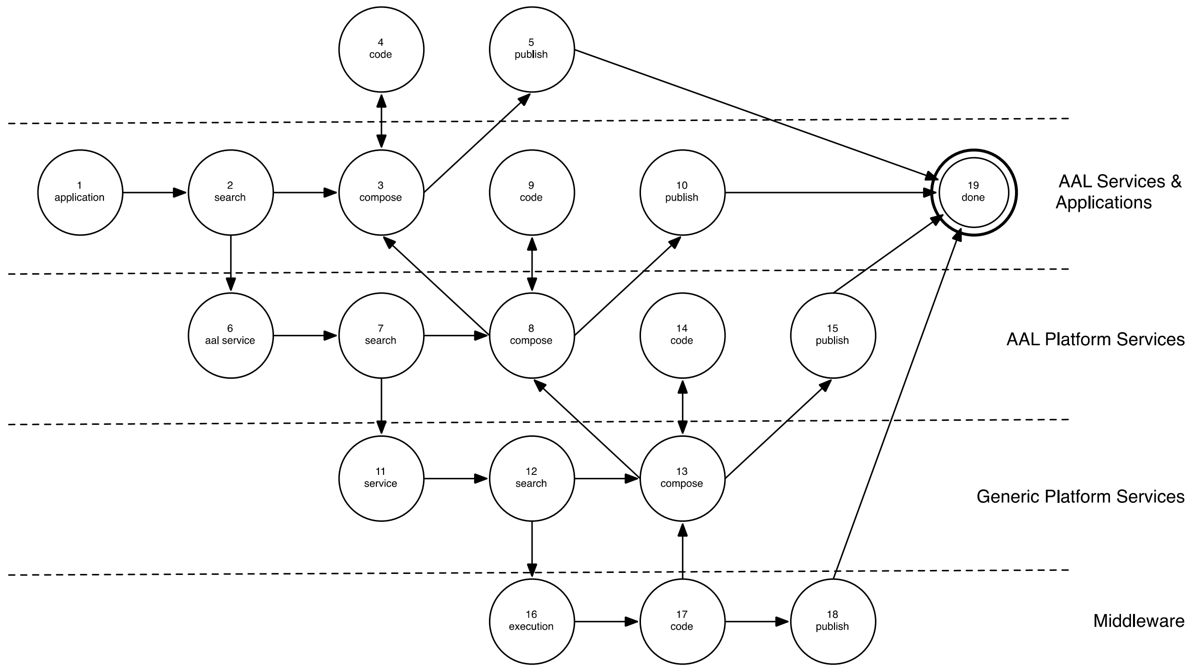


Figure 19 State diagram for the dashboard

The initial state is the application state where the developer wishes to develop a new AAL service or application. To do this, a search for existing applications or services is conducted. Activating this state allows the developer to specify the parameters required to describe the project[5]. Assuming that all components are present, the developer can move onto the next state (3 compose), where the corresponding eclipse view will be available. After the composition is done, the new ALL service can be published to the uStore (5 publish) and the final state is reached (19 done).

Given that not all underlying components are present, the developer will move to the *all service* state (state 6). The principle in the AAL Platform Services layer is exactly the same as above. That is: search for existing components (7 search); compose existing components (8 compose); code any necessary “glue” (9 code); publish (10 publish); and terminate the state machine (19 done). As in the above layer, it is possible that not all underlying components are present. In that case, the developer can move onto the Generic Platform Services layer (state 11 services).

The sub-states for this layer are also identical to the one above: initially a search for components is carried out (state 12 search); assuming that all underlying components are present they can be composed into a suitable new component (13 compose); any additional “glue” code can be produced (state 14 code); finally, the new component is published (15 publish), and the state machine terminates (19 done).

Finally, assuming that underlying components are also missing here, the developer can move onto the bottom layer, the Middleware. As this is the lowest layer there are no underlying components to search for. Thus, the developer must write the required code for any required components (state 17 code). Once the code has been developed, it can be published[6] (state 18 publish), and the state machine terminates (state 19 done).

Employing theses different states and state transitions allows a developer a clear overview of the process of developing and publishing universAAL services, components and applications.

[5] The exact parameters are to be decided at a later state. Examples could be, but are not limited to: input, output, goals, pre-conditions and post-conditions.
[6] It is currently undecided if Middleware components are to be published at uStore, Developers Depot or both.

5.4 Service/Application Project Build

The role of the Service/Application Project Build tool is the creation of compatible executable services/applications that are to be integrated in the universAAL platform. Since the provider has created a compatible to universAAL service or application using the Project and Items Wizard tool, then an executable jar file containing the developed service/application is created and uploaded to the artefact repositories.

5.4.1 Relation to Eclipse

The current implementation of the Service/Application Project Build tool uses the following Eclipse extension:

Eclipse extension point	Description of provided realization(s)
org.eclipse.ui.actionSets	This allows the main functionality of this tool (building, running and debugging a project) to be accessible through the main toolbar of Eclipse (under the menu universAAL). The following Id's have been utilized: • org.universaal.tools.buildserviceapplication.actions.BuildAction (Action for building a universAAL project) • org.universaal.tools.buildserviceapplication.actions.RunAction (Action for running universAAL project) • org.universaal.tools.buildserviceapplication.actions.DebugAction (Action for debugging universAAL project)

Plugin features are accessible through the menu bar of the Eclipse plugin.

5.4.2 Design

The building process utilizes the Maven plugin for Eclipse IDE (m2eclipse) and its underlying Maven embedder API. The created project is a Maven project, where all jar dependencies are downloaded from the universAAL Maven repository to the local project. This process ensures that all dependencies are resolved with all the necessary compatible libraries.

When building universAAL compatible eclipse projects, a jar file is created and uploaded to the local Maven repository, where all the Maven data are imposed by the Project and Items Wizard tool. After a successful build, the jar can be used by other projects in the same computer, just by referring to the applied *Group Id*, *Artifact Id*, and *Version*. In order to be visible to all service providers, the jar should be deployed to the universAAL Maven repository (it is not implemented yet).

Since the Persona service model was adopted as an initial model in universAAL, the deployed jar file is a Persona OSGi bundle, which needs all the Persona middleware dependencies in order to run. This tool also has the capability to run or debug a new developed Persona project. This is done by automatically creating an Eclipse launch configuration file that starts up all the middleware bundles that are needed in order to run the new service/application.

5.5 Developer Depot Client

The Developer Depot Client is a tool in the universAAL tool-chain that is still in the early phase of inception. The overall idea for this tool is to provide simple access from the AAL Studio environment to the resources provided by the developer depot, and to simplify sharing of projects and reusable resources between developers. Functionality that may become part of this tool is:

- support for searching, viewing and reusing existing components and services, e.g. based on the use-cases they support;
- support for viewing and using the reference architecture actively during development;
- publishing of projects to the developer depot server.

5.5.1 Relation to Eclipse

In Eclipse terms, the Developer Depot Client is likely to provide several views, several commands and menu entries, probably one or more editors, and possibly a new Eclipse perspective to organize the work. The table below will be filled out in a later version of this deliverable, once the design of this tool has progressed.

Eclipse extension point	Description of provided realization(s)
org.eclipse.ui.commands	…
org.eclipse.ui.views	…

5.5.2 Design

A design description for this tool will be provided in a future version of this deliverable.

5.6 Domain Specific Language (DSL) Tool

In some cases plain UML is not feasible to use as a modelling language to describe processes, information or structures of a domain. Languages that are specialized for the target domain should be used. These domain specific languages (DSL) are defined using a metalanguage such as eCore in a metamodel. The universAAL AAL Studio will provide DSLs and editors for these where appropriate. Utilizing the functionality of EMF and GMF, an easy-to-use and robust modelling environment for designing AAL Services will be provided as a component on the Eclipse platform.

The universAAL DSL is a modelling language, a model notation that represents core concepts in the domain. The DSL allows the user to express valid domain structures such as an AAL service model.

5.6.1 Relation to Eclipse

The current realization of DSL Tool uses the Eclipse EMF/GMF editors.

Eclipse extension point	Description of provided realization(s)
org.eclipse.ui.newWizards	Create a new service model based on the servicemodel language
org.eclipse.ui.editors	A tree based editor for editing the service model

5.6.2 Design

Creating a new DSL is a daunting task, especially for a complex and multidisciplinary domain such as the AAL domain. The DSL provided in AAL Studio is a result of

1) Identify reusable, domain specific artefacts in the development process
    a. Map existing artefacts and structures to a model
    b. Review domain standards for concepts and structures
2) Identify desirable artefacts for code generation or documentation
    a. Code skeletons
    b. Test protocols
    c. Traceability reports
    d. Deployment descriptors
3) Develop the model and editor in Eclipse EMF/GMF
    a. Optional: create transformation scripts

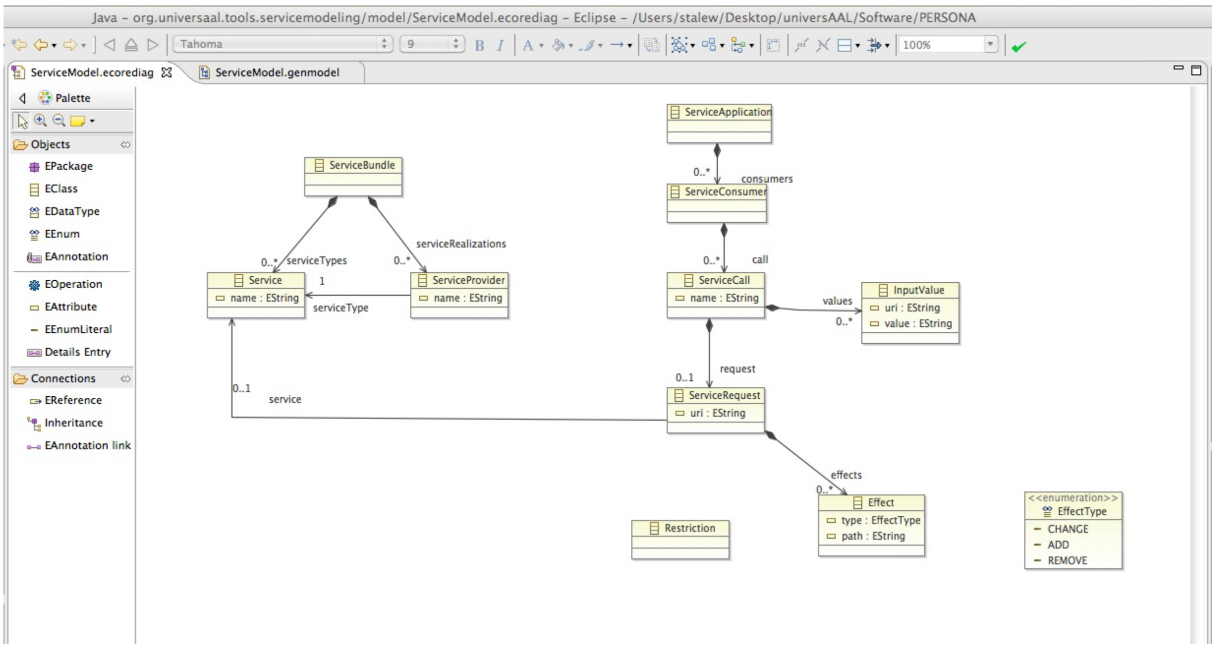


Figure 20 The first version of the AAL Studio service DSL

5.7 Modelling Tool

The AAL Studio modelling tool is a UML editor with domain specific UML profiles that are provided as an extension of the Papyrus tools for Eclipse. The developer can create both behavioural and structural design models for the application (domain) that can will be the basis for the actual implementation of the service (application or bundle).

A UML model can take the role of both system specification and documentation. The Object Management Group's MDA® approach describes a process that has three main phases:

1) Computation Independent Models (CIM): Describe the domain focusing on work processes, actors and information flow
2) Platform Independent Models (PIM): Describe the system components in terms of generic classes and structures.
3) Platform Specific Models (PSM): Describe the system in terms of concrete technology or platform specific components. Here, specific technological aspects are modelled. Many PSM models can be transformed from a rich PIM, e.g., a Oracle Database model from an information model in PIM.

To increase the expressiveness of UML, the AAL Studio Modelling tool comes with a set of UML profiles. Similar to the DSL model in the DSL Tool, these profiles contain elements that can be used to mark UML elements (such as class, component, actor, association) with stereotypes, tagged values or constraints. Figure 19 shows the initial uaal_ml UML profile.

5.7.1 Relation to Eclipse

The AAL Studio modeling tool uses the UML plugin that is provided to eclipse through the Eclipse Modeling Project.

Eclipse extension point	Description of provided realization(s)
org.eclipse.papyrus.extensionpoints.uml2.UMLProfile	The current version provides a simple stereotype with a two tagged values. One of the tagged values are of type enumeration.
org.eclipse.emf.ecore.uri_mapping	Maps to UAAL to simplify referencing.

5.7.2 Design

Creating a UML profiles is very much like creating a DSL. The main difference is that a UML profile extends an existing language (the UML) whereas a DSL restricts an existing language (eCore). Also, using UML profiles does not require a specific editor as almost all UML editors/frameworks support UML profiles.

SoaML(http://www.soaml.org) is being considered as a relevant profile for the AAL Studio modelling tool.

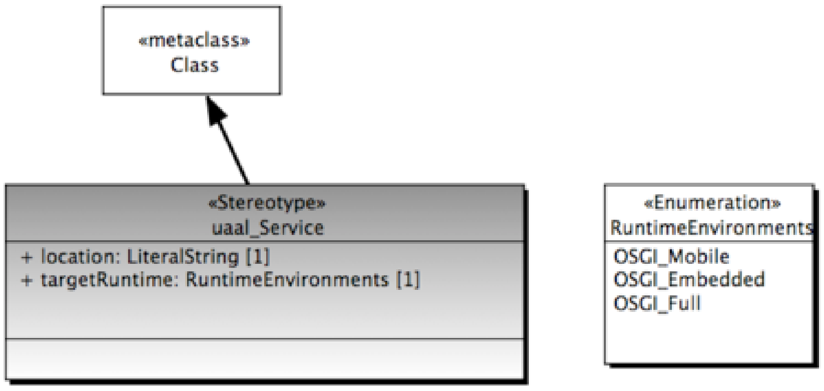


Figure 21 First version of uaal_ml profile

5.8 Transformation Tool

The transformation tool is a plugin that invokes transformation of a source artefact into a target artefact. The traditional example is a UML class model that is transformed into java classes. One source artefact can be transformed into several target artefacts; e.g., the class model can be Java Classes, a SQL script, and documentation in html and rich text format. This is achieved using different transformation scripts that reads the source artefact and output text as target.

Transformation can be two-way, but this is considered more complex than one-way (forward) transformation only. The current focus of the AAL Studio Transformation Tool is forward transformation.

5.8.1 Relation to Eclipse

The current transformation tool is provided as a command plugin for Eclipse. The commands can be invoked from the menu line and

Eclipse extension point	Description of provided realization(s)
org.eclipse.ui.commands	A command that starts the list classes transformation Id: org.universaal.tools…
org.eclipse.ui.handlers	To handle the command invocation. Will start SmallTest.mt2
org.eclipse.ui.menus	Adds universaal->transform to menu bar
org.eclipse.core.expressions.definitions	Used to evaluate the file type used as input to transformation. The file must have extension ".uml"

5.8.2 Design

The transformation command plugin design is in progress. The main questions when that must be addressed when creating a transformation tool are:

1) Which target artefacts are needed and what are common (elements) of these?
2) Which source artefacts are available and what is missing to be able to transform into the target artefacts?
3) How should the transformation be provided to be useful for the majority of source-target transformations?

The identification process related to questions 1&2 is in progress. Summarized the following activities have been identified as relevant for transformation candidates:

- Create services and define its service-profiles and available functions
- Create service calls that take use of the available services
- Define Context-Events and publish them
- Add listeners to available context-events

From these, the following transformations are to be investigated:

1) Tool for create services in form of Java classes including needed ontology extensions
2) Tool that help to create service-calls according the available services
3) Tool to create and manage (send on bus, check if it is available, ...) context-events
4) Parser for the service and ontology to change classes
5) Tool to export the ontology into OWL
6) Support transformations (by using external tools) between OWL <--> UML

5.9 Conformance Tool

This section introduces the conformance (or compliance) tool as important support in software development process. These kind of tools are actually adopted as further guarantee that the provided software implementation complies with a standard or a specification. For the purpose of universAAL project such test should ensure that the AAL service design "complies with the reference architecture and design principles" (as reported within DoW).

The purpose of this section is currently not to choose ready-to-use conformance tools, instead to introduce and investigate such working area within the universAAL project. For this reason, it will be provided to the reader an overall view for conformance test issues, by first introducing universAAL development tool chain section 5.9.1 and by providing the conformance definition section 5.9.2. Section 5.9.3 will provide an overall view of potential area of usage for conformance tool. In upcoming versions of this deliverable, selected tools will be described in Part I and Part II using the same approach as the for the other tools that are currently presented in these.

5.9.1 universAAL tool chain

The development process for AAL service should be approached by exploiting the features provided by AAL Studio. The role of the tool chain is to assist developers in design, develop, test and deploy the software artifacts. Figure 12 (page 21) reports an overall sketch of the building blocks designed with the AAL Studio. The conformance tool building block fits within AAL Studio and will offer to developers some tools able to guarantee the correctness of the approach during some of the application life cycle phases.

5.9.2 Writing conformance software

The conformance test aims at investigate how an implementation conforms to a standard or specification, in other word how the implementation meets the requirements. Conformance test exist only against a standard or a specification from which gather the requirements. ISO/IEC Guide 2[7] identifies three major terms in conformance test field:

- conformity - fulfilment of a product, process or service of specified requirements
- conformity assessment - any activity concerned with determining directly or indirectly that relevant requirements are fulfilled
- conformity testing - conformity assessment by means of testing

The conformance test aims not to measure the software quality by analysing e.g. design patterns, instead to measure the software adherence with respect to a standard or specification.

The optimal goal for these kind of test is to prove the software implementation with respect to *all* the requirements gathered from the specification. This approach, often, results unavailable and alternative solutions must be adopted. One of them is named *falsification testing* technic, this technic assumes to split the test into two phases. The first phase consists in the selection of a subset of relevant requirements and program input, also the expected outcomes are defined. The second phase consists in test the implementation against the condition defined during the first phase and check the results. If the test fails it is possible to claim that the software does not comply with the specification however the

[7]ISO/IEC Guide 2:2004 Standardization and related activities -- General vocabulary, http://www.iso.org/iso/catalogue_detail.htm?csnumber=39976

converse can not be assumed. For this reason as more requirements and inputs are selected during the first phase, as more accurate will be the outcome in the second one.

The general-purpose conformance building blocks are depicted in Figure 20.

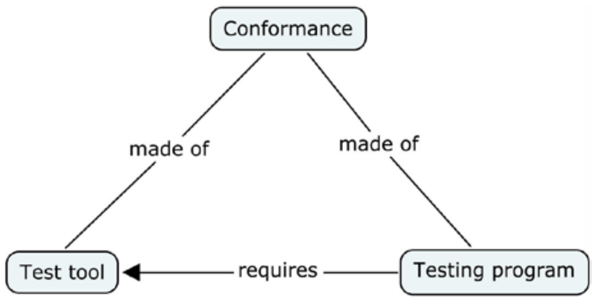


Figure 22 Conformance testing

The Conformance testing is made of two building blocks: the Test tool and the Testing program. The Test tools define which artifacts are needed to perform specific test, while the Testing program plans the whole test process. It's important to consider that the Testing program block can not exist without Test tools, however the converse is true that is the tools can be implemented also without the definition of a Testing program. This aspect avoids the conformance tool task to plan and define a Testing Program, that could introduce an unnecessary overhead.

More accurately the Testing program, usually, consists of the following items:

- Standard or specification
- Test tool (e.g. tool suite, and/or reference implementation)
- Procedures for testing
- Organization(s) to do testing, issue certificates of validation and arbitrate dispute.

The conformance testing policies and procedures should de designed in order to explain the general intention to the tester. The description includes the documentation of how the testing is to be done and the directions for the tester to follow. The documentation should be detailed enough so that testing of a given implementation can be instrumented and repeated with no change in test results. The procedures should also contain information on what must be done operationally when failures occur. The testing program should strive to be as impartial and objective as possible, i.e., to remove subjectivity of both the procedures and the testing tools.

Not all industries accept to invest part of the overall development process time for testing issues, in these cases no test are planned before software releases. Other industries relying on testing business units while other ones on third-party collaborations.

As far as the testing program is concerned, one notable example is the Java Community Process (JCP)[8]. JCP program, started from Sun Microsystems, aims at involve industries and individual experts for the definition of new Java Specification Request (JSR) made of the following contributions:

- technology specifications
- reference implementation (RI)
- technology compatibility kit (TCK).

The JSR flow follows four major steps one of them concerns the design and development of the TCK. The intention of TCK is to release a suite of test, tools and documentation able to guarantee that the software implementation adhere to the JCR. Sun Microsystem also provides Java Compatibility Test Tools (Java CTT) designed to assist developers in TCK design and implementation. Both TCK and

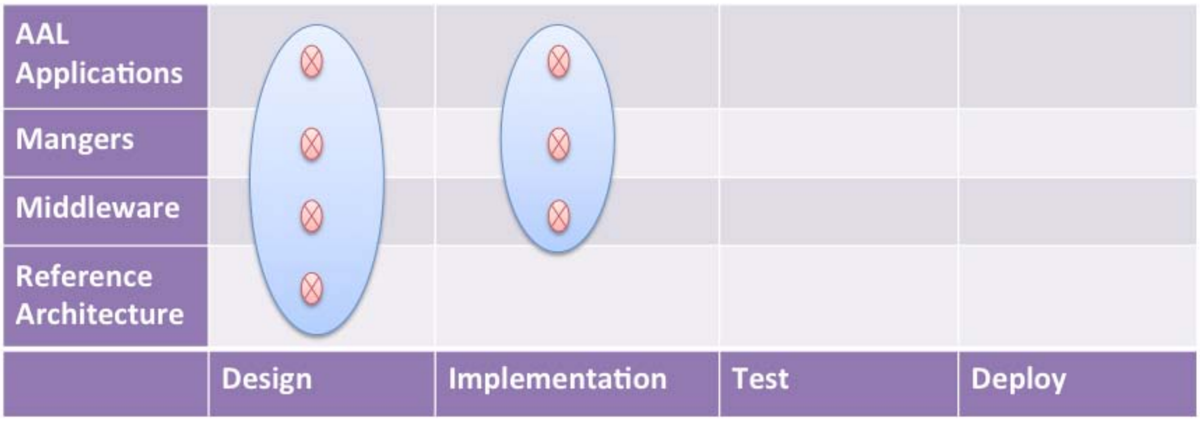
[8] Java Community Process (JCP), http://www.jcp.org/en/home/index

Java CTT highlight the importance of third-party software able to claim the software implementation conformity.

5.9.3 Matrix of interests for universAAL conformance tools

According to the definition previously introduced, it's possible to plan a broad range of conformance tools for universAAL platform. Table 1 reports the spectrum of possibilities for the development of conformance tools. The matrix has been obtained by combining the different development phases (matrix columns) with the layers of the universAAL platform (matrix rows) where such tools could be adopted.

Table 1 Matrix of interests



	Design	Implementation	Test	Deploy
AAL Applications	⊗	⊗		
Mangers	⊗	⊗		
Middleware	⊗	⊗		
Reference Architecture	⊗			

Conformance tools can embrace one or more layers of the reference architecture by fully or partially filling out the matrix. As an example conformance tools will be able to claim different properties among the layers:

- the reference architecture is *designed* according to the reference model
- the middleware *implementation* meets the reference architecture
- the *unit test* for a Manager (e.g. Context-Manager, Activity-Manager) completely stress the component
- AAL applications correctly adopt the *deployment* procedure

The above cited examples highlight that conformance tools can be designed for one or more development life-cycle steps and for one ore more layers of the universAAL reference architecture. Moreover the conformance tool task will be fully autonomous only after the complete acceptance of the reference architecture from the universAAL consortium. This will give to the conformance task the chance to discuss about the design and implementation of tools able to check properties related to the upper layers.

For this project phase only the design and implementation columns are investigated, and the circles indicate the area of interest that will be further investigated. It is intention of next deliverable releases to assess which of the columns and rows will be subsequently considered. The following two sub-sections describe examples of the kind of tools that could be implemented during the design and implementation phases.

5.9.4 *Conformance testing for the design step*

Task 3.2 is working on the definition of a meta-model for design AAL services (Domain Specific Language (DSL) ToolModelling Tool). The design step will be approached exploiting a new UML profile and/or a custom DSL definition. In both of the cases EMF framework will be adopted. One of the EMF components is called the Validation Framework[9], the EMF Validation Framework provides a generic and extensible framework for defining constraints on EMF meta-models and for checking models against those constraints. The Validation framework offers to developers some features for the definition and implementation of custom checks:

- provides API for defining constraints for any EMF meta-model
- provides support for parsing the content of constraint elements defined in specific languages (Java and OCL)
- provides API support to define "client contexts" that describe the objects that need to be validated and to bind them to constraints that need to be enforced on these objects.

5.9.5 *Conformance testing for the implementation step*

The conclusion of the design steps gets off the implementation one. During the implementation process developers are involved in coding the software, here ad-hoc AAL Studio plug-ins should be introduced in order to guarantee that the code adhere to some constraints:

- naming convention check: the plug-ins provide support to check convention infringement in naming definition
- programming style check: the plug-ins provide support to adopt specific convention in coding. For example correct use of brackets, white spaces, indent, programming statement layout check, programming pattern adoption
- programming problems: the plug-ins provide support to find dead code, incorrect logical statements, duplicate code or tenuous code
- test definition: the plug-ins can force unit test coding for every e.g. developed class or developed package.

Some available tools that can be easily integrated within AAL Studio are reported below:

- PMD[10] tool
- Maven check style tool [11]

[9] http://www.eclipse.org/modeling/emf/?project=validation
[10] PMD http://pmd.sourceforge.net/
[11] Maven check style, http://maven.apache.org/plugins/maven-checkstyle-plugin/

6 Source code build instructions

To build the tools described in this document, you need an Eclipse installation set up with the prerequisites for the universAAL tools that are described in Part 1 of this deliverable, under “Installing the AAL Studio (Eclipse based tools)”. Link to the wiki page:

http://a1gforge.igd.fraunhofer.de/mediawiki/index.php?title=uaaltools:Installing_the_AAL_Studio

Once the external prerequisites are installed, the source code of the tools will build automatically when downloaded into Eclipse from the Subversion repository. The code is available from the tools project under the universAAL GForge server:

http://a1gforge.igd.fraunhofer.de/gf/project/uaaltools/