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

Universal Open Architecture and Platform for Ambient Assisted Living

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

Document Identification			
Deliverable ID:	**D1.3-A**	Deliverable title:	universAAL Reference Architecture
Release number/date:		V1.0 25.06.2010	
Checked and released by:		Sergio Guillén/ITACA	

Key Information from "Description of Work" (from the Grant Agreement)	
Deliverable Description	*Specification of the Reference Architecture. Text, reference figures, UML diagrams. Rules about how to interpret the specification. Includes universAAL protocol specifications, API specifications and ontology.*
Dissemination Level	PU=Public
Deliverable Type	R = Report
Original due date (month number/date)	Month 3 / 30.04.2010

Authorship& Reviewer Information	
Editor (person/ partner):	Gema Ibáñez/ITACA
Partners contributing	Erlend Stav (SINTEF), Ståle Walderhaug (SINTEF), Saied Tazari (FhG), Stefano Lenzi (CNR), Francesco Furfari (CNR), Peter Wolf (FZI), CERTH, Laura Belenguer (ITACA), Juan Carlos Naranjo (ITACA), Gema Ibáñez (ITACA), Patricia Abril (UPM), Sandeep Kumar (Philips), Venelin Arnaudov(ProSyst), Oliver Höftberger (TUW), Miran Mosmondor (ENT)
Reviewed by (person/ partner)	Ivan Benc (ENT) – Sten Hanke (AIT)

Release History

Release number	Date issued	Milestone*	eRoom version	Release description /changes made
0.1	26.03.2010	TOC approved	V1	ToC First internal release.
0.2	29.03.2010	ToC approved	V2	Chapter sections extended.
0.3	29.03.2010	ToC proposed	V3	ToC external review proposed.
0.4	30.03.2010	ToC proposed	V4	Sections assigned to partners
0.5	30.03.2010	ToC approved	V5	ToC external review approved
0.6	15.04.2010	Intermediate	V5	Editor changed
0.7	28.04.2010	Intermediate	V6	Added section 4.1
0.8	11.05.2010	Intermediate	V7	Added section 6.2
0.9	12.05.2010	Intermediate	V8	Added section 4.2
1.0	25.06.2010	Release	V19	Technical manager release
1.0	22.09.2010	Release	V20	Front cover, template and footer 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 7th Framework Programme, Priority 7.1.b (ICT & Ageing). The consortium members are:

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

Table of Contents

Table of Figures

List of Tables

Executive summary

This deliverable D1.3 is related to Task 1.4 “Consolidated AAL Reference Architecture specification” in work package 1 (WP1), and is connected with D1.1 “universAAL Reference Requirements” and D1.2 “universAAL Reference Use Cases” deliverables, and several work packages. Both requirements and use cases drive the architecture development. From the context and requirement view described in D1.1 and D1.2, the component and distribution views will be defined in this deliverable. In WP2 will be implemented the components defined in the reference architecture. Scope and boundaries of services can be implemented with the platform, which are defined in WP3 and includes development tools and services. The reference architecture is main result of the project, which is subject of standardization in WP8.

Also, the deliverable acts as roadmap for the current and the future work concerned to this deliverable and it captures initial work done at this first stage in the analysis and consolidation phase of the development process in universAAL, giving a first look on a universAAL Reference Model.

Terminology and layered reference model, as component views, are presented and consolidated from various input projects (SOPRANO, MPOWER, PERSONA, OASIS, AMIGO and GENESIS) in order to establish a common understanding of the AAL domain. The AAL reference terminology captures most important concepts used in documentation of the AAL Reference Architecture. The AAL layer model presents a generic pattern for structuring AAL software components. Both terminology and layered reference model correspond with the first version of the reference model that will be validated in order to be used as base for the universAAL Reference Architecture in further deliverables. Later versions of this deliverable will report next steps undertaken to arrive at the universal Reference Architecture that will be used to define platform components and services and their relationships. Further, the resulting Reference Architecture will be described according to the ARCADE methodology.

1 About this Document

1.1 Role of the deliverable

Deliverable 1.3 will report on work done in Task 1.4 "Consolidated AAL Reference Architecture specification". As a specification tailored to the AAL domain the reference architecture facilitates the development of an AAL platform that subsequently enables provision of AAL services.

In general, software architecture comprises software components, the externally visible properties of those components, and the relationships between them. The specification or documentation of a system's software architecture facilitates communication between stakeholders, documents high-level design, and enables reuse and maintainability of components and patterns between projects. The specification of the universAAL reference architecture follows the ARCADE [1] architecture description framework. The ARCADE framework was developed to assist the software architect by providing documentation formats and structure, by handling important quality related concerns and by ensuring successful reusability and maintainability of architectural components.

Furthermore, a consistent AAL terminology and AAL layered reference model is presented. This deliverable describes the component and distribution views that can be considered follow ups to context and requirement view described in D1.1 and D1.2, respectively. Both views consolidate input coming from various projects including SOPRANO, MPOWER, PERSONA and OASIS. The AAL reference terminology captures most important concepts used in documentation of the AAL Reference Architecture. The AAL layered reference model presents a generic pattern for structuring AAL software components. Both terminology and layered reference model have been derived from various AAL-related and technical sources to ensure a complete and useful representation from a domain and technical point of view.

The AAL Reference Architecture will define platform components and services and their relationships. To some extent it will also outline possibilities of future platform extensions, scope and design of applications providing services to the user, possibilities for the integration of tools and maintainability and reusability of software components. Further, terminology and reference model will outline the general scope the system and the domain that is taken into consideration.

In particular, this deliverable is related to the following universAAL deliverables and work packages:

D1.2 -- universAAL Reference Requirements and D1.1 – universAAL Reference Use Cases for AAL: Both requirements and use cases drive the architecture development. They define scope and boundaries that the system's architecture has to comply with. Accordingly, the ARCADE framework defines that component and distribution view as follow-ups to context and requirement view (see Figure 1).

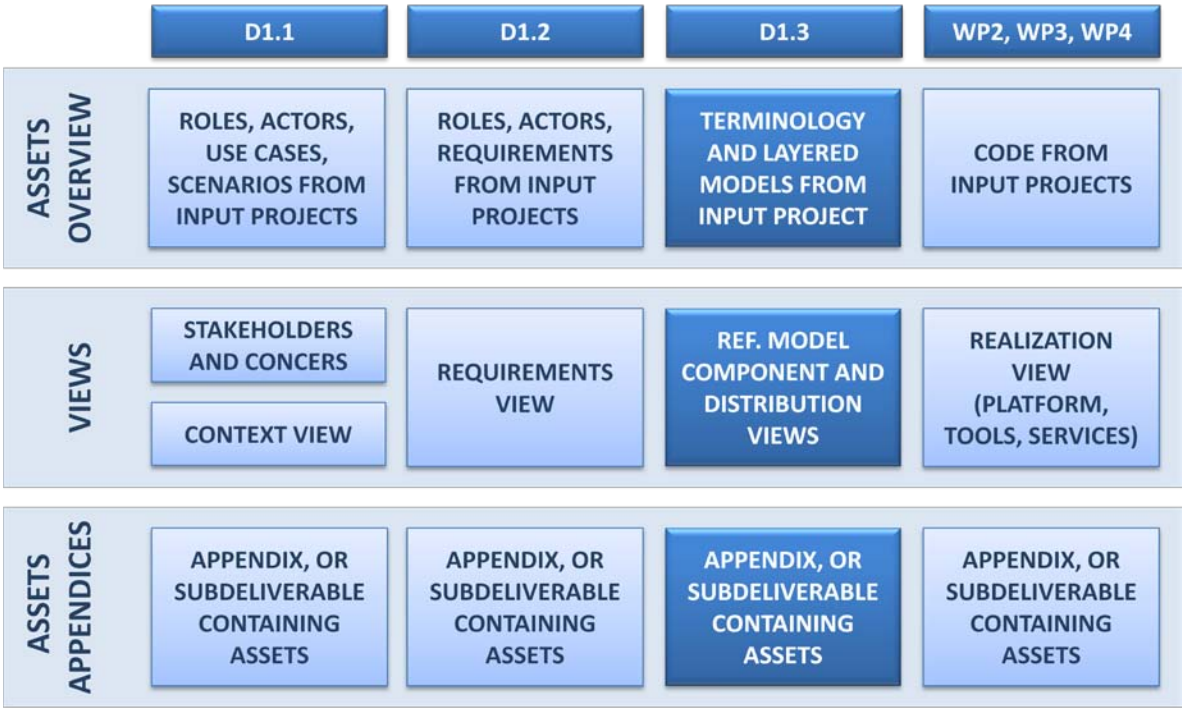

Figure 1: ARCADE artifacts addressed in the different universAAL deliverables

- **WP2 – Open Source AAL Platform and Implementation:** WP2 implements the components as defined in this reference architecture. The process of implementation as well as the consideration of runtime properties of the system contributes to the realization view of the system (see Figure 1)
- **WP3 – Tools and Tutorial and WP4 – Innovative service concept implementation:** As explained above the platform architecture will to some extend define scope and boundaries of services that can be implemented with the platform. This includes development tools and services of the Developer Store as well as services stored within the uStore targeting at end users and uAAL authorities.
- **WP8 – Community building & standardization and WP9 -- Dissemination & Exploitation:** The AAL reference architecture will be a main result of universAAL project. Parts of it or the whole architecture are subject of standardization in WP8. Due to its importance for the AAL community and its stakeholders, the reference architecture is a major constituent of scientific and community dissemination as well as exploitation.

Note that roles and actors are common in both deliverables, but are involved in different processes, Appendix, or subdeliverable containing assets could be different information contained in each deliverable.

1.2 Relationship to other versions of the deliverable

Later versions of this deliverable will report further steps undertaken to arrive at the universAAL Reference Architecture. This includes a more detailed description of how distribution and consolidation view have been achieved. Further, the resulting Reference Architecture will be described according to the ARCADE approach. The layered reference model as well as the terminology may be updated according to information gathered in the later stages of the project. For a broader overview on future work regarding universAAL Reference Model and Architecture, please have a look at Section 6 “Future Work”.

1.3 Structure of this Document

This deliverable captures initial work done on the universAAL Reference Architecture and provides a first look on a universAAL Reference Model. Section 3 describes the relationship between and the rationale behind the reference model and the reference architecture and acts as a roadmap for this and further deliverables. Section 4 provides the input to the terminology as part of the reference model. This input is structured according to the external projects that it has been extracted from. The next section lists the similar information as input to the layered model. Section 5 closes the discussion on the reference model by presenting the consolidated universAAL result. Finally, Section 6 outlines future work in relation to the universAAL Reference Model and Architecture.

2 Roadmap to the universAAL Reference Architecture

2.1 From Reference Model to Reference Architecture

Figure 2 shows a scheme of the development process as outlined in the Description of Work (DoW) Annex of the project contract. At a first glance, this looks quite similar to common development processes; however, there is at least this decisive difference when considering that the analysis phase comprises also the consolidation of known and accessible results from earlier R&D. The rationale behind this and the scope of the consolidation, however, is discussed in the next section.

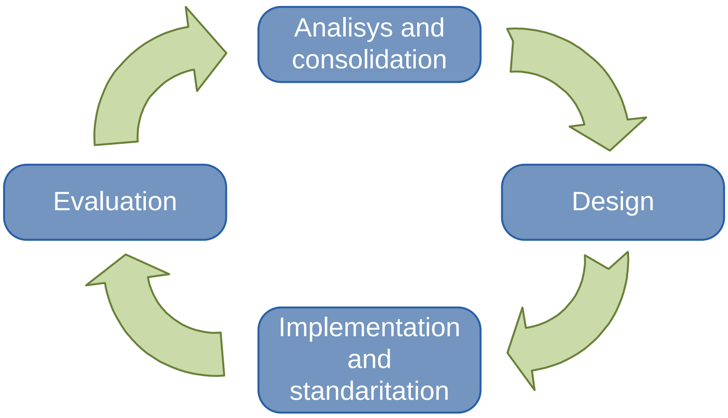

Figure 2: A scheme of the development process in universAAL

If we map the project results from the first few months to the first iteration in the above process, the scene will change as shown in Figure 3. This means that we see the version A of D1.3 (this report) as the result of consolidating the design work done in the input projects in order to establish a common understanding of the AAL domain. We call this common understanding of AAL systems the universAAL reference model. Then the version B will introduce a first version of the universAAL reference architecture based on this reference model.

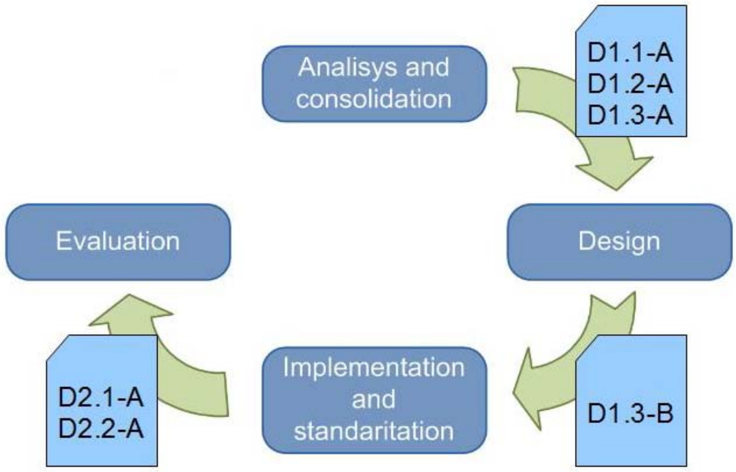

Figure 3: Project results in the first iteration

Putting the above thoughts together, we end up with the relationships of the main technical deliverables of the project as summarized in Figure 4. This is also conform with the common definitions of the terms reference model, reference architecture, and realization architecture, for instance, the definitions provided by the Organization for the Advancement of Structured Information Standards[1] summarized in Table 1.

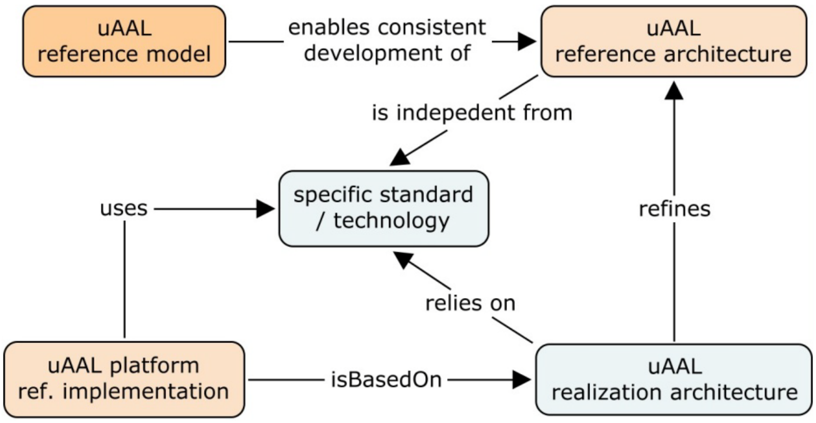

Figure 4: The relationships between the main technical deliverables of universAAL

Accordingly, the reference model defined in this report is supposed to enable the consistent development of the reference architecture without any technological bias, while the universAAL platform as the reference implementation of that architecture will have to make technological choices.

Table 1: Term definitions relevant for architectural design

Term	Definition	Source
Reference Model	A reference model is an abstract framework for understanding significant relationships among the entities of some environment. It enables the development of specific reference or concrete architectures using consistent standards or specifications supporting that environment. A reference model consists of a minimal set of unifying concepts, axioms and relationships within a particular problem domain, and is independent of specific standards, technologies, implementations, or other concrete details.	SOA-RM [2]
Reference Architecture	A reference architecture models the abstract architectural elements (building blocks) in the domain independent of the technologies, protocols, and products that are used to implement the domain. It differs from a reference model in that a reference model describes the important concepts and relationships in the domain focusing on what distinguishes the elements of the domain; a reference architecture elaborates further on the model to show a more complete picture that includes showing what is involved in realizing the modelled entities.	SOA-RA [3]
Realization Architecture	By increasing the level of detail in a reference architecture, we can end up with a concrete architecture	universAAL (derived from

[1] Actually, by a technical committee at this standardization body that is working on the topic of Service-Oriented Architectures. OASIS – not to be mixed up with the EU-FP7 project OASIS that is one of the input projects to universAAL

	that specifies all the technologies, components and their relationships in sufficient detail to enable direct implementation. We refer to such a concrete architecture as the realization architecture.	SOA_RA)

2.2 Rationale behind the Collection of Former Project Architectures

The number of research projects and industrial labs dedicated to the area of Ambient Assisted Living (AAL) is constantly increasing. It becomes more and more crucial to identify reusable results in different areas of architecture, technologies, protocols, and standard building blocks. Apart from those domain-specific solutions for rather constrained scenarios, there have also been general-purpose results coming from several research projects in the field of AAL. It seems to be high time for an evaluation of significant existing solutions in order to foster the identification and re-usability of domain-independent units in an AAL environment and avoid re-inventing the wheel over and over.

Hence, universAAL decided to be a pioneer in reusing existing AAL technology by identifying and utilizing available solutions in a consolidation process. These include the results from the projects Amigo, GENESYS, MPOWER, OASIS, PERSONA, and SOPRANO.

To force reuse, we chose to break the normal software engineering sequence of use case and requirement analysis, design and implementation, and evaluation and feedback and rely on parallel work in several threads, each collecting, categorizing, comparing, harmonizing, merging, and prioritizing one of the engineering results from the input projects. We basically believe that well-founded work based on software engineering techniques has already been done in those input projects and we should benefit from it.

One of those parallel threads is dedicated to the architectural design of AAL systems. Taking the roadmap from the previous section into account, it should be obvious now that the collection of the basic concepts from the input projects is an essential step towards the provision of the universAAL reference model for AAL. The question of further steps in this process is discussed in the next section.

2.3 Methodology to Consolidate former Architectures: ARCADE

The ARCADE methodology for developing architectural descriptions is being used in universAAL as the overall methodology with local customizations to meet the needs of the various tasks. ARCADE defines a small set of high-level artefacts that constitute the software architecture (For a description of these artefacts please consult ARCADE handbook [4]). Central artefacts for this deliverable are the *views* that will be defined as the result of the work done in Task 1.4 “Consolidated AAL Reference Architecture Specification”. In addition, a number of *assets*[2] have been imported from the input projects and constitute the basis for the architecture work in Task 1.4.

It is important to note that this version of the deliverable does not yet contain all the necessary views. Views will be defined gradually and will be available in later versions of this deliverable. For this version of the deliverable the main focus has been the collection, analysis and structuring of some of

[2] System assets are sources of information that can be used when developing the architecture descriptions. System assets can be considered as implicit requirements, which are not necessary to include in the requirement view, however assets may be included in component, deployment and realisation views. Examples of assets that are available are: a dictionary as a reference list of important concepts, standards that is a formalised model or example developed by a standardisation organisation or established by general consent and patterns, which are descriptions of a recurring, well-known problem and a suggested solution.

the assets that are central to Task 1.4. This chapter describes how ARCADE will be used throughout the task, and how end results will look like in terms of artefacts and deliverables.

2.3.1 Views

Views in ARCADE define ways of looking at the software architecture. ARCADE proposes a set of standard views, and allows the definition of new views or customization of existing views. For the purpose of universAAL a new Reference model view has been defined, while two of the standard views are used in this deliverable. Figure 1 in section 2.2 gives an overview of which deliverables that address the other ARCADE views.

2.3.1.1 Reference model view

Reference model view is defined specifically for universAAL. universAAL aims at defining a reference architecture that can be instantiated in various forms. The reference model view is constituted by two parts:

- Terminology model: A set of terms and concepts, and the relationships among them, which are defined by the universAAL reference model.
- Layered model: A set of architectural layers with the corresponding service areas and responsibilities. The collection of these layers will cover the areas of responsibility for the entire universAAL platform.

In the context of universAAL the reference model will be used to:

- Compare and consolidate concepts from the different input platforms
- Compare and consolidate the architectural layers of the input platforms (using the layered model)
- Guide the development of the reference architecture (use both terminology and layers)

2.3.1.2 Component view

The component view defines the logical/functional components of the reference architecture, the interfaces among the components, and the interfaces to the external world. Component view is a standard view defined by ARCADE. UML and similar formalisms will be used to define the component view. This view will be added and further described in the next version of this deliverable.

2.3.1.3 Distribution view

The distribution view defines the logical distribution of the components from the component view. The distribution view will define which components logically belong together and which don’t, and how communication among the different groups of components will be realized.. This view will be added and further described in the next version of this deliverable.

2.3.2 Assets

In the current version of this deliverable, the type of asset most relevant for universAAL architecture are the ones provided by the input projects. For each of the views above there is a large body of knowledge residing in the input projects. The following assets have been identified to be of importance to current work:

- Terminology models: The terminology models from input projects are collected and are presented in this version of the deliverable. See Section 4.1.

- Layered models: An analysis of layering in input projects is done compared to the initial layered model in the universAAL description of work. For each input project attempt is done to find a mapping to the universAAL layered model. This is documented in Section 4.2.
- Component models: An analysis of existing components from input projects will be done in the next version of D1.3. The goal is to identify existing components and overlaps in functionality in these components.
- Distribution models: Similar to component models, distribution models from input projects will be analyzed in the next version of the deliverable and be used as input to constructing the universAAL reference architecture.

2.3.3 Applying ARCADE to D1.3

Although ARCADE will be deployed in universAAL using various tools such as modelling and design tools, the resulting artefacts will be documented in the contractual deliverables. The set of deliverables in WP1 (D1.1, D1.2 and D1.3) together with design and realization deliverables from WPs 2, 3 and 4 will document all the resulting artefacts as described in ARCADE. Figure 1 in Section 2.2 illustrate what each deliverable will contain. For D1.3, the content of the final D1.3 will consist of the following main parts:

- Assets overview: A list of references to input material, including material from input projects but also input from e.g. standardization bodies, other methodologies, dictionaries, and other reference material.
- Reference model view: will document the reference model for universAAL. Will consist of section for universAAL terminology and layered model. UML class diagrams, entity-relationship diagrams and similar will be used.
- Component view: will document universAAL component model. UML class diagrams, component diagrams and similar will be used.
- Distribution view: will document universAAL distribution view. UML distribution diagrams and similar will be used.
- Appendices: will document assets used in Task 1.4 as described in Assets overview chapter.

3 Input to reference model from other projects

3.1 Terminology model (or concepts and relationships)

This subchapter presents a collection of relevant terminology, concepts and ideas from each of the following input projects: AMIGO, GENESYS, MPOWER, OASIS, PERSONA, and SOPRANO.

The terminology from the input projects is an important input to creating a consolidated terminology which is part of the initial reference model of universAAL. The terminology from each project is summarized in a table with the following columns:

- **Concept**: This is the concept in question. These are concepts which can be central to the universAAL reference model and the reference model's purpose (see 3.3.1.1 for a definition of what the reference model will be used for)
- **Definition**: This is a textual definition of the term, explained in a way that makes sense to a person not knowing the platform in detail. It can also identify relation to other terms in the list.
- **Relevance to universAAL**: This gives a brief and concrete explanation for why the term is relevant to the universAAL reference model and why it should be in the reference model.
- **Reference**: When possible, references are provided to the original resource in which the term is explained. A summary of the reference documents with URLs are provided before the table of each project.

Due to the large extension of the tables where the terminology from each project is summarized, they have been moved and can be found in the Appendix A at the end of this deliverable.

3.2 Layer model

This subchapter presents the layer models of the following input projects: AMIGO, GENESYS, MPOWER, OASIS, PERSONA, and SOPRANO. In cases where the input project does not define an explicit layer model, a description has been provided of the "de-facto" layer model used.

The description of the layer models consists of an introduction, one (or more) figure(s), and a table with the following columns:

- **Layer/sidecar**: This name used for the layer.
- **Description**: A description of what the layer does and/or contains.
- ***Project* use**: An explanation of how this layer was used in the input project

3.2.1 AMIGO Layer Model

The Amigo Open Source Software follows the paradigm of Service Orientation, which allows developing software as services that are delivered and consumed on demand. The benefit of this approach lies in the loose coupling of the software components that make up an application. Discovery mechanisms can be used for finding and selecting the functionality that a client is looking for. Many protocols already exist in the area of Service Orientation. The Amigo project supports a number of these important protocols for discovery and communication in an interoperable way. This makes it possible for programmers to select the protocol of their choice while they can still access the functionality of services that are using different methods.

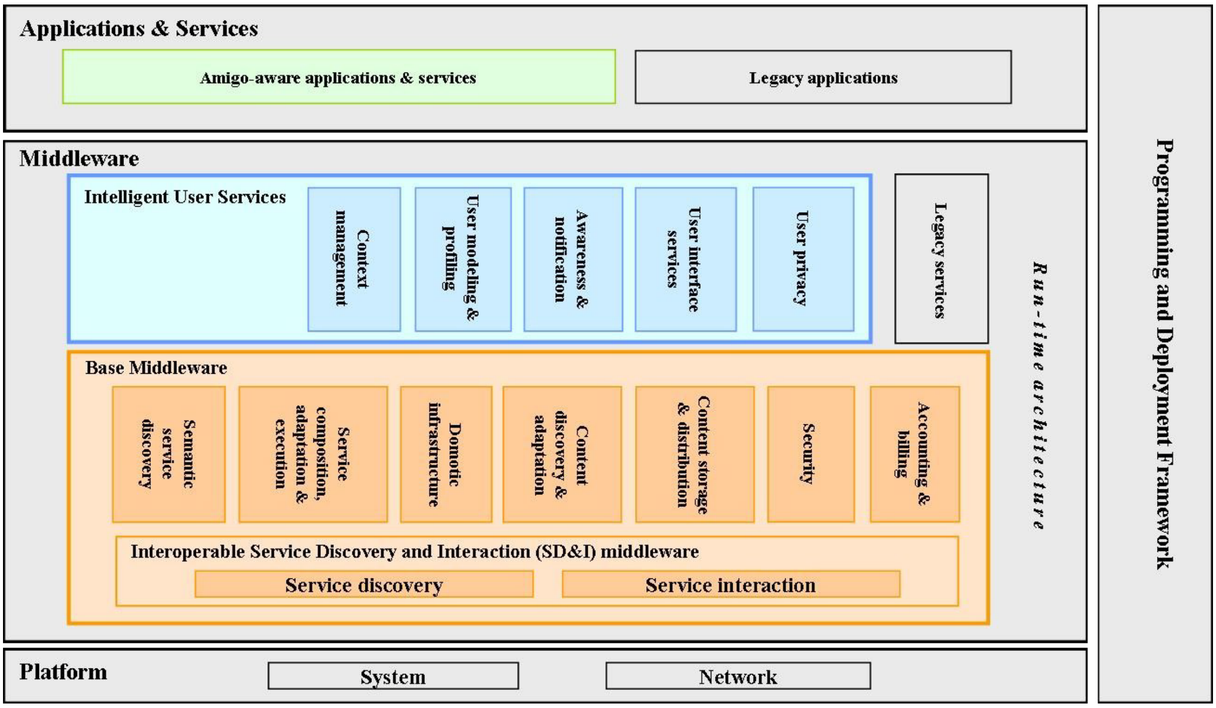

Figure 5: Amigo layer model

Table 2: Layers from Amigo project

Layer/sidecar	Description	Amigo use
Applications & Services	Both functional and non-functional properties of services are specified, both syntactically and semantically.	In the application layer of the Amigo abstract reference service architecture, Amigo services enjoy an enriched service description.
Base Middleware	The Base Middleware contains the functionality that is needed to facilitate a networked environment. It provides the semantics to communicate and discover available services and devices in the network, including the ones that are based on existing communication and discovery standards, such as UPNP, WS, or SLP. This implies that independence is accomplished for existing hardward- and software, and new services can be discovered and composed. In addition, security mechanisms for authentication, authorisation, and encryption are provided.	Amigo project develops middleware that dynamically integrates heterogeneous systems to achieve interoperability between services and devices.

Intelligent User Services	The Intelligent User Services broker between users and service providers, and provide context information, combine multiple sources of information and make pattern-based predictions. Information is tailored to user profiles and adapts to the user's situation and changes in the context.	The Intelligent User Services in Amigo contain the functionality that is needed to facilitate an ambient in-home network.
Programming and Deployment Framework	The Programming and Deployment Framework contains modules that facilitate the development of services in by providing support for interoperability, security and service description to service developers. Amigo supports and abstracts over several important protocols used for discovery and communication. Therefore, heterogeneous services can be integrated into the networked home independently of their underlying software and hardware technologies.	Facilitates the development of Amigo-aware services in .NET or Java.

3.2.2 GENESYS Layer Model

In GENESYS services are structured forming a waistline, as depicted in Figure 6. This layer model is inspired by the Internet, where the Internet Protocol (IP) forms a waist between underlying communication technologies (e.g., Ethernet networks, wireless protocols) and higher level protocols on top of the IP (e.g., UDP, TCP). These higher level protocols can further be refined to more application specific protocols, like HTTP, FTP, etc.

Similarly to this, GENESYS defines *core services* that represent the waist. These core services (i.e., global time, communication, configuration and execution control) are required in all instantiations of GENESYS. For each of the core services different underlying implementation options exist. As an example, the communication service can be based on a switched Network-on-a-Chip (NoC), Ethernet or wireless protocol.

Towards the top of the waistline, platform services can be successively refined and extended. This way, more powerful and specialized platform services can be obtained. At first, *domain-independent optional services* are built above the core services. These services can further be refined to construct *domain-specific services*, where *central* and *optional* services can be distinguished. Actual a*pplication services*, that use domain-specific services of underlying layers, are situated at the top of the waistline. In the subsequent table the rationale of each layer will be explained.

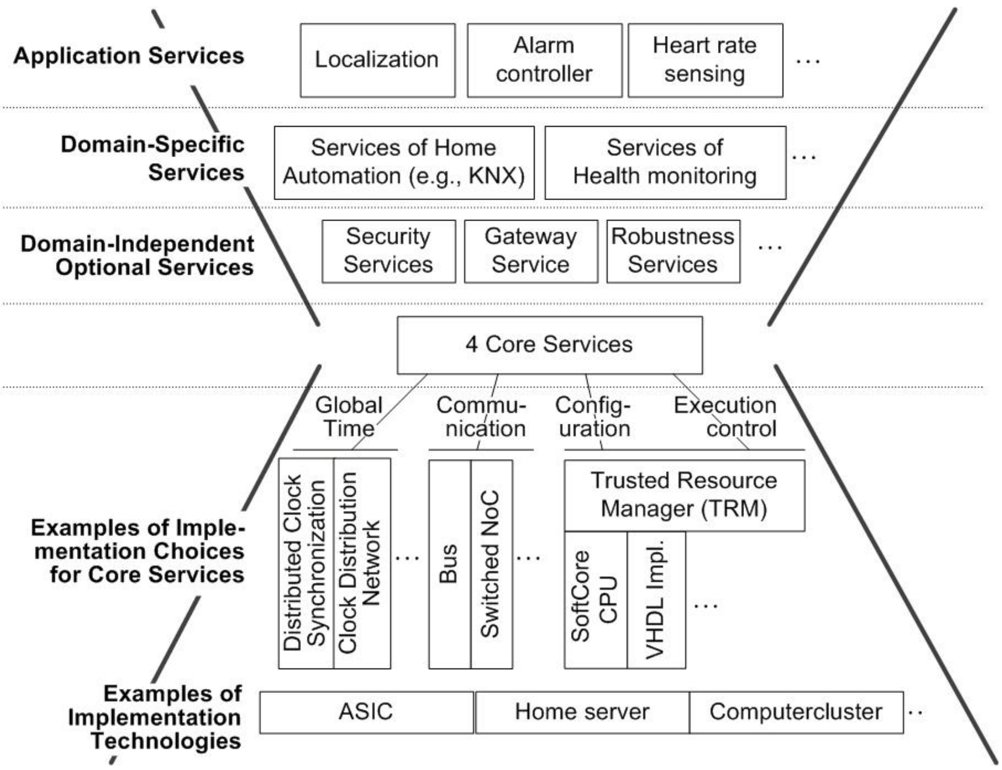

Figure 6: Waistline structure of GENESYS services

Table 3: Layers from GENESYS project

Layer/sidecar	Description	GENESYS use
Core Services	Provide capabilities which are required in every target application domain (e.g., Home Automation, Health care, etc.). These services are at the waist of the layer model. The actual implementation of core services depends on the choice of underlying implementation technologies.	In the GENESYS architecture core services comprise global time services, communication services, configuration services and execution control services.
Domain-Independent Optional Services	Are built on top of the core services and provide functionalities that can be used for different application domains.	Domain-independent optional services are not required for every instantiation of GENESYS, but extend the capabilities of the core services. Exemplary services are security services, gateway services, robustness services, etc.
Domain-Specific Services	These services are focused towards a specific application domain. Domain-independent optional services and core services below the domain-specific services are further enhanced.	GENESYS distinguishes *domain-specific central services*, which are considered essential for the specific application domain, and *domain-specific optional services*, that supplement the service set provided to the layer above. An example of a domain-specific service would be the implementation of a KNX-interface for the home automation domain.
Application Services	Application services are situated at the top of the waistline structure. Services of all underlying layers can be used. This layer provides services that represent the actual value for the user of the platform.	At the application service layer of the GENESYS architecture services are implemented which the user of the platform actually requires. For example an emergency service, that can locate the assisted person, perceive its heart rate and in case of need trigger an alarm, may be such an application service.

3.2.3 *MPOWER Layer Model*

The MPOWER layered model is an adaption of the layer model from the IBM SOA Reference Architecture [11]. As shown in the figure, it consists of five main layers – each layer comprising a set of “components” that conforms to the rules and requirements specified for the layer. In addition, three “sidecars” and their relations to the layers are included in the model. Each layer and “sidecar”, and their application in MPOWER is briefly described in the table. The figure also indicates how the different groups of MPOWER services map to the reference architecture.

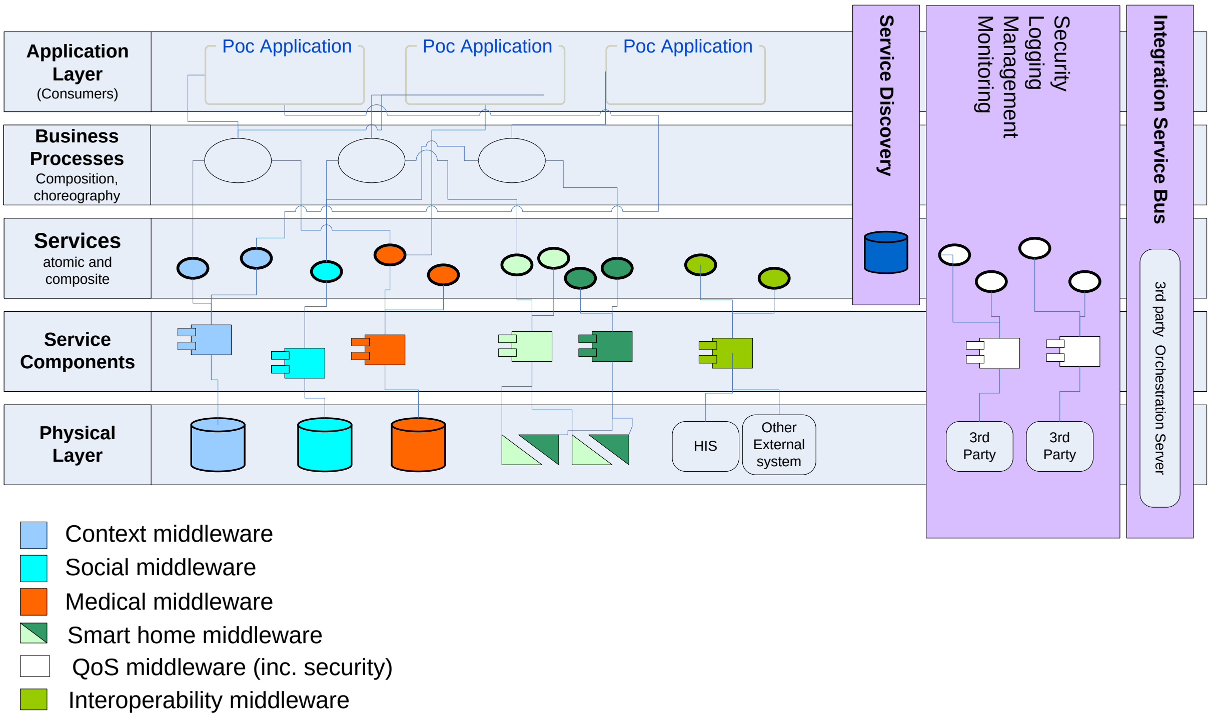

Figure 7: The MPOWER layer model

Table 4: Layers from MPOWER project

Layer/sidecar	Description	MPOWER use
Application layer	Provides user interface and application specific components, decoupling these from the underlying (business) services on which they build.	Applications built using MPOWER, including the pilots, belong to this layer. These provide the access point through which the users of applications access the services.
Business processes	Defines the business rules and process of the applications. Services are bundled into a flow through orchestration or choreography, and thus act together in supporting use cases and business process of the application.	Used to define business rules of the MPOWER pilot applications. An example of an assistive care business process is management of a shared calendar where calendar, patient and caregiver information, and medical plans are accessed through a set of services and service components.

Services	Provides services available for invocation. Service implementations may use service components in their realization, and expose their functionality through service interface descriptions. Services can be made available for service discovery through a registry.	The main functionality of the MPOWER platform is provided as services. Examples of services from MPOWER are Authentication, Calendar Management, Medication Management, External Notification, Door Control Management, and iCal Calendar Export.
Service components	Exposes the functionality of the components and databases in the resource layer. The Service components provide a high-level access to their information and control functions.	A typical service component in MPOWER is a smart house sensor driver that encapsulates and implements the sensor communication logic for the higher layer services.
Physical layer	Consists of databases, existing custom built applications, and low level resource such as physical sensors and actuators.	In MPOWER, examples are databases storing medication and administrative information, and (smart) sensors for e.g. physiological monitoring and door control.
Service discovery	Service discovery is referring to finding a suitable service for given task. It could be described as “the automatic identifying of a software-based service which allows processing functions to be offered and then executed after they have been located. Also includes design time notification”.	In MPOWER, a service discovery implementation is based on the UDDI. (Universal Description, Discovery and Integration). The UDDI is a platform-independent, XML-based registry for businesses worldwide to list themselves on the Internet. The UDDI specification defines a way to publish and discover information about web services. The service requestor or web service client locates entries in the broker registry using a service discovery component (which uses various find operations) and then invokes the requested web service.
QoS layer	This layer provides the capabilities required to monitor, manage, and maintain QoS such as security, performance, and availability. This is a background process through sense-and-respond mechanisms and tools that monitor the health of SOA applications, including all important standards implementations of WS-Management and other relevant protocols and standards that implement quality of service for a SOA.	In MPOWER, QoS layer includes security. The objective of the MPOWER security middleware is to ensure sufficient protection (i.e. security level) for any of the MPOWER enabled services when they are used. This implies that security middleware is orthogonal to the other services in a way that it is an implicit part of each service, ensuring a satisfactory security level of any combination of services in the MPOWER platform.

Integration Service Bus	This layer enables the integration of services through the introduction of a reliable set of capabilities, such as intelligent routing, protocol mediation, and other transformation mechanisms, often described as the Enterprise Service Bus (ESB). Web Services Description Language (WSDL) specifies a binding, which implies a location where the service is provided. On the other hand, an ESB provides a location independent mechanism for integration.	MPOWER project uses OpenESB which is SUN's implementation of ESB. The ESB is the piece of software that lies between the business applications and enables communication among them. It works as distributed infrastructure for enterprise integration and consists of service containers and provides services for transforming and routing messages.

3.2.4 *OASIS Layer Model*

Although OASIS has not been explicitly designed having a layer-based approach in mind, its architecture can be described from a layer-based point of view. Figure 8 provides an overall picture of the major components that participate in the OASIS architecture and how these inter-operate with each other.

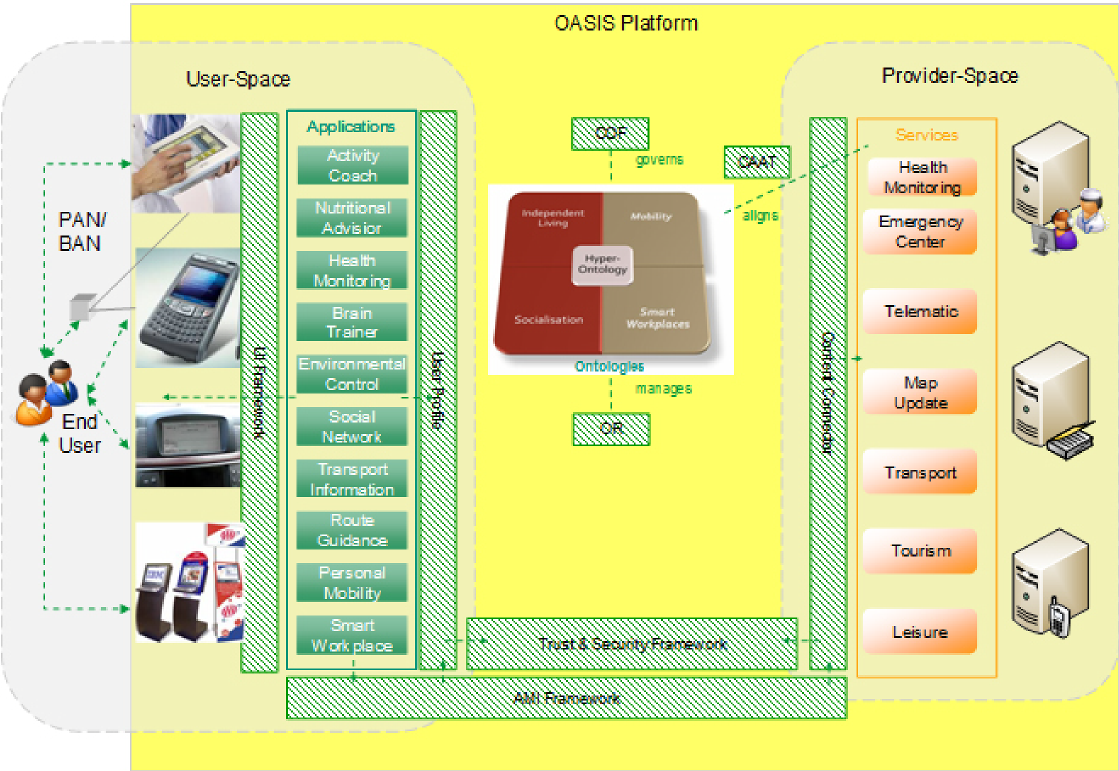

Figure 8: OASIS Conceptual Architecture

All the components involved in the OASIS architecture may be arranged on different architectural layers that are derived from a top level conceptualization of the OASIS architecture. The specific concepts that are involved in the OASIS architecture are explained in what follows and the corresponding layer to which each component belongs is indicated in parenthesis.

- *Services and devices*. Multiple services will be part of OASIS in order to provide all the desirable functionality to the rest of the OASIS components. In OASIS there are two types of services: local services that will reside inside the platform and remote services provided by all the external application providers (Service Layer).
- *AMI Framework*. The Ambient intelligence framework that provides seamless interactivity between OASIS services, applications and the hyper-ontology. It is comprised of the multi-agent platform (Middleware Layer).
- *Common Ontological Framework (COF)*. The COF defines a formal specification of ontology modules, and how they relate. The COF defines a methodology and best practice for ontology construction. It makes possible to define a hyper-ontology and also facilitates the integration of new emerging ontologies (Middleware Layer).
- *Content Anchoring and Alignment Tool (CAAT)*. This tool aligns the functionality of the provided WS to the ontologies stored in the Ontology Repository. The concepts of the same or different application areas, after being aligned with other ontological concepts, will be able to anchor in the hyper ontology framework, thus being ready to be used seamlessly through the CCM (Support Application Layer).

- *Content Connector Module (CCM)*. The role of the CCM is twofold: it supports automatic integration of WS and devices, which takes place when new service providers or hardware developers are willing to register their assets in OASIS, and it receives a request for service by the end-user (client) application via the AmI and invokes the appropriate service that returns the required content to the client (Middleware Layer).
- *Ontology Repository (OR)*. It is the physical infrastructure that supports ontologies storage and management. The COF provides one specific repository for OASIS, the ORATE (Support Application Layer).
- *Trust & Security Framework* (TSF). The TSF is a module responsible for identification, authentication, authorization, including delegation, federation between domains and the integration of the identity services (Trust & Security Layer).
- *User Profile*. It contains all the context information related to a specific user. If one OASIS components needs to retrieve some information related to the user context but out of its own scope, it should make a query to this user profile (Trust & Security Layer).
- *UI Framework*. Allows automatic user interface self-creation for new connected services and self adaptation to the device used, the context of use and the user needs and preferences (End-user Application Layer).

The various layers are presented in more detail in the following table.

Table 5: Layers from OASIS project

Layer/sidecar	Description	OASIS use
Service Layer	This layer includes all external and internal services that are provided to the system. On one hand external services can be seen as the various resources provided by external service providers who register themselves in the system and semantically align their services to the system ontologies. It also supports hardware developers that aim at aligning the functionality of a new device with respect to the ontologies.	In OASIS this layer is used to encompass all assets that are registered in the system including WS and hardware devices that are interconnected and invocable. Service layer is used to integrate all available service in a seamless and semantics-aware way in order to make them visible to the other architectural layers and implement SOA functionalities.
Middleware Layer	Provides a reference implementation of the reference architecture and OASIS platform. This layer includes the technical infrastructure required for the semantic search and integration of services and devices with respect to the ontologies, as well as the invocation of services, consumption of the available resources that are provided in a service-oriented way and the deliverable of requested resources to the end-user applications.	It includes all major OASIS middleware elements and components, such as the AMI, COF and CCM.
Support Application Layer	This is a support layer that provides all appropriate tools that are required by various operations related to elements of the middleware layer.	This layer consists of all necessary concepts and frameworks for ontology storage / management, service alignment and integration. Specifically it includes the CAAT, as well as various ontology support tools such as the ontology backup, update and maintenance tool, mappings visualisation tools, etc.

Trust and Security Layer	The trust and security layer is responsible for identification, authentication, authorization, including delegation, federation between domains (local/remote and OASIS/third party providers), user profiles and the integration of the identity services.	The Trust and security layer is the core subsystem for performing user registration, authentication and profile management, including privacy management through the security module, using central and/or federated identity management functionality.
End-user Application Layer	This layer encompasses the end-user applications that run on the user's device. It covers issues and concepts also related to the user interface adaptation mechanisms.	It is used to support the functionality and design aspects of the end-user applications.

3.2.5 PERSONA Layer Model

The aim of PERSONA architecture was not necessarily focused in following a traditional 3-layered model nor the IBM SOA layered Reference Architecture. However, looking backward it is possible to find many similarities with any of those layered models or even others.

PERSONA is not a unique monolithic system that runs in a unique runtime environment. PERSONA is based on a model where several nodes that can be located in a unique runtime environment or in different runtime environments (in different machines, in different instances of Java Virtual Machines…) can communicate one to the other thanks to the transparent communication mechanisms offered by the middleware. It is in fact a dynamic ensemble of networked nodes, where the middleware helps that this ensemble takes form by supporting seamless connectivity and facilitating communication based on goal-based interoperability[3].

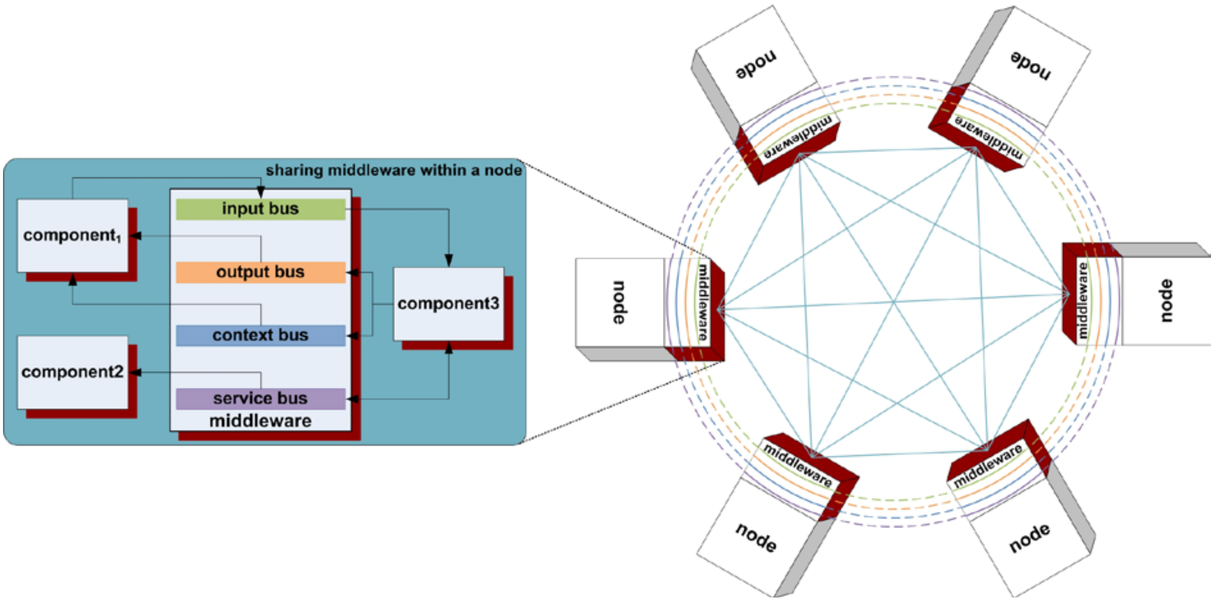

Figure 9: Using peer-to-peer connections to form a dynamic ensemble of networked nodes in PERSONA

From the logical perspective of a layered model, PERSONA can be divided into layers shown in Figure 10, namely the middleware layer, the platform core layer, the platform plug-ins layer, and the AAL services layer. The middleware is responsible for resolving the challenges of seamless connectivity (e.g. node discovery) and goal-based interoperability (e.g. providing a message brokering mechanism) while hiding the distribution and possible heterogeneity of underlying operating systems and networking protocols. After having guaranteed integration and interoperability by the middleware, the question that had to be answered was about shared functionality needed by AAL applications and services. PERSONA divides such functionality into two parts: the mandatory part and the plug-in part. Components that provide the platform core / general-purpose services are mandatory, and hence an integral part of every installation of a PERSONA-based system. The plug-in part, which is represented by the *Platform Pluggable (special-purpose) Services* layer, consists of all components that provide installation-specific shared functionality.

3 For further explanations, please refer to the definition of the PERSONA concepts in PERSONA Terminology Model of APPENDIX A especially the terms Seamless connectivity, Goal-based interoperability, Message brokering, Self-organizing system, Middleware and its distributed realization, Sodapop Model, and Virtual communication bus.

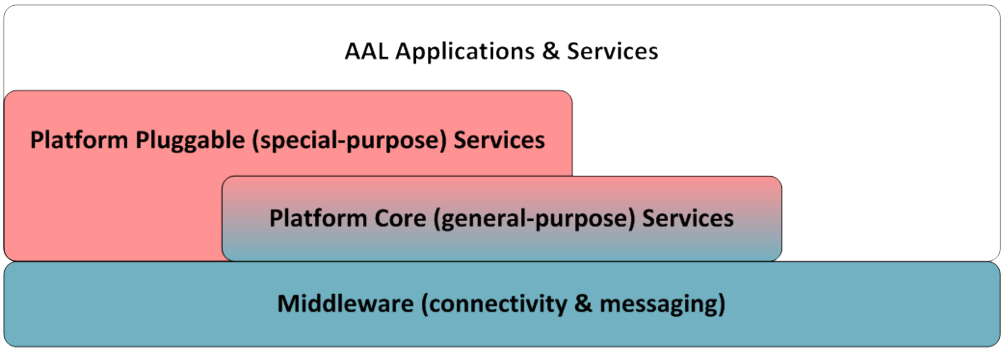

Figure 10: The PERSONA layer model

All components residing on the top three layers, no matter on which of the three logical layers above the middleware, are supposed to perform their communications through the middleware; in other words, "outsourced" functionality from a layer below or from the same layer should be requested by sending a request to the middleware. The middleware is then responsible to (1) find out which concrete component on which layer is providing the requested functionality, (2) send the request in an appropriate form to that component, (3) get its response, and (4) return it in an appropriate form to the original requester. That is, direct component-to-component communication is forbidden in PERSONA. In this way, the only syntactical interface on which the components are dependent is the one of the middleware (cf. Figure 11).

The relevant or visible middleware components that provide these interfaces are the PERSONA buses: Service Bus, Context Bus, Input Bus and Output Bus. As indicated by Figure 9, local instances of these buses in the different nodes cooperate to provide the components using them with a virtually global view on them, this way hiding the distribution of the system.

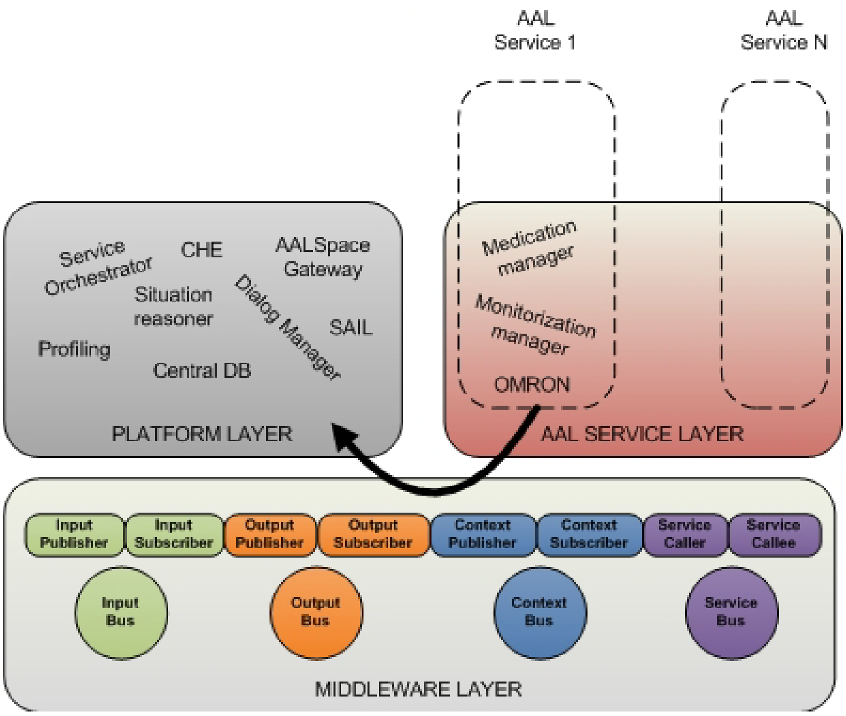

Figure 11: Example of communication ways in PERSONA when components from the application layer need to access platform core services

The above discussion reveals that a layer model as shown in Figure 10 is not reflecting the reality of interfacing between the different layers because there is no API of the pluggable or core components! Getting back to Figure 9, we can see that at a physical level, all the components from the three top layers can be distributed on the nodes of an ensemble freely and from the viewpoint of the middleware all of them are its “users” with “equal rights”. Despite the fact that the set of components residing on the application and the plug-ins layers is undetermined, such a hierarchical **view** like in Figure 10, however, can indeed be derived for each concrete configuration / installation based on the real dependencies between the components, although it is likely that the result would differ from installation to installation.

It is worth to mention that the internal architecture of the middleware itself was originally described using a layer model due to strict hierarchical dependencies that were defined during the conceptual design of the middleware. However, the details of the implementation of the middleware are not so relevant at the level of Reference Model or Reference Architecture though it has a lot of interesting features from a requirements perspective or implementation point of view.

Table 6: Definition of the layers in PERSONA

Layer/sidecar	Description	Persona use
Middleware	The “middleware” is the **intermediate piece of software** allowing the ensemble to take form by defining high-level protocols and providing uniform interfaces for ● **integrating** components into the system ● enabling the **communication** between them It hides: ● **distribution** of components ● **heterogeneity** of the various hardware components and their operating systems and networking protocols	The integration of components from all the other layers into the system is done through the visible interfaces of the middleware that are the PERSONA buses: Context Bus, Service Bus, Input Bus and Output Bus… In PERSONA, middleware also provides a level of security mechanisms that ensure that components are allowed to call other components’ services. PERSONA middleware also ensures that communications between nodes are encrypted.
Platform core services	The logical grouping of components that provide shared funtionality that is mandatory and application-independent. It is also possible that they publish and consume services among themselves. They attach to the middleware using the visible interfaces of the middleware provided by the PERSONA Buses.	Components that belong to this logical layer: Service Orchestrator, Context History Entrepôt, Profiling component, Dialog Manager, IOHandlers, Situation reasoner, Activity Monitor, Sensor Abstraction and integration layer…
Platform plug-ins	Components that enhance the platform toward a certain configuration by providing shared functionality beyond the functionality provided by the mandatory components, without realizing any use case for the human users of the system, belong to this logical layer.	By defining a placeholder for platform plug-ins, PERSONA has made its platform extensible with high potential for customizing it based on real needs and preferences.

AAL applications and Services	Those components in charge of executing the business logic of the AAL Services and realizing use cases for the human users of the system belong to this layer. They might use platform services to accomplish this task as well as provide services to their siblings on the same layer.	This layer is used to symbolize the relation between AAL applications and the PERSONA platform.

3.2.6 SOPRANO Layer Model

In the following, the different layers of abstraction, their planned manifestation in the system architecture and their role for the SOPRANO [5] system are explained in more detail. For a better understanding, a scheme is introduced that depicts the layers and illustrates the information flow within the proposed system architecture. The system behaviour will be determined by rules and assumptions processed at each level of the scheme, the SOPRANO pyramid. Therefore, the pyramid is a representation of the logical data processing in SOPRANO, and corresponds to the actual system architecture with its different hardware and software components.

In the sequel, the layers of the pyramid and their respective concepts are explained in more detail, starting from the bottom. The pyramid is meant as a reference model for a partition of the different components of the SOPRANO system.

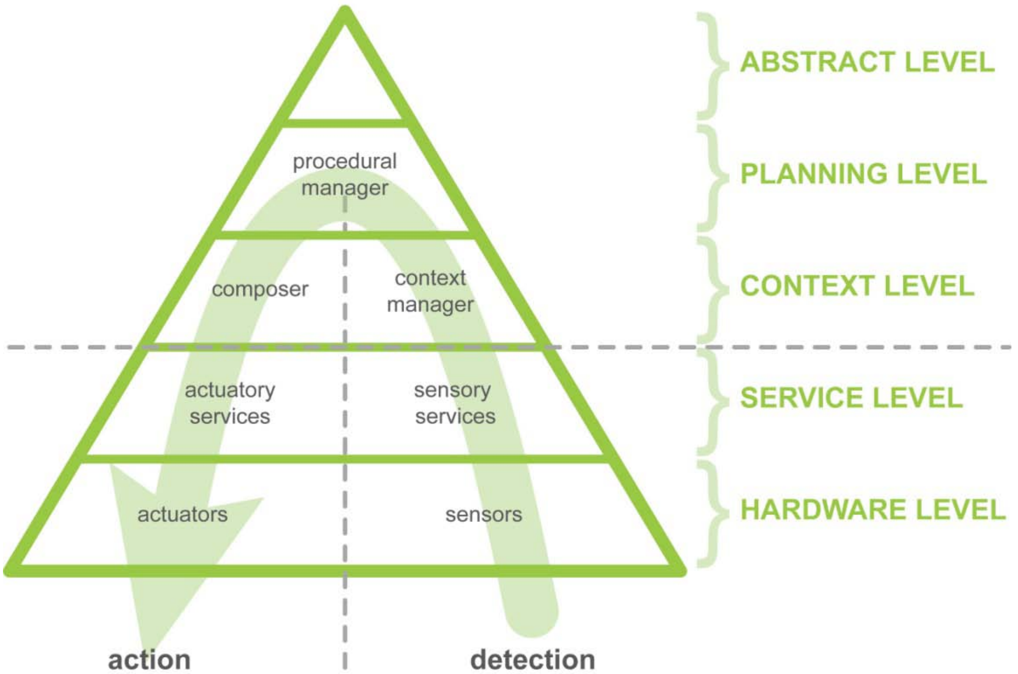

Figure 12: The SOPRANO pyramid. The semantic hierarchy from top level abstract system behaviour down to attached devices is depicted

Table 7: Layers from Soprano project

Layer/sidecar	Description	Soprano use
Hardware Level	The lowest level of the SOPRANO pyramid is the level of the actual hardware devices. There are two types of hardware devices: sensors and actuators.	The logical entities introduced on this level are the providers of raw data on the sensoric side, and the consumers of low-level commands on the actuator's side. The Hardware Level exposes its interface to the above-lying service level. The corresponding interface in the system architecture will be the hardware drivers that expose the sensor's functions to the service level.
Service Level	The Service Level provides the first abstraction and aggregation of raw data, yielding semantic data. The creation of meaningful semantic data can be achieved by temporal aggregation ("is still in bed") or by means of semantic interpretation of raw data, e.g. by crossing thresholds ("has fever").	The Service Level makes use of the underlying Hardware Level for the triggering actuators or receiving sensor data. It offers its services to the above-lying Context Level. In the proposed system architecture, this will be done via OSGi Service Bundles. Hereby, the interface definition evolves out of the concepts introduced by the SOPRANO ontology. The Service Level (and likewise the Hardware Level) is individual to a single SOPRANO installation in a household, depending on the actual sensors and actuators installed. The levels on top of the Service Levels are common for the entire SOPRANO system, hence for the entirety of SOPRANO installations. This entails the necessity of a service registration with the upper level components of the proposed system architecture.
Context Level	The Context Level aggregates Services to high-level sensory events and offers aggregated functionality to the upper planning level.	The Context Level offers high-level semantic events to the planning component above and is ready to take goal-oriented instructions. While at the lower Service Level, a dedicated device is triggered, actions on the Context Level refer to the desired impact only, e.g.: "The AP needs to be notified" instead of "display message on interactive TV". The Context Level delivers the context conditions that are processed at the Planning Level. At the Context Level, the desired system behaviour is describes regardless of the actual devices attached to a local SOPRANO installation. Hence, the context level modelling is valid for all local installations; modifications of the procedures concern all local installations. The corresponding system components that implement the system behaviour on the context level are the context manager (for the sensory branch) and the composer (for the actuatory branch). Their common set of concepts is the context ontology.

Planning Level	At the Planning Level, the stack of sensory aggregation is evaluated, and appropriate action is triggered.	This level contains and applies the rules for taking decisions. The events are evaluated in the current context and the result could be the triggering of a workflow of high-level procedures. The responsible SOPRANO component is the procedural manager. The procedures described in the workflow are passed for realisation to the composer.
Abstract Level	The top level of the SOPRANO pyramid as a representation of the semantic hierarchy is meant as the theoretical superstructure only.	The SOPRANO allows abstraction at several levels of both knowledge and functionality. The reasoning engine allows uplifting of the collected knowledge (from low-level sensor bound statements to high-level statements events handled by the system can lead to creation of new logical statements). The Composer is capable to select the best matching low-level services that realise the high-level procedures.

4 universAAL Reference Model

This chapter defines the first version of the reference model. As mentioned in Chapter 4, in this version of the deliverable we present only the terminology and layer model. The component and distribution models will be defined in the future version. The terminology comprises the definition of a set of basic concepts and their interrelationships. The layer model specifies a set of very high-level and abstract architectural layers along with a generic representation of entities that can reside on each layer. The first version of the universAAL reference model is based on the alignment of the terminology and layer models from the previous projects as described in previous chapter. More details about the consolidation of terminology and layer model are given in the sub-sections of this chapter.

4.1 universAAL Terminology Model

The terminology model for universAAL was done by consolidating the terminology used in the various projects. For the purpose of consolidation, the terminology from the previous projects were categorised into terminology groups. The terminology groups that emerged from the process were the following: Architecture, Behaviour, Bus, Components, Content, Devices, Domotic Infrastructure, Frameworks, Infrastructure, Messages, Ontology, Participant, Quality of Service, Reasoner, Service, and System. This grouping of terminology was determined from the overlap in the concepts from the different projects and also using terminology from the existing standards. SoaML [6], AALIANCE [7] and Continua [8] were used as existing standards and external projects to guide the grouping of terminology and identify missing terminology. This grouping is preliminary and will be refined or extended with other groups in the future version of the terminology model.

Once the grouping of the terminology was done, the important terms from each group were identified that were considered to be the most relevant concepts for the universAAL reference model. For certain terms that required a broader definition than the ones provided by existing projects or standards, a universAAL specific definition has been provided. Figure 12 below shows the reference model terms and the relationships between them. The types of relations used here are *specialize* (solid line with filled arrow), *realize* (dotted line with filled arrow) and *use* (dotted line with unfilled arrow with <<use>> stereotype). Here, the *use* relation should be regarded as a generic relationship.

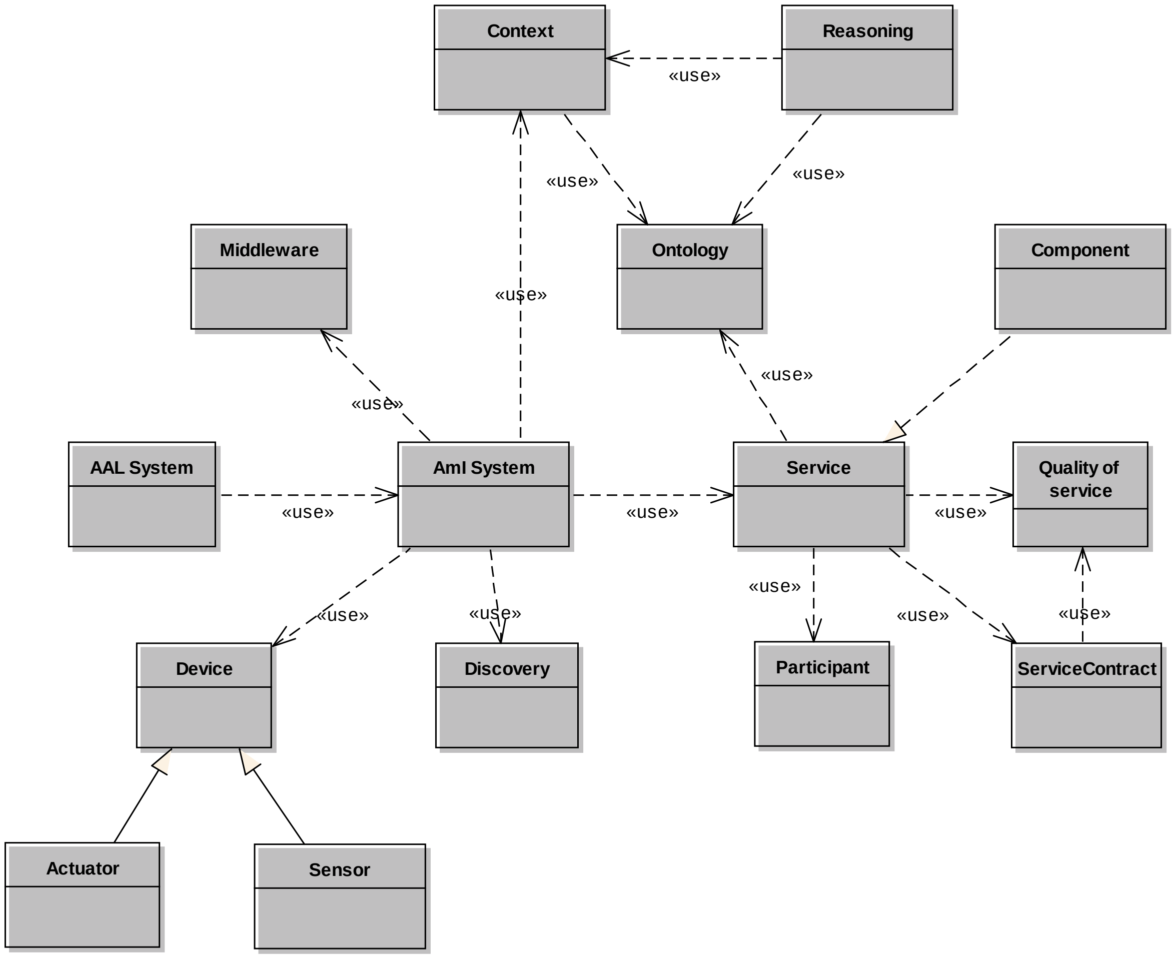

Figure 13: Overview of universAAL terminology and relationships

For certain terminology multiple definitions are listed from the previous projects. All the term definitions listed in the table below are taken directly from their source. In this first version of the Reference Architecture Terminology they are not adapted to universAAL. However, this will be done in the next version of the deliverable. In a future version of the deliverable we aim to consolidate these definitions into a single definition. Some terms are also still to be defined – these are generally the more generic terms that were added to represent a group of terms from the input projects. When the source is listed as OASIS_Std this refers to the terminology from the OASIS Standard SOA reference model, and not to the OASIS project.

Table 8: universAAL Terminology Model

Term	Source(s)	Definition
AAL System	«PERSONA»	A system consisting of networked physical and virtual resources that are set up to collectively provide intelligent assistance towards wellbeing in preferred living environments.
Actuator	«PERSONA»	A device that is able to cause certain changes in the physical realm upon receipt of related requests through an interface provided in the virtual realm.
AMI System	«PERSONA»	A highly distributed system that uses different facilities for bridging between the virtual and physical realms (e.g., I/O channels, sensors, and actuators), in addition to utilizing pure virtual resources and services, in order to provide human users with ambient assistance in performing their tasks and reaching their goals. The provision of assistance in AmI systems happens normally in a personalized and

		multimodal way. Usually AmI systems also provide automatic assistance in terms of automatic reactions to environmental changes and / or detected intentions, referred to as context-awareness.
Component	«GENESYS, Continua»	To be consolidated in next version of reference model. Continua: A Component is an entity in the Continua architecture. In general, for any Interface, there is a Service Component, with a well-defined set of functions depending on its type, on one side of the interface and one (or more) Client Components on the other side. Each Component is contained within a Device Genesys: A component is regarded as a self-contained composite hardware/software subsystem that can be used as a building block in the design of a larger system. The component can have a complex internal structure that is neither visible, nor of concern, to the user of the component. The behavior of a component, which is visible at the component's *LIF*, has to be specified in the value and time domain
Context	«universAAL»	To be specified in next version of this reference model
Device	«Continua»	A Device is a physical entity (box) and contains one or more Components (functionality)
Discovery	«universAAL»	To be specified in next version of this reference model
Middleware	«PERSONA»	To be consolidated in next version of reference model. PERSONA: A piece of software that glues the distributed components of a self-organizing system to each other, thus allowing the system to emerge. It resolves the challenges of seamless connectivity (e.g. node discovery) and goal-based interoperability (e.g. providing a brokering mechanism) while hiding the distribution and possible heterogeneity of underlying operating systems and networking protocols - no architectural layer but a piece of software! Other sources: "Middleware Architecture with Patterns and Frameworks" [9] states that "intermediate software layers have come to be known under the generic name of middleware". More specifically, it states that this intermediate software "resides on top of the operating systems and communication protocols to perform the following functions. 1. Hiding distribution, i.e. the fact that an application is usually made up of many interconnected parts running in distributed locations. 2. Hiding the heterogeneity of the various hardware components, operating systems and communication protocols that are used by the different parts of an application. 3. Providing uniform, standard, high-level interfaces to the application developers and integrators, so that applications can

		easily interoperate and be reused, ported, and composed. 4. Supplying a set of common services to perform various general purpose functions, in order to avoid duplicating efforts and to facilitate collaboration between applications."
Ontology	«SOPRANO»	To be refined in next version of reference model. This is not a functional component of SOPRANO, but it is a very fundamental part of the system. All other modules rely on it as a common means of understanding. It defines information that can be exchanges, knowledge that can be stored and its datatypes.
Participant	«SOAML»	A participant is the type of a provider and/or consumer of services. In the business domain a participant may be a person, organization or system. In the systems domain a participant may be a system, application or component.
Quality of service	«Continua»	Quality of service is the collection of properties that define characteristics of an interface connection. This set of properties includes aspects of the communication link such as reliability, latency, bandwidth, and etc.
Reasoning	«universAAL»	To be specified in next version of this reference model
Sensor	«PERSONA, Continua»	To be consolidated in next version of reference model. Continua: A Sensor Service Component allows access to digital representations of external conditions and events. This includes measurements of temperature, motion, or electrical conditions. Persona: A device that can measure something in the physical realm and represent the related info in terms of data in the virtual realm.
Service	«universAAL, SOAML, PERSONA, GENESYS»	To be consolidated in next version of reference model. GENESYS: The service delivered by a system is its intended *behaviour* as it is perceived by its users. The behaviour is the sequence of observable outputs of a system. PERSONA: The provision of something of value, in the context of some domain of application, by one party (service provider) to another (service consumer); more precisely: the actual value provided to achieve a consumer's goal. In the virtual realm, provision of value has traditionally been called functionality; hence, service can be seen as a general abstract way of talking about accessible functionality that can be utilized using pull mechanisms. Services accessible in the virtual realm can be utilized by activating a related service utility (e.g., using the terminology of Web Services, an "operation" of a "Web Service"), which in turn will start a provision process realized by the corresponding service providing component (e.g. the Web Service component). In such a process, human participants as well as other service components may be involved. The process may also incorporate access to several physical or virtual resources,

		such a printer or a database. However, the process is encapsulated by the SOAML: A service is value delivered to another through a well-defined interface and available to a community (which may be the general public). A service results in work provided to one by another.
ServiceContract	«SOAML»	A ServiceContract is the formalization of a binding exchange of information, goods, or obligations between parties defining a service.

4.2 universAAL Layer Model

This section describes the consolidation of the layer models, and the initial definition of a layer model for universAAL.

When describing the layer model of universAAL, we have some main usages in mind:

- Comparing and relating other architectures to the universAAL architecture
- Guidance during positioning attempts for components which are being considered for integration with the universAAL platform
- Guidance for the design of the more detailed reference architecture of universAAL and for universAAL compliant components

The approach taken here to describing the layered model is inspired by the "layered style" introduced by Clements [10] in addition to the layer view of ARCADE described in chapter 3.3.

4.2.1 Consolidation of the layer models from the input projects

As a first step towards consolidation of the layer models of the input projects, we arranged the layer models of the input projects along with the description of the universAAL platform from the universAAL Description of Work (DoW) in a common diagram (see Figure 1613). This work revealed that the model used in universAAL DoW and the models introduced by GENESYS and PERSONA are very similar. Further, most of the other layer models have a good mapping to these models. The alignment lines in the figures were added to help visualize this mapping.

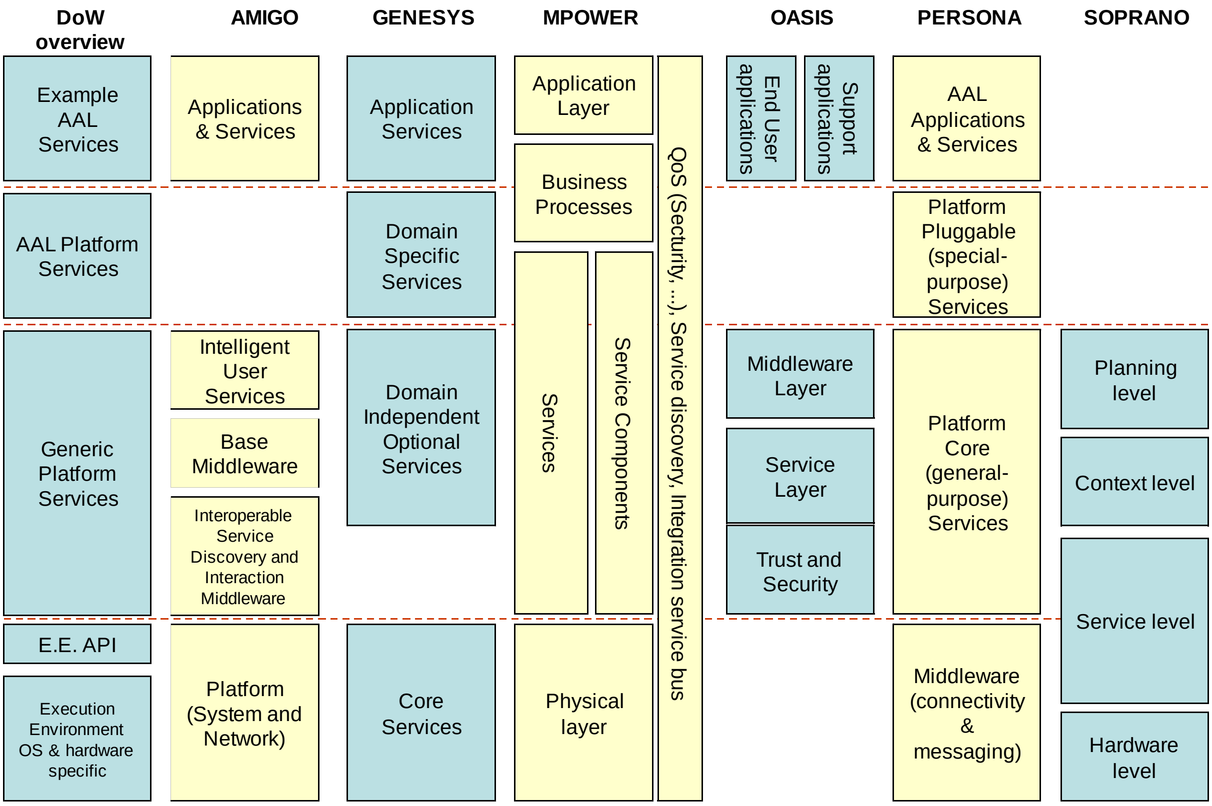

Figure 14: Consolidation of layer models

4.2.2 The universAAL layer model

Based on the mapping between the layer models of the input projects, we have designed a first version of the universAAL layer model. As most of the input models mapped well to the model in the DoW, GENESYS, and PERSONA, our initial layer model is close to these. A first variant of a simple (strict) layer model is presented in Figure 15. Note that the layer view is a logical view of the system, and does not include aspects such as distribution.

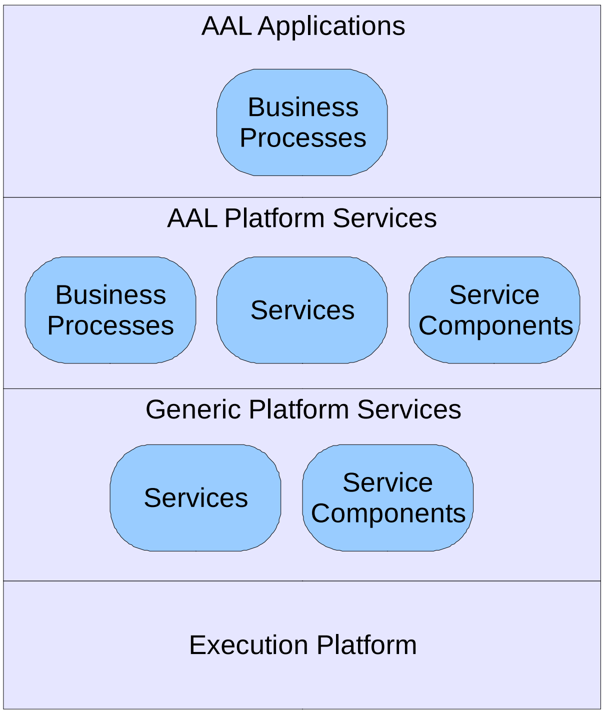

Figure 15: universAAL layer model (simple, strict variant)

In the model, the universAAL platform is divided into three layers, namely the Execution Platform, the Generic Platform Services, and the AAL Platform Services. As a result, the application layer resides on top of the AAL Platform Services.

The execution platform is assumed to extend the native system layer of the different physical nodes participating in an AAL system and hence hide the distribution of these nodes as well as the possible heterogeneity of their native system layers. In addition to that, this layer is supposed to act as a container for integration of all components from the above layers and facilitate the communication among them.

We distinguish between domain specific services (AAL platform services) and domain independent services (generic platform services) because the generic services are common to all AmI-based systems and facilitate the construction of all kinds of smart environments. To this end, we are emphasizing that the AAL platform services tend to be higher level services which depend on generic services for their realization.

Compared to the DoW model, the universAAL layer model does not represent the API of Execution Environment as a separate layer. Rather, this API defines the interface to the Execution Environment layer, and the layer interfaces are not shown explicitly in the figure.

In the figure, we have also mapped the layers from the IBM SOA model (used also in MPOWER) into the proposed model as the types of entities that can reside on each of the layers, as they can present a different dimension of decomposition. The notion of Service is taken from the terminology introduced in the previous section and is meant as an abstract unit for referring to functionality. Service Components are software components that bring with themselves a possible realization for such services. And, last but not least, the business processes emphasize the need for composability at a meta level, based on a workflow involving services.

In a layered model the main elements are layers, and the main relations expressed are “allowed to use” relations between the layers. The simple layer model of Figure 15 can be interpreted as a strict model where usage between layers are restricted to only allow usage of the layer directly below. A more relaxed variant is presented in Figure 1616. While the strict model can be seen as the ideal to fully profit from a layered system, the relaxed model can map more easily to real-life situations. For example, with the above understanding of the execution platform, if this layer is going to be responsible for dealing with the distribution of functionality and heterogeneity of the networked nodes, then it would be more convenient to assign the brokerage task for realizing the communication between all components to the execution platform and hence allow all layers to directly use this functionality.

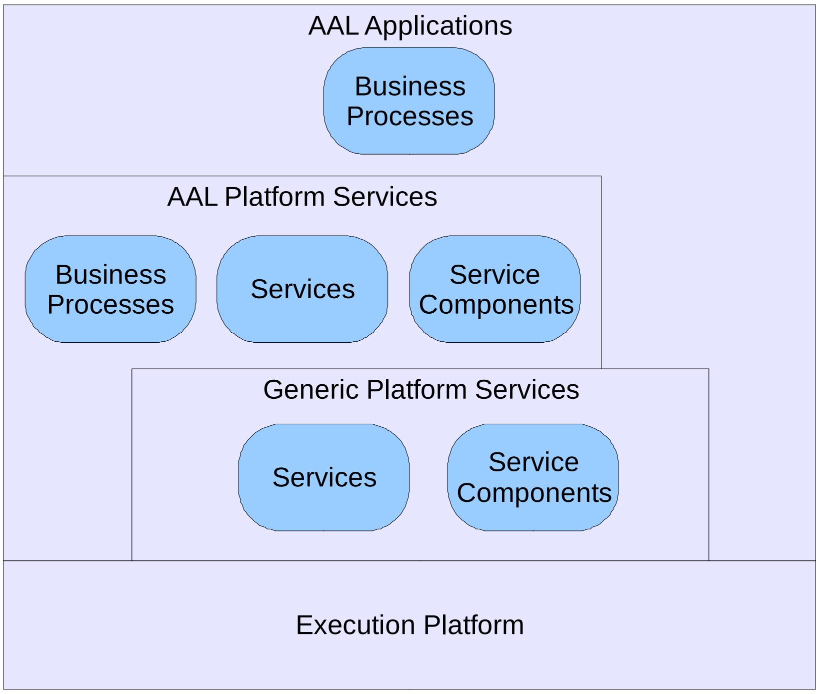

Figure 16: universAAL layer model (relaxed variant)

Since a well-defined layer model should also describe the intra- and inter-layer usage rules, we close the description of the universAAL layer model with the following rules:

1. The inter-layer rules
   - components belonging to one layer can freely use the services provided by the layer below through its interface
   - components belonging to one layer can, when necessary, use the services of other layers below (in addition to the one directly below it) through their interfaces, but this usage should be avoided when possible because it weakens the layering. The rules for such usage is subject to further detailing in the reference architecture
   - components of one layer are not allowed to make direct calls on or to have any other dependencies on layers above unless such a call involves the realization of an interface that is defined by the lower layer itself; in such a case, layers above are allowed to register the related realizations for receiving notifications / callbacks
2. The intra-layer rules
   - components of a layer are allowed to interact with other components defined in the same layer using well-defined interfaces, subject to rules that will be further defined in the reference architecture
   - more specifically, for business processes, services, and service components in a layer, the following rules apply: service components may utilize services realized by other components on the same layer and business processes can use services. Other uses are not allowed.

Table 9: Definition of the layers in universAAL

Layer/sidecar	Description	Interface
AAL Applications	Ambient Assisted Living applications using the platform for their realization.	
AAL Platform Services	Reusable services for the AAL domain	The interface of this layer to layers above consists of selected services and business processes. Service components are not visible outside this layer, but are the only parts that use the layers below directly.
Generic Platform Services	This layer provides ambient intelligence functionality and other domain independent services to the layers above.	The interface of this layer to layers above is a selected set of services. Service components are not visible outside the layer. Service components depend on the Execution platform, while the services do not use this directly.
Execution platform	This layer extends the native system layer of the different physical nodes participating in an AAL system, hiding distribution and heterogeneity issues for the layers above.	The interface of this layer to layers above is a platform-independent API.

The layered reference model does not indicate which layers contain component frameworks or other means for extensibility. This will be covered in the reference architecture. Also, the layer model does not define the set of components that will be mapped to the layer. The reference architecture will cover this, and will also consider whether any particular subset of components will be mandatory for the platform, and/or whether we will have different “profile” versions of the platform (i.e. for mobile / desktop / server).

The current layer model does not cover any aspects of communication enforced by the platform (e.g. should all communication go through the execution platform?). Such aspects will be defined in the reference architecture.

5 Future work

Based on the methodology described in Section 3, we can summarize the role of the achievements presented in Section 5 in the process of architectural design of the universAAL platform as shown in Figure 171916.

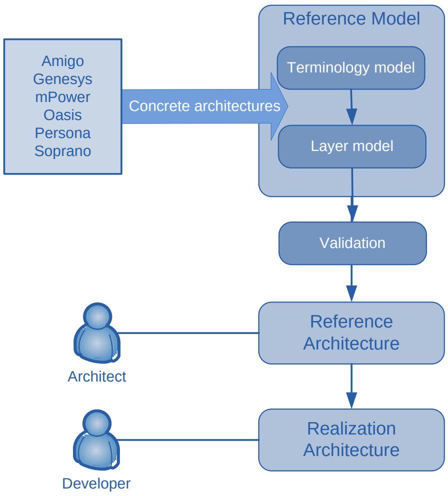

Figure 17: The process of architectural design in universAAL

The first version of the universAAL reference model is based on the alignment of the terminology and layer models from the previous projects. The terminology model requires further refinement to consolidate multiple definitions used for certain terms. Additionally universAAL specific terminology needs to be defined in the future versions of the reference model.

In the next version of the document, we also intend to include concept maps, similar to the one provided in Figure 1817 below. Note that the current content of the concept map in the figure should currently only be regarded just as an example.

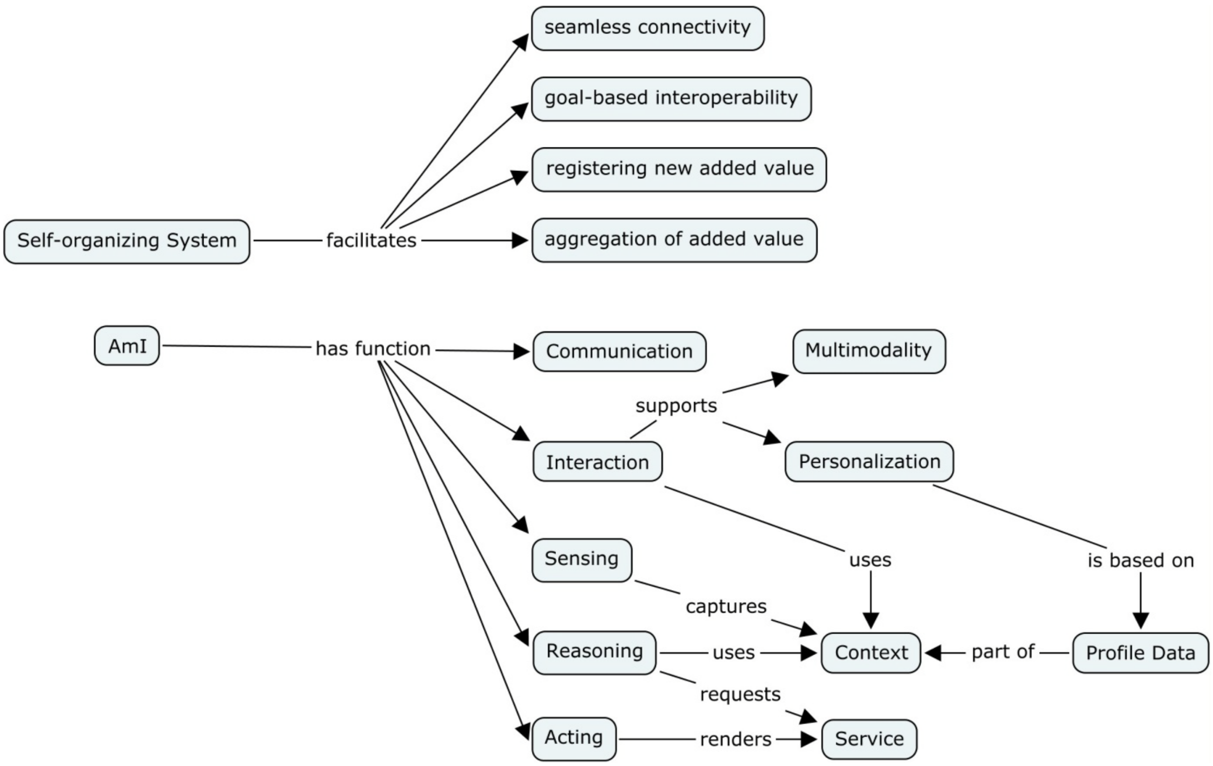

Figure 18: Example concept map

For the layer model, further detailing is needed for the services, service components and business processes and how these are mapped into the reference model. Also, the inter- and intra-layer rules will be defined in more detail.

The main work in the next version of this deliverable, however, will be to define the first version of our reference architecture based on our reference model through a validation step. As part of this initial step, we plan to map the main components of each input project to the reference model, and to group these components based on the functionality they provide. Further, the initial reference architecture will be derived covering the initial selected groups of functionality.

References

[1] *ARCADE, An Open Architectural Description Framework* [online at: http://www.arcade-framework.org/], accessed on: 09.06.2010

[2] *Reference Model for Service Oriented Architecture 1.0*, [online at: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf], accessed on: 09.06.2010

[3] Boston, MA, *Reference Architecture Foundation for Service Oriented Architecture*, Committee Draft 02, [online at: http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-cd-02.pdf], accessed on: 09.06.2010

[4] Stav E., Walderhaug S., Johansen U., *ARCADE - An Open Architectural Description Framework*, SINTEF

[5] Content taken from SOPRANO Deliverable: *Analysis and abstraction of the identified needs and requirements*

[6] Beta 2 version from OMG at the time of writing, *Service oriented architecture Modeling Language (SoaML)*, [online at: http://www.omg.org/spec/SoaML/], accessed on: 09.06.2010

[7] *AALIANCE Ambient Assisted Living Roadmap* [online at: http://www.aaliance.eu/], accessed on: 09.06.2010

[8] *Continua Health Alliance*, [online at: http://www.continuaalliance.org/], accessed on: 09.06.2010

[9] Krakowiak, S., *Middleware Architecture with Patterns and Frameworks* [online at: http://sardes.inrialpes.fr/~krakowia/MW-Book/], accessed on: 09.06.2010

[10] Clements P., et. al., *Documenting Software Architectures: Views and Beyond*

[11] Arsanjani A., *Service-oriented modeling and architecture: How to identify, specify, and realize services for your SOA*, vol. 2007: IBM developerWorks, 2004

APPENDIX A

1. AMIGO Terminology Model

The AMIGO terminology is extracted from the AMIGO project deliverables, which can be found at:

http://www.hitech-projects.com/euprojects/amigo/deliverables.htm

Concept	Definition	Relevance to universAAL	Reference
Programming and Deployment Framework	The .NET / OSGi programming framework is an essential part of the Amigo Software which is used as a basis by nearly all application/component developers. The goal of the framework is to support developers to write their application or component software in a short timeframe by relieving them of time consuming and complex tasks, such as protocol-specific details for remote communication and discovery.	universAAL requires a programming and deployment framework for developers	Deliverable D3.1b Detailed Design of the Amigo Middleware Core Section 4.3 P77-87
Context Management Service	The Amigo Context Management Service (CMS) is an open infrastructure for managing context information. The role of the CMS is to acquire information coming from various sources, such as physical sensors, user activities, and applications in process or internet applications and to subsequently combine or abstract these pieces of information into "context information" to be provided to context aware services.	universAAL will require some form of context management	Deliverable D4.7 Intelligent User Services 2 - Context Management Service Software Developer’s Guide
Awareness and Notification	The Awareness and Notification Service (ANS) provides the basic functionality required to develop applications allowing people and other applications to stay aware of any significant change in context with minimal effort. ANS is able to keep track of changes in various types of context, for example activities and presence of people. ANS makes application layer services aware of context changes by notifying them. Applications register monitoring rules that specify what changes in context should be notified to them. From the user perspective, the Awareness and Notification Service provides notifications with appropriate rendering of intensity, based on the user's preferences and current context.	universAAL requires a mechanism to notify or make aware other people or components of the system to major changes to other parts of the system	Deliverable D4.7 Intelligent User Services 4 - Awareness and Notification Service Software Developer’s Guide

Privacy and Security	This component provides access to the Amigo authentication and authorization service. It encapsulates the communication and cryptographic primitives that are used for device/user registration, authentication, and authorization with the centralized Amigo security service.	universAAL with 24x surveillance requires a good privacy and security component	Deliverable D4.7 7 - Intelligent User Services Privacy and Personal Security
User Modeling and Profiling	User modeling and profiling provides the methodology to enhance the effectiveness and usability of services and interfaces in order to (a) tailor information presentation to user and context, (b) reason about user's future behavior, (c) help the user to find relevant information, (d) adapt interface features to the user and the context in which it is used, (e) indicate interface features and information presentation features for their adaptation to a multi-user environment. These goals are achieved by constructing, maintaining and exploiting user models and profiles, which are explicit representations of individual user preferences.	universAAL caters to various categories of users with different needs which requires user personalization using profiles	Deliverable D4.7 Intelligent User Services 3 - User Modeling and Profiling Service Software Developer's Guide
Interoperable Service Discovery & Interaction Middleware	The role of the interoperable service discovery & interaction (SD&I) middleware is to identify the discovery and interaction middleware protocols that execute on the network and to translate the incoming/outgoing messages of one protocol into messages of another, target protocol. The system parses the incoming/outgoing message and, after having interpreted the semantics of the message, it generates a list of semantic events and uses this list to reconstruct a message for the target protocol, matching the semantics of the original message. The interoperable SD&I middleware acts in a transparent way with regard to discovery and interaction middleware protocols and with regard to services running on top of them. The supported service discovery protocols are UPnP, SLP and WS-Discovery, while the supported service interaction protocols are SOAP and RMI.	universAAL needs to work in a very heterogenous environment and requires good interoperability and discovery mechanisms	Deliverable D3.1b Detailed Design of the Amigo Middleware Core Section 4.1 and 4.2 P44-77

Domotic Infrastructure	The Amigo Domotic Infrastructure aims at presenting heterogeneous physical hardware devices as unified software services using standard service technologies. Nowadays, there is a great diversity of physical device technologies and protocols. Further, there are a number of service technologies that should be supported within the Amigo system. Therefore, the purpose of the Amigo Domotic Infrastructure is to enable the integration of different device technologies presenting them by means of software services, but isolating the final users (service clients) from the specific base technologies.	universAAL needs to connect to various sensors with different interfaces and therefore requires abstraction at the physical layer.	Deliverable D3.1b Detailed Design of the Amigo Middleware Core Section 4.4 P87-97
Content Distribution	The Content Distribution service provides available content in the Amigo home to Amigo services and applications according to the DLNA standard. This is done by gathering available content descriptions (not the actual content to avoid time-consuming and unnecessary copying of content) from UPnP Digital Media Servers (like Windows Media Connect, etc.). Moreover, it has the ability to provide content in a format which suits the renderer's capabilities in the best possible way.	universAAL requires various kinds of content (related to educational material for an illness etc) to be distributed in a secure and copyright protected way.	Deliverable D3.1c Detailed Design of the Amigo Middleware Core Security & Privacy, Content Distribution, Data Storage Section 3 P15-29
Content Storage	This component offers a generic storage service to other components and applications inside an Amigo system. There is no restriction on the kind of content that can be stored, and each component or application can open and control access to a sub-store inside the Data Store. It supports also notifications on changes in a sub-store. Data is automatically backed up and restored when necessary.	universAAL needs to consider how to store the huge amount of data that will be generated from sensors and consumed by the end-user	Deliverable D3.1c Detailed Design of the Amigo Middleware Core Security & Privacy, Content Distribution, Data Storage Section 4 P29-33

Accounting and Billing	The Accounting and Billing component of the Amigo middleware offers a basic service for managing IPDR documents. Authorized applications will be able to introduce, search for and filter and share IPDR documents via the Accounting and Billing Service. This component offers validation of IPDR documents as well as service specific IPDR schema caching. Furthermore it enables advanced searches with criteria based on IPDR creation time, service type and service specific element matching.	universAAL needs good business models to be adopted by industry and accounting and billing is the first step in that.	Deliverable D3.3 Amigo Middleware Core Enhanced: Prototype Implementation & Documentation Section 10 P110-112
User Interface Services	Encompasses several interface related services, such as a multimodal dialogue manager and services supporting interaction via specific modalities (e.g., speech, GUI, gesture).	universAAL needs an good UI customized to the target end-users.	Deliverable D2.3 Specification of the Amigo Abstract System Architecture Also: D4.7 series
Semantic Service Description	This component offers a comprehensive approach to semantic service description, discovery, composition, adaptation and execution in the Amigo home, collectively called SD-SDCAE, using the Amigo-S language, thereby enabling integration of heterogeneous services into complex services based on their abstract specification.		Deliverable D3.3 Amigo Middleware Core Enhanced: Prototype Implementation & Documentation Section 4 P55-88
VantagePoint	The VantagePoint component is a Java application that can visualize, query and edit OWL ontologies that model a user-specified physical environment.		Deliverable D3.5 Amigo overall middleware: Final prototype implementation & documentation Section 6 P70-96
Management Console	The management console provides a single point of control and diagnostics for the whole connected home. It is able to connect (remotely) to the different deployment platforms on the devices for control (software update) and diagnostic purposes.	universAAL needs to be managed in an easy way and allow easy diagnosis of problems.	Deliverable D3.5 Amigo overall middleware: Final prototype implementation & documentation Section 10 P126-130

2. *GENESYS Terminology Model*

The GENESYS terminology is defined in the GENESYS book, which is available from:

http://www.genesys-platform.eu/genesys_book

or internally for the universAAL project from:

https://project.sintef.no/eRoomReq/Files/ikt/ICT-20097-ICTAgeing/0_30b12/Genesys%20book%20-%20requirements%2C%20architecture%2C%20implementation.pdf

Concept	Definition	Relevance to universAAL	Reference
Application Service	The application service is the intended sequence of messages produced by a job via output ports at the *LIF* and the *controlled object* interface in response to the progression of time, inputs and state.	Conceptual basis for component-based design	GENESYS book (Glossary, page 175)
Architectural Style	The architectural style consists of rules and guidelines for the partitioning of a system into subsystems and for the design of the interactions among subsystems. Subsystems must comply with the architectural style to avoid a property mismatch at the interfaces between subsystems.	Architectural principles facilitate significant properties (e.g., robustness, composability, etc.)	GENESYS book (Glossary, page 175)
Architecture Model	A set of descriptions that define an architecture or a configuration or a combination of an architecture and a compatible configuration (that obeys the rules defined by the architecture).	An architecture model is also basis of universAAL.	GENESYS book (Glossary, page 175)
Architecture	The architecture is a framework for the construction of a system for a chosen application domain. It provides generic platform services and imposes an architectural style for constraining an implementation in such a way that the ensuing system is understandable, maintainable, and extensible and can be built cost-effectively.	It is the goal of universAAL to develop an architecture.	GENESYS book (Glossary, page 175)
Behavior	The sequence of messages (i.e., intended and unintended) produced by a subsystem at its *LIF*.	When different components are integrated to an universAAL implementation, their behavior (as perceptible from outside the component) is important, but not the internal structure.	GENESYS book (Glossary, page 175)
Behavioral Model	A model that describes the dynamic internal evolution (operation) of the object of reference (system, subsystem, component) and its response to external stimuli.	(see behavior)	GENESYS book (Glossary, page 175)

Channel	A channel serves for the exchange of messages between ports. It is associated with a communication topology, a data-direction (e.g., unidirectional or bidirectional), temporal properties and dependability properties.	Necessary for interaction of universAAL components.	GENESYS book (Glossary, page 175)
Cluster	A cluster is a physically distributed computer system that consists of a set of nodes interconnected by a physical network.	Facilitates description of physical structure of a universAAL system.	GENESYS book (Glossary, page 176)
Component	A component is regarded as a self-contained composite hardware/software subsystem that can be used as a building block in the design of a larger system. The component can have a complex internal structure that is neither visible, nor of concern, to the user of the component. The behavior of a component, which is visible at the component's *LIF*, has to be specified in the value and time domain.	universAAL systems will be composed of self-contained hardware/software subsystems which internal structure is not of concern to the user.	GENESYS book (Glossary, page 176)
Composability	Composability is a concept that relates to the ease of building systems out of subsystems. A system, i.e., a composition of subsystems, is considered composable with respect to a certain property (e.g., timeliness, certification) if this property, given that it has been established at the subsystem level, is not invalidated by the integration.	A universAAL implementation will be composed of multiple subsystems which are individually developed and tested.	GENESYS book (Glossary, page 176)
Constrained Access	The access of the platform services through the application is temporally constrained in order to ensure consistency in read/write operations without explicit synchronization. This depends on clock synchronization between application and platform for temporal access coordination.	In order to constrain access to shared resources (e.g., shared memory) the instant and duration of the access to this resource should be temporally constrained.	GENESYS book (Glossary, page 177)
Controlled Object	The controlled object is the home environment, sensors and actuators that are to be controlled by the computer system.	The universAAL platform controls several devices (controlled objects) in the environment of the assisted person.	GENESYS book (Glossary, page 177)
Core Platform Services (Core Services)	Core platform services are mandatory in every instantiation of the reference architecture template (e.g., networking service, robustness service, etc.). The core platform services provide the foundation for higher-level, *optional services*.	The universAAL platform needs to provide a set of services that are a stable foundation for each instantiation.	GENESYS book (Glossary, page 177)

Cross-Domain Architectural Style	The cross-domain architectural style consists of views, concepts, and design principles that have been consolidated from the different application domains. This includes the description of fundamental architectural principles, the identification of commonalities between application domains and the identification of different *integration levels* required in each application domain.	universAAL may profit from cross-domain architectures as the development cost of devices and applications can be amortized to multiple domains (e.g., including automotive). "Economics-of-scale"	GENESYS book (Glossary, page 177)
Cross-Domain Development Methodology	The cross-domain methodology framework consists of a set of methods, techniques and tools for diverse development processes that are applicable across multiple application domains.	(see Cross-Domain Architectural Style)	GENESYS book (Glossary, page 177)
Declared State	The declared state is the state of a subsystem, which is considered as relevant by the system designer for future *behavior* of the subsystem.	Necessary for robustness w.r.t. transient faults.	GENESYS book (Glossary, page 177)
Determinism	A model behaves deterministically if and only if, given a full set of initial conditions (the initial state) at time t0, and a sequence of future timed inputs, the outputs at any future instant t are entailed.	Foundation of robustness by active redundancy, and of certification	GENESYS book (Glossary, page 177)
Distributed Application Subsystem (DAS)	A Distributed Application Subsystem is a nearly independent distributed subsystem of a large distributed real-time system that provides a well-specified *application service*. For example the multimedia system in an AAL system can be such a DAS. Since DASs may be of different criticality, the probability of error propagation across DAS boundaries must be sufficiently low to meet the dependability requirements. A DAS is further decomposed into smaller units called *jobs*.	Logical structuring of universAAL systems.	GENESYS book (Glossary, page 178)
Error	An error is that part of the system state which is liable to lead to a subsequent *failure*. A failure occurs when the error reaches the service interface.	Conceptualization of dependability issues.	GENESYS book (Glossary, page 178)
Error Containment	Although a *fault containment region* can demarcate the immediate impact of a fault, fault effects manifested as erroneous data can propagate across the boundaries of fault containment regions. Therefore the system must also provide error containment for avoiding error propagation by the flow of erroneous messages.	One universAAL component cannot affect other universAAL components.	GENESYS book (Glossary, page 178)

Error Containment Region (ECR)	The set of *fault containment regions* that performs error containment is denoted as an error containment region. An ECR consists of at least to independent fault containment regions. The error detection mechanism must be part of a different FCR than the message sender, otherwise the error detection service can be affected by the same fault that caused the message failure.	Errors in one subsystem of the universAAL implementation may not propagate to affect other subsystems. Thus ECRs have to be defined to detect errors within subsystems.	GENESYS book (Glossary, page 178)
Event Message	An event message is a message that contains event observations. An event observation contains the difference between the "old state" and the "new state". The time of the event observation denotes the point in time of the state change. In order to maintain state synchronization, the handling of event messages requires exactly-once semantics.	To notify distinct components of the system of the occurrence of an event, this type of messages is necessary (e.g., when the doorbell is ringing).	GENESYS book (Glossary, page 178)
Fail-operational System	A fail-operational system is able to tolerate one or several *faults*. Fail-operational systems send correct messages despite the *failure* of their subsystems.	The universAAL platform will contain services which are essential to the life of the assisted person (e.g., fire alarm, fall detection,…). These services must also operate correctly despite the failure of subsystems.	GENESYS book (Glossary, page 179)
Fail-safe System	In a fail-safe system all *failures*, to an acceptable extend, only minor ones. In case of a failure, the system responds in a way that harm to persons and things is reduced as much as possible (e.g., if the fire detection system fails, an automatic alarm is triggered as maybe the failure was caused by the fire itself).	Subsystems of universAAL may require being fail-save.	GENESYS book (Glossary, page 179)
Failure	A failure occurs when the delivered service deviates from fulfilling its specification.	(see Error)	GENESYS book (Glossary, page 179)
Fault	A fault is the adjudged or hypothesized cause of an *error*. Faults can be internal (e.g., a design fault) or external (e.g., a malicious attack) of the system. A fault can remain in a system without having any effect (e.g., a design fault). It needs to be activated to become an *error*.	(see Error)	GENESYS book (Glossary, page 179)

Fault-Containment Region (FCR)	A Fault-Containment Region is a collection of components that operates correctly regardless of any arbitrary logical or electrical fault outside the region.	Appropriate design of fault containment regions in the universAAL architecture ensures that one faulty subsystem cannot influence computations of other FCRs.	GENESYS book (Glossary, page 179)
Fault Hypothesis	The fault hypothesis is the specification of the *faults* that must be tolerated without any impact on the essential system services. The fault hypothesis states the assumptions about units of failure (i.e., *Fault-Containment Region*), failure modes, failure frequencies, failure detection, and state recovery.	In the universAAL architecture it needs to be defined which kind of faults the system is able to tolerate without the failure of the whole system. For example, if a new application, which contains design faults, is installed on the universAAL system, it should be able to tolerate this fault without requiring explicit repair action.	GENESYS book (Glossary, page 179)
Host	The host is the unit used to execute *jobs*.	Physical structuring of a universAAL system.	GENESYS book (Glossary, page 180)
Integrated Resource Management	Integrated resource management is the simultaneous management of multiple resources (e.g., bandwidth, power, energy, memory) in order to globally optimize different resources.	In universAAL most services depend on the actual context, thus also resource requirements vary in different situations.	GENESYS book (Glossary, page 180)
Integration Level	The integration level denotes the layer in a system-of-systems at which it is composed out of its components. Different integration levels can be distinguished, e.g., chip level, device level, system level.	Physical structuring of universAAL systems.	GENESYS book (Glossary, page 180)
Job	A job is a constituting element of a *DAS* and forms the basic unit of work. It interacts with other jobs through the exchange of messages in order to work towards a common goal and provide the *application services*.	Logical structuring of universAAL systems.	GENESYS book (Glossary, page 180)

Linking Interface (LIF)	A *job* provides its real-time services, and accesses the real-time services of other jobs by the exchange of messages across its Linking Interface. These messages have to be fully specified in a LIF specification which consists of an operational specification and a LIF service model specification.	The universAAL platform needs to provide precisely specified interfaces for services provided as well as to provide services to other components. This improves composability.	GENESYS book (Glossary, page 181)
Linking Interface Specification	The linking interface specification is the mediating middle between a service supplier and the service user. It comprises a syntactic specification, a temporal specification, and a LIF service model specification. The syntactic specification forms out of the sequence of bits in a message larger chunks (e.g., a number, a string, a method call, etc.) and assigns a name to each chunk. The temporal specification of the messages defines their send and receive instants, e.g., at what instants the messages are sent and received, how the messages are ordered, and the rate of message arrival.	An exact specification of the interfaces in the value and temporal domain of services provided helps to reduce the potential of mismatch between service suppliers and service users. Furthermore, erroneous components can be detected by means of assertions at these interfaces. This improves composability.	GENESYS book (Glossary, page 181)
Message	A message is any data structure that is formed for the purpose of inter-job communication. In order that errors in a message may be detected, an output guard and an input guard can be associated with a message. Such a guard is a predicate on values of the message, and relevant state variables that define an application-specific acceptance criterion. Using such assertions, it is possible to classify messages as: valid, checked, permitted, timely, value-correct correct, or insidious.	Concept for interaction between universAAL components.	GENESYS book (Glossary, page 181)
Optional Platform Services (Optional Services)	The optional platform services which are built upon the *core platform* service can be generic in the sense that they can be used in multiple application domains or specific for a focused domain. These are not required in every instantiation of the architecture, but extend the provided services of the *core platform services*.	universAAL will provide services which are not required in every instantiation of the architecture, but available for optional use.	GENESYS book (Glossary, page 182)

Platform	A platform is the hardware/software foundation for the execution of applications. The platform comprises generic services for the development of applications, which are denoted as platform services (see *Core Platform Services* and *Optional Platform Services*).	Foundation for universAAL services.	GENESYS book (Glossary, page 183)
Platform Services	Platform services facilitate the development of distributed applications and separate the application functionality from the underlying platform technology to reduce design complexity and to enable design reuse. Platform services can be distinguished into *Core Platform Services* and *Optional Platform Services*.	universAAL should provide a set of platform services which can be selected for a specific universAAL instantiation.	GENESYS book (Glossary, page 183)
Reference Architecture Template	The reference architecture template is a template for building concrete architectures. The reference architecture template provides specifications for a comprehensive set of *platform services*, including domain-independent services that can be used across application domains. In a specific application, a subset of these platform services can be selected and implemented. The selection and implementation of the platform services is part of the instantiation of the template used to arrive at a concrete architecture.	To facilitate building of concrete architectures, universAAL should provide an appropriate template.	GENESYS book (Glossary, page 183)
Reliability	Reliability is the ability of a system or component to perform its required functions under stated conditions for a specific period of time.	For several universAAL services it will be essential that the user can rely on the correct operation over a long period of time. For example, high reliability of the fire detection system means that the probability of a malfunction within a specified period of time is very low.	GENESYS book (Glossary, page 183)

Replica Determinism	Replica determinism is a desired property between replicated systems. A set of replicated subsystems is replica determinate if all subsystems in this set produce exactly the same output messages that are at most an interval of *d* time units apart, as seen by an omniscient outside observer.	In case of replicated computation in the universAAL architecture for fault tolerance reasons (e.g., for live critical applications), the replicated subsystems need to be replica determinate in order to come to the same result.	GENESYS book (Glossary, page 183)
Robustness	Robustness is the capability of a system to deliver an acceptable level of service despite the occurrence of transient and permanent hardware faults, design faults, imprecise specifications, and accidental operational faults. A system must be resilient with respect to unanticipated behavior from the environment of the system or of subsystems. In case such unanticipated behavior occurs, the system should still exhibit some sensible behavior, and not be completely unpredictable.	The universAAL architecture can comprise several subsystems that need to operate even in the presence of faults (e.g., the fire alarm subsystem). Thus, the universAAL must follow the concept of robustness.	GENESYS book (Glossary, page 184)
Service	The service delivered by a system is its intended *behavior* as it is perceived by its users. The behavior is the sequence of observable outputs of a system.	Conceptual foundation of component-based design.	GENESYS book (Glossary, page 184)
Sparse Time Base	If the time base of the global time in a distributed system is dense (i.e., the events are allowed to occur at any instant of the timeline), then it is in general not possible to generate a consistent temporal order of events on the basis of the time-stamps. Due to the impossibility of synchronizing clocks perfectly and the denseness property of real time, there is always the possibility that a single event is timestamped by two clocks with a difference of one tick. By introducing the concept of a sparse time base this problem can be solved. In the sparse time model the continuum of time is partitioned into an infinite sequence of alternating durations of activity and silence. Thereby, the occurrence of significant events is restricted to the activity intervals of a globally synchronized action lattice. In this time model, the costly execution of agreement protocols can be avoided, since every action is delayed until the next lattice point of the action lattice.	Enables the consistent ordering of events without agreement protocols.	GENESYS book (Glossary, page 184)

State	The state enables the determination of a future output solely on the basis of the future input and the state the system is in. In other word, the state enables a "decoupling" of the past from the present and future. The state embodies all past history of the given system. Apparently, for this role to be meaningful, the notation of the past and future must be relevant for the system considered.	Important concept for robustness.	GENESYS book (Glossary, page 184)
State Message	A state message is a periodic message that contains state observations. An observation is a state observation, if the value of the observation contains the state of a real-time entity. The time of the state observation denotes the point in time when the real-time entity was sampled. The handling of state messages occurs through an update in place and non-consuming read.	Many applications need periodic update of the state. This information is transported by the state messages (e.g., the actual heart rate of the assisted person is checked several times per minute).	GENESYS book (Glossary, page 184)
State Recovery	State recovery is the action of (re-) establishing a valid state in a subsystem after a failure of that subsystem.	After the failure of a subsystem in an universAAL implementation a valid state needs to be established, e.g., after rebooting the faulty component.	GENESYS book (Glossary, page 185)
Unconstrained Access	Unconstrained access does not restrict the points in time of access operations performed by the application. In order to support consistency, asynchronous handshake protocols are employed that do not require clock synchronization between application and platform.	(antonym to Constrained Access)	GENESYS book (Glossary, page 185)

3. MPOWER Terminology Model

The MPOWER terminology is extracted from the two sources. The first source is the MPOWER project deliverable "MPOWER D1.1 Overall architecture". This deliverable is available from:

https://project.sintef.no/eRoomReq/Files/ikt/ICT-20097-ICTAgeing/0_30b1e/Mpower%20Overall%20architecture.pdf

The second source is the presentation "MPOWER - Basic architectural concepts" created for a training session in the universAAL project. It is available from:

https://project.sintef.no/eRoomReq/Files/ikt/ICT-20097-ICTAgeing/0_35458/universAAL_MPOWER_ENT.ppt

Concept	Definition	Relevance to universAAL	Reference
Service Platform	A set of software services and components offering secure interfaces to access local and central communication and information services.	universAAL must define platform (some ideas already defined in DoW).	MPOWER D1.1 Overall architecture (section 3.1, page 14)
Common Services	A set of reusable information and communication services enabling the development and deployment of smart home care solutions. - Is part of service platform.	universAAL must define what a service is.	MPOWER D1.1 Overall architecture (section 3.1, page 14)
Applications	Two Proof-of-Concept applications have been developed using the MPOWER Middleware services as the core artefacts; one demonstrates information access and sharing aspects, while other demonstrates MPOWER smart home environment. Applications provide functionality to end users. Applications use (common) services.	universAAL must define what we name what the end users/seniors see and use.	MPOWER - Basic architectural concepts (slide 31)
Architecture	The MPOWER Architecture package consists of the: (1) Reference Architecture, (2) MPOWER HL7 Information models, and (3) UML models that specify reusable services and components. Using UML Patterns defined in the MDSD Healthcare Framework, and Profiles defined in the MPOWER UML Extensions, domain-specific and technology independent UML models are described as Platform Independent Models (PIMs). The PIMs can be transformed into PSMs adding platform specific mappings using the transformation scripts described in the framework.	universAAL must define the role of the reference architecture. An architecture could be an instance of a reference architecture?	MPOWER D1.1 Overall architecture (section 6, page 25.)

Reference architecture	The MPOWER project uses IBM Service Oriented Architecture (SOA) as reference architecture. This example shows how MPOWER used a reference architecture.	The universAAL reference architecture must justify itself, that is, how it should be used.	MPOWER D1.1 Overall architecture (section 6.1, page 25., and Appendix A, page 56.) MPOWER - Basic architectural concepts (Slide 10)
SOA Architect-ural style	IBM defines SOA architectural style as: "A set of patterns and guidelines for creating *loosely coupled, business-aligned* services that, because of the separation of concerns between description, implementation, and binding, provide unprecedented flexibility in responsiveness to new business threats and opportunities."	universAAL should define what it means with service orientation.	http://www.ibm.com/developerworks/library/ar-archtemp/ OR MPOWER D1.1 Overall architecture (section 6.2, page 26.)
Middleware	The MPOWER Middleware holds reusable and compiled (runnable) services and components that can be easily utilized by application developers. The MPOWER middleware consists of five categories of services: Sensor services, Contextual services, Information (Medical and Social) services, Security services, and Interoperability services.	In MPOWER middleware is the same as service platform. universAAL should choose one (my suggestion is that platform is better).	MPOWER D1.1 Overall architecture (section 7, page 32.) OR MPOWER - Basic architectural concepts (Slide 12)
Sensor system	Systems that provides measured information through a defined interface. A sensor can be both physiological and non-physiological. Automation services rely on sensor information from e.g. door sensors, water-temperature sensors, light sensors and movement sensors. A sensor system can be composed of several sensor-s/systems).	universAAL must define what a sensor is (and probably actuator).	MPOWER - Basic architectural concepts (Slide 13)

Frame Sensor Adapter (FSA)	Sensor/Actuator network abstract architecture model which solves problem of communication among services and different sensor protocols It defines a framework providing unified access to sensors and actuators that use different communication channels and different data formats.	universAAL must to define a layer/service towards the physical world of sensors and actuators. Suggest other name than FSA, but something similar in nature.	MPOWER - Basic architectural concepts (Slide 14)
Context information model	The context understood as a current state of the MPOWER environment describes the environment state at a certain moment. Two spaces have been defined to manage context within MPOWER: first one – "user personal profile" and second - "the Inward environment info".	universAAL could define context.	MPOWER - Basic architectural concepts (Slide 18-21)
Service components	In order to be interoperable, the MPOWER platform implements a set of service components that contain the functionality needed to interoperate with external system (1. Health information systems –e.g. information exchange with Google Health, 2. SMS messaging, external calendar services, voice and video over IP)		MPOWER - Basic architectural concepts (Slide 11- figure)
Service discovery	A reference implementation of a service discovery implementation based on the standards UDDI. The UDDI allows for lookup of services that is important in cases when loose coupling of services is needed.		MPOWER - Basic architectural concepts (Slide 11- figure)
Service provider	Is the implementation of a service.		MPOWER D1.1 Overall architecture (page 25)
Service description	Describes the services.		MPOWER D1.1 Overall architecture (page 25)
Service consumer	can either use the uniform resource identifier (URI) for the service description directly or can find the service description in a service registry and bind and invoke the service		MPOWER D1.1 Overall architecture (page 25)

Service broker	provides and maintains the service registry. The next diagram shows the main interaction between the above mentioned parties.		MPOWER D1.1 Overall architecture (page 25)
Integration service bus	A reference implementation of an Enterprise Service Bus. ESB works as distributed infrastructure for enterprise integration and consists of service containers and provides services for transforming and routing messages.		MPOWER - Basic architectural concepts (Slide 19)

4. OASIS Terminology Model

The OASIS terminology is extracted from the following OASIS project deliverables.

OASIS D1.6.1, which is available at:

http://www.oasis-project.eu/docs/OFFICIAL_DELIVERABLES/SP1/D1.6.1/OASISDeliverableD1_6_1_v1_1.pdf

OASIS D1.3.1 (after peer-review), which is available internally at the universAAL eRoom:

https://project.sintef.no/eRoomReq/Files/ikt/ICT-20097-ICTAgeing/0_33866/OASIS%20Deliverable%20D1_3_1_version_1.0_after_peer-review.pdf

Concept	Definition	Relevance to universAAL	Reference
Adaptive Multiagent Integration framework (AMI)	Agent-based framework that provides seamless interactivity between OASIS services, OASIS applications and the hyper-ontology	Can provide interactivity between universAAL applications and ontologies. Provision of personalization services.	OASIS D1.6.1 p.55
Common Ontological Framework (COF)	The COF defines a formal specification of ontology modules, and how they relate. The COF defines a methodology and best practice for ontology construction. It makes possible to define a Hyper-Ontology and will also facilitate and optimize the integration of new emerging ontologies. This Hyper-Ontology will reside in the OOR (OASIS Ontology Repository) also provided by the COF.	This framework can be used for ontology support in conjunction with Ontology Repository (OR).	OASIS D1.6.1 p.33
Content Anchoring and Alignment Tool (CAAT)	This tool aligns the functionality of SOAP-compatible web services as well as hardware devices with the ontologies stored in the OASIS Ontology Repository. The concepts of the same or different application areas, after being aligned with other ontological concepts, will be able to anchor in the hyper ontology framework, thus being ready to be used seamlessly through the CCM.	Can be used for the alignment of external web services without significant effort.	OASIS D1.3.1 (after peer-review) p.31

Content Connector Module (CCM)	The role of the Content Connector Module (CCM) is twofold: it supports automatic integration of Web services, which takes place when new service providers are willing to register their Web services in OASIS, and it receives a request for service by the end-user (client) application via the AMI and invokes the appropriate service that returns the required content to the client.	Middleware for invocation of web services and delivery of appropriate content to any requesting party.	OASIS D1.6.1 p.35
OASIS Application	A wide range of connected applications, all integrated within the OASIS System, and interoperating in integrated Use Cases defined, covering the needs of the elderly and their caregivers in terms of Independent Living, Socialisation, Autonomous Mobility and Smart Workplaces.	The OASIS paradigm for application integration could be useful in universAAL.	OASIS D1.6.1 p.106
OASIS Platform	Defines the logical platform, on which resources from different providers can be shared in an integrated way.	A variation of this platform can be used for services integration.	OASIS D1.6.1 p.43
OASIS Reference Architecture	Defines the content that can be shared on the OASIS Platform by means of ontologies. It is composed of the COF, the ontologies and the support tools (Content Connector Module and other ontology management modules), both available as open source, that allow the automatic or semi-automatic connection of existing and emerging ontologies and services to the OASIS Architectural Framework	Part of the reference architecture can be re-used.	OASIS D1.6.1 p.25
OASIS Service	Multiple services will be part of OASIS in order to provide all the desirable functionality to the rest of the OASIS components. In OASIS there are two types of services: local services that will reside inside the platform and remote (web) services provided by all the external application providers.	OASIS services may be used also as universAAL services.	
Ontology Repository (OR)	It is the technological layer that supports the Ontologies storage and management. The COF (Common Ontological Framework) provides one specific repository for OASIS, the OOR.	Can be used as ontology storage and management for universAAL ontologies.	http://ontologies.informatik.uni-bremen.de/

Trust & Security Framework (TSF)	The TSF will be implemented inside an OASIS module which will be responsible for identification, authentication, authorization, including delegation, federation between domains (local/remote and OASIS/third party providers) and the integration of the identity services.	Framework for universAAL user identification, authentication and authorization.	OASIS D1.6.1 p.70
UI Framework	Allows automatic UI self-creation for new connected services and self adaptation to the device used, the context of use and the user needs and preferences. All the UI`s that comprises this framework have followed a specific user-centric methodology for its development called OPAF, so we can ensure that all the user requirements will be accomplished.	Adaptive UI principles may be adopted for the end user applications.	
User Profile	It contains all the context information related to a specific user. If any OASIS component needs to retrieve some information related to the user context but out of its own scope, it should make a query to this user profile.	Can be used as a basis for universAAL user profile.	OASIS D1.6.1 p.60

5. *PERSONA Terminology Model*

Concept	Definition	Relevance to universAAL	Reference
I/O channel	Channels between the virtual realm and the physical realm for capturing user input that was intended to be used by the virtual system (then talking about an input channel) or making info prepared by the system perceivable for human users (then talking about an output channel). Displays, keyboards, loudspeakers, and microphones are examples of devices that realize I/O channels.	I/O channels belong to the reality of AAL systems and must be utilized in an intelligent way in the course of providing assistance.	PERSONA internal reports IR3.1.1 & IR3.1.2 & deliverable D3.1.1
Sensor	A device that can measure something in the physical realm and represent the related info in terms of data in the virtual realm.	ditto	ditto
Actuator	A device that is able to cause certain changes in the physical realm upon receipt of related requests through an interface provided in the virtual realm.	Ditto	ditto
AAL System	A system consisting of networked physical and virtual resources that are set up to collectively provide intelligent assistance towards wellbeing in preferred living environments.	Obvious	ditto
AAL Space	A physical space equipped with a concrete setup of an AAL system, thus able to effectively contribute to the provision of intelligent assistance. Two important characteristics of AAL spaces are awareness and reactivity; that is, they are always gathering info about certain changes in the space and may be able to automatically react upon recognition of certain situations. Example AAL spaces are the near-body AAL space, the home, the neighbourhood (with its shopping centers, local authorities, etc.), and the village or town.	Defines some locality, a physical boundary,that helps to decide about security policies of concrete setups of AAL systems. Provides an abstraction for discussing certain situations, such as relationships between users and spaces (e.g., ownership or interaction while being inside or outside the space) and interoperability between AAL	PERSONA DoW

		spaces.	
Service	The provision of something of value, in the context of some domain of application, by one party (service provider) to another (service consumer); more precisely: the actual value provided to achieve a consumer's goal. In the virtual realm, provision of value has traditionally been called functionality; hence, service can be seen as a general abstract way of talking about accessible functionality that can be utilized using pull mechanisms. Services accessible in the virtual realm can be utilized by activating a related service utility (e.g., using the terminology of Web Services, an "operation" of a "Web Service"), which in turn will start a provision process realized by the corresponding service providing component (e.g. the Web Service component). In such a process, human participants as well as other service components may be involved. The process may also incorporate access to several physical or virtual resources, such a printer or a database. However, the process is encapsulated by the service utility and hence hidden to the service consumers.	Nowadays, the most frequently used concept that provides an abstract unit for sharable functionality.	see the eRoom doc "WP1 Terminology" for several references
Open distributed system	A system with several communicating physical nodes, each possibly hosting several logical units, that allows to dynamically add and remove components – physical as well as logical – and nevertheless guarantees a certain level of operation without having to recompile, reinstall or restart any part of the existing and running system. The components of an open distributed system may be redundant, competing with some existing components, or bring new functionality with them. In order to join to such a system, a component must follow the provided specifications and be somehow authorized. The WWW is the largest known open distributed system constantly in dynamic evolution. As components are removed and added, the WWW continues to work without essential affection even if some end-points and users may experience difficulties with certain changes.	The concept of open distributed systems is conform with the reality of AAL systems as it respects the two most important characteristics of them, namely distribution of resources and evolvability. Especially relevant for universAAL, because the platform should be open for dynamic configurability and hosting new components.	see AmI-08 paper on PERSONA CASF (Framework Supporting Context-Awareness)
AmI system	A highly distributed system that uses	one of the major	See, for instance,

	different facilities for bridging between the virtual and physical realms (e.g., I/O channels, sensors, and actuators), in addition to utilizing pure virtual resources and services, in order to provide human users with ambient assistance in performing their tasks and reaching their goals. The provision of assistance in AmI systems happens normally in a personalized and multimodal way. Usually AmI systems also provide automatic assistance in terms of automatic reactions to environmental changes and / or detected intentions, referred to as context-awareness. However, they are bound to a certain locality and hence physically limited. Therefore, it is crucial for such systems to possess enormous potential for sharing functionality in an evolvable process. As a result, AmI systems realized as open distributed systems fit their circumstances best as they allow for dynamic configurability and respect the reality of distribution as well as the physical limitations adequately.	input disciplines for realizing AAL systems and spaces.	Emile Aarts and José Encarnação, True Visions: The Emergence of Ambient Intelligence, Springer, 2006
Seamless connectivity	Seamless connectivity is given if nodes participating in an open distributed system find each other and connect in a dynamic way and can use the connections to communicate in an appropriate way. Seamless connectivity is especially appropriate for AmI systems because they provide a boundary for such automatism per se due to being bound to a certain locality.	Contributes towards openness and dynamic configurability of AAL spaces	See the strategic research agenda of ARTEMIS
Goal-based interoperability	Utilization of functionality (also known as service utilization) takes place in a goal-based way if requests simply express the meaning of what is requested to be achieved and hence both addressing concrete target components and using syntactical artifacts, such as interfaces, can be avoided. That is, goal-based interoperability can be achieved when in communication there is no dependency on technical details of the "how"s, but participants just focus on the "what"s. To realize goal-based interoperability, usually a brokering mechanism takes over the responsibility of finding an appropriate responder, mapping the request to a related "call", routing the "call" to that responder,	Goal-based interoperability is particularly useful in open distributed systems because the independence from physical addresses and syntactical artifacts leads to more openness of such systems.	See "Goal-based Service Utilization Using SPARQL and OWL-S" in the latest OWL-S site

	getting the response, and forwarding it back to the requester, thus hiding all the technical details of the "how"s. SQL-like query languages generally provide a suitable formalism for goal-based communication as they just rely on a *data model* and per se pose no technological or methodical requirements on query handling. In such queries, the goal can be formulated starting with simple "speech acts", such as DELETE, INSERT, SELECT, and UPDATE, while the remaining of the query allows to unambiguously interpret what the query is supposed to achieve.		
Message brokering	In PERSONA used as opposed to object brokering. Object brokers can find objects based on their interface specification and return a reference to them so that their methods can be called on a syntactical basis. PERSONA chose a message brokering approach to realize the brokering mechanism defined under goal-based interoperability to increase the chances for avoiding syntactical dependencies. In this approach, the broker takes over all the tasks of (1) matchmaking between requests and offers as well as between subscriptions and notifications, and (2) handing over messages between the endpoints without need for direct connection between the endpoints, thus avoiding endpoint-specific interfaces.	Contributes towards more independence in the development of pluggable components	Cf. WebSphere Message Broker & wikipedia message broker versus wikipedia object broker
Self-organizing system	An open distributed system that builds on seamless connectivity and realizes goal-based interoperability. Important characteristics of such a system are: 1) It is usually based on the provision of certain *connection points* (CPs) with related interfaces, protocols, and roles possibly at different levels of detail. 2) Each component respecting those interfaces and protocols that is plugged into the system must initially announce its roles with related offers and subscriptions on each CP. Similarly, it must announce its requests on related CPs whenever it needs functionality that it assumed to be available in the system. 3) As a consequence of plugging a new component into the system, its offers will	Obvious	PERSONA IR3.1.1, IR3.1.2 & D3.1.1 + several publications of the German research projects EMBASSI and DynAMITE, the SODAPOP spec, the strategic research agenda of ARTEMIS, and wikipedia

	immediately be available to the system. All the consequent communication will be performed through the CPs to which it has connected, no matter if it is on the sender side or on the receiver side. Being a part of an open distributed system, the component must however come with a high level of fault tolerance as there is no guarantee about the constant availability of sources that can fulfil its needs. 4) In addition to abstraction over messaging routes, the dependency between the endpoints (of requests, responses, and notifications) is limited to shared vocabularies used in constructing messages, thus avoiding the definition of additional interfaces and protocols beyond those of the system CPs. 5) Such a system must provide facilities for aggregating over available values so that effort needed for composing new offers (services) out of other available offers (services) is minimized. The same applies also to generating high-level notifications that are derived from previous relatively low-level notifications. The above conditions imply that there is a good level of automatism in the organization of the system even if the selection and authorization of components may need human intervention.		
Middleware	A piece of software that glues the distributed components of a self-organizing system to each other, thus allowing the system to emerge. It resolves the challenges of seamless connectivity (e.g. node discovery) and goal-based interoperability (e.g. providing a brokering mechanism) while hiding the distribution and possible heterogeneity of underlying operating systems and networking protocols → no architectural layer but a piece of software!	Obvious	see INRIA book, several publications of the German research projects EMBASSI and DynAMITE, the SODAPOP spec, the strategic research agenda of ARTEMIS, and PERSONA publications
Distributed realization of the middleware	In self-organizing systems, the realization of the middleware in a distributed way simplifies the physical architecture of the system just as an ensemble of networked nodes in which each instance of the middleware represents a node in the system. Seamless connectivity will become	Obvious	see several publication of the German research project DynAMITE

	an issue to be solved at the level of middleware instances, i.e. it will be sufficient when middleware instances are able to discover each other and form the ensemble in this way. The middleware can then provide two types of CPs. The first type would be the CPs between instances of the middleware: nodes discovering each other join to each other at these CPs and organize themselves in this way into an ensemble. The cooperation between the instances of the middleware can then solve the two important challenges of self-organizing systems, namely distribution and heterogeneity of nodes in a transparent way without involving components running on each node. The second type of CPs is provided to the local components of each node to facilitate their hookup to the system. From the point of view of such a component, the problem is reduced to using the runtime facilities of its platform to find the relevant CPs of the shared instance of the middleware and connect to them. In this way, each component integrates itself into the system independently from any other local or remote component participating in the emerging system.		
Sodapop Model	SODAPOP (Self-Organizing Data-flow Architectures suPporting Onotology-based problem decomPosition) is a conceptual model based on which the distributed middleware solution of PERSONA has been realized. It introduces two important concepts, namely *transducer*s and *channel*s. Each pluggable component joins to the system through its transducers (one transducer per middleware CP and role). Each of the transducers is a specialist in exchanging certain types of messages with the rest of the system and takes over the responsibility of mapping internal representation of those messages onto an appropriate external representation and vice versa (a kind of adapter for CPs). A channel is a medium for message brokering between connected transducers. Channels may be seen as the “cutting points” for distributing a system: several instances of the same channel distributed across multiple physical nodes cooperate in	A reliable conceptual work for the distributed realization of the middleware that provides a basis for modular brokering mechanisms, one per SODAPOP channel.	the SODAPOP spec

	order to virtually form a single global channel. Each piece of the channel on each node provides an API to transducers and defines the possible roles with which transducers can connect to it along with protocols based on which they can put messages on the channel or are expected to process a message given to them. On the other hand, such pieces of the same channel cooperate with each other based on the channel strategy to hide the distribution of the system from the transducers connected. A middleware based on Sodapop model realizes the channels as the CPs of a self-organizing system towards transducers and all other components of the system realize transducers that connect to channels. Hence, the essentials of a Sodapop-based system are defined by identifying its set of channels and for each of them specifying the channel strategy and API. There are two types of channels: event-based channels, on which messages are posted without expecting any reply, and call-based channels, on which posted messages will be replied with an appropriate response. Possible roles on an event-based channel are *publisher* and *subscriber*, on a call-based channel, however, *caller* and *callee*. A publisher transducer expects that for each published event, the corresponding event-based channel will find (an) appropriate subscriber transducer(s) that know(s) how to continue with the event. In case of call-based channels, the calling transducer determines the further processing of the result after the channel arbitrates a call and returns the response.		
Virtual communication bus, or simply “bus”	The term used in PERSONA to refer to a Sodapop channel. Reasons for using another name: (1) the term I/O channel was already used in PERSONA with another meaning, (2) in order to benefit from the analogy with hardware notion of a bus, and (3) a PERSONA bus does not respect the Sodapop constraint of being “memory-less”. The communication buses reflect the loose connections needed in a dynamic environment and represent, in a modular	obvious	Cf. Enterprise Service Bus

	way, the need for interface/ontology definitions, protocol specifications for communication, and strategies for “dispatching incoming messages” to an appropriate (set of) receiver(s). The PERSONA middleware provides four buses: an *input* and an *output* bus for covering interoperability needs related to handling explicit interaction with human users and a *context* and a *service* bus for realizing push and pull mechanisms for all other interoperability needs between components attending a self-organizing system. PERSONA also avoids using the term transducer and uses more specific per combination of the bus to which a transducer connects and the role played by that transducer on that bus, this way inventing different names for different transducer types.		
The PERSONA input bus	An event-based bus for sharing explicit input provided by a human user.	obvious	PERSONA IR3.1.2 & D3.1.1
Input event	The type of messages brokered by the input bus. Such messages encapsulate explicit user input captured through channels to the physical realm. Explicit user input may be provided to a PERSONA system in the context of a dialog (see below) or in a context-free way. For each input event not encapsulating a context-free input, the publisher of the event must enrich the input event with the ID of the dialog in whose context the input is provided. If such a dialog ID is specified in an output event, only an input subscriber (See below) that has subscribed for input related to that dialog will be notified to handle the input event, whereas all context-free input is forwarded to a default handler.	obvious	PERSONA IR3.1.2 & D3.1.1
Input publisher	The type of transducers that publish input events onto the input bus. Important metadata to be provided by an input publisher: (1) the identity of the user providing the input, and (2) if the input is provided in the context of a dialog, the ID of that dialog.	obvious	PERSONA IR3.1.2 & D3.1.1
Input subscriber	The type of transducers that subscribe to the input bus for certain input events. A specific input subscriber is assumed to exist	obvious	PERSONA IR3.1.2 & D3.1.1

	that is subscribed forever for all context-free input. Otherwise, subscriptions to the input bus are made very dynamically by specifying the dialog ID in the context of which the subscriber is willing to wait for input, just before publishing the corresponding output event (see below) that starts the dialog. The subscriber is then automatically unsubscribed as soon as the cycle of the dialog is closed. Applications must normally have such a transducer.		
The PERSONA output bus	An event-based bus for handing over info prepared for being presented to (a) certain human user(s). This brokerage is supposed to be done in a personalized and context-aware way so that the info is eventually presented to the user using output channels that are most appropriate for the current situation of the addressed user and are the best match for his / her possible impairments, capabilities and preferences.	obvious	PERSONA IR3.1.2 & D3.1.1
Output event	The type of messages brokered by the output bus. Such messages encapsulate a modality- and layout-neutral representation of info prepared for being presented to (a) certain human user(s) through channels to the physical realm. It is assumed that an output event already contains content-specific metadata, such as content language and privacy level, and addressed user. The output bus first enriches the output event with adaptation parameters (location of the user, possible impairments, proposed modality for the current situation, modality-specific parameters derived from his / her possible impairments, capabilities and preferences, etc.) and then for all output subscribers (see below), it checks (using ontological matchmaking) if the enriched output event can be validated as an instance of the class of output events in which the output subscriber is interested. If yes, that output subscriber is notified to handle the enriched output event.	obvious	PERSONA IR3.1.2 & D3.1.1
Output publisher	The type of transducers that publish output events onto the output bus. Output publishers are supposed to already include content-related metadata (e.g., content language and privacy level, addressed user) to each published output event. Applications must normally have such a	obvious	PERSONA IR3.1.2 & D3.1.1

	transducer.		
Output subscriber	The type of transducers that subscribe to the output bus for certain output events. The subscription is supposed to be rather static and occur once at the registration time with the output bus. The subscription provides criteria for output events that can be handled by the subscriber depending on output channels controlled, e.g. how privately the channels can be used, related modalities and locations, set of users (e.g., because of the limitations of the used user identification mechanisms). An output subscriber selected by the output bus for handling an enriched output event, looks for metadata that provide instructions regarding the addressed user, his / her location, the modality to use, and modality-specific parameters derived from his / her possible impairments, capabilities and preferences.	obvious	PERSONA IR3.1.2 & D3.1.1
Dialog	A cycle of publishing an output event and receiving a related input event.	obvious	PERSONA IR3.1.2 & D3.1.1
The PERSONA context bus	An event-based bus providing a general-purpose push mechanism that can be used if the info to be shared is neither addressing a human user nor the representation of captured user input. It is called the context bus, because it is used for the exchange of sharable info about changes in the system and its environment, and for an interested subscriber the event has happened in the context of its runtime environment.	obvious	PERSONA IR3.1.2 & D3.1.1 + AmI-08 paper on PERSONA CASF (Framework Supporting Context-Awareness)
Context element	A distinct characteristic or feature of a distinct resource. Using the RDF representation techniques, a context element in this sense can be identified uniquely by a pair of URIs, namely the URI of the resource and the URI of the corresponding property.	obvious	PERSONA IR3.1.2 & D3.1.1 + AmI-08 paper on PERSONA CASF
Context event	A statement reporting the state of a context element at a specific time where the state was changed into the reported value. In terms of RDF, this could be as simple as an RDF statement – the two URIs identifying the underlying context element would form the subject and predicate of the RDF statement and the state value would be the object of the statement; however, as a reporting statement bound to a specific	obvious	PERSONA IR3.1.2 & D3.1.1 + AmI-08 paper on PERSONA CASF

	time, a context event should be treated as a reified statement in order to be able to specify the associated time, too. Additionally, listeners to such events may need to also know who is reporting this value with which level of confidence and / or temporal validity. For each pair of received context event and context subscriber (see below), the context bus checks (using ontological matchmaking) if the context event can be validated as an instance of the class of context events in which the context subscriber is interested. If yes, that context subscriber is notified to handle the context event.		
Context publisher	The type of transducers that publish context events onto the context bus. Important metadata to be provided by a context publisher consists of the provider info (see below), a timestamp, confidence level, and temporal validity.	obvious	PERSONA IR3.1.2 & D3.1.1 + AmI-08 paper on PERSONA CASF
Context subscriber	The type of transducers that subscribe to the context bus for certain context events. Subscriptions to the context bus can be made very dynamically and / or at the registration time with the context bus. The subscription provides criteria for context events that can be handled by the subscriber depending on their subjects and subject types, properties, reported value, temporal validity, confidence value, and / or provider.	obvious	PERSONA IR3.1.2 & D3.1.1 + AmI-08 paper on PERSONA CASF
Context provider	A component that has a transducer for publishing context events onto the context bus. Major subgroups of context providers are: (1) *Controllers* that have the states of some context elements under their control, like an actuator controlling the lights in a place that can also provide info about the state of the controlled light sources, (2) *Gauges* that wrap a sensor, and (3) *Reasoners* that estimate the state of some context elements by combining different known information and applying certain methods of aggregation, statistical analysis and / or logical deduction. The values reported by gauges could be measured dimensions. If so, the concepts *DimensionMeasure* or *MultiDimensionMeasure* (or subclasses of them) are used for building the object part	obvious	PERSONA IR3.1.2 & D3.1.1 + AmI-08 paper on PERSONA CASF

	of the reified statements. Such "measure"s provide info about the unit and accuracy of measurement in addition to the measured value.		
The PERSONA service bus	A call-based bus providing a general-purpose pull mechanism that can be used for utilizing accessible functionality. As explained earlier, "service" can be used as a general abstract term for talking about accessible functionality that can be utilized in the virtual realm using pull mechanisms. The bus API, communication protocol, and strategy altogether realize a brokering mechanism that is providing for goal-based interoperability. Obviously, the most important of message types exchanged on the service bus correspond to service requests, service calls (forwarding a request to matching callees), and service responses.	obvious	PERSONA IR3.1.2 & D3.1.1 + SMR2 paper on PERSONA Semantic RPC
Service Profile	For each service made available to the service bus, a service callee (see below) must register a service profile that describes both the "what"s (what is the service for) and the "how"s (how it can be utilized). The bus uses the "what"s in the course of matchmaking and the "how"s when the match was successful and the service is going to be utilized. The structure of such a profile is taken from OWL-S specification for service profiles. For more info & examples, please refer to "Goal-based Service Utilization Using SPARQL and OWL-S" in the latest OWL-S site.	obvious	PERSONA IR3.1.2 & D3.1.1 + SMR2 paper on PERSONA Semantic RPC
Service Request	The type of messages used by a service caller (see below) to request a service. They are modeled as SPARQL-like queries in the following form: 'CALL' [('ALL' | 'ONE' | 'BEST') '(' VAR ')']+ ['DELETE {' <statements possibly using variables> '}'] ['INSERT {' <statements possibly using variables> '}'] ['SELECT' <list of constants or variables (with/without aggregation)>] 'WHERE {' <statements & filters clarifying the context of the variables used> '}' Each variable (VAR) used in the CALL clause represents a service to be called. It	obvious	PERSONA IR3.1.2 & D3.1.1 + SMR2 paper on PERSONA Semantic RPC

	must be bound to a specific class of services (e.g., the class of lighting services) in the WHERE clause. In this way, the query is basically stating that one or more services are requested. The combined or standalone usage of DELETE and INSERT altogether specifies the expected effects either in the virtual realm or in the physical realm or both. SELECT specifies the list of expected return values. Only one of DELETE, INSERT, and SELECT is mandatory. Combined usage of DELETE and INSERT is SPARQL-equivalent of UPDATE in SQL. For more info & examples, please refer to "Goal-based Service Utilization Using SPARQL and OWL-S" in the latest OWL-S site. Upon receipt of such a query, the service bus checks the set of service profiles that describe services from the same class (e.g., the class of lighting services) to see which one of them has the same effect and output as required by the query while fulfilling all conditions specified in the WHERE clause. In this way, a set of matching services will be found by the service bus. Then, the directives ALL, ONE, and BEST are considered to respectively determine if all matched services should be called, just one in a random way, or even just the one that is the best match. Besides that, the WHERE clause may provide QoS or other non-functional criteria (e.g., a related entity being the nearest to a specific location) that already restrict the number of matched services to one so that the above directives may make no difference.		
Service Call	The type of messages sent by the service bus to a concrete service callee (see below) in order to instruct the addressed component to start the process of providing a concrete service registered with the bus. Before sending such a message, the bus must derive the needed input parameters from the query (actually, the input parameters are determined already during the matchmaking as some services will match only if a certain constant value from the query is used as a certain input parameter). Hence, a service call specifies which process should be started with which	obvious	PERSONA IR3.1.2 & D3.1.1 + SMR2 paper on PERSONA Semantic RPC

	concrete input parameters.		
Service Response	Both messages in reply to a service call returned by a service callee (see below) to the service bus and those in reply of a service request returned by the service bus to a service caller (see below) are called service response. In both cases, it consists of a *call status* (e..g., succeeded, no matching service found, service-specific failure, or timed out) and a set of return values, however, a response sent by a service callee to the service bus distinguishes the output values in terms of IDs already used in the service profile whereas in case of a response sent by the service bus to a service caller, the returned values are distinguished in terms of variables specified in the original query (service request). That is, the job of the service bus is not limited to forwarding original responses from callees to callers, but it must first perform a backward mapping. This is possible because, similar to the case of determining the set of input values in case of service calls, also the mapping between output IDs used in service profiles and output variables used in service requests had certainly belonged to the conditions of the match. In addition to such a backward mapping, the service bus may have to also combine several responses received from different callees into one single response if more than one callee was called as a result of service brokering.	Obvious	PERSONA IR3.1.2 & D3.1.1 + SMR2 paper on PERSONA Semantic RPC
Service Caller	The type of transducers that attach to the service bus and are allowed to send service requests to the bus and will receive service responses in return.	Obvious	PERSONA IR3.1.2 & D3.1.1 + SMR2 paper on PERSONA Semantic RPC
Service Callee	The type of transducers that register service profiles with the service bus and might be addressed by the bus to perform a specific service provision process. They are expected to send service responses in reply to each such “address” (call).	obvious	PERSONA IR3.1.2 & D3.1.1 + SMR2 paper on PERSONA Semantic RPC
I/O Handler	Each I/O handler manages a set of I/O channels and subscribes to the output bus by specifying its capabilities, which is used by the output bus in the course of match-making with adaptation parameters	obvious	PERSONA IR3.1.2 & D3.1.1 + chapter in AISE book on PERSONA

	associated with each output event. That is, using the adaptation parameters, the output bus tries to find a best-match I/O handler that receives the content to be presented to the user along with instructions in regard to modality and layout derived from the adaptation parameters. The selected I/O handler is then responsible for converting the application output to the format appropriate for the channel selected in accordance with received instructions. It then monitors its input channels to catch the related user input. Upon recognized input, it must convert it to the appropriate format in accordance to the previously handled application output and publishes it as an event to the input bus. I/O handlers are application-independent, pluggable technological solutions that manage their respective I/O channels to concrete devices using one or more of the following alternatives: A tight connection using low-level protocols, a loose connection using device services on the service bus, and / or a loose connection using contextual events on the context bus.		platform
Dialog Manager	The Dialog Manager is an application-independent component that handles the system-wide dialogs and hides the complexity of utilizing the application services from the user. Another important task for the Dialog Manager is the provision of a mechanism for associating service calls with situations as means for providing a configurable management of the reactivity of an AmI environment. For this purpose, the Dialog Manager relies on a configurable repository of rules schematically in the form of "situation → action" (abbreviated as "s[i]→a[j]"). Then, it must subscribe to the context bus for all situations s[i], for which it has an associated action a[j] in its repository. The association "s[i]→[j]" is the heart of controlling system behavior and hence it will be very advantageous to store it in a central configurable repository. Concrete tasks of the DM in PERSONA are: Providing system dialogs, such as navigation through available services, providing standard and common dialogs,	obvious	PERSONA IR3.1.2 & D3.1.1 + chapter in AISE book on PERSONA platform

	such as "login" or "notify", handling system reactivity using rules that associate situational events with actions (service requests), handling context-free user input in terms of service search (if an I/O handler detects user input that has no relation to any previous output, the Dialog Manager receives the input and tries to interpret it as search for services).		
Context History Entrepôt	The Context History Entrepôt (CHE) gathers the history of all context events in a central repository not only to fill the gap caused by context publishers that provide no query interface, but also to provide a fallback solution for those that cannot maintain the whole history of data provided by them. Additionally, it guarantees the essential support to reasoners that perform statistical analysis and need context stored over time. As a singleton component, the CHE takes care of logging every context event that is published in the context bus by specifying a "pass-all" filter when subscribing to the bus. In order to have the growth of the repository under control, the CHE also implements a deletion policy based on the likeliness of the data to be needed further on.	obvious	PERSONA IR3.1.2 & D3.1.1 + chapter in AISE book on PERSONA platform
Situation Reasoner	A general-purpose context reasoner that uses the database of the CHE and infers new contextual info using the logical power of the RDF query language SPARQL. It stores "situation queries" persistently and indexes them based on context events that must trigger its evaluation. It provides two services on the service bus, one for accepting new situation queries and the other for dropping them. These services are also used by a graphically interactive tool for administrators in order to facilitate the introduction of new relevant situations to the system by providing an overview of existing context providers, allowing drag-and-drop interaction using artifacts for accessible context elements, catching logical errors made by the user, and generating the appropriate SPARQL query string, to name a few of its features.	obvious	PERSONA IR3.1.2 & D3.1.1 + chapter in AISE book on PERSONA platform
Profiling	In order to guarantee the adaptability of an AAL space to the wishes and preferences of its users, it is essential that a special-	obvious	PERSONA IR3.1.2 & D3.1.1 + chapter

	purpose component is foreseen for the management of the profiles and the provision of needed shared mechanisms. We call this component the Profiling Component.		in AISE book on PERSONA platform
Service Orchestrator	Services may exist only at a meta-level in terms of "composite" services made from combining really-existing "atomic" services. The Service Orchestrator (SO) is the component in charge of interpreting the metadata describing a composite service and performing the instructions within it. These descriptions are added / removed / modified by a GUI for system administrators. The SO registers the composite services to the bus like any other service callee would do so for its atomic services. This way, whenever a composite service is called on the service bus, the bus will find the SO as the only object that "implements" that service; hence the SO implements the callee interface for handling service requests. At this stage, the SO starts to execute the corresponding composite service by calling the sub-services through its capabilities as a caller until it finishes and then returns the results to the bus that will forward them to the original caller. Summarizing the admin tool aspect so far, it is worth to mention that three repositories must be kept configurable for administrators of AAL spaces: a) the database of the Situation Reasoner regarding "conditions→situation" rules, b) the database of the Dialog Manager regarding "s[i]→[j]" associations, and c) the database of the SO regarding composite services.	obvious	PERSONA IR3.1.2 & D3.1.1 + chapter in AISE book on PERSONA platform
AAL Space Gateway	In order to facilitate remote access to AAL spaces and, the other way around, to support AAL spaces in notifying an absent native user, as well as to enable the bridging between AAL spaces and, furthermore, to provide a possibility for external service providers to advertise their services to the occupants of AAL spaces, we suggest to employ a special-purpose component called the AALSpace Gateway. The gateway provides access to the hosted services in the AAL space under a fixed URL. For this purpose, it must act within	obvious	PERSONA IR3.1.2 & D3.1.1 + chapter in AISE book on PERSONA platform

	the AAL space as input publisher and output subscriber so that in case of incoming remote access and after authentication, the remote user can start a dialog with the smart home to access info and services for which he or she has the required access rights.		
PISM	The middleware must control the access to services with the help of a component that we call the Privacy-aware Identity & Security Manager (PISM) that is also supposed to act as a service provider. The main responsibilities of the PISM are: a) management of the entities' identities and credentials, b) management of permissions for accessing "hosted" services, c) providing authentication services, and d) providing a tunable mechanism for deciding on the disclosure of private data.	obvious	PERSONA IR3.1.2 & D3.1.1 + chapter in AISE book on PERSONA platform

6. SOPRANO Terminology Model

Concept	Definition	Relevance to universAAL	Reference
SOPRANO Ambient Middleware/platform or openAAL	Provides the intelligence of the system. SAM collects incoming sensor information, analyses them (in the Context Manager), decides with the help of a procedure database which actions in form of workflows to be taken (in the Procedural Manager) and executes them through different actuators in the house (in the Composer)	Middleware platform that provides decoupling, enables independent contribution and provides central platform services	D2.1.1: Initial SOPRANO Architecture p. 6/25
Service Integration Component	An OSGi service middleware provides the technical basis for this system part. All local services in the house as well as external services are registered in its central service registry	Provides support for service management, remote management, message exchange etc.	D2.1.1: Initial SOPRANO Architecture p.25
Administration Component	It contains the (graphical) administration interfaces for the different administrator roles in SOPRANO: the healthcare consultant, the case manager as well as the carer.		D2.1.1: Initial SOPRANO Architecture p.25
SOPRANO Ontology	This is not a functional component of SOPRANO, but it is a very fundamental part of the system. All other modules rely on it as a common means of understanding. It defines information that can be exchanges, knowledge that can be stored and its datatypes.		D2.1.1: Initial SOPRANO Architecture p.25
SOPRANO Context Ontology	Part of SOPRANO ontology that describes the context information that can be stored in the Context Manager. It acts also as the central data model.	Context model whose principles can be reused in universAAL context modelling	D2.2.1: User Context (section 1-3)

Context Manager	The task of the Context Manager is to host SOPRANO's user context database which stores context information as well as the user's profile data. It receives events of all installed sensors and updates the database accordingly. Other components can register with the context manager using state patterns to be notified about specific status changes.	Enables complex situation detection, captures and deals with history data and error-prone sensor information.	D2.1.1: Initial SOPRANO Architecture p.29 & D2.2.3: First version of Context Manager
Procedural Manager	The procedural manager processes the SOPRANO high-level rules, called procedures, as well as the procedure templates. The procedural manager provides interfaces for storing, deletion, and retrieval of procedures and procedure templates. Furthermore, the procedural manager executes the procedures as reactions to status changes with the help of Context Manager and Composer	Can be used to store and execute abstract, reusable, easy-to-define system behavior in form of BPEL workflows.	D2.1.1: Initial SOPRANO Architecture p.29f & D2.3.1: Initial version of SOPRANO procedural Manager
Composer	The composer queries the service integration component for suitable concrete services, matches them against the abstract goal descriptions in a context-aware manner and invoke the best-fitting service. Semantic service description languages as well as appropriate matchmaking algorithms are used for this task.	Provides semantic services matchmaking.	D2.1.1: Initial SOPRANO Architecture p.30 & D2.4.1: Initial version of SOPRANO Composer
Sensor Services	Services that typically represent individual sensors or sensor systems. In general, every service that delivers information to the context manager.		D2.1.1: Initial SOPRANO Architecture p.16 & D 1.3.1 Abstract description of entities involved in the identified application scenarios

Actuator Service	Services that typically represent actuators. In general, every service that takes information and puts it out in the real world		D2.1.1: Initial SOPRANO Architecture p.16 & D 1.3.1 Abstract description of entities involved in the identified application scenarios
Analytical Services	Services that process the raw data of sensors and actuators to make them “SOPRANO-compliant”. They are typically part of sensor services.		D2.1.1: Initial SOPRANO Architecture p.25
Complex or Interactive Services	More complex services that can provide more complex behavior. Typically they act as both Sensor and Actuator Service and can, for example, provide complex user interaction.		D2.1.1: Initial SOPRANO Architecture p.18
Procedures	Procedures are stored in the Procedural Manager and define specific situations and the system’s reaction to those situations.		D2.1.1: Initial SOPRANO Architecture p.19/25f
Service Matchmaker and Invocator	Matches service requests and offers by means of Diane Service Description framework and executes the best fitting service.		D2.4.1: Initial version of SOPRANO Composer (section 1-3)
Workflow Engine	Executes Procedures with help (by invocating) Composer and Context Manager in a context-aware manner (by invocating the Context Manager)		D2.4.1: Initial version of SOPRANO Composer (section 1-3)
Ambient System	Systems that works primarily in the background by understanding the current situation in the house via the connected sensors and influencing it via the connected actuators.		D2.1.1: Initial SOPRANO Architecture p.17f

Architecture	Architecture is viewed from 5 different perspectives: In the *Logical View,* the functional decomposition of the system is regarded. In the *Use-Case View,* application scenarios of the system are defined from a user's point of view. In the *Development View,* the implementation of the software as well as its packaging is defined. In the *Deployment View*, the mapping of the software components to the underlying hardware is defined. In the *Process View* the aspects of the system at run-time such as tasks, threads, or processes as well as their interactions are defined.		D2.1.1: Initial SOPRANO Architecture p.24ff