
 <p>SEVENTH FRAMEWORK PROGRAMME: PRIORITY 7.1B LARGE SCALE INTEGRATING PROJECT (IP)</p>	IP project number 247950      Project duration: February 2010 – February 2014 Project coordinator: Joe Gorman      Project Coordinator Organisation: SINTEF, Norway Strategic Objective: 7.1.b      website: <a href="http://www.universaal.org">www.universaal.org</a>	
	  <b>Universal Open Architecture and Platform for Ambient Assisted Living</b>	
<b>Document Type:</b>		Project Deliverable, with independent sub-parts. <i>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.</i>
	<b>X</b>	Project Deliverable (single document, no sub-parts).
		Sub-part of a Project Deliverable.

<b>Document Identification</b>			
Deliverable ID:	<b>D2.3-A</b>	Deliverable title:	universAAL integration and testing strategy and issue tracker
Release number/date:		V1.0 2010-06-12	
Checked and released by:		Sergio Guillen/ITACA	

<b>Key Information from "Description of Work" (from the Contract)</b>	
Deliverable Description	<i>Definition of tools and procedures for integration strategy, software management, issue tracker, bug management, risk management.</i>
Dissemination Level	PU=Public
Deliverable Type	R=Report
Original due date (month number/date)	Month 3 / 30.04.2010

<b>Authorship &amp; Reviewer Information</b>	
Editor (person/ partner):	Alvaro Fides/ITACA
Partners contributing	CERTH, Prosyst, Fh-IGD, ITACA, UPM, CNR, ENT
Reviewed by (person/ partner)	Peter Wolf/ FZI

## Release History

Release number	Date issued	Milestone *	eRoom version	Release description /changes made
	23.02.10	ToC	1	First Table of Contents
	26.03.10	PCOS Proposed	2	Table of Contents agreed by working group, with brief section descriptions
	31.03.10	PCOS Approved	3	Internal reviewers comments applied to PCOS Proposed
	13.04.10	Intermediate Proposed	4	Gathered partners contributions to compose Intermediate proposal (approximately 50% of content)
	20.04.10	Intermediate Approved	5	Applied comments that derive on minor changes. Deeper changes and/or additional content will be applied in External Proposed.
	26.04.10	External Proposed	6	Gathered partner's contributions to compose the 100% of the deliverable content. (Annex pending)
	28.04.10	External Reviewed	7	Applied comments from review and added the annexes
	20.05.10	External Revised	8	Applied comments from technical meeting. Reordered section 3. Rewritten Risks section and integrated into section 7. Rewritten and reordered section 7.
	30.04.10	External Approved	9	Minor corrections. Reference section changed to footnote
1.0	12.06.10	Released	11	Minor editorial corrections

\* 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 7<sup>th</sup> Framework Programme, Priority 7.1.b (ICT & Ageing). The consortium members are:

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

## Table of Contents

Release History.....	2
universAAL Consortium .....	3
Table of Contents .....	4
Table of Figures.....	6
List of Tables.....	6
1 Executive Summary.....	7
2 About this Document.....	8
2.1 Role of the deliverable.....	8
2.2 Relationship to other universAAL deliverables .....	8
2.3 Relationship to other versions of this deliverable .....	9
2.4 Structure of this document.....	9
3 Initial Assumptions and universAAL Development Requirements .....	11
3.1 Development process requirements.....	11
3.1.1 Requirements to create a successful software project.....	11
3.1.2 Requirements to create an open source software community .....	12
3.2 Generic assumptions and requirements .....	13
3.2.1 Selection of programming framework.....	13
3.2.2 OSGi framework .....	14
3.2.3 Maven project management and build automation .....	15
3.2.4 Eclipse IDE.....	15
3.2.5 Development teams .....	16
3.2.6 Documentation .....	16
3.2.7 Implementation in other frameworks .....	17
4 IPR Management.....	18
4.1 Tools.....	19
4.1.1 IPR directory .....	19
4.2 Regulations, guidelines and good practice .....	19
4.2.1 Software license in brief.....	19
4.2.2 The license used by universAAL .....	21
4.2.3 How to apply a license .....	21
4.2.4 Disclaimer and notice file examples.....	21
5 universAAL Development Environment and Coding Standards .....	24
5.1 Specific requirements .....	24
5.2 Tools.....	24
5.3 Regulations, guidelines and good practice .....	25
5.3.1 Guidelines for managing a group .....	25
5.3.2 Guidelines for developers.....	25
5.4 Code conventions .....	25
5.4.1 Code styling.....	26
5.4.2 Code documentation.....	26
5.4.3 Checkstyle and Maven .....	26
6 Project Management Tool and Issue Tracker.....	27
6.1 Specific requirements .....	27
6.2 Tools.....	27
6.2.1 General features.....	28
6.2.2 Eclipse integration.....	28
6.2.3 Issue tracker.....	28
6.3 Regulations, guidelines and good practice .....	28

6.3.1	Project planning.....	28
6.3.2	Handling of technical problems.....	29
7	Integration Strategy .....	30
7.1	Release management .....	30
7.1.1	Initial specification .....	31
7.1.2	Alpha specification.....	31
7.1.3	Beta specification .....	32
7.1.4	Final specification .....	32
7.2	Integration strategy of software components .....	32
7.2.1	Candidate survey .....	33
7.2.2	Positioning.....	34
7.2.3	Planning.....	35
7.2.4	Completing & Binding .....	36
7.2.5	Stability test.....	37
7.2.6	Accepted .....	38
7.2.7	Archived .....	39
7.2.8	New development.....	39
7.3	Integration risks.....	39
8	Testing and Deployment Strategy .....	43
8.1	Specific requirements .....	43
8.2	Tools .....	44
8.3	Regulations, guidelines and good practice .....	46
8.3.1	Unit testing .....	46
8.3.2	Automated Test Generation.....	47
8.3.3	Code Coverage .....	48
8.3.4	Load testing .....	48
8.3.5	Deployment .....	48
8.4	universAAL deployment strategy.....	49
9	Future Work.....	50
Appendix A. Quick Developer Reference.....		51
A.1	Setup Instructions .....	51
A.1.1	Generic .....	51
A.1.2	IPR.....	51
A.1.3	Development environment .....	51
A.1.4	Coding standards .....	51
A.1.5	Project management and issue tracker.....	51
A.1.7	Testing .....	52
A.1.8	Deployment .....	52
A.2	Summary of selected tools and responsible people.....	53

## Table of Figures

Figure 1: OSGi layer model .....	14
Figure 2: Example artifact procurement.....	15
Figure 3: Eclipse and developers.....	16
Figure 4: Main open source license categories.....	20
Figure 5: Source file header with the license disclaimer.....	22
Figure 6: Notice file example .....	23
Figure 7: Integration Process.....	33
Figure 8: Candidate survey stage .....	34
Figure 9: Positioning stage .....	35
Figure 10: Completing stage .....	37
Figure 11: Release process in the development stages.....	38
Figure 12: Possible outcomes for a universAAL Compliant Component.....	38
Figure 13: Testing cycle .....	44
Figure 14: JUnit.....	46
Figure 15: JUnit Test Case .....	46
Figure 16: Simple Test Case.....	47
Figure 17: EclEmma.....	48
Figure 18: Grinder's console .....	48

## List of Tables

Table 1: Simple comparison between different software frameworks.....	13
Table 2: Feature comparison between Scripted HTML and Wikis .....	17
Table 3: Release month for components in first and final set.....	31
Table 4: Risks during the component integration phase .....	41
Table 5: Testing tools .....	45
Table 6: Complementary testing tools.....	47

# 1 Executive Summary

This deliverable deals with the integration of components from other projects (or newly created) and their testing, for which a set of tools and development guidelines are studied and selected. These tools and procedures set up a development process that each piece of software must follow to be integrated into universAAL, ranging from the very inception of the process, selection of existing components, code development, testing and deployment.

This document first starts with the common decisions a software project must make at its beginning, like deciding the code language and running framework. For universAAL, Java and OSGi have been selected due to their usage in most of the input projects. In international projects like universAAL, with so many different partners, it is necessary to use tools that help in synchronizing the work of developers when coding, building and documenting the software. In this aspect, we will use Maven for building, GForge for development management, Subversion for code sharing, and scripted HTML and Wiki documentations. A set of requirements should be addressed throughout the whole process in order to not only properly manage the project but also make it a successful open source one.

The IPR management section gives recommendations and advice about how to decide for a license for the software developed at universAAL, despite it is not its purpose to decide which one must be use. It also advices on how to behave when using third-party software, as this may have different licenses, and defines a procedure and registry to deal with this issue. Lastly, it gives guidelines to follow when writing code in regard to the license applied.

Development Environment section specifies how to properly use Eclipse IDE, by installing some plugins that help developers maintain an integrated interaction with the rest of tools and guidelines, namely Subclipse, ADJT, M2Eclipse and Pax runner. Other tools and advices are commented related to style and code documentation.

Project Management and Issue Tracker is a section that introduces GForge as main managing tool for the development of code. It covers a wide range of features of project management, from version control to documentation and, one of the most important, includes an issue tracker to keep trace of the appearing bugs and issues and manage them as they are solved.

The Integration Strategy defines the process through which an external software component is adopted into universAAL reference implementation, by passing different approval stages, adaptation and evaluation. In the case of a new development, it describes the path it would follow in an analogous way. Responsibilities are adjudicated for each of the major releases in universAAL, by describing how to lead the release plan.

Integration Strategy also contains the risks of its process identified so far along with possible mitigation strategies.

The Testing and Deployment Strategy describes the tools and techniques used for automated code testing that developers should follow (JUnit, for instance, among others), along with the tools selected for deployment (Nexus)

## 2 About this Document

### 2.1 Role of the deliverable

According to what is expressed in the Description of Work, this deliverable tries to cover the following aspects:

- An **integration strategy** that describes the process by which external components from other projects or newly created ones are integrated into universAAL.
- A **testing regime** for the approval of such integrated components.
- **Monitoring** and planning the development, integration and testing process throughout the **technical work packages**, therefore involving partners from WP1, 2, 3, 4 and also 5 for evaluation.
- Describe the tools and procedures to **document** the whole process, from writing raw code to release management.
- Define a **strategy** for identifying and cataloguing **risks**, and find and classify their possible **mitigation** strategies.
- Select a tool for **tracking all possible bugs**, classify them and assign responsibilities to solve them. This bug resolution process can be monitored to track the status of the resolution (and already solved) of the bug.

In addition to what was expressed in the Description of Work, this deliverable determines the development environment and associated tools that will be used by code developers, as these tools and the environment provide features that cover many of the objectives for the deliverable stated before, like, for instance, documentation or issue (bug) tracker.

In order to ensure the success of universAAL open source results, it is important that universAAL offers a development environment that has the most relevant tools in a professional software management project and that universAAL uses the most common adopted tools in order to capture the interest of external developers.

### 2.2 Relationship to other universAAL deliverables

- **D1.2 Reference Architecture Requirements:** This (D2.3) deliverable includes requirements for a successful process management that could be considered as part of the requirements of the project. However these requirements are not specific to the objectives of the universAAL scope, but related to any kind of software project and therefore no relation will be established from within D1.2, though it could be done in the future. Requirements from D1.2 are also used as input in the integration process.
- **D1.3 universAAL Reference Architecture:** D1.3 is used as input in the integration process for matching external components from other projects to components in universAAL.
- **D3.1 Tools for Design and Development:** References are made in this document that point to the work reflected in D3.1 about selecting a tool to implement the Developer Depot. Its outcome may affect the selection of tools in this deliverables' section "Testing and Deployment Strategy".
- **D3.3 Conformance tools:** Tools described in D3.3 will help in the integration process to check compatibility of platform components with the Reference Architecture.
- **D3.4 uStore:** References are made in this document that point to the work reflected in D3.4 regarding the implementation of uStore. Its outcome will affect the selection of tools and procedures in this deliverables' section "Testing and Deployment Strategy"



- **D5.1 Technical Verification Plan Definition:** This document presents an evaluation plan that defines evaluation criteria and procedures that must be considered in order to conduct scientifically sound evaluations
- **D9.3 Exploitation Plan:** This document includes detailed information about the IPR Directory and the right procedures for the IPR management. Content from D9.3 will specify tools regarding IPR management that can be referenced in this document, or may modify it.

## 2.3 Relationship to other versions of this deliverable

As this is the first version of this deliverable we can here only foresee that further versions will include additional content on aspects that could not have been planned at the first stage of the writing of this first version. As development tasks start, there is a chance that new or different tools have to be adopted than those presented here.

Also new versions will deal with the content foreseen on the Future Work section at the end of this document.

## 2.4 Structure of this document

This document is structured so it tries to follow the selection of development tools and decisions to make from the most generic to the most specific, and from first stages of code development to the very last ones. Therefore its sections follow the path: Environment – License – Coding – Bug fixing – Testing. This ordering doesn't follow any standard but common sense and experience on previous projects based on the usual set of tools used on these. There is also an alteration of this order by including an additional section that is central to this document, the Integration Strategy, which is placed in its respective order as will be explained below.

The content starts with a section that focuses on two initial aspects to consider when starting software development. One of them is a set of common requirements that must be addressed throughout the development process. The other is the selection of the main software characteristics of the project, such as language, running framework, and their versions.

The following section covers license adoption following the order expressed at the beginning of this section. This section is very straight-forward and didn't need special structuring.

Next section describes the setup of the development environment and coding guidelines. These two aspects were bond into the same section due to the strong relation of proper code writing and the selected development environment, as all IDE's include format and style tools. Like any section in which a set of tools is selected, it is divided into subsections for requirements of the tools, list of available tools, and hints on how to uses the selected tools.

The following section, Project Management and Issue Tracker, selects the tool(s) for managing the code of the project among different partners and the tracking of the bugs found. Again two aspects are included into the same section because of the possibility of the selected tools for Project Management to handle the Issue Tracking with their own features. This section is also divided into requirements, tools and guidelines.

The next section, Integration Strategy, doesn't follow the path expressed previously but it is included here since the development environment is already set in the previous sections. It contains one of the main purposes of the deliverable: how to integrate external component. It is divided into three sections. The first one describes the plan to follow on each release period by code developers, the second one describes the integration process itself, and the last one identifies the risks to encounter.

After this, the Testing Strategy section is presented. It has been placed after the Integration Strategy because its tools are the last ones to be used, both in the code development chain and in the integration process. Since it is a tools section, it follows the requirements/tools/guidelines structure.

The Future Work subsection lists a set of points that will have to be addressed in following versions of this deliverable.

The annexes of this document are thought as a quick reference for developers to find, in the first one, instructions and links to each tool described here, and in the second one, a summary of the selected tools, the reasons to use them, and a responsible contact person for each in case of doubts or issues.

## 3 Initial Assumptions and universAAL Development Requirements

Building large and complex software, in different teams spread in different countries is a task that needs very good management. The entire software development lifecycle must be organized, supported by proper tools and continuously monitored. For this reason it is of utmost importance to agree on a common process management, as well as a common set of tools that is to be used by all partners involved. This chapter showcases the solutions chosen for the universAAL project. The choice was based on the state of the art of programming tools, as well as previous experience of the partners, gained in preceding projects. Additionally, a set of good practice for successful software project management has been added.

### 3.1 Development process requirements

A good selection of development tools is not enough to create a successful open source project. Management of the project and basics before to start the development are necessary. In the case of universAAL, in which the development community will determine the success of the project, the correct definition, management and control of forces involved in the development is a key point. This section pretends to establish the initial points to be in account to start the development of universAAL platform. Future version of this deliverable will deal with how universAAL faces these requirements.

#### 3.1.1 Requirements to create a successful software project

There are some common features for every successful software project, no matter the programming language or the development environment is used.

Any software project has to rely on common basic procedures in order to be successful, regardless of programming language and development environment:

- The project is based on the user's needs and requirements.
- There is a well defined set of requirements established, determined by evaluating necessary features with potential users.
- There is a methodology for requirement specification and developers strictly try to implement those.
- The developers are using a shared development environment and tools. If necessary they should be trained to get the necessary experience with said applications.
- There are several milestones within the project timeframe, determining a certain status the software should have at specific times.

Therefore, the development team should control various elements to finish successfully a software project:

- universAAL is a large project, so the software needs to be divided into specific parts according the project architecture. Each of these parts will be independently executable, clearly defined, use well-defined, minimal interfaces and be developed by a small team that should not exceed two or three developers. universAAL does not have a single user group. The platform should be functional under a wide set of user's needs and divergent operational conditions. The development teams have to understand the user requirements. For that, a clear and deep system specification is need.
- Regarding the development technology and tools, the development team should agree to use the same development environment, as well as they should be familiar with all its tools and resources. Effectiveness and appropriateness are desirable in the selected tools.

- Once the project starts, collaboration between partners is essential. This collaboration should include technical support for developers and should be guaranteed by training or mentors.
- A version control system must be used such as Subversion to follow up the different code version. The developers must undertake to properly use this system. To maintain a change log file is useful too.
- Following the project target, the developers must use the existing code, libraries or software from the previous projects. Don't waste time developing things that have already been developed.
- Validation and testing strategies well-defined in order to avoid integration problems at the end of the project.
- Documentation of the project is a key point to finalize successfully a project. This document must contain a short description of what the system does: a list of all user requirements, with examples of the use cases, overview of the interfaces and architectural concepts, detail interface specification, a small installation guide and the user handbook.

### *3.1.2 Requirements to create an open source software community*

The main step to create a successful open source community is a clear differentiation from other existing communities. The objectives should be clear, not only for the initiators of the project, but also for future developers and other project participants. Concrete the efforts to encourage developers and editors, despite of they will be the smallest percent of the members a community, most of the members will be only watchers, passive members of the community. It is a valid method of participation, but active member must be encourage and take over.

When starting a new open source project it is fundamental to encourage participation and create a working network between all developers:

- In order to grow an open source community needs to be well-known, Therefore universAAL needs to use the contacts established by all project partners to promote the projects
- A community can be located at:
  - On its own site
  - As part of a larger site with unrelated projects
  - As part of a larger site with related projects

But a successful community needs to be presence in all three. The objective is to extend the human social networks and to obtain beneficial contributions. Social networks are a good way to expand the community. Facebook, Twitter and other communities can facilitate acquiring new members, simplifying contact between them and provide a way to publish community alerts

- Large projects are difficult to manage. Because of this, it is better to start small and work into great complexity. But an expansion plan must be created for future.
- Features of popular communities:
  - Discussion groups or forums are successful tools to discuss, collaborate and give help to each other.
  - Wikis are not impressive, but could be useful for self-learning.
  - Software repositories are completely necessary for shared software. Methods of distribution, collaboration, conflict avoidance and access control must be defined.

- Documentation is essential to a successful community. A good and complete documentation allow those who lack technical skills to contribute in the community so the potential developers increase. Usually wikis are the vehicle for community-based documentation.
- A content management system (CMS) integrates administrative functions, login, page management, etc. There are open source projects that can be adapted to the specific features of our community.

All the process to create the community should be managed by a single person or a small team. They will lead the community, encourage participation and guide the community towards the target. The management team must have good communication skills, vast knowledge and a talent for guiding people. They must establish clear and firm rules to handle unusual or challenging situations or disagreement situations, but being in account disagreements could be a good opportunity to solve problems if the leader controls the situation.

## 3.2 Generic assumptions and requirements

When starting a project it is a necessity to define certain initial parameters on which the participants have to rely when evaluating more specific tasks. In case of software development there needs to be consensus on the programming language, framework and tools used. While the standard procedure is the evaluation of suited utilities this approach is not entirely suited for universAAL. A target is to consolidate various preceding project into a generic framework. The integration process, which requires the conversion of software, is simplified if the target is using similar frameworks similar to the base projects. This aspect is granted increased attention in the following chapter.

### 3.2.1 Selection of programming framework

The universAAL middleware is required to run on a large number of possible devices. Therefore it is necessary to use a programming framework that is readily available on numerous platforms, be it embedded systems, fully equipped personal computers or mobile devices.

The largest platforms for cross-platform programming nowadays are Java<sup>1</sup> and .NET<sup>2</sup> for PCs and Java ME<sup>3</sup> for mobile devices. Each of those frameworks requires a virtual machine (VM) running on the target system that interprets the intermediate code generated by a compiler.

	Java	.NET	Java ME
<b>Supported Languages</b>	Java, Jython, JRuby...	C#, VB.NET, C++/CLI...	Java, Jython, JRuby...
<b>Supported OS</b>	Windows, Mac OSX, Linux, Solaris	Windows	Various Mobile and Embedded OS
<b>VM Capabilities</b>	Extensive hardware access	Extensive hardware access	Minimal subset of VM

**Table 1: Simple comparison between different software frameworks**

Concerning an integrating project like universAAL, the selected framework should be compatible to those of the preceding projects. Having a common platform simplifies integration tasks considerably and prevents having to rewrite large parts of the source code. Almost all of the development work in the input projects has been done in Java. Considering also the wide-spread availability of Java on a

<sup>1</sup> Created by Sun Microsystems in 1990

<sup>2</sup> Created by Microsoft in 2000

<sup>3</sup> Java Mobile Edition

large number of devices, the logical consequence for universAAL is to choose the Java framework as the default programming language and runtime.

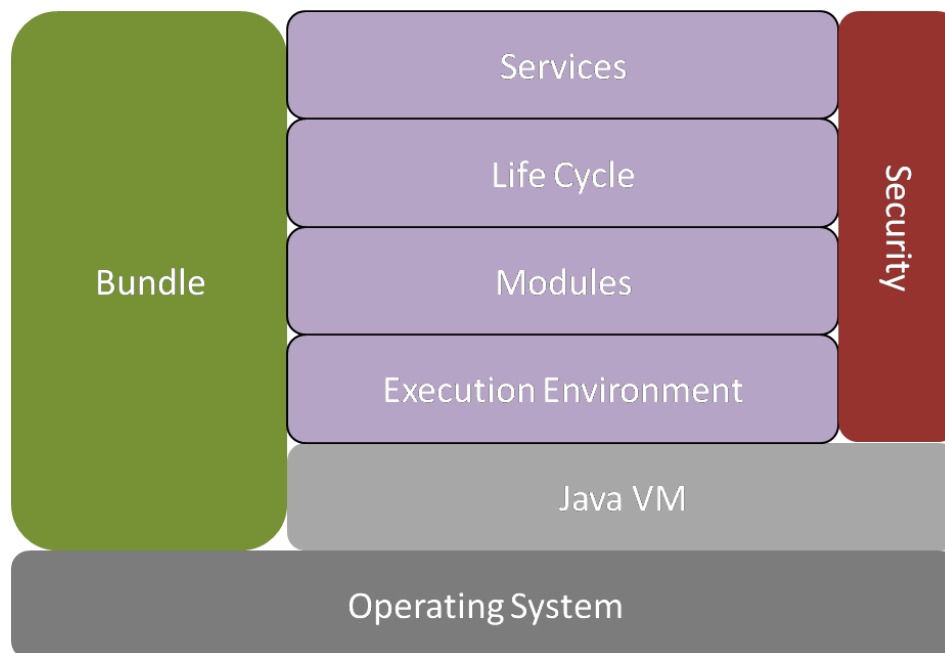
Java is available in various editions, including the already mentioned Java ME. Each of those editions has developed considerably throughout its lifetime, resulting in consecutive versions having different compatibility levels.

- Java 1.4 – Released in 2002
- Java 5 – Released in 2004 – Last version compatible with Windows 98
- Java 6 – Released in 2006

While newer VM versions are able to run projects created in previous versions the opposite usually does not apply. It is important for universAAL to support the largest number of available devices, including systems running on old operating systems and embedded devices that might also rely on older cores. Consequently it was decided that Java version 1.4 will be used for the universAAL.

### 3.2.2 OSGi framework

OSGi<sup>4</sup> is a service platform and module system for Java. It implements a component model that allows objects to be remotely started, stopped, installed, uninstalled or updated without requiring a reboot. Applications and components are deployed as bundles, packages of classes, libraries and resources that can import and export packages that can be consumed by other bundles. Furthermore OSGi implements a life cycle layer that handles the dynamic management of bundles and a service layer where a bundle can provide services – Java objects that implement an agreed interface. This basic layer structure is shown in the following figure. Bundles can have a variable amount of abstraction – between hardware-near development and service level implementation.



**Figure 1: OSGi layer model**

universAAL and several of its predecessors are relying on a service-oriented approach in software engineering that allows simplified modularization of components, implemented by distributed development teams. The different groups are able to develop components independently if services

<sup>4</sup> <http://www.osgi.org/>

and functionality are determined instead of having to rely on uncertain class hierarchies or libraries. While the preceding projects have been using different approaches, e.g. SOAP (MPOWER), most projects used the OSGi service platform (PERSONA, Soprano, and Amigo). Since there are also means to connect OSGi and SOAP<sup>5</sup> it was decided to use an implementation of the OSGi service platform for universAAL. The most recent version, which will be used in universAAL, is R4.2.

### 3.2.3 Maven project management and build automation

Large software projects created by distributed groups of developers can easily lead to source code that is relying on a vast number of imported packages that have to be individually integrated into the project of all developers. This can lead to situations where a developer has to spend a considerable amount of time to get all required packages and settings to get the build working on his machine, before he can do actual work. In order to mitigate this issue it is necessary to use utilities that allow for an easy, preferably automated build process that store packages and settings remotely.

Maven<sup>6</sup> is as much a build-automation and project management tool for Java as it is a set of guidelines to develop software projects that use a simple and generic structure and are therefore easy to understand for any developer familiar with Maven.

The main concepts are configuration based on a POM (Project Object Model) file that stores build and project settings and automated structuring based on software development best practices including setup of unit testing. Furthermore it is possible to automatically generate useful information about the software, e.g. change logs and dependency lists. The POM allows Maven to reference remote repositories and automatically download resources that are called artifacts. Accordingly it is possible to automatically deploy generated artifacts to global repositories. Used artifacts are cached locally and updated dynamically, minimizing the issue of having the same versions installed for each developer.

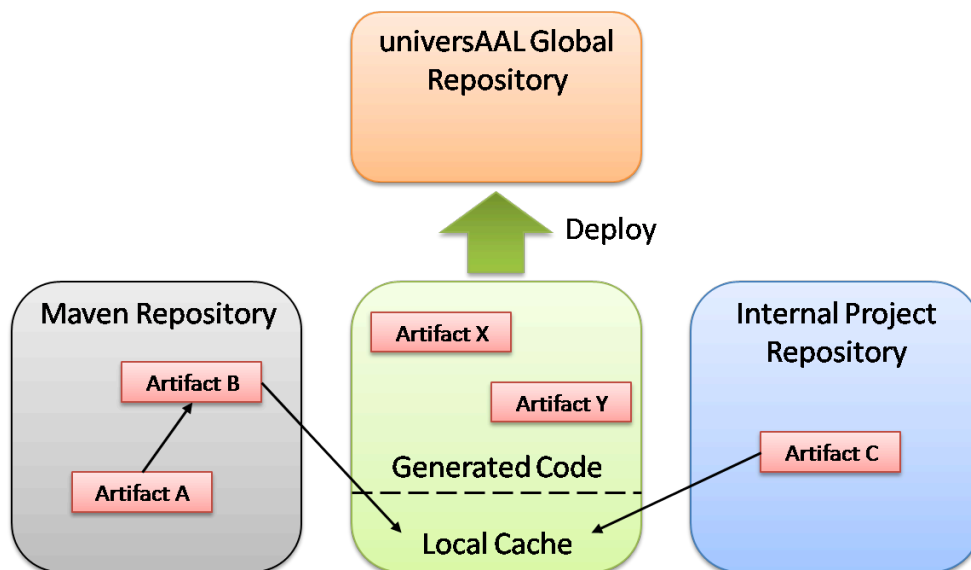


Figure 2: Example artifact procurement

### 3.2.4 Eclipse IDE

One of the main targets of universAAL is to create a community of developers and service providers that will work together on enhancing the system and adding new functionality. In order to allow developers to work with the universAAL middleware they have to be provided with a development

<sup>5</sup> <http://xf.apache.org/distributed-osgi.html>

<sup>6</sup> <http://maven.apache.org/>

environment they can work in. This framework should be open, wide-spread and easily adaptable, in order to achieve a broad adoption and consequently a large and active community.

Eclipse<sup>7</sup> is the most popular software development environment for Java. It is an open source project that provides an integrated development environment (IDE) together with a large amount of components adding certain functionality, called plug-ins. Concerning universAAL development there are plug-ins available for Maven, that further simplify the creation and life-cycle management of those projects. Therefore it was chosen to use Eclipse as internal development environment and publish the results as an Eclipse plug-in that developers can use to start working with universAAL.

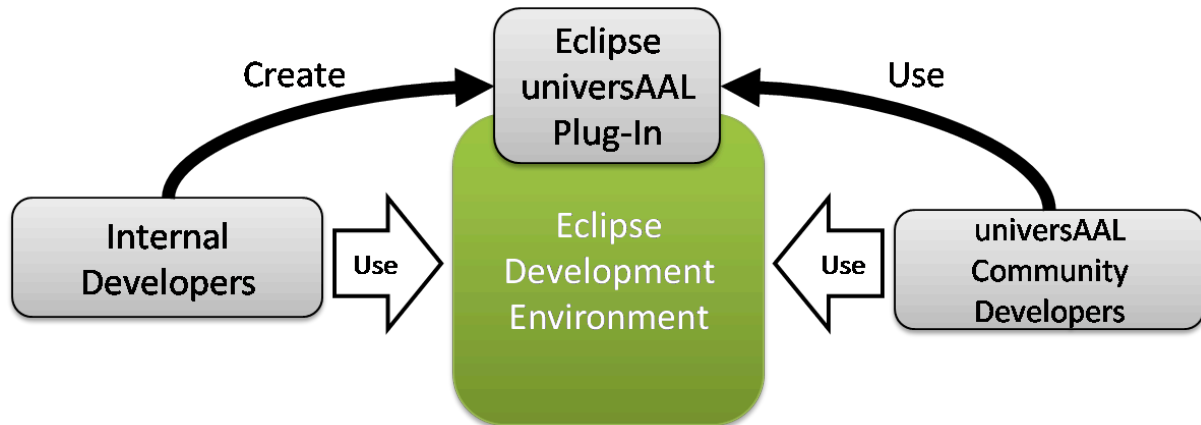


Figure 3: Eclipse and developers

### 3.2.5 Development teams

The universAAL project is consisting of many partners. Accordingly there will be multiple groups of developers that develop the universAAL framework in parallel. In international projects members of the different groups might operate in different geographical locations. Therefore it is necessary to provide them with utilities for communication, sharing code and documentation, as well as time-management and possible surveys. On a project-scale universAAL is using the eRoom<sup>8</sup> system for storing contacts, calendars and created documents. The developers are provided with an additional collaborative software development utility called GForge<sup>9</sup>. This utility combines a code repository with additional forums and wikis, helping the developers on software-related tasks. The server administrator has fine-grained access control and can grant individual users or groups thereof permission to either read or write certain aspects, respectively allow them to administrate projects on their own.

It is required that all developed components use defined interfaces and are well documented, in order to minimize potential issues when integrating the subprojects into the main framework. While Maven simplifies these routines, it is also necessary for all developers to follow the guidelines presented in Code Guidelines section.

### 3.2.6 Documentation

One of the most important aspects in collaborated software development is proper documentation – not only of the code itself but also of the implementation process itself. It is required that all interfaces and classes are properly documented and that all fixed bugs and added features between different versions of the software.

<sup>7</sup> <http://www.eclipse.org/>

<sup>8</sup> <https://project.sintef.no/eRoom/>

<sup>9</sup> <http://www.gforge.org/>



Concerning large projects it is important that the documentation is readily accessible for everyone involved and potential contributors from outside. Consequently an online format should be used, that has the added advantage of supporting hyperlinks that considerably simplify browsing through source code documentation. Currently the most popular methods for online documentation are HTML-pages that are either created automatically using scripts, being deployed manually to a web-site using a CMS system or user-generated wikis.

	<b>Scripted HTML</b>	<b>Wiki</b>
<b>Layout</b>	Individual Layout	Generic Layout – limited Skin support
<b>Features</b>	Interactive elements - feature set according to HTML standard and installed browser plug-ins	Static elements – feature set limited by installed wiki application eventually expendable by plug-ins
<b>Versioning</b>	Not available unless additional versioning software is used	Integrated versioning system supporting restoring of previous pages
<b>User Management</b>	Additional CMS systems necessary	Supported by GForge plug-ins including role management
<b>Content</b>	Either automatically generated or user generated	User generated

**Table 2: Feature comparison between Scripted HTML and Wikis**

In conclusion, universAAL will follow a combined approach. A large amount of content, especially source code documentation and change logs, is easy to generate automatically using tools like Maven and Javadoc. However, these tools generate HTML code that can't be trivially converted into wiki-compatible code. Versioning is not implicit for these parts of the documentation as they are created quickly from available code, which itself is stored on a repository supporting versioning. Strictly user generated content, including tutorials is not supported by these generators and therefore it is reasonable to use a wiki system. There are numerous different types of wiki software that are not compatible between each other. The most common is MediaWiki, which is also used for the encyclopaedia system Wikipedia. It was decided to use this system, in order to reduce the necessary learning for participants and because of available plug-ins for collaboration environments, like GForge, as well as plug-ins that allow the creation of PDF documents from Wiki pages.

### 3.2.7 Implementation in other frameworks

Referring to the discussion in the previous subsection, “Selection of programming framework”, porting the universAAL middleware to other cross-platform development environments is planned for future iterations (including mobile platforms, such as iPhone OS and Android). The reasoning behind this is to further extend the support for available devices and consequently increase universAAL market share. Later in the project the partners will investigate their connections to universities and outsource this task to undergraduate and graduate students that are interested in this line of research. Using this as topic for bachelor/master projects is also a viable option.

## 4 IPR Management

IPR (Intellectual Property Rights) is a generic term that encompasses several different issues that are covered by different laws and practices. Generally, all issues related to *copyright*, *patents*, *trademarks*, *trade secrets* and *sui generis database rights* are collectively indicated as IPR.

IPR management is of basic importance in universAAL, because most of the software artifacts we are going to release will be distributed and accessible to a large community of different users (Stakeholders). The documents that rule the software copyright and access right are in order the Description of Work (linked to the Grant Agreement) and the Consortium Agreement. In particular the Consortium Agreement has got a section dedicated to the IPR and access right that each developer should read carefully to avoid disputes for the software exploitation. Concerns may arise for both the software produced in the project life span (Foreground Knowledge) and for the software already developed by a consortium member (Party) before the start of the universAAL project (Background knowledge).

The main problem we need to avoid since the beginning is the coding of everything without a clear and **non-repudiation agreement** on the copyright license that will be used for the software component we are going to develop. The need of an initial agreement is evident in case of joint development, however also in case of a development performed by a single Party, or just of background software, the definition of copyright license must be clear since the beginning. The reason is that any further dependency introduced with either foreground or background components cannot be based on ambiguous copyright license assumptions. The risk is that after months of development and effort spent in component design we cannot reach an agreement with full satisfaction for each partner.

The second problem we need to avoid is the **infringement of copyright licenses** of the software we may use in case of derivative work. If we start the development by using third party software (external), we clearly need to state which kind of restrictions or permission we have by using that external software. Note that even the inclusion of a single file or snippet can cause a legal dispute.

The **Release Manager** is the person in charge of checking all the above conditions are properly met before the software release. Obviously not all the code can be examined, especially in case of large sized software packages. Hence, each developer is responsible for his/her software contribution and he/she should be properly instructed by the Party about the right behaviour. In particular each Party shall nominate one or more people, **IPR Manager(s)**, in charge of creating an entry in the **IPR directory** that summarizes the licence adopted for each component developed by the Party and the list of software included by the component and related copyright licence, namely the *Notice file*.

The universAAL software development will be organized in several projects corresponding to logical or physical subsystems that implement the reference architecture. A (software) **Project Leader** is the person granted with administrative permissions who is in charge of the management and development of a component suite belonging to a specific subsystem. With such an organization in mind, the process for the proper management of IPR issues can be roughly outlined in two phases:

- The Project Leader together with IPR Managers of the Parties involved in the development will agree on the type of copyright licence that will be used for the project. Then each IPR Manager will create an entry in the IPR directory for the copyright license. The Exploitation Manager shall verify any incompatibility with other system components.
- Approaching the deadline of a new release, each IPR Manager checks for any inconsistency in the Notice file in relation to the contributions provided by the developers, this is an internal procedure defined by the Party. It then updates the list of software included in the component in the relevant entry of the IPR directory.

## 4.1 Tools

### 4.1.1 IPR directory

Each partner must designate one or more persons as having the authority to add entries in the IPR registry, namely the IPR Manager(s). The formal process to designate the IPR Managers will be managed by the Project coordinator together with each partner legal representatives. The IPR registry as well the formal documents nominating the IPR Managers will be available, through the collaborative instrument used by the project consortium (eRoom), within specific folders at the root level of the WP9 document repository. Detailed instructions on the meaning of a registry entry and the related approval process will be provided both in the deliverable D9.3 and within the IPR registry folder.

This section is strongly connected to work on D9.3 and no final assumptions can be made here until that work is completed. Therefore next versions of this deliverable will complete this section using input from D9.3 or may even remove it, in which case a reference will be included here.

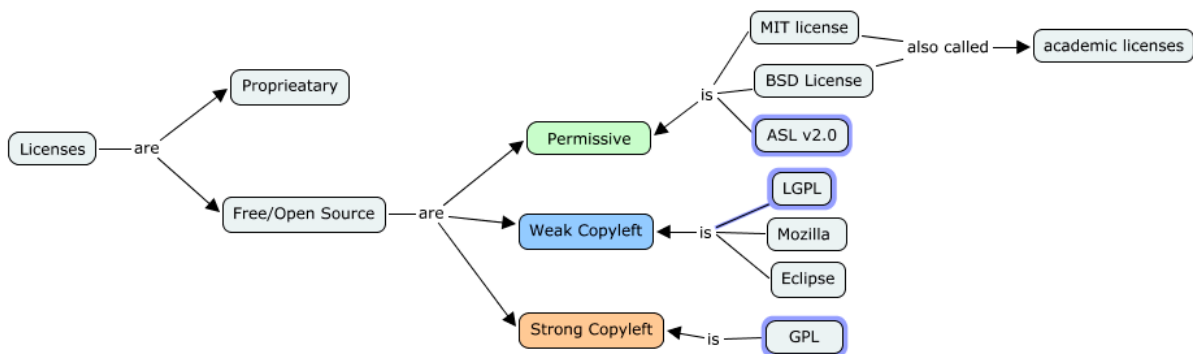
## 4.2 Regulations, guidelines and good practice

### 4.2.1 Software license in brief

A Software License is a legal document that accompanies software, it is a copyright license. Without a software license, according to the provisions laid down within the Berne copyright convention, software cannot be distributed or modified without the explicit permission of the authors. The Web page at <http://www.templetons.com/brad/copymyths.htm> provides useful answers to common myths about copyright seen on the net and cover issues related to copyright and USENET/Internet publication.

A good reading is the book “Understanding Open Source and Free Software Licensing”<sup>10</sup>. It provides several chapters for the most common license and many of them are freely available on internet.

The Open Source Initiatives has approved 66 different licenses<sup>11</sup> as conforming to the Open Source Definition, but only 9 licenses are recommended for general use. To make things simple we divide the OS licenses in three categories: **Permissive**, **Weak Copyleft** and **(Strong) Copyleft** as represented in the following diagram:



<sup>10</sup> Andrew M. St. Laurent: Understanding Open Source and Free Software Licensing, O'Reilly Media. August 2004.

<sup>11</sup> Note that the version of a license makes a license a completely new license. So ASL 1.0 and ASL2.0 are counted as two different licenses

**Figure 4: Main open source license categories**

- **Strong copyleft licenses:** Usually, Open Source software licenses allow any recipient of a program to use it for any purpose, to copy, modify, and redistribute it. The strong copyleft licenses include a particular restriction on these freedoms.

The recipient can only redistribute the software under the same licensing terms.

If you want to be sure that all copies of your program, even modified ones, are accompanied by the corresponding source you should use a copyleft license.

GNU GPL (General Public License) is the most popular license of this type. It is used for the programs developed by the GNU Project and many others. Examples include the Linux kernel, the GNU Emacs editor and the portable GCC compiler.

Different markets have been created thanks to GPL license in the embedded system and mobile computing (e.g. Linux-based).

- **Weak copyleft licenses:** If you want to be sure that your software remains Open Source even after modification, and you want to leave people free to link it against **any program, be it Open Source or proprietary**, you should use a weak copyleft license. LGPL (Lesser General Public License) is the most popular license of this type. On the contrary of GPL, software released under the LGPL may be linked against non GPL compatible software, even proprietary software, **provided that the source of the LGPL part is made available**, and that the LGPL part is upgradeable independently from the rest of the program linked against it.

This is usually feasible if the LGPL part is a shared library, a DLL, a loadable module or a component.

- **Permissive licenses:** The Apache Software License (ASL 2.0) is the most popular license in this category. It is conceptually similar to the BSD license (Berkeley Software Distribution), one of the first invented software licenses together with the MIT license (Massachusetts Institute of Technology). The disclaimer of a BSD license practically says:

“Do what you want with this code, keep the copyright note. We don’t give you any guarantee”

Because you need credit the original authors of the modified software it is not a very scalable license. Some cases exist of software distributed with 40 different licenses in order to give credits to all the authors.

The users can do whatever they want with the software, including **releasing it as proprietary software**, whether modified or not. Generally these licenses are compatible with GPL license model and the software released with a permissive license can be modified and distributed with a GPL license or copyleft license.

The important aspect of the modern license like ASL 2.0 is that it gives some protection against the bad surprise of **submarine patents**. Submarine patents are patents that apply to a certain technology, but are hidden— unpublished or unasserted—during the development of that technology. Once the market matures and the patented technology is in wide use, the submarine patent surfaces and the company owning the patent tries to claim patent royalties from across the industry used that licensed technology.

#### 4.2.2 *The license used by universAAL*

According to the Description of Work, the main results produced within universAAL will be made available for free as open source under the MIT<sup>12</sup> license.

Although the default license for the main results produced within universAAL is the MIT license, it can be changed with the agreement of the partners. Any decision about the type of license to be used must follow two principles:

- Legal concern: applicability of the license due to compatibility concerns.
- Commercial concern: choice of the license depending on the target business model.

#### 4.2.3 *How to apply a license*

1. Include a **Disclaimer** for the header of every source file
  - a. By reporting a copyright notice for all the involved authors
  - b. Do not remove the copyright notices of included/modified source files
  - c. Include a reference to the **Notice file** for the list of all required licenses
2. List all the required licenses for the external code
  - a. included in your software
  - b. from which your code was (partially) derived
  - c. used at runtime
  - d. optionally used at compile time
3. Check the **compatibility** of licenses used
  - a. i.e. you cannot license with ASL if you include GPL
4. Prepare a copy of each different license
5. Create the **Notice file**
  - a. By providing all the required credits
  - b. By listing all the included/modified/used software
6. Include in your software package or project repository
  - a. The **Notice file**
  - b. The **License file** adopted for the developed code
  - c. All the **License files** adopted by the software you used
7. Maintain regularly updated the Notice file as soon as you create dependency with other software

#### 4.2.4 *Disclaimer and notice file examples*

Every source file of a software module released under OS License should have as header a disclaimer reporting copyright information, the license applied to the software and a link to the Notice file with detailed information about the all the software licenses associated to the software module.

---

<sup>12</sup> <http://www.opensource.org/licenses/mit-license.html>

The next figure shows an example of disclaimer for a derivative work, with copyright of three different subjects underlined at the top. The two lines of text underlined at the bottom refer to the “Applied license” and to the “Notice file” distributed with the software for a complete list of the copyright ownership.

```
/*  
Copyright 2008-2010 Andrew Rapp, http://code.google.com/p/xbee-api/  
  
Copyright 2008-2010 CNR-ISTI, http://isti.cnr.it  
Institute of Information Science and Technologies  
of the Italian National Research Council  
  
Copyright 2008-2010 ITACA-TSB, http://www.tsb.upv.es/  
Instituto Tecnológico de Aplicaciones de Comunicación  
Avanzadas - Grupo Tecnologías para la Salud y el  
Bienestar (TSB)  
  
See the NOTICE file distributed with this work for additional  
information regarding copyright ownership  
  
Licensed under the Apache License, Version 2.0 (the "License");  
you may not use this file except in compliance with the License.  
You may obtain a copy of the License at  
http://www.apache.org/licenses/LICENSE-2.0  
Unless required by applicable law or agreed to in writing, software  
distributed under the License is distributed on an "AS IS" BASIS,  
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
See the License for the specific language governing permissions and  
limitations under the License.  
*/
```

**Figure 5: Source file header with the license disclaimer**

The Notice file is a text file reporting the most important information about the software module. The typical information reported is:

- The included software in case of derivative work
- The software used, for example the libraries linked by the module
- Any other important information for example the reference to any specification implemented by the software
- And finally a summary of the licenses applied by the software

```
I. Included Software

This product includes software developed by
Andrew Rapp from the project XBee-API (http://code.google.com/p/xbee-api/)
Licensed to CNR-ISTI under the ASL License (CNR-ISTI protocol 0002638)

II. Used Software

This product uses software developed at
The OPS4J (http://www.ops4j.org)
Licensed under the Apache License 2.0.

This product uses software developed at
The RXTX (http://users.frii.com/jarvi/rxtx)
RXTX License v 2.1 - LGPL v 2.1 + Linking Over
Controlled Interface (see LICENSE.RxTx file ).

III. Specification used

Part of this software is based on the specification
provided by The ZigBee Alliance (http://www.zigbee.org)
Licensed under ZigBee specific license for non-member
of the ZigBee Alliance (see LICENSE.ZigBee file )

IV. License Summary

- Apache License 2.0
- ZigBee Non-Member License
- RXTX License v 2.1 - LGPL v 2.1 + Linking Over Controlled Interface
```

Figure 6: Notice file example

## 5 universAAL Development Environment and Coding Standards

The following chapter describes the setup for the universAAL development environment for project partner development. In order to simplify the process of combining generated code into a single working project it is required that every developer is following the guidelines described below.

### 5.1 Specific requirements

As described in chapter 3, the chosen development environment is based on Eclipse. In order to create compatible code and not run into any issues with plug-ins it is necessary for each developer to have the same version of Eclipse and additional plug-ins.

In order to set up Eclipse properly developers should follow the procedure described below:

1. Install a version of Java SE<sup>13</sup> (at least JDK 1.4), and the latest versions of Maven<sup>14</sup> and Eclipse<sup>15</sup>

Adding updates to Eclipse follows a similar routine. They are distributed using update sites that have to be entered in the dialog reachable in the top menu bar through *Help -> Install New Software*.

2. Install Subclipse<sup>16</sup> plug-in – update site [http://subclipse.tigris.org/update\\_1.6.x](http://subclipse.tigris.org/update_1.6.x)
3. Install AspectJ<sup>17</sup> plug-in – update site <http://download.eclipse.org/tools/ajdt/35/update>
4. Install M2Eclipse<sup>18</sup> plug-in – update site <http://m2eclipse.sonatype.org/sites/m2e>
5. Install Pax Runner<sup>19</sup> plug-in – update site <http://www.ops4j.org/pax/eclipse/update>

A description of these tools, how they are required in the process of building universAAL components and how to run a demonstration application are described in the following section.

### 5.2 Tools

Chosen plug-ins and tools are shortly described before guiding through the process of successfully building applications using this framework.

- Subclipse is a plug-in for Eclipse that allows integration into SVN version control repositories. It allows linking projects to a repository location, simplifying the commit process. All aspects of versioning including compare operations with older versions are supported.
- AspectJ is an aspect-oriented, platform-compatible extension for the Java programming language. ADJT its associated Eclipse plug-in. The target of aspect-oriented programming is the modularization of cross-cutting concerns, like error checking, synchronization, performance optimization or logging. ADJT includes a specific compiler, debugger, documentation generator and program structure browser.
- M2Eclipse is a Maven plug-in for Eclipse. It supports the life-cycle management of artifacts, automatically creating Maven-conform projects and GUI supported POM editing.
- Pax Runner is an Eclipse plug-in, simplifying OSGi platform bundle provisioning. The process of running created bundles in the different framework implementations of the OSGi specification, e.g. Apache Felix<sup>20</sup>, is made easier.

---

<sup>13</sup> <http://java.sun.com/javase/downloads/index.jsp>

<sup>14</sup> <http://maven.apache.org>

<sup>15</sup> <http://www.eclipse.org>

<sup>16</sup> <http://subclipse.tigris.org/>

<sup>17</sup> <http://eclipse.org/aspectj/>

<sup>18</sup> <http://m2eclipse.sonatype.org/>

<sup>19</sup> <http://paxrunner.ops4j.org/display/paxrunner/Eclipse+Plugin>

<sup>20</sup> <http://felix.apache.org>



## 5.3 Regulations, guidelines and good practice

In order to achieve a development process that is manageable and results in decent quality software it is necessary to follow certain guidelines regarding the development environment, most specifically regarding updates of the development environment. A group of senior developers needs to be available, that is monitoring the status of the development environment and according to initial and updated requirements will decide about required plug-ins, and which version is supposed to be used. These regulations can be divided into guidelines for this managing group and guidelines for each single developer.

### 5.3.1 *Guidelines for managing a group*

- Select required tools and decide on initial development environment. In case of universAAL this step has already been done and the results are noted down in this document. It is critical that this initial version is fit for productive use. Accordingly there should be no instances of alpha versions or nightly builds.
- Each tool requires a certain set of configuration. If it is possible to share this configuration via files those have to be shared throughout developers. If that is not possible tutorials have to be given out how to set the tools correctly.
- Changing requirements are likely to occur in the process of projects with such an extended running time. The managing group has to keep an eye on changing requirements, evaluate their impact on the current development environment and give out update instructions if required.
- Most tools used in this project are in continuous development, regularly leading to new versions that are fixing issues and adding new functionality. The managing group needs to decide if the new version contains features or fixes bugs that are critical for the universAAL development. If a newer edition is to be used, it has to be tested on a separate system. If it is deemed fit for productive usage developers are notified and given instructions about how to successfully integrate this new version.

### 5.3.2 *Guidelines for developers*

- All developers have to start with the same development environment. The IDE as well as all necessary plug-ins need to have the same version as decided by the managing group. Developers need to follow the rules specified by the managing group in order to generate a common setup for all participants and accordingly less complications in future setups.
- It is important that developers implement only the specified interfaces and consequently avoid unnecessary work and guarantee a good integration of the different modules.
- The written code needs to be fully documented. This includes class descriptions, method descriptions and commentaries to implemented algorithms. The documentation needs to be clear and understandable even for developers that are not directly affiliated to the project.

## 5.4 Code conventions

Following coding conventions is a necessity when working on large, collaborative software development projects. The majority of actual work with a source code is maintenance. This includes bug-fixing, as well as adding features or improving used algorithms. Furthermore it is uncommon for a single developer to work on a portion of the code throughout the project lifetime. Consequently it is important that code is written and documented in a way that allows outside developers to quickly understand the source and start working more efficiently.

### 5.4.1 Code styling

Code styling describes the conventions used when writing the source code itself. This includes standardized formation of the code, such as setting brackets, indentations and wrapping lines on long expressions.

The Eclipse IDE supports automated formatting of the code to follow these simple guidelines. There are various presets including the common Sun Code Conventions for Java Programming Language and Maven Code Conventions. The formatter can be set individual for each project in *Project -> Properties -> Java Code Style -> Formatter*. Afterwards the source code can be automatically formatted using the shortcut *Ctrl+Shift+F*. The active profile needs to be set to “Java Conventions [built-in]”.

### 5.4.2 Code documentation

While proper code styling increases the readability of the code it is also essential to properly document any created interfaces, classes, methods and algorithms. Primarily this helps developers understand existing code. While having a lot of comments within the code is often seen as distraction by the current developers they always have to keep in mind that code needs to be reusable and thus simple to understand by outside developers. The rule of thumb is rather to have a comment too much than spending hours afterwards understanding the created code.

Another important aspect is the automated creation of accompanying documentation. Javadoc is a tool for software documentation that is able to automatically generate HTML documentation files from source code comments. Sun is providing several tutorials for Javadoc, including examples of the documentation on their own projects<sup>21</sup>. Another interesting document is the Sun guidelines on proper API specifications<sup>22</sup>. Developers are required to read the important parts of those documents and adapt their code documentation if necessary.

### 5.4.3 Checkstyle and Maven

Checkstyle<sup>23</sup> is a coding standard enforcement plug-in for Eclipse that can also be integrated into Maven. On demand Checkstyle is testing the current source code and indicates violations of the defined coding standard. In a Maven environment it is possible to automatically generate reports of these violations and take the appropriate measures. Optionally custom coding standard files can be used. universAAL will use the Sun code styling and consequently provide configuration files for adapting Maven.

---

<sup>21</sup> <http://java.sun.com/j2se/javadoc/writingdoccomments/>

<sup>22</sup> <http://java.sun.com/j2se/javadoc/writingapispecs/index.html>

<sup>23</sup> <http://eclipse-cs.sourceforge.net/>

## 6 Project Management Tool and Issue Tracker

The project management is the process of planning, organizing, and managing resources to bring about the successful completion of specific project goals and objectives.

### 6.1 Specific requirements

The universAAL project is a large and complex international project managed by different teams in different regions and it is necessary to provide them with utilities for communication, sharing code and documentation, as well as time-management and possible surveys. Therefore the project needs an environment in which to host projects in a way that the code, documentation, binaries etc. were publicly accessible to all who wishes to see and use them, contribute feedback, report bugs, ideas and suggestions, and even help to develop code/modules/documentation/resources for the software.

The management tool should provide a centralized access point for:

- Communication between project members using mailing lists
- Discussion Forums - for discussions between teams
- Support and enhancement requests with an issue tracking system
- Sharing of documentation – modules specifications, APIs, Release plans, known problems etc.
- Handling of TODO lists, tasks, etc
- File uploads/releases
- Posting of news - every project can have its own news items.
- Code Snippets - Provides of a basic knowledgebase that can contain code fragments, HOWTOs, etc.
- Version control repository (CVS or SVN) with well maintained release log
- Usage statistics, including the project members, the number of mailing lists, CVS statistics, the number of items in the discussion forums, etc.
- Fine-grained access control

The Issue tracker should support:

- Administration
- Defining a various type of tracked items
- Browsing of Items
- Reporting
- Assigning responsibilities
- Submitting a new item
- Tracking the time spent on given task (optional)
- Cross-reference item with source code modification and to the release
- Voting on item for prioritize the item based on the active community needs

### 6.2 Tools

GForge is a modular system that is easy to install and maintain. It is based on object-oriented plug-in architecture and thus it is very easy to be extended. It provides a full configured development system with versioning, a project web site and tools for communication between members of development teams.

### 6.2.1 General features

A complete configured GForge system gives the following features:

- A Web space for every project
- Source code versioning via several systems, such as SVN or CVS
- Shell access to the server for the developers
- A set of communication media for project coordination and exchange between team members – mailing lists, discussion forums, support tracker, bug tracker, documentation sharing, TODO lists/tasks, news, code snippets

The GForge server administrator has fine-grained access control and can grant individual users or groups thereof permission to either read or write certain aspects, respectively allow them to administrate projects on their own.

### 6.2.2 Eclipse integration

The GForge management system provides an Eclipse plug-in that brings the full power of the GForge Tracker and Document Manager to the desktop. This plug-in, however, is only available to GForge Group customers and there is no trial version. Many users would be able to perform all their GForge work without ever opening a web browser, and without ever leaving Eclipse. The Tracker plug-in includes custom query building and editing, and add/update/delete of tracker items and the doc manager plug-in allows drag and drop between your desktop and the GForge document manager.

Just to evaluate, a short overview on the features of the Eclipse plug-in is presented in this video: <http://gforgegroup.com/training/gfas/gfas-eclipse2/gfas-eclipse2.html>. The first decision is not to use this plug-in, as it is a proprietary restricted tool only available under purchase, and its impact is relative (and null in the development itself). Only if at a later stage the management of projects becomes cumbersome for developers, it may be decided to purchase this tool. In that case, it would be reported in the version of this deliverable following the purchase.

### 6.2.3 Issue tracker

The Tracker is a generic system where you can store items like bugs, feature requests, patch submissions, etc. The GForge system can track virtually any kind of data, with each tracker having separate user, group, category, and permission lists, easily move items between trackers when needed. New types of trackers can be created when needed.

The GForge Tracker provides:

- Custom Field Admin
- Item Details
- Saved Query and Export Page
- Time Tracking with Reporting And Exports to Excel

## 6.3 Regulations, guidelines and good practice

### 6.3.1 Project planning

The web space provided by GForge could be used by the software project leaders and contributors for sharing and updating the development schedules and responsible persons.

A detailed description of the project is a must, especially in case of multiple contributors working in different locations. This description should contain the verbose description of the functionality of each component, its dependencies and public interfaces. For planning and tracking purposes it is

recommended to have a table describing the progress status of each component (what is working stable, what is still missing, known problems etc).

A very powerful tool for management of the internal tasks and tracing their progress is the TODO tracker provided by GForge. See below.

### 6.3.2 *Handling of technical problems*

For this purposes the GForge provides several types of issue trackers that have a core of similar functions and a set of specific attributes.

- **To-Do tracker:** Here one can add a task, describe it in details, set deadline, estimated efforts and priority. The tasks could be assigned to specific user/contributor in the project. This tracker supports attachments.
- **Support tracker:** Very suitable tool for communication between developers using the functionality of given module and the developers working on this module. Here they can discuss problems or request advices. A problem should be classified in shorter time in a proper product (or module) category and sub-component category. It is very important that the developers monitor this tracker and react as soon as possible on requests that concern their module(s). A general responsible for the tracker takes care of the timely handling of the requests and the correct assignment to the proper developer. **The most important recommendation before raising a support issue is checking if this problem has been already handled.** Use the search features of the system. The support issue should provide enough information that helps its reproduction and thus better analysis.
- **Bugs tracker:** It is recommended that this tracker is operated only by the developers of the components with the reported bugs. The bugs are retrieved out of the problems reported in the Support tracker or from the developers' internal tests. Once a fix for the bug is found and included in some release of the given component, the version of this release should be provided in the note closing the bug.

GForge provides e-mail notifications upon submission of new issues or updates on the existing ones. The recipients of these notifications are:

- The users participating in an issue thread.
- The users monitoring the tracker.
- Specified by the system admin.

An alternative and very handy way to work with the GForge trackers could be the one provided by the plug-in for Eclipse IDE.

## 7 Integration Strategy

Since the philosophy of universAAL when it comes to implementation is to reuse as much as possible code from its Input Projects or other Open Source projects, it is necessary to define a strategy for the integration of such existing code into what will become the Reference Implementation.

By checking if and how an existing software component matches to a component defined into the reference model and then into the reference architecture, such component will be made the necessary changes to be considered “integrated” and compliant with universAAL.

Only when it is not possible to reuse components from other projects, whether by not passing the integration process or not finding a suitable candidate, a new development will take place inside universAAL for creating such component. However, the resulting new component would go through the whole integration process in a different fashion, as there is no software to be tested, but a new one is designed from scratch. This is explained further in the “New Development” subsection.

Dates for stages of each process are included when possible in order to keep this planning feasible.

### 7.1 Release management

The Description of Work defines “iterations” on which a certain set of functionalities must be reached. These can be referred to from the point of view of the development as “releases”.

In order to properly manage the release of all the components that will form the reference implementation, these will be released in different periods according to a division into different classifications depending on level of completion and importance of its role, namely:

Division into components sets:

- First set: Basic components needed to assemble the reference implementation. The components to develop will have been selected taking into account work from WP until M6: Architecture model and requirements from D1.2A, D1.3A and D1.3B.
- Final set: The rest of components needed to form the reference implementation. Also those components from the first set that for some reason had to be postponed. These will take into consideration the final version of architecture definition from D1.3C, as this is due in M15.

Division into completeness of implementation:

- Alpha release: First version of the component, which covers a certain set of core requirements considered as basic and/or prioritized.
- Beta release: Enhanced version of the component that covers all the expected requirements from it, and corrects any error found in the alpha release. Also solves any issue found by WP5 during the evaluation of the component.
- Final release: Final version of the component that corrects any error and issue found in the beta release, whether by testing or evaluating it in WP5.

In certain situations there may be fewer releases, because of time or resources limitations or early success (e.g. alpha version already covers all requirements).

How and what requirements are selected for alpha and beta releases is explained in the following subsection “Integration process”. Here only the management of the release process is explained. According to the division into these two aspects, the expected release dates for each component is shown below matching the month number of the iterations:

	<b>M9</b>	<b>M18</b>	<b>M27</b>	<b>M36</b>
<b>First set</b>	Alpha	Beta	Final	
<b>Final set</b>		Alpha	Beta	Final

**Table 3: Release month for components in first and final set**

At the beginning of each release (iteration) period, and for every component that is to be developed and/or integrated to perform a role in the reference implementation there will be a specification document. Such document will contain a concise definition of the component that will evolve as different versions (alpha, beta and final) are developed. The specification is available for every partner so that they can check if any principle defined in other work packages is not being addressed in the release plan, for instance, missing requirements. At each stage of the release process (on each “release milestone”) the document is updated with the report of the previous release and the plan for the next one, as shown in next subsections, but take into account that these only reflect the status of the specification as it should be at each release. The specification is a live document that can be updated constantly, following the work progress of partners as usual.

### 7.1.1 Initial specification

At the beginning of the release no reports are of course expected, but only plans for the development of the component. The specification will contain the following, reflecting the planning made at this stage:

- A text description of “black-box” behaviour of the component.
- The list of requirements to fulfil, divided into alpha and beta, to be implemented in each version.
- The key technological decisions made for the component.
- Internal design of the component based on interfaces.
- Definition of Test Cases to apply to the component.
- Organization of the code (artifacts, packages, etc).
- Work division: composition of development team and assignation of responsibilities and time scheduling.

### 7.1.2 Alpha specification

Alpha specification reflects the results of the development of the alpha release and plans the beta release. The specification will be updated with the reports of the tests passed by alpha release (it must have passed them to be released) and the evaluation made by WP5. Any change to the design or requirements of the component that has arisen during alpha development is also reported by updating such information in the specification. The planning of the beta release is added, not updated over the alpha one, so that the specification keeps track of the development process. For this release and any of the following, it will be checked that code documentation is present. Summarizing, the specification will now reflect the following:

- Update to the initial specification according to outcome of alpha development and plan of beta development.
- Link or reference to code documentation.

- Reports of the tests made upon the component, if relevant (for instance, performance test)
- Link, reference or inclusion of WP5 reports on evaluation of alpha release.

### 7.1.3 *Beta specification*

Beta specification does the same than alpha specification but applied to beta release, and also taking into account that next release is the final one, and therefore no other requirements will be added. The other main point of beta specification is reporting how issues identified by WP5 evaluation of alpha release have been solved. The specification will therefore contain:

- Update to the alpha specification according to outcome of beta development and plan of final development.
- Link/reference to code documentation.
- Reports of the tests made upon the component, if relevant (for instance, performance test)
- Description of the solution of the issues identified by WP5 on the alpha release
- Link, reference or inclusion of WP5 reports on evaluation of beta release.

### 7.1.4 *Final specification*

The final specification will reflect the final component design and features along with its results on tests and WP5 evaluations. As the final release, it will not contain any development plan, although it could be possible to identify a responsible for future maintenance and deal with open issues. The reports to be included in the final specification would be:

- Update to the beta specification according to outcome of final development and plan maintenance.
- Link/reference to code documentation.
- Reports of the tests made upon the component, if relevant (for instance, performance test)
- Description of the solution of the issues identified by WP5 on the beta release
- Link, reference or inclusion of WP5 reports on evaluation of final release. These will be marked as open issues.

## 7.2 **Integration strategy of software components**

The Description of Work document proposed a draft for the integration process of existing components, which was developed taking into account the Eclipse Development Process<sup>24</sup>. The process described here develops that draft further by precisely describing its phases and decisions, renaming them when appropriate, and breaking down or adding some stages in order to cover all possible situations during the integration.

This integration process is conceived initially for software components that might become part of the reference implementation. For communication protocols or information models (like ontologies) other processes may be defined by the work packages/subtasks that deal with them more deeply. However it may be possible to follow this process if it is devised applicable.

The full integration process is exposed in the following scheme, which has been derived and enhanced from the Description of Work, Figure 3 “Initial draft of universAAL component integration process”. Each stage will be focused on in the next subsections.

---

<sup>24</sup> [http://www.eclipse.org/projects/dev\\_process/development\\_process.php](http://www.eclipse.org/projects/dev_process/development_process.php)



Take into account that this integration process is deeply interrelated to the release management described in the previous subsection, as it includes a part of development. They are two processes that a component under development (whether newly developed or adapted from an input project) must traverse in parallel. The stages at which the processes are related to each other will be clearly pointed out here.

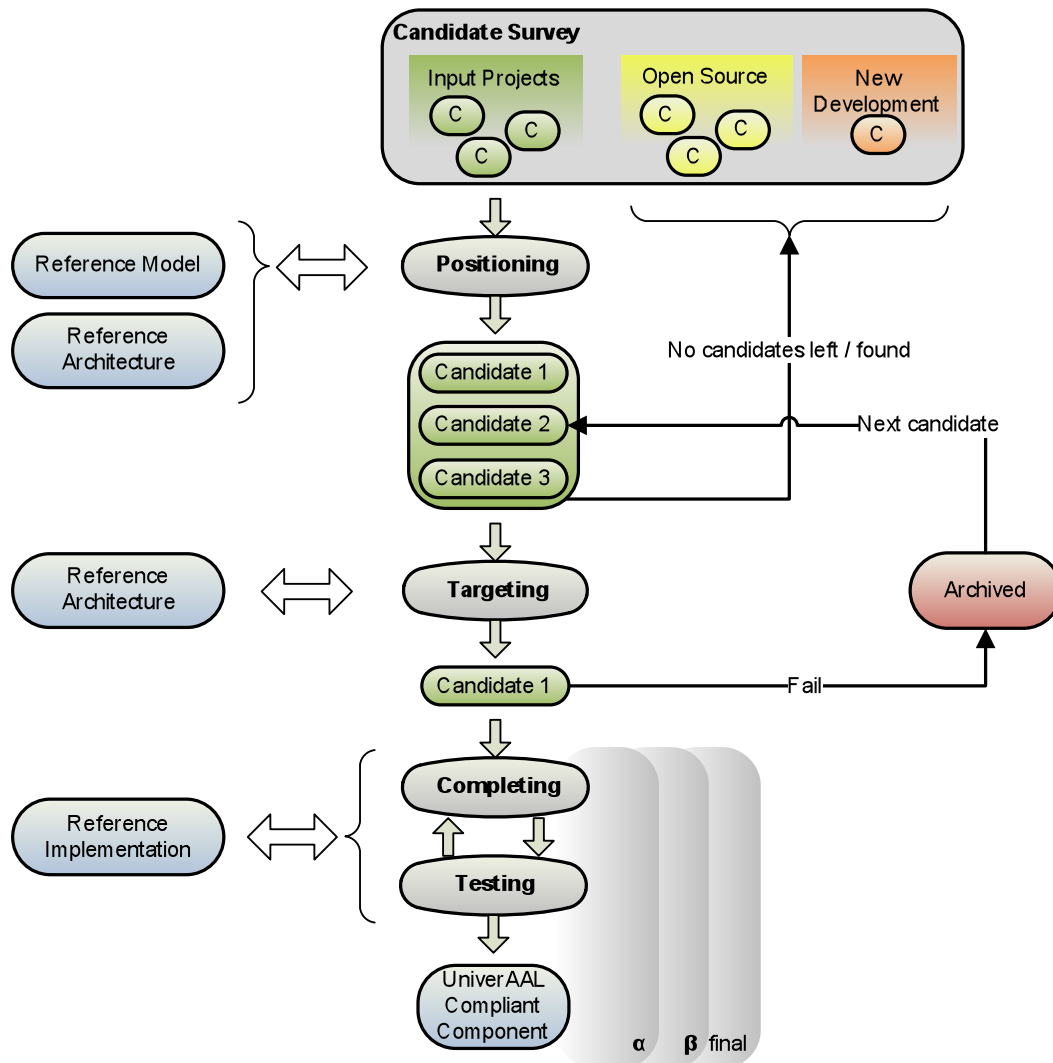


Figure 7: Integration Process

### 7.2.1 Candidate survey

At the beginning of the release process described before a survey searching for components from input projects shall be made. This listing will permit to identify components that are similar and therefore group them. In this way there will be groups of components coming from the input projects that do similar things and so they will give an idea of a generic, “gestalt” reference model. These groups will be created and matched to the universAAL reference model in the positioning stage. By doing so, the components inside each group can be more easily fit into the reference model.

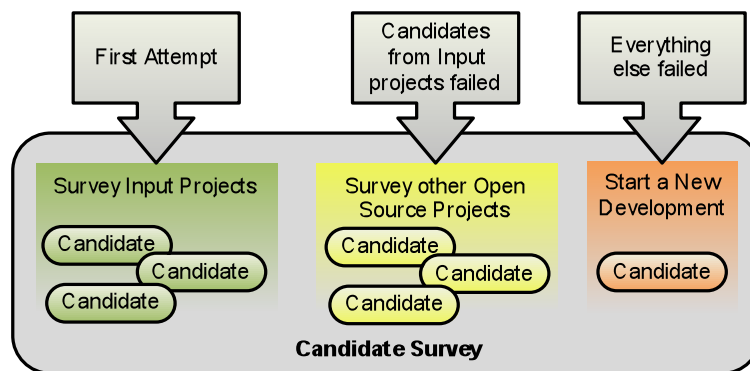
This selection of components from input projects takes place at the beginning of each of the release process for both the first set and the final set of components. That matches M6 and M15 respectively, although at M15 the selection of components can be reused from the one of M6. If so, it will be stated in next versions of the deliverable.

However, this is only applicable to components coming from other input projects. For other open source projects, though, the beginning of this stage can take place at any other time depending on some considerations: If no component has passed with success the integration process, we will have to

browse external solutions from other open source projects. This can be achieved either by browsing other projects on our own or by issuing a call to the community to present candidates that can fulfil the requirements that have become orphan due to candidates failure. This will only be possible once the community is successfully built and the appropriate communication channels have been set up. Therefore this depends on the work by WP9. Next versions of this deliverable will be updated with the procedure for this as soon as the proper tools and methods are described by WP9.

Once communication channels with the community have been established, there could be a chance that members of an external open source project propose a candidate of their own to play a certain role in the reference model and architecture. Whether this role has already been adopted by an integrated component, or it hasn't yet (or is still being integrated), we should study the proposed component and consider it for integration. If decided it is worth being integrated, it will start the integration process, only that if it fails at any stage it will be archived and no alternative will be searched, thus ending the integration process. The first step will be to place the suggested component inside the proper group of similar components.

Only if no candidates from input projects and no suitable solutions are found in the open source alternatives a new development shall be started, in which case the process is slightly different, as will be described in the "New development" subsection at the end of this section.



**Figure 8: Candidate survey stage**

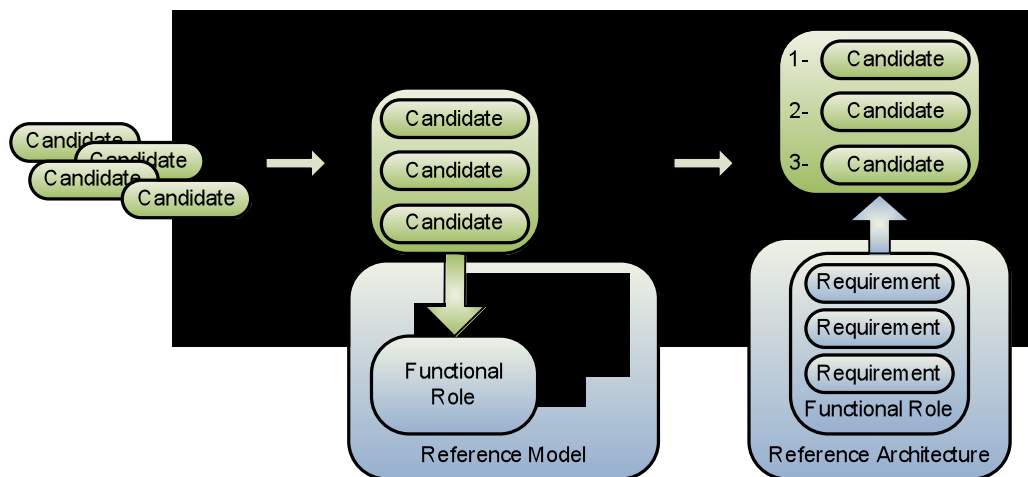
In the cases of components from other open source projects and new developments, they follow the integration process on their own, without being aligned with the release management further than reporting their final integration in the nearest release month available for the final set of components. This also means that no open source or new developed components will be part of the first set of components release process.

A component registry shall be updated with any new candidate that is selected to enter the integration process. Whether it is an eRoom database, part of the specification document, or an independent document stored elsewhere, it will contain for each entry the name and version, as well as the status of the component throughout the process. It will also contain the feedback from the stage evaluation (in whatever format is considered best). This is only to keep track of the status of the component throughout the process.

### 7.2.2 Positioning

The output of the candidate survey is not really a single candidate but a list of candidates that will be grouped at this stage into groups that perform a certain role that can be associated to a role in universAAL reference model. Work done in D1.3 (A version for first set of components, C version for final) with the layer model and component view is used as input in the positioning stage to identify which component of the model suits the group of candidates. Although the properties of each candidate in the group can spread further than those it was selected for, it is imperative that it fulfils all of those that are required for the role in universAAL model. This is caused by the different abstraction

levels, since WP1 defines the model in terms of functional building blocks and here the candidates are pieces of actual software (for instance, Maven artifacts).



**Figure 9: Positioning stage**

Therefore now each grouped candidate is expected to perform as the role assigned to its group. For achieving that, it will have to accomplish a set of requirements that are assigned to that role in the reference model, plus some others that may be general to every part of universAAL implementation. These requirements are gathered in D1.2, where they have also been prioritized. If the component accomplishes all (but not necessarily only) the requirements that are desired and is compatible with the reference architecture specification, then the candidate is able to be integrated. Otherwise it will be discarded and marked as archived.

But before it passes to the next stage, the rest of the candidates of the same group must be addressed as well. The objective is to rank all the candidates inside a group. Before ranking the candidates in a group, the requirements for the role that group is assigned must be analysed, based on their priority from D1.2.

Based on the level of accomplishment of requirements taking into account the priority assigned to them, and the expected amount of work foreseen to integrate each candidate component, they will be given a rank, and will be ordered in the group. If a candidate receives a rank too low, it may be decided to discard it and will be marked as archived. The ranking procedure will be determined during the first positioning stage reached and therefore will be redacted in the next version of this deliverable.

Whether a candidate passes or fails the Positioning stage, all the issues found while trying to fit it into the reference model will be collected. This will serve as feedback for the definition of the reference model and architecture at WP1, especially at the first stages of the project, when it still needs to be refined. Also a big input for WP1 is the grouping of components, which gives a sense of how the input projects architecture their model. Another feedback for WP1 and also other tasks of WP2 is observing the technological decisions made by the input projects for the functional roles represented in the groups. This could help ease the decision on the same aspects in universAAL.

Note that for the final set of components the process starts in M15 and D1.3C should be available. In this case a gap analysis will be carried out to identify missing functional roles in the input projects and act accordingly (new development or call for candidate to the community).

### 7.2.3 Planning

In the positioning stage candidates were assigned to a group and that group was matched to a certain role in the reference model in the positioning stage. Inside that group candidates are ranked. Now the candidate with the highest rank is analysed in order to plan its adaptation development for completing stage, and the selection of requirements for alpha and beta releases.

During this stage it must be taken into account that what is being evaluated from the component is its *business logic*, not its *platform logic*. This means that what must fit in the reference architecture and requirements is its “inner behaviour”, not its borders or how it connects to its original platform. This (the platform logic) is what will be modified if necessary in the next stage to integrate it in the reference implementation. However the business logic could be modified under certain conditions as explained here.

It is hard to define a development process that is only addressed partially, as happens here: First, the software component already exists; therefore there is no freedom to develop from scratch. Secondly, it is only the platform logic that is being modified. This has an impact on the requirements that can be implemented and as a consequence in the features of alpha and beta releases. An initial classification could be that the core functionalities (functional requirements) are implemented in alpha version, and then additional requirements for performance, security, etc (non-functional requirements) are addressed in beta release. This is feasible for new developments, but, as an example, the internal business logic of a candidate may implement all of the alpha (functional) requirements -it must if it has reached this stage- but also some or all of the beta (non-functional) requirements. Hence, since only the platform logic should be modified, once it is adapted into the reference implementation it will feature all the requirements, both for alpha and beta releases.

At this point there is already a prioritization of requirements coming from WP1. There is also enough knowledge to adequately divide the requirements into alpha and beta releases. The issue is now that the development for adaptation to the reference implementation may not match 100% with the division in alpha and beta requirements. This is addressed taking into consideration the following:

- Alpha release: It will undertake the minimum amount of code modification upon a candidate needed so it accomplishes two points: meeting the initial requirements for alpha release and being adapted and able to run in the reference implementation platform. There are two possibilities by excess or defect:
  - In some cases the result of this alpha release can also be the beta release if all requirements have already been reached by just adapting the platform logic to the reference implementation platform.
  - In other cases it may be necessary to also modify the business logic if adapting the platform logic was not enough to meet the alpha requirements.
- Beta release: Every needed code modification so that the candidate implements all of the expected requirements. In some cases this may imply modifying business logic as well, to a certain extent.

Just like in the positioning stage, whether the component passes or fails, any issue during the process will be reported to be used as feedback for WP1. This will help refining the reference architecture.

When this stage is finished there should be a clear idea of the amount of work to be done to integrate the candidate component into the reference implementation platform, and assign the development work accordingly. At this point there is enough information to redact the initial specification introduced in the release process.

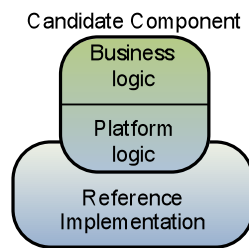
Please note: For the first set of components the positioning stage should have used D1.3A, but it is expected that at this point, in this stage, D1.3B is available. Therefore, a final check will be made against the functional roles defined in that deliverable and the proper measures will be taken if changes are found: postpone candidate component integration for the final set, modify prevision of work, requirements, or rearrange ranking.

#### 7.2.4 *Completing & Binding*

Until now, everything that has been made was preparation for release process. In this stage the candidate component is considered to be compatible with universAAL reference architecture, and then

the development team will modify the *platform logic* of the candidate component as necessary to connect it to the rest of the reference implementation. This represents the beginning of the development managed in the release process. It is the development carried out here the one that is divided into alpha, beta and final versions.

It may happen that the modifications are so much and so complicated that it is not worth to continue developing upon this candidate component. It may happen as well that the *platform logic* of the component is not properly separated from its *business logic* and the adaptation would require a modification of the *business logic*, thus modifying the component behaviour to something that no longer matches what we were looking for. If any of these happens, the candidate component will have not passed the completing stage, and will be marked as archived. However this should have been foreseen in previous stages and the risk of finding such situation is very low, as the most complicated components to integrate would have been discarded or received a low rank.



**Figure 10: Completing stage**

Like in the previous stages, issues found during the completing stage and their solutions are treated as feedback for the creation of the reference implementation and the stage itself.

When the candidate component has been modified to connect and work properly with the rest of the reference implementation (for alpha, beta or final release), it passes to the Stability Test stage. As will be explained in such stage, candidate may return to this completing stage after the stability test.

It is imperative that a base framework is selected prior to integrating a component. Therefore it may be difficult at the first stages of universAAL to address this adaptation development until we decide on a common basic framework (whether it is completely new or, most probably, comes from another input project). Until then, work in this stage, and the reference implementation itself, will be always temporary.

### 7.2.5 Stability test

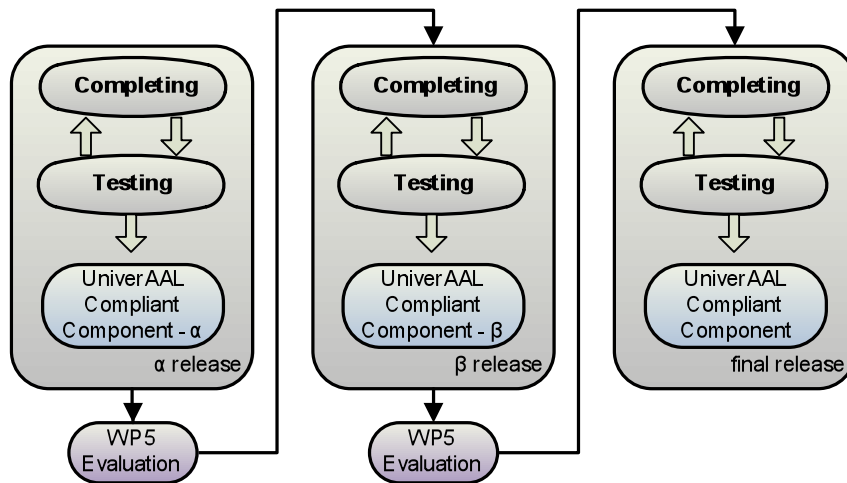
This stage performs different tests upon the now adapted candidate component. Do not mistake this stage with the Testing strategy defined in this document; despite they may use identical techniques. While testing strategy is a continuous process throughout all the development of code (including the completing stage), this stability test stage performs a suite of tests upon the component to guarantee its proper operation considering several aspects: Stability, non-degradation over time, load charge, and other factors according to the purpose of the candidate component and its requirements. This will confirm that the component will work without problems upon any situation it may face as part of universAAL.

If the candidate component does not pass the Stability Test, it may be decided to return it to completing stage, in case it is possible to modify the adaptation of the component in order to pass the test. If they decide however that it is not possible for whatever reason, the component will be discarded and marked as archived. Again, this is not likely to occur; as such problems should have been foreseen in previous stages.

### 7.2.6 Accepted

A component that passes the Stability test with success is marked as “universAAL compliant” with a certain degree depending on its version: alpha, beta and final. Only final versions will be considered 100% compliant, while alpha and beta have a temporary meaning.

If the version accepted is alpha or beta then it is still not the end of the integration process, as this is only another milestone in the release process. An alpha accepted version will be submitted to completing stage to continue its development when required (after being evaluated by WP5). The same happens to beta accepted version towards final version. In this way the integration process is synchronized with the release process.

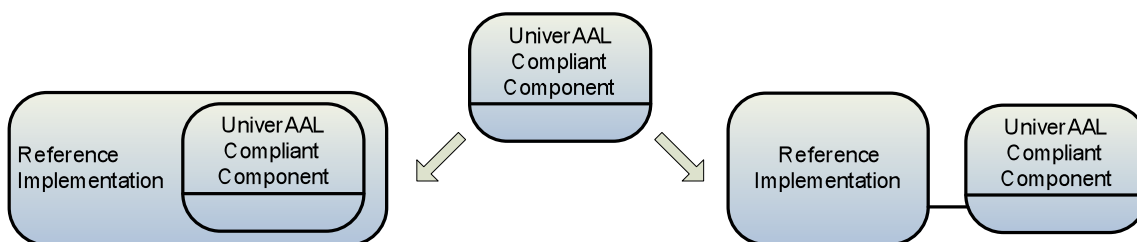


**Figure 11: Release process in the development stages**

There is still one decision left to make. Partners from WP2 will decide if a compliant component will be part of the reference implementation, with the target functional role identified by the positioning stage. If so, it is because, apart from passing the stages, it is either the best option or the first component to have passed the process for the target role. In successive evaluations of candidates, there may appear a candidate that passes the process and gave better results during the test stage or was integrated more easily, or is just considered to be a better option for whatever reason. If this happens, it can replace any previous candidate that reached reference implementation for the same role. This replaced component will be moved out of the reference implementation and be considered only as universAAL compliant.

However this probably will not happen, since there is only one candidate expected to be integrated for each functional role. It would apply, in any case, to suggestions from the community or exceptional cases.

When a component is accepted is ready to be released and there is enough information to compose the specification of that release, as stated in the release process.



**Figure 12: Possible outcomes for a universAAL Compliant Component**

### 7.2.7 *Archived*

When a component fails to pass any stage of the integration process it is marked as archived and its status is updated in the component registry accordingly, so it cannot be tried again. Only under exceptional circumstances a candidate that had been marked as archived could be evaluated again, such as a huge emphasis by the community to reconsider the candidate or a change in the reference model, architecture or implementation that could allow the candidate to pass the process under these new circumstances.

In any case the candidate that is being re-evaluated must pass the whole process again from the beginning, as if it was a suggestion from the community or a new development.

A new version of an archived candidate, if considered to be evaluated, would be treated as a completely new candidate, that is, with its own entry in the registry and having to pass the whole process on its own.

When a candidate is archived, the next candidate in the ranking (see positioning stage) of the same group will then begin its integration process starting at the planning stage. If there are no remaining candidates in the rank, it will have to be decided whether to issue a call of candidates to the community or start a new development from scratch, which should be the last option. In any case it would consider the feedback from the previously failed process.

### 7.2.8 *New development*

If a new development has to be undertaken, it will go through the evaluation process in a different manner. As there is no “actual” code or documentation to be evaluated in the positioning and planning stages, the purpose of these will be to design the component in order to match 100% with the role it is intended for. The role it must fulfil is that of the group that has ran out of candidates, and the new development will have to implement all the requirements associated to the group in the positioning stage. After these stages, the component will be fully developed during the completing stage, and then tested in the stability test stage. This is actually the only stage that the newly developed component can fail. The previous stages are automatically passed, since they are not really an evaluation but a definition (and development) of the component. If the component fails to pass the test stage, it will be redelivered to development stage to be corrected and then tested again. This loop is performed as many times as needed until the component passes the test.

There is a differentiation in the adaptation development of completing stage, however: In this case requirements for alpha and beta releases no longer have the problem stated in planning stage regarding the business & platform logic versus alpha & beta requirements.

Despite being a new development it could be helpful to reuse, as long as it is possible, parts of business logic or algorithms from the previously failed candidates, if these are considered good enough but failed the integration due to other reason. Of course, if licenses allow for it.

There is a risk that the reference model, architecture or implementation change during the development. If this happens and the development of the new component is too mature or the change is too deep as to adapt it on the fly, the development may have to be halted and start from the beginning. If this happens, the component which development was halted will be marked as archived and a new entry is added to the registry representing the new development, with the same identification but increased version number.

## 7.3 **Integration risks**

The integration of software components within universAAL platform includes various types of technical risks. These should be identified at this premature stage of the project, as a prevention mechanism for detecting and reacting early to any kind of problem may occur. The integration process

will be continuously monitored, in an iterative manner, in order to avoid malfunctions or errors that can be observed when integrating existing or new components to universAAL platform.

Following the process of the integration of software components, the risks presented in the next table were identified.

Risk	Probability/Consequence	Mitigation	WP	Milestone cleared
<b>Technical Risks</b>				
<b>R1: A candidate for integration component, is impossible to integrate</b>	Probability: Medium Consequence: Delay in the platform provision.	New alternative components will be looked for integration. If no alternatives are found, a new component will be developed.	WP1	M6
<b>R2: A candidate component cannot be adequately positioned in the reference model.</b>	Probability: Medium Consequence: Delay in the platform provision.	Possible refinement of the reference model in order to consolidate this component. Otherwise, replacement of an alternative component.	WP1	M6
<b>R3. A candidate component requires excessive amount of work to be done for consolidation within the platform</b>	Probability: Low Consequence: Delay in the platform provision and tool implementation.	New alternative components will be looked for integration. If no alternatives are found, a new component will be developed, estimating that the required amount of work is less than the candidate's one.	WP2	
<b>R4. The <i>platform logic</i> of the candidate component is not properly separated from its <i>business logic</i> and the adaptation would require a modification of the <i>business logic</i>, thus modifying the component behaviour to something that no longer positioned in the reference model</b>	Probability: Low Consequence: Delay in the platform provision and tool implementation.	New alternative components will be looked for integration. If no alternatives are found, a new component will be developed.	WP1	
<b>R5. A candidate component does not pass the Stability Test</b>	Probability: Low Consequence: Delay in the platform provision and tool implementation.	The component should be returned to the development stage, and if impossible to modify the adaptation in order to pass	WP2	



		the Stability Test, archive it and search for new one		
<b>R6. Poor performance evaluation of an adapted component</b>	Probability: Low Consequence: Delay in the platform provision and tool implementation.	The component should be reconsidered and optimized for maximum performance	WP2	
<b>R7. Evaluation of an adapted component shows poor quality of expected results</b>	Probability: Low Consequence: Delay in the platform provision and tool implementation.	The component should be reconsidered and optimized for better quality. Alternatively, new components will be looked for integration	WP2	
<b>R8. A new developed component fails to pass the Test stage</b>	Probability: Low Consequence: Delay in the platform provision and tool implementation.	The component will be delivered to development stage to be corrected and then tested again	WP2	
<b>R9. No 1 to 1 relationship between selected components and building blocks in D1.3 B/C</b>	Probability: Medium Consequence: More work to do in the adaptation of the component	This is more likely to happen with version D1.3 B. Mismatching components will be left for final set	WP1	M6
<b>R10. Incompatibility of technological choices of different selected components</b>	Probability: Low Consequence: More work to do in the adaptation of the component	More development work will have to be made to make it compatible	WP2	
<b>R11. Difficulties in time and resource planning</b>	Probability: Low Consequence: Delays in development and release	Development process under tight control, scrum meetings of WP2	WP2	

**Table 4: Risks during the component integration phase**

The majority of the risks can be compensated at a relatively small cost or in some other cases, an achievable solution may be possible at reasonable cost, or a reasonable solution is available at modest cost. In those cases that the technical risk can potentially create a severe problem to the universAAL platform, one can see that the mitigation possibility is high, which means that the universAAL consortium can address the specific problem at a relatively small cost. Moreover, the majority of the moderate problems can also be faced at a relatively small cost, while for some of them an achievable solution may be possible at reasonable cost, or a reasonable solution is available at modest cost.

This analysis shows that as universAAL has ambitious goals, matching new technological developments with a complex problem has inherently high risk. This is compensated within universAAL by the high previous expertise in this area of several of its partners, their multidisciplinary and the consortium's width.

In order to provide a general prevention mechanism we will continuously monitor the integration progress in order to detect and react early to any problems that may occur. The organisation of work in an agile manner and in short iterations will help us detect risks and deal with them early.

If new risks are identified later once the process has begun, they will be reported in the next version of the deliverable, along with the mitigation strategy followed.

## 8 Testing and Deployment Strategy

It is always good practice to automate the testing of every piece of code developed. This avoids performing manual tests of the code every time there is a change, and identifying errors that could appear later in the development if they were overlooked. When integrated with the automated build and deployment, it increases productivity by confirming that code committed is properly working. Thus all available code is ready to use.

Developing a test strategy includes:

- Identify and describe the approach to test: a statement describing how the testing will be implemented
- Identify the criteria for test: objective statements indicating the value(s) used to determine when testing is complete, and the quality of the application-under-test
- Identify any special considerations for test

Testing can be performed in different levels of the system. Going from the inside (code) to the outside (whole system) these levels are:

- Unit Testing: A unit test is a test of functionality for a specific unit. In object oriented design, unit is always class.
- Integration Testing: The goal of this test level is to test the software project as a whole group of modules that have been tested in unit level. Automatic build script has to be developed in order to automatically build system as a whole.
- System Testing: In this test level we are testing system as a whole in the run-time phase. Purpose of system testing is to test the integration of system modules and how the software product satisfies system requirements as a whole. These tests range the following aspects: functional, user interface, security, performance, smoke (check basic, core test cases), regression (testing fixes that solve previous failed tests) and installation.

There are different techniques to follow when testing. Some are more adequate for a certain type of test, while others can be used more widely or can improve others:

- Manual Testing: A human tester interacts with the system as an end user.
- Automated Testing: Testing is automated based on preconditions and expected results.
- Regression Testing: Code changes and fixes may result in the appearance of new errors. Regression Testing techniques deal with this issue by re-testing a minimum set of units.
- Environment Testing: Tests the compatibility of the code with its “surroundings”, that is, other software it interacts with.
- Stress Testing: Pushes the system to its limits in order to determine its stability.
- Load Testing: Very similar to Stress Testing, it overwhelms the system with demands that would be normally performed if isolated.
- Performance Testing: Tests the resources the system consumes and its variation during its operation.

### 8.1 Specific requirements

The test strategy addresses the risks and presents a process for mitigating those risks in line with the testing policy. Only exhaustive testing can show a program is free from defects. However, exhaustive testing is impossible.

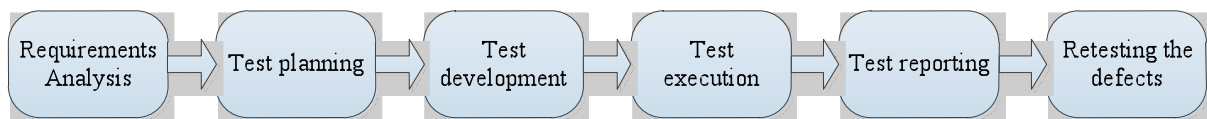
Requirements-based testing is a validation testing technique where each requirement is considered and a set of tests for that requirement derived. A general principle of requirements engineering is that requirements should be testable.

The requirement for test must be an observable, measurable behaviour. If the requirement for test cannot be observed or measured, it cannot be assessed to determine if the requirement has been satisfied. There is not a one-to-one relationship between each use case or supplemental requirement of a system and a requirement for test. Use cases will often have more than one requirement for test, while some supplemental requirements will derive one or more requirements for test and others will derive none.

The requirements for test may be identified from many sources; so all the materials available for the application or system to be developed should be reviewed. The most common sources of requirements for test include requirement lists, use cases, additional specifications, interviews with end-users, and review of existing systems.

Testing guidelines

- Choose inputs that force the system to generate all error messages;
- Design inputs that cause buffers to overflow;
- Repeat the same input or input series several times;
- Force invalid outputs to be generated;
- Force computation results to be too large or too small.



**Figure 13: Testing cycle**

Once it has been identified what is being tested and how, there is the need to identify who will do the testing and what is needed to support the test activities. Identifying resource requirements includes determining what resources are needed, including the following:

- Human resources (number of persons and skills)
- Test environment (includes hardware and software)
- Tools
- Data

## 8.2 Tools

This is a brief compilation of widely used tools for different aspects of integrated code testing, along with tools for deployment of such code that can be integrated with the testing process.

It was decided that in universAAL Java will be used so the selected tools are all related to that programming language:

**Test Libraries:** Code libraries that provide methods for testing certain situations and expected results from the code.

**Code Coverage Test:** Code coverage tools measure the amount of code that has been tested.

**Load Test:** Tests the code performance and stability under stress conditions of load.

**Deployment:** Code builds have to be made available to project developers and also to the community. For this a tool must be used that makes them available from the result of the build process by Maven.

Tool	Description [type]	License
<b>JUnit</b> <a href="http://www.junit.org/">http://www.junit.org/</a>	A Java library that implements a framework for <b>unit testing</b> . It's one of the most spread. It is most suitable for testing inside components. It is possible to integrate it in Eclipse via plug-ins, possibility to test J2EE server components. Ships with GUI as well as command line interface. [Library]	Common Public License
<b>TestNG</b> <a href="http://testng.org/">http://testng.org/</a>	Similar to JUnit but allows <b>functional</b> and <b>integration</b> tests and not only <b>unit test</b> . Supports Test Distribution. Application Server Testing support. JUnit tests can be converted to TestNG. Covers the entire core JUnit4 functionality. Generates test reports in HTML and XML formats. [Library]	Apache License, v 2.0
<b>Mockito</b> <a href="http://mockito.org/">http://mockito.org/</a>	A mocking library which allows <b>verifying</b> specifically what you want to test. [Library]	MIT License
<b>FEST-Assert</b> <a href="http://fest.easytesting.org/assert/wiki/pmwiki.php">http://fest.easytesting.org/assert/wiki/pmwiki.php</a>	An <b>assertion</b> framework, which more or less replaces JUnit Assert. Its main goal is to improve test code readability and make maintenance of tests easier. Can be used with either JUnit or TestNG. [Library]	FEST
<b>JBehave</b> <a href="http://jbehave.org/">http://jbehave.org/</a>	A Java framework for Behaviour-Driven Development. [Library]	BSD-style license
<b>JUnitDoclet</b> <a href="http://www.junitdoclet.org/home/index.html">http://www.junitdoclet.org/home/index.html</a>	It generates skeletons of Test Cases based on your application source code. [Library]	GNU LGPL
<b>JTest</b> <a href="http://www.parasoft.com/jsp/products/jtest.jsp;jsessionid=aaaavOWbct-Flh">http://www.parasoft.com/jsp/products/jtest.jsp;jsessionid=aaaavOWbct-Flh</a>	Automated Java testing and static code analysis product. It aims to improve Java code reliability, functionality, security, performance, and maintainability. Basic functionality includes Unit test-case generation, static analysis, regression testing, and code review. [Library]	Proprietary software
<b>Grinder</b> <a href="http://grinder.sourceforge.net/">http://grinder.sourceforge.net/</a>	<b>Load testing</b> framework that makes it easy to run a distributed test using many load injector machines. Load tests anything that has a Java API. [Load]	BSD-style open-source license
<b>EclEmma</b> <a href="http://www.eclEmma.org/">http://www.eclEmma.org/</a>	A plug-in for Eclipse based upon the EMMA Java <b>code coverage</b> tool. It adds a so called <i>launch mode</i> to the Eclipse workbench. [Coverage]	Eclipse Public License
<b>Cobertura</b> <a href="http://cobertura.sourceforge.net/">http://cobertura.sourceforge.net/</a>	A Java tool based on jcoverage which is a free <b>code-coverage</b> tool for Java™ programmers that allows them to measure the effectiveness of their Java tests and how much of a software program's code has been tested. [Coverage]	Apache Software License, v 1.1
<b>Nexus</b> <a href="http://nexus.sonatype.org/">http://nexus.sonatype.org/</a>	A Maven repository that allows <b>publishing of software artifacts</b> . [Deployment]	GNU GPL

Table 5: Testing tools

From the above, the following tools have been selected based on the previous experience of the partners involved: JUnit, JUnitDoclet, EclEmma, Grinder and Nexus, which are examined in more detail in the following subsection (Also included TestNG as will be seen on it).

## 8.3 Regulations, guidelines and good practice

### 8.3.1 Unit testing

Unit Testing Tips:

- Tests should be in the same package as tested code
- Separate tests for packaging
- Start small and work up
- Give useful names and assertions
- Run often and review the results
- Build tests around bugs you are fixing

For unit testing framework JUnit is proposed as testing framework. However, when dealing with software that needs parameterize testing, dependency testing and suite testing, TestNG tool would also be used since it is meant for complex integration tests and high-level testing. The choice for TestNG should be made as soon as such kind of testing is foreseen for the software under development.

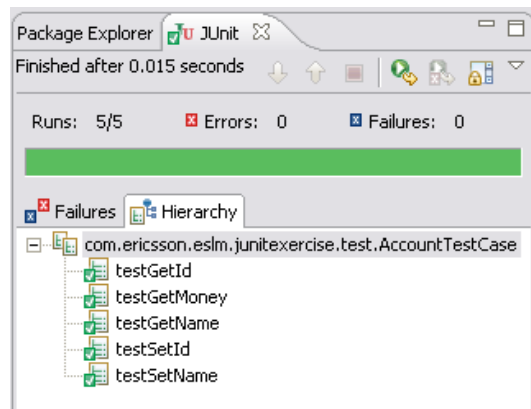


Figure 14: JUnit

To create Test Cases in JUnit there is a wizard available to make a skeleton containing a TestCase

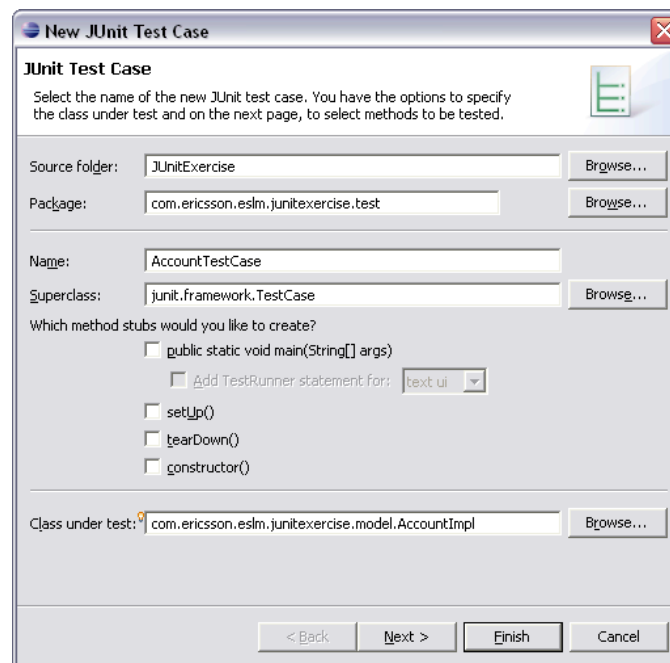


Figure 15: JUnit Test Case

Then a TestCase must be implemented for each method, like the following:

```
import junit.framework.TestCase;

public class SubscriberTest extends TestCase
{
    private static final String IMSI = "1234567890";

    /*
     * Test method for
     * 'com.ericsson.eslm.j2eecourse.junittest.model.Subscriber.getImsi()'
     */
    public void testGetImsi()
    {
        Subscriber subscriber = new Subscriber();
        subscriber.setImsi(IMSI);
        assertEquals("Imsi getter as setter", subscriber.getImsi(), IMSI);
    }
}
```

Figure 16: Simple Test Case

For detailed information about creating Test Cases with JUnit refer to the tutorial and links in the annex. Many other tools are complementary to JUnit that can be used for testing under special conditions or different kinds of applications:

JUnit extension	About	Web link
<b>JExample</b>	Uses dependencies between test cases to reduce code duplication and improves defect localization.	<a href="http://scg.unibe.ch/research/jexample/">http://scg.unibe.ch/research/jexample/</a>
<b>DbUnit</b>	Performs unit testing with database-driven programs	<a href="http://www.dbunit.org/">http://www.dbunit.org/</a>
<b>JUnitEE</b>	For testing Java EE applications	<a href="http://www.junitee.org/">http://www.junitee.org/</a>
<b>Cactus</b>	For testing Java EE and web applications. Cactus tests are executed inside the Java EE/web container	<a href="http://jakarta.apache.org/cactus/">http://jakarta.apache.org/cactus/</a>
<b>GroboUtils</b>	Provides automated documentation, class hierarchy unit testing, code coverage, and multi-threaded tests.	<a href="http://groboutils.sourceforge.net/">http://groboutils.sourceforge.net/</a>
<b>Mockrunner</b>	For testing servlets, filters, tag classes and Struts actions and forms.	<a href="http://mockrunner.sourceforge.net/">http://mockrunner.sourceforge.net/</a>
<b>XMLUnit</b>	Extends JUnit and NUnit to enable unit testing of XML. It compares a control XML document to a test document or the result of a transformation, validates documents, and compares the results of XPath expressions.	<a href="http://sourceforge.net/projects/xmlunit/files/xmlunit%20for%20Java/XMLUnit%20for%20Java%201.3/">http://sourceforge.net/projects/xmlunit/files/xmlunit%20for%20Java/XMLUnit%20for%20Java%201.3/</a>

Table 6: Complementary testing tools

### 8.3.2 Automated Test Generation

For automated test generation JUnitDoclet tool is proposed. It makes the process of testing software easier by offering a convenient way to create, organize, and maintain JUnit tests. JUnitDoclet Eclipse Plug-in<sup>25</sup> leverages this tool thanks to Eclipse APIs and UI functionalities.

<sup>25</sup> <http://sourceforge.net/projects/e-junitdoclet/>

### 8.3.3 Code Coverage

For code coverage tool EclEmma is proposed. It has a fast develop/test cycle which means that it launches from within the workbench. Coverage results are immediately summarized and highlighted in the Java source code editors. It does not require modifying projects or performing any other setup.

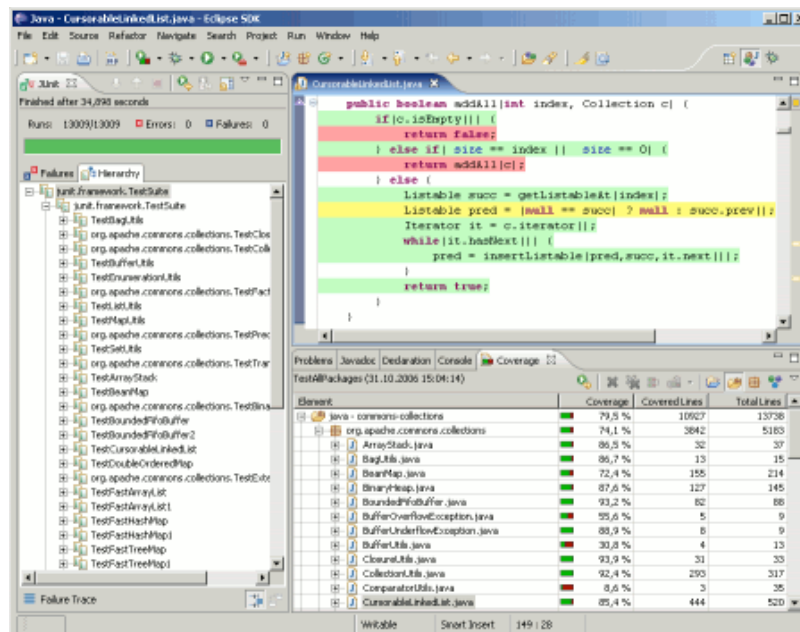


Figure 17: EclEmma

### 8.3.4 Load testing

The Grinder is a framework for running test scripts across a number of machines. Tests are written in the powerful Jython scripting language. It has a mature HTTP Support, automatic management of client connections and cookies. Sophisticated record and replay of the interaction between a browser and a web site is also possible.

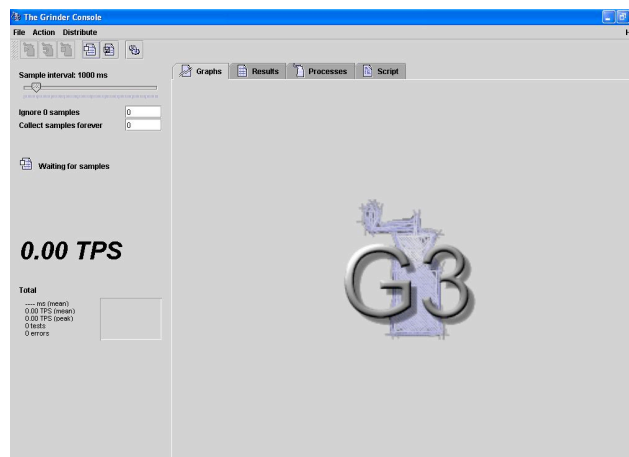


Figure 18: Grinder's console

### 8.3.5 Deployment

Nexus has been chosen as deployment tool due to its capability of integration with Maven. This way, Maven acts as the bridge from the testing to the deployment, as it can be integrated with both sides: A developer can use Maven to automatically test, build and deploy a software artifact. This is how the integration between testing and deployment is achieved as commented in the introduction of this section.



Take into account that this deployment is for internal development within the integration process, that is, a place where to find dependencies to developed code. It's just a "place" where to store the "jar" files for dependency resolution with Maven. Do not mistake it for the final deployment of binaries to the end user or external developer in universAAL. This is explained in the following section.

## **8.4 universAAL deployment strategy**

universAAL deployment strategy involves the Developer Depot and the uStore. Since the selection of tools for implementing these items is still work in progress in WP3, no assumption can be made here yet. Next versions of the deliverable will specify the connection between the deployment of developed code and the uStore and Developer Depot. This will be discussed between this tasks and subtasks from WP3 dealing with uStore and Depot, once these have been selected, taking into consideration the use cases defined for these tools in WP1. WP3 is already aware of this issue, though.

There is a possibility that the tools selected for deployment in uStore and/or Depot are the same than the one selected here (Nexus). In such case, any additional/different configuration needed for it would be updated in next versions. On the other hand, a different tool could be used, in which case Nexus could be replaced for it, or kept as an intermediate step between development and final deployment.

Whatever the case, the choice of Nexus in the previous section must be regarded in this initial version of the deliverable as a temporary measure in case development reaches a point in which code must be deployed but uStore and/or Depot are not yet ready.

## 9 Future Work

It is certain that the content of this deliverable will suffer modifications and additions. As no development has been started yet, we have no practical feedback on the coordination of all the selected tools, and we rely on the experience on previous projects where similar or the same tools have been used. There are chances that some of those tools are not performing as expected or not coordinating well with the rest. There is also a possibility that new and better tools or versions are launched or found. In any case, future versions of this deliverable will reflect any change in the selection or setup of the development tools.

Another point that must be addressed is the specific identification of the partners in each role identified here. In this version only belonging to a certain work package has been specified for certain roles, while next version should state the exact partner in each role for those that are constant (for instance, a responsible administrator for a tool). This will be possible when work on these aspects begin and partners can express their availability in willingness and resources for assuming that role.

Relationship to other deliverables and tasks must also be updated. Due to being one of the first deliverables to be released, there weren't many chances to coordinate the content, although other tasks have been notified of the intention of this document. However, in next versions of this deliverable, as work will be more mature and there will be time to share the common ideas, the contents will be enriched and specialized from the synergy.

For remaining decisions to be made which have not been taken in this version they will be made as soon as needed and partners will be reported about them at that moment. In other words, despite being reflected in next version of this deliverable in M15, the decision will have been made before, when it was needed.

Annexes will be more explicit and exact as the first developers start using the tools and solve any issues they may find. Also some administrator or contact persons for support on each tool may still have to be decided.

As closing word, future work will deal with those aspects in each section of this document where it is stated that they will be developed in next versions.

## Appendix A. Quick Developer Reference

### A.1 Setup Instructions

#### A.1.1 Generic

- **Language:** Java. Installation of JDK 1.4
  - <http://java.sun.com/javase/downloads/index.jsp>
- **Build lifecycle:** Maven.
  - <http://maven.apache.org>

#### A.1.2 IPR

- **IPR Directory:** To be refined in D9.3. As a temporary reference, use WP9 folder in eRoom.

#### A.1.3 Development environment

- **IDE:** Eclipse. Installation of Eclipse IDE for Java Developers
  - <http://www.eclipse.org/downloads/>
- **Plug-ins:** Installation of latest version plug-ins unless other is specified. Use Eclipse plug-in update feature to download and install the following plug-ins from their respective update sites:
  - Subclipse: [http://subclipse.tigris.org/update\\_1.6.x](http://subclipse.tigris.org/update_1.6.x)
  - ADJT: <http://download.eclipse.org/tools/ajdt/35/update>
  - M2Eclipse: <http://m2eclipse.sonatype.org/sites/m2e>
  - Pax Runner: <http://www.ops4j.org/pax/eclipse/update>
- **Configuration:** Configuration of all tools and selection of plug-ins is strongly dependant on decision made later in the development process regarding universAAL middleware. Accordingly configuration data will be shared among developers as soon as possible.

#### A.1.4 Coding standards

- **Style:** The Maven code styling conventions will be used. They are closely related to Sun Java code styling conventions. Automated code formatting in Eclipse can be configured using Project -> Properties -> Java Code Style -> Formatter. Choose “Java Conventions”.
- **Comments and Documentation:** Commenting the code properly is very important if other developers need to work on the same source in order to increase readability. Javadoc allows generating HTML pages from source code commentary which simplifies understanding code for outside developers. Read the following tutorials and guidelines:
  - <http://java.sun.com/docs/codeconv/>
  - <http://java.sun.com/j2se/javadoc/writingdoccomments/>
  - <http://java.sun.com/j2se/javadoc/writingapispecs/index.html>

#### A.1.5 Project management and issue tracker

- It has been decided to use GForge, which includes Issue Tracker. Information and downloads are available through:
  - <http://gforgegroup.com/es/download.php>

- <http://gforge.org/gf/>
- Video tutorials can be found in:
  - <http://gforgegroup.com/training/>

### *A.1.7 Testing*

- **Unit testing:** JUnit library can be downloaded from its homepage. Integration with maven lifecycle will be addressed in next versions.
  - <http://www.junit.org/>
- Instructions and tutorials can be found in these links:
  - <http://junit.sourceforge.net/>
  - <http://junit.sourceforge.net/doc/cookbook/cookbook.htm>
- **Automated test generation:** JUnitDoclet plug-in for Eclipse will be used. It can be downloaded from:
  - <http://sourceforge.net/projects/e-junitdoclet/>
- **Code coverage:** EclEmma Eclipse plug-in will be used. Download through Eclipse from its update site:
  - <http://update.eclEmma.org/>
- **Load testing:** Grinder can be downloaded from:
  - <http://grinder.sourceforge.net/download.html>
- Tutorials and documentation are in:
  - <http://grinder.sourceforge.net/g3/getting-started.html>

### *A.1.8 Deployment*

- **Deployment:** Deployment tool depends on the results of tasks in WP3. In the meantime, if any deployment is needed, Nexus should be used:
  - <http://nexus.sonatype.org/>

## A.2 Summary of selected tools and responsible people

Tool	Key Decisions	Administrator/contact person
<b>Generic</b>		
<b>Java Development Kit</b>	<ul style="list-style-type: none"> <li>• Most common OS-independent framework</li> <li>• Various preceding projects relying on Java</li> </ul>	Andreas Braun (Fh-IGD)
<b>Maven</b>	<ul style="list-style-type: none"> <li>• Build automation</li> <li>• Guidelines and best practices for Java projects</li> </ul>	Saied Tazari (Fh-IGD)
<b>IPR</b>		
<b>IPR Directory</b>	<ul style="list-style-type: none"> <li>• To be defined in D9.3</li> </ul>	Francesco Furfari (CNR)
<b>Development environment</b>		
<b>Eclipse IDE</b>	<ul style="list-style-type: none"> <li>• Most common Java IDE</li> <li>• Large set of plug-ins</li> </ul>	Andreas Braun (Fh-IGD)
<b>Eclipse Plug-ins</b>	<ul style="list-style-type: none"> <li>• Currently used plug-ins <ul style="list-style-type: none"> <li>○ Subclipse</li> <li>○ ADJT</li> <li>○ M2Eclipse</li> <li>○ Pax Runner</li> </ul> </li> <li>• Some results also published as Eclipse plug-in</li> </ul>	Andreas Braun (Fh-IGD)
<b>Coding standards</b>		
<b>Maven Formatting</b>	<ul style="list-style-type: none"> <li>• Best practice and default setting in Maven</li> <li>• Highly readable and easy to understand</li> </ul>	Andreas Braun (Fh-IGD)
<b>Javadoc</b>	<ul style="list-style-type: none"> <li>• De-facto standard for Java code commenting</li> <li>• Automated generation of HTML documentation</li> </ul>	Andreas Braun (Fh-IGD)
<b>Project management</b>		
<b>GForge</b>	<ul style="list-style-type: none"> <li>• Includes Issue Tracker</li> <li>• Eclipse plug-in available for purchasers</li> </ul>	To be decided
<b>Testing</b>		
<b>JUnit</b>	<ul style="list-style-type: none"> <li>• For complex integration and high level testing, consider using TestNG</li> <li>• Several complements are available for JUnit for different conditions or applications</li> <li>• Integration with Maven to be described later</li> </ul>	To be decided
<b>JUnitDoclet</b>	<ul style="list-style-type: none"> <li>• Use plug-in for Eclipse</li> </ul>	To be decided
<b>EclEmma</b>	<ul style="list-style-type: none"> <li>• Use plug-in for Eclipse</li> </ul>	To be decided
<b>Grinder</b>	<ul style="list-style-type: none"> <li>• Specially for web applications where load capacity must be tested</li> </ul>	To be decided
<b>Deployment</b>		
<b>Nexus</b>	<ul style="list-style-type: none"> <li>• Awaiting result of tasks in WP3</li> </ul>	To be decided