The PERSONA Service Platform for AAL Spaces

Mohammad-Reza Tazari and Francesco Furfari and Juan-Pablo Lázaro Ramos and Erina Ferro

1 An Introduction to the PERSONA Project

The project PERSONA aims at advancing the paradigm of Ambient Intelligence through the harmonization of Ambient Assisted Living (AAL) Technologies and concepts for the development of sustainable and affordable solutions for the independent living of senior citizens. PERSONA is one of the integrated projects funded by the European Commission within the 6th Framework Program for IST (Information Society Technologies) on AAL for the Aging Society. It involves the participation of 21 partners, from Italy, Spain, Germany, Greece, Norway and Denmark, with a total budget of around 12 million Euros.

Ambient Assisted Living (AAL) is the concept that groups the set of technological solutions, named AAL Services, targeting the extension of the time that elderly and disabled people live independently in their preferred environment, i.e. their own four walls, neighborhood and town where they are used to live [1]. AAL Services provide personalized continuity of care and assistance, dynamically adapted to the individual needs and circumstances of the users throughout their lives.

The main objective of PERSONA is the development of a scalable open standard technological platform to build a broad range of AAL Services. A number of AAL Services are being implemented over such a platform in order to demonstrate, test and evaluate their social impact and potential exploitation scenarios. AAL Services

Mohammad-Reza Tazari
Fraunhofer-IGD, Darmstadt, Germany e-mail: saied.tazari@igd.fraunhofer.de

Francesco Furfari
CNR-ISTI, Pisa, Italy e-mail: francesco.furfari@isti.cnr.it

Juan-Pablo Lázaro Ramos
ITACA-TSB, Valencia, Spain e-mail: jualara@upvnet.upv.es

Erina Ferro
CNR-ISTI, Pisa, Italy e-mail: erina.ferro@isti.cnr.it

H. Nakashima et al. (eds.), *Handbook of Ambient Intelligence and Smart Environments*,
DOI 10.1007/978-0-387-93808-0_43,

under development are divided into four different categories according to an analysis of elderly needs in regard to a life with the highest level of independence:

- AAL Services supporting social inclusion and experience exchange,
- AAL Services supporting elderly users in their daily life activities,
- AAL Services supporting elderly people to feel more confident, safe and secure, and helping their relatives to manage risky situations, and
- AAL Services fostering mobility and supporting elderly people outside their homes.

Figure 1 shows an example of a scenario where the user is supported to live in a more confident, safe and secure way. It shows how the home environment is able to detect potential risky situations, and to propose actions to the user for mitigating the risks. The interaction mechanisms between the user and the system are compliant with the paradigm of natural and contextual communication, in order to minimize the impact on the user's life.

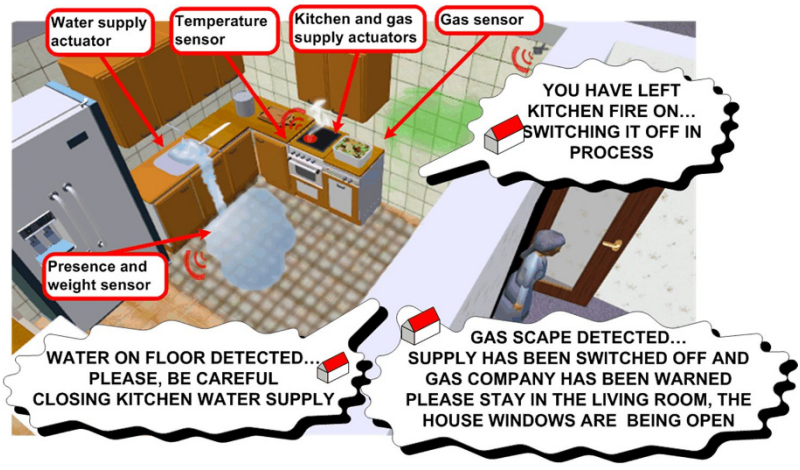

Fig. 1 Virtual scenario of AAL Services related to support elderly people to feel safe and secure

The targeted AAL technological platform is expected to be developed under the umbrella of a reference architecture for AAL spaces and AAL service provision as the main scientific and technological outcome of the project. The reference architecture of PERSONA for AAL systems has been designed to enable the smooth, flexible and modular integration of service-providing components using appropriate Information and Communication Technologies, and to create systems which are cost-effective, reliable & trusted, and adaptable to the requirements of each specific user within both the house and urban environments.

Of course, a such open reference architecture promotes its wide adoption by the related R&D community when it can be used as a powerful tool to build affordable AAL Services oriented to help elderly people to live independently. Hence, one

of the most important challenges of PERSONA in the medium term is to transfer the project's results into a commercial scenario within two or three years. For this reason, a number of resources are being spent in order to define the business strategy to exploit the proposed scenarios into a real life market environment on a European scale, where the different types of stakeholders participate and coordinate to provide innovative, affordable, sustainable and effective AAL Services for elderly people.

2 Requirements on a Service Platform for AAL Spaces

To derive both user and technical requirements, we investigate in the following the requirements on an AAL service platform from the view point of both the services to be hosted and the users of such services.

2.1 User Requirements

An AAL Service is a composition of one or more functionalities that covers specific needs of elderly or disabled people, allowing them to live more independently. It ensures the continuity of care along the time and in the preferred environments (home, neighborhood, village, also named AAL Spaces) by using an Ambient Intelligence infrastructure that provides the basis for service development and deployment.

Before starting to list the requirements on such an ambient intelligence infrastructure, also known as AAL Service Platform, we would like to analyze the requirements of end users on the final AAL Services to be delivered to them. In the scope of PERSONA, end users are elderly people. Alone this fact has a lot of implications in terms of both coverage of needs and service delivery, because the targeted services involve technologies that are unusual or even unknown to this group of users, at least within the early decades of the 21^{st} century, and therefore can cause lack of acceptance or even rejection.

Our analysis was based on the four categories of elderly needs (social integration, support for daily life activities, support for feeling safe and secure, and mobility) as the starting point, and targeted the realization of an ambient assisted living system that covers those needs or minimizes their impact.

The sources of information used to gather the list of generic requirements for AAL Services were:

- ISTAG documents [7, 13] describing the Ambient Intelligence paradigm
- ARTEMIS Strategic Research Agenda [2]
- The AAL program [1]
- PERSONA Project investigations with experts about elderly care and ethical and legal issues.

The following list summarizes requirements derived from the combination and the analysis of the above sources with focus on the development of AAL Services:

- General requirements
  - The new services must be as effective as the usual way of dealing with situations
  - The user must be able to disconnect the service at all times
  - The system must not be designed as a full replacement of personal care by relatives, neighbors or friends
  - The system should always take into account the current context of the user in order to avoid inconsistency and ensure coherent service provision
  - It should be possible to easily add new functionalities and services to a concrete physical infrastructure
  - It should be possible to add new devices to the system while enabling their easy adoption and usage by the rest of the logical or physical components
  - Devices used for building up the ambient infrastructure should be "invisible" as far as possible; in particular, installed sensors and those used in wearables shall be hidden / unnoticeable
  - The automatic decisions of the system must have the user's confirmation and remain traceable for him/her
- User interaction with AAL Services
  - Services should conform to usability standards and apply design-for-all with a focus on older adults
  - The system must be easy to learn
  - Adaptation: User Interfaces shall incorporate features for coping with access impairments and capability changes due to aging
  - Communication with the user should be done in a consistent way when using different modalities, such as voice, text, graphics, signals, actions, and state changes
- Privacy and data management:
  - Data obtained must be relevant for the purposes of the service and from early design stages, emphasis must be put on the necessity of including data items
  - The user shall be able to prevent devices from gathering information about him/her at any time he/she wishes
  - For various purposes, such as provision of services and information, exercise of rights, and gathering of data, the system must be able to authenticate who the user is
  - Transmission of data to third parties must be kept to a minimum and security of data transmission to external entities must comply with legal requirements
  - The system shall allow the verification of access rights of third parties with possibility for rectification, opposition and cancellation

2.2 Technical Requirements

The most important deliverable of the PERSONA project is the AAL Service Platform. Hence, for defining the technical requirements, which must correspond to the targeted platform, we had to establish a common understanding of such a platform for AAL services.

Normally, platform is understood as a framework on which applications (in this case AAL services) may run. The framework may be very specific and tied to certain hardware and operating system or more open by abstracting those levels based on a more generalized runtime environment, such as Java. Apart from the runtime environment, the platform may readily provide services that perform various common functions in order to avoid duplicating efforts. For the purpose of interoperability, a platform must also facilitate communication and collaboration between various components, either application components or those providing the platform services. Ontologies, communication protocols and data exchange formats can be provided as part of such an interoperability framework. AAL systems, as applications of Ambient Intelligence, are highly distributed systems which increases the importance of the interoperability framework.

In order to create a list of requirements based on the above understanding, we explored the existing research initiatives in the field of ambient intelligence architectures and middleware implementations thoroughly at a national, European, and international level. Finally, the most important inputs came from the research priorities of ARTEMIS [2] in the fields of "Seamless Connectivity and Middleware" and "Reference Designs and Architectures", the SODAPOP notion of self-organization [12], and even more specific initiatives such as the Jaspis architecture [25]. Some of the more general requirements can be summarized as follows:

- guarantee a high-level of flexibility in the distribution of functionalities and facilitate the integration of arbitrary numbers of sensors, actuators, control units, appliances, and applications into the system
- support ad-hoc networking to enable components to immediately act as a node in the networking infrastructure
- support different communication patterns, such as event-based and call-based
- provide service discovery and binding mechanisms: explicit service requests must be resolved and responded as far as the corresponding service is available per se
- support service chaining: meaningful continuation with (intermediate) results reached / generated by one component in another component by discovery of alternatives and choosing a reasonable next "step"
- provide mechanisms for event aggregation: seamless combination of events generated / captured by different components in order to infer a possibly more significant event
- support for parallel processing of events while detecting and resolving / prohibiting possible conflicting interpretations and actions in favor of more reasonable ones

- provide for service composition and orchestration: serving explicit service requests that cannot be resolved directly but by intelligent strategy planning algorithms that use simpler services available as building blocks to compose the demanded service. To serve the original service request, the composed service must then be executed[1]
- facilitate the explicit user interaction with the system while separating the presentation mechanism from application logic and supporting multimodality in an ensemble of devices distributed at different locations
- support personalization and context-awareness in all layers of the system and adaptability in the presentation layers
- hide the complexity of utilizing services from the users accessing them (cf. the SET concept of eMobility[2])
- provide for identity management, privacy-awareness, and QoS-awareness
- follow a modular design to better support distribution and extensibility of the system, higher maintainability / dependability, and greater possibility for reusing existing and composing new functionality
- At the realization level, characteristics to be considered include stability and diagnosis efficiency and performance, and completeness and simplicity of the API

3 Evaluation of Existing Solutions

A declared goal of PERSONA is to avoid re-inventing the wheel in its technical developments, especially by identifying promising conceptual approaches while combining them with the newest technological trends. The development of the platform followed this principle using an appropriate methodology outlined in [3]. Starting with the above mentioned requirements, over 30 R&D projects with promising solutions in the fields of architectural design and middleware solutions for AmI systems were gathered. After two quick iterations for filtering out those with comparatively less significant outcomes, this list was reduced to six: Oxygen (launched by MIT in 1999), EMBASSI / DynAMITE (1999-2006, two German national research projects introducing the SODAPOP model), RUNES (2004-07, EU-IST project), Amigo (2004-08, EU-IST project), and ASK-IT (2004-08, EU-IST project). The theoretical and practical evaluations (e.g. by testing the provided software toolkits) showed that a combination of the concepts from the SODAPOP model and those developed within Amigo could lead to an improved solution for AmI systems. Section 4 discusses different aspects of the PERSONA solution derived in this way. In

[1] The latter five requirements collectively facilitate self-organization of the distributed system in an intelligent way, guaranteeing a higher level of service in comparison with using each of the nodes in an standalone way.

[2] The technology platform for mobile and wireless communications in the seventh framework program of the European Union for the funding of research and technological development in Europe – www.emobility.eu.org

the remaining of this section, an overview of the two major "inputs" (SODAPOP & Amigo) will be given followed by a discussion of our evaluation results.

A system built according to the SODAPOP[3] model [11, 12] consists of two types of components: *transducers* and *channels*. Transducers read one or more messages during a time interval and map them to one (or more) output message(s). They are the functional units of the system that may have a memory and accept messages selectively. They provide for temporal aggregation of multiple input messages to a single output message. A transducer may attach to multiple channels; they, in turn, read a single message at a point in time and map them to one or more messages which are delivered to transducers (conceptually, without delay). Channels can be realized within a middleware solution and may be seen as the "cutting points" for distributing a. They have to accept every message and provide for spatial distribution of a single event to multiple transducers. Channels accept (and deliver) messages of a certain type t, transducers map messages from a type t_1 to a type t_2. A system is defined by a set of channels and a set of transducers connecting to channels. There are two types of channels: event-channels, on which messages are posted without expecting any reply, and RPC-channels, on which posted messages will be replied with an appropriate response. Events and RPCs are (in general) posted by transducers to channels without specific addressing information: in a dynamic system, a sender never can be sure which receivers are currently able to process a message. It is up to the channel to identify a suitable (set of) receiver(s) as the result of its arbitration. How this process of choosing the receiver set works is defined by the channel's *strategy*. The channel strategy must ensure that message routing is transparent to transducers, no matter if the message contains an event, a call or a reply. This has the following effects: *a*) a coarse-grained self-organization is achieved based on data-flow partitioning through the definition of several channels to which transducers may attach, *b*) a fine-grained self-organization for functionally similar transducers is achieved by channel strategy based on a kind of "pattern matching" approach, *c*) transducer designers are relieved from the burden of dealing with the distribution of functionality and from the task of finding and selecting the appropriate receiver set, and *d*) the risk of misbehaved transducers that, e.g., always select the receivers from the same vendor is minimized.

The architecture of Amigo [14, 23] is based on a special view on a logical software layer in computing units called the middleware layer that is placed between the platform layer (comprising the operating system and the networking facilities) and the application layer. This way, the middleware is not a single piece of software responsible for ensuring seamless connectivity and interoperability but exists only logically comprising all the Amigo platform services that provide common general-purpose functions. On a more abstract level, Amigo identifies the need for context information to be accessible from all three layers while the application and the middleware layers require privacy and security services. An enriched platform layer may provide some support for the quality of service (QoS) and info on reusable resources and device capabilities. An enriched middleware layer uses the resource

[3] Self-Organizing Data-flow Architectures suPporting Ontology-based problem decomPosition – www.igd.fhg.de/igd-a1/projects/sodapop/sodapop.zip

constraints and device capabilities to provide QoS support and may turn the service discovery into a semantic, QoS-aware and context-aware mechanism with support for service composition. Finally, the service description on the application layer can be enriched by semantic functional specification, syntactic and semantic QoS specification, and a context specification. The logical view on the middleware layer divides it into two main blocks, namely the base middleware and the "intelligent user services". The former has a core part called the "interoperable middleware core" that provides for service discovery and interaction while incorporating QoS parameters. On top of this core, diverse sub-components are suggested, such as mobility management, billing, security & privacy, and even content storage and interoperability. The "intelligent user services" comprise context management & notification services, user modeling & profiling, and user interface services. A device capable of being integrated into an Amigo environment is defined as a device that: *a*) implements (a subset of) the abstract reference architecture employing specific technologies, *b*) provides some Intelligent User Services, and *c*) may implement some interoperability methods for service discovery and interaction.

A comparison of the two solutions shows the following differences:

- The SODAPOP model provides a message brokering solution whereas the Amigo solution is based on object brokering.
- In SODAPOP, the middleware is a piece of software that implements the distributed channels and is responsible for seamless connectivity and interoperability among transducers. This view on the middleware is comparable with the "interoperable middleware core" of Amigo, but the interesting thing in SODAPOP is that all the common general-purpose functions to be provided by the integral components of a concrete AmI system use the middleware the same way like integrable components. This would mean that even such system services do not enjoy any special privilege and may face competition and be substituted by alternative realizations; a characteristic that guarantees a higher level of system openness.
- Amigo uses modern approaches, such as service orientation and ontological description and match-making of services. Although SODAPOP also talks about channel ontologies, but at the time its two realizations were developed, ontological technologies were not mature enough so that those ideas were never realized.
- Although service orientation was chosen as the fundamental principle in Amigo, it provides the application layer with a special broker for context sources with special protocols for supporting event-based communication. Similarly, the interaction between human users and the system is handled specifically, e.g. with special protocols for device selection. This is comparable with the definition of several channels in SODAPOP, each responsible for a certain communication "field". However, the SODAPOP approach was evaluated more positively because of its inherent neutrality in modularization of such communication "fields" that allows for modular specification of ontologies, protocols and strategies.

- The comprehensive set of system services proposed by Amigo can be used as a good basis for deriving the set of PERSONA platform services.

4 The PERSONA Architectural Design

Based on the analysis presented above, PERSONA chose the following strategy for deriving a reference-architecture for AmI systems:

- rely on the bus[4]-based architecture of SODAPOP,
- simultaneously adopt the service-orientation and the ontological approach of Amigo in service discovery, match-making, and composition, and still
- examine the set of integral system services to be considered within the architectural model based on proposals made by Amigo and even other solutions.

In the following subsections, we present the results of this process leading to a proposal for a reference-architecture for AAL spaces.

4.1 The Abstract Physical Architecture

A vision of ambient intelligence is that distributed functionality embedded in appliances, controllers, actuators and sensors should be utilized seamlessly and made available to the human users based on natural interaction paradigms. This leads immediately to the intuitive conclusion that an AAL space is an open distributed system that can be modeled as a dynamic ensemble of networked nodes (see figure 2), where each of the nodes may be such a networking enabled physical component, as a sensor, an appliance / a consumer electronic device, a controller, an actuator, or a full-scaled computing device like a PC. As shown in figure 2, we call the gluing software facilitating the integration of, and the collaboration among the nodes through support for seamless connectivity and enabling interoperability the *middleware* that each node must bind in order to play a role in the ensemble, i.e. utilize and/or offer functionality.

A point to consider here is that at least in case of general-purpose computing devices, such as PCs, a physical node may host several logical components, each with certain functionalities. So, when zooming in on each node, a question is how such logical components may bind the middleware. Figure 3 shows different scenarios in this regard and in regard to wrapping legacy nodes / components[5]. It is worth to mention that in case of node #1 of figure 3, it will act as two separate nodes even if it is a single physical node, because each instance of the middleware represents a distinct "node" in the ensemble.

[4] The term "communication bus" (or shorter "bus") was preferred to the original SODAPOP term "channel".

[5] See section 5 for the PERSONA standard solution for wrapping thin components like sensors.

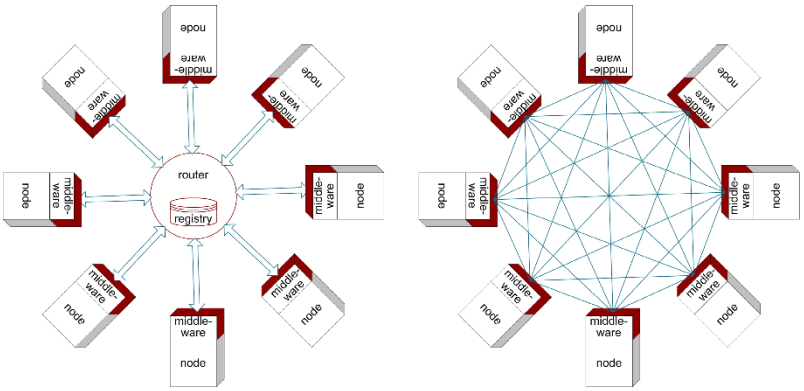

Fig. 2 An ensemble of networked nodes arranged whether with infrastructure (left) or ad-hoc (right)

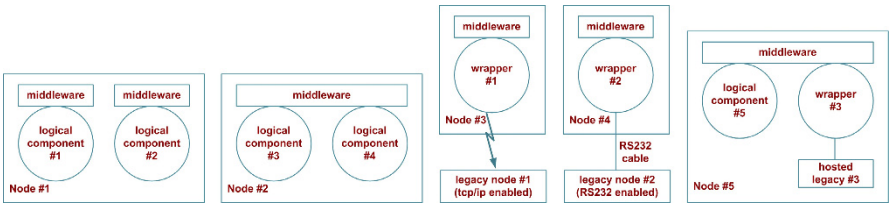

Fig. 3 Different scenarios of hosting logical components and wrapping legacy nodes / components

The project also made the following decisions in the early design stages:

- PERSONA goes for ad-hoc networking based on existing node discovery solutions, such as UPnP and Bluetooth.
- As per definition in [15], middleware is responsible for "hiding the heterogeneity of the various hardware components, operating systems and communication protocols", we decided to rely on the existence of a Java virtual machine providing the needed abstraction.
- In order to facilitate the sharing of the middleware within nodes in such cases as depicted by nodes #2 and #5 in figure 3, the OSGi dynamic module system for Java[6] was chosen.

4.2 The Interoperability Framework

According to the SODAPOP model, for the specification of a system it would be sufficient to *a*) determine the set of its communication buses, *b*) specify the ontology, protocol, and strategy for each bus, and *c*) identify the set of components that

[6] See www.osgi.org.

connect to them. For a reference-architecture, however, the steps 2 and 3 may be taken only partially in order to address a class of systems, which in the PERSONA case would be the class of AmI systems for AAL spaces.

With this understanding of the striven reference architecture, it would be possible to limit the specification in its second step to the specification of an upper ontology shared in the corresponding class of systems and in its third step to the identification of only those components that are shared among all of the instances of the corresponding class of systems as mandatory or recommended components. To be compliant with such a reference-architecture, a concrete system realizing concrete use cases must then employ the shared specifications, enhance the ontology on each bus according to its needs and add components that are needed for the realization of its use cases.

To initiate the derivation of the targeted architecture, we now refer to another guideline from the beginning of this section, namely the adoption of the SOA-based approach of Amigo. The question that arises is: does service-orientation mean that we should have an abstract view on all functionalities as services and consequently define only one bus, say a "service" bus? The answer to this question has been already given in section 3: even Amigo has developed special-purpose brokering frameworks for handling contextual events and user interaction. PERSONA has also decided to abstract all functionality other than those related to the above two topics as "service" and consequently to provide a set of four communication buses, namely the context, input, output, and service buses. Referring to the SODAPOP model that provides an event-based class of buses and a call-based one, we defined the first three buses event-based and the service bus call-based (cf. figure 4).

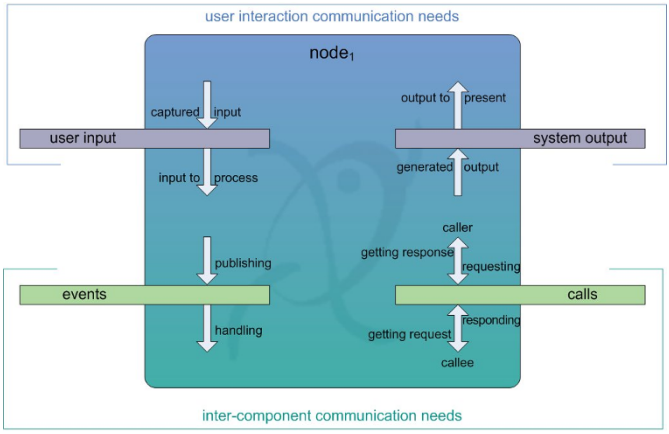

Fig. 4 The PERSONA set of communication buses responsible for brokering four rough types of messages

The rationale behind the type of the context and service buses is self-explanatory. In case of the input and output buses, one may pose the remark about sufficiency

of one call-based bus instead of two event-based buses. It seems that from the view point of the user an input may be handled like calling a system function that may be followed by some system output as the corresponding response. Similarly, system output can be viewed as if the system is calling the user to give his/her answer. Although such a solution is not disqualified from our point of view, but we chose the other alternative due to the following concerns: In addition to a special timeout handling that comes into play when the user is expected to react to a system output, the fact that the communication between the user and the system is handled by components that bridge the gap between the virtual world of the system and the real world of the user have to be considered. Hence, on both sides of the I/O buses there are virtual components that in case of the input bus are responsible for either capturing user input or processing it in the context of a dialog. Similarly, the components attached to the output bus generate the system output on one side, and present it to the user on the other side. Generally, it seems that only components that process the user input and / or generate system output in the context of a dialog are able to decide about the direction of the communication, i.e. if the system is being called by the user or the system is calling the user. Additionally, the strategy for dispatching a captured user input posted to the input bus to input processing components interested in that input can be designed in a specific way that differs from the output bus strategy. In the next section we will get back to this issue again.

Another discussion point is that if both the input bus and the context bus are event-based buses that capture some event in the real world, either by sensing or by capturing explicit user intervention, why not merge them into one bus as it was the case in DynAMITE [12]. Also here we have a preference for separating the two buses in order to have a more modular approach in the development of the bus ontologies, protocols and strategies. Factual concerns for doing so are twofold: explicit user interaction in AmI environments necessitates a special treatment because it happens in an openly distributed I/O handling environment with possibility of merging several modalities while following the user, e.g. from one room to the next one. Additionally, user input in contrast to sensory data may follow in the context of a dialog.

Adding the four identified buses to the simplified physical view of a PERSONA system from the previous section results in a scene depicted by figure 5. Components must be linked with the middleware that realizes the four buses – on the left side, the example of a node in the ensemble hosting three components that share the same instance of the middleware connecting to different buses with different roles – and instances of the middleware can find each other and collaborate, resulting in virtually connected communication buses over all of the participating nodes and components. That is, through the cooperation of different instances of the middleware, local pieces of the same bus will find each other and so will be able to cooperate with each other based on strategies specific to each bus. This way, the middleware can hide the physical distribution of functionality within the ensemble.

This means that the interoperability framework in PERSONA can be summarized as shown in figure 6. The communication buses reflect the loose connections needed in a dynamic environment and represent, in a modular way, the need for interface /

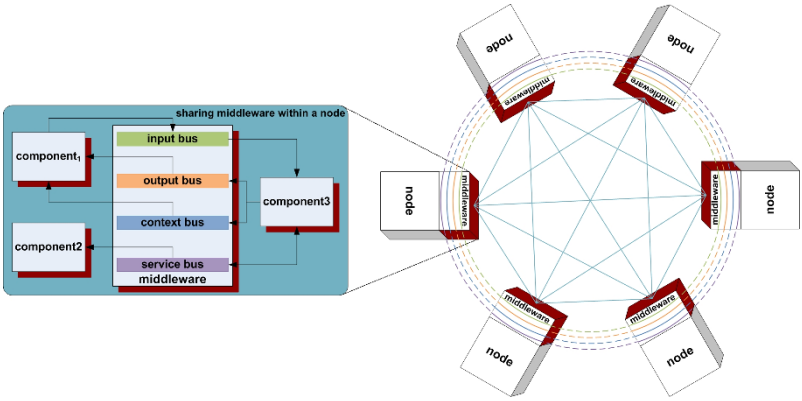

Fig. 5 The distributed communication buses of PERSONA and the formation of virtually global buses

ontology definitions, protocol specifications for communication, and strategies for "dispatching incoming messages" to an appropriate (set of) receiver(s). A component simply registers to some buses by specifying the role(s) it is going to play on each of them. Possible roles for a registering component on an event bus are *publisher* and *subscriber*; possible roles on the service bus are *caller* (or service client) and *callee* (or service provider). Based on this terminology, figure 6 furthermore identifies groups of similar functional roles in such an AmI system and their dependencies, *without specifying any concrete component*. It guarantees the coherence of the system by modeling the basic data flow in AmI systems: The explicit user input through *input publishers* and the contextual / situational events posted by *context publishers* trigger dialogs that will finally terminate with explicit system output through *output subscribers* and / or changes in the environment performed by appropriate *service providers*. The *dialog handlers* (formally equivalent to simultaneously playing some of the roles of input and context subscriber, output publisher, and service client) are therefore responsible for the behavior of the whole system. The horizontal arrows attached to the vertical buses symbolize the availability of services and context in the whole system, independent from any layering. For example, the input and output layers may access the s-bus for utilizing transformation services, such as automatic speech recognition or text-to-speech, and the dialog handling layer accesses the s-bus, mainly for procurement of application services to the user.

4.3 The PERSONA Platform Services

A reference-architecture may specify a set of mandatory / recommended components shared in the whole class of systems complying with that architecture. As discussed earlier, the set of the PERSONA platform services is determined by not only

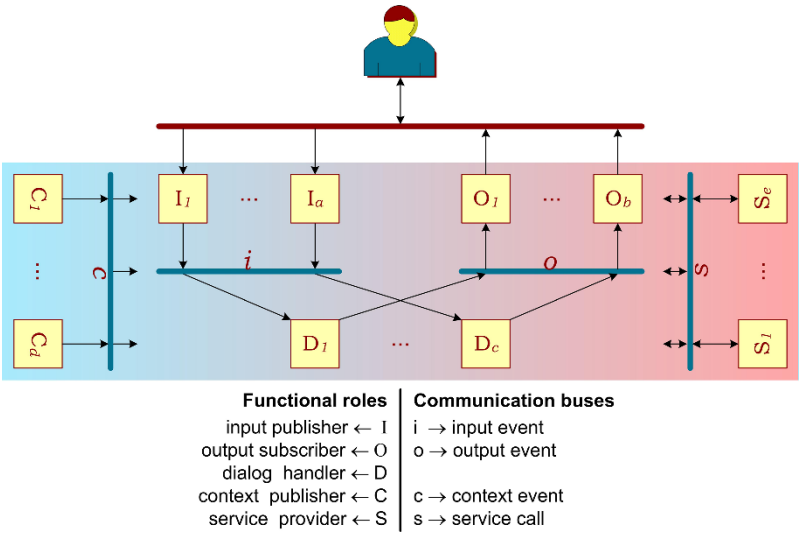

Fig. 6 The abstract interoperability framework in PERSONA

relying on the specific use cases from the PERSONA scenarios but also checking such results against the platform services in other solutions, like Amigo, ASK-IT, EMBASSI, and RUNES. However, it showed very quickly that this should not be set equal to the set of all functions that are shared somehow but the number of the related components must be kept manageable towards the definition of a basic configuration for AAL spaces. Hence, even borderline cases such as contact lists and personal calendars whose services are widely shared at least in private AAL spaces are considered as pluggable components in PERSONA. The criteria for considering a component as an integral part of all AAL spaces were therefore twofold: *a*) the component does something towards producing aggregated added value[7], and / or *b*) it plays a complementary role for the function of the middleware. This way, we came up with the following set of components:

- First of all, an application-independent component is needed that handles the system-wide dialogs and hides the complexity of utilizing the application services from the user (e.g. hierarchical presentation of the services available to the user, supporting some sort of search for services, or handling login dialogs). It should be obvious that this component has to be a dialog handler (in the sense of the functional roles discussed in the previous section); therefore it is called the Dialog Manager. Another important task for the Dialog Manager could be the provision of a mechanism for associating service calls with situations as means for providing a configurable management of the reactivity of an AmI

[7] In a modular and dynamically evolving system, it is essential to be able to utilize the whole potential of the system on a meta-level that does not necessitate any sort of re-compile, re-build, re-deploy, or restart of any other component when new components are added to or old ones are removed from the system.

environment. For this purpose, the Dialog Manager may rely on a configurable repository of rules schematically in the form of "situation → action" (abbreviated as "$s[i] \rightarrow a[j]$"). 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 "$s[i] \rightarrow a[j]$" is the heart of controlling system behavior and hence it will be very advantageous to store it in a central configurable repository. An enhanced version of the Dialog Manager can additionally *a*) initiate dialogs for pro-actively suggesting services that seem to fit to the current situation and *b*) offer services to other components for handling common dialogs with the user, such as ok-cancel, warning and notifying.

- The Context History Entrepôt[8] (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[9] that perform statistical analysis and need context stored over time[10]. 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; the policies consider a time-based threshold as well as the abstraction level of the data. For making the context history available to context consumers, the CHE registers to the service bus as a callee supporting some kinds of context queries, such as full-fledged SPARQL[11] queries. The latter is possible just because the CHE relies on such a repository containing data from all possible sources.
- A general-purpose context reasoner called the Situation 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 – as they are not meant as a one-time query that are answered and then forgotten, but they must generate related situational events whenever the situation changes – 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

[8] For a more detailed discussion on the PERSONA framework for supporting context-awareness please refer to [8].

[9] Context reasoners estimate the state of some context elements by combining different known information and applying certain methods of aggregation, statistical analysis and / or logical deduction.

[10] Many of reasoners that predict context need even long term histories that pluggable context providers may not be able to maintain, as discussed in [17]

[11] SPARQL Protocol and RDF (Resource Description Framework, www.w3.org/RDF) Query Language specified as part of the Semantic Web technologies in the context of OWL (Web Ontology Language, www.w3.org/2004/OWL/), www.w3.org/2001/sw/DataAccess/.

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. The SR takes its name from a modeling theory for situations introduced in [22].

- 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 "$s[i] \rightarrow a[j]$" associations, and *c*) the database of the SO regarding composite services. This demonstrates the total power of the PERSONA system regarding the generality, configurability, and adaptability of the solution and also admits the necessity of such service providers in the business model of PERSONA that are specialists in installation, configuration, and maintenance of PERSONA-based AAL-spaces.
- In order to guarantee the adaptability of an AAL space to the wishes and preferences of its users, it is essential that a special-purpose component is foreseen for the management of the profiles and the provision of needed shared mechanisms. Almost all of the projects from the STAR analysis suggest such a component as a mandatory one. Also the analysis of the realization of PERSONA use cases confirmed this general assumption. We call this component the Profiling Component.
- 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 middleware must verify the legitimacy of components that register to its buses by consulting the PISM. Likewise, trusted services serving anonymous entities must use PISM services in order to decide if the data to be returned can be disclosed. Hence, 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.
- 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 AAL-Space Gateway. The gateway provides access to the hosted services in the AAL space under a fixed URL. For this purpose, it must act within 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. For each of the modalities supported by the gateway, it may register a different input publisher to the input bus (resp. a different output subscriber to the output bus). So, the gateway can make an SMS notification functionality available to the output bus. If an output event is triggered on the output bus that is addressed to an absent user, then the SMS output of the gateway may be selected for presenting the output to the user as an SMS. The phone I/O of the gateway can be utilized by the AAL space, if the absent user is required to interact with the system rather than just being notified. This way, a voice-based session will be initiated actively by the AAL space. Finally, external entities that would like to make advertisement for their services may use an interface of the gateway to be provided as a Web Service for this purpose. Then there must be some authorization mechanism upon which the gateway can decide to ignore or forward the notification to the user; if the authorization process fails, the message will be ignored. Otherwise the gateway calls an appropriate service of the Dialog Manager for notifying the user and hence must register to the service bus as a service caller, too.

Figure 7 incorporates the above components into the logical interoperability framework of PERSONA summarizing the discussions so far. To keep the figure well-arranged, the pluggable components are shown as separate context publishers, special-purpose reasoners, or service clients / providers although a component may play several of these roles simultaneously. Also, the possibility for them to play the role of a dialog handler is not depicted in this figure at all. Additionally, the PISM is omitted from the scene due to the difficulty of showing its double role as a sub-component of the middleware and as a service provider.

4.4 The Middleware

The middleware is composed of a set of OSGi bundles organized in three logical layers (cf. figure 8):

- The lowest layer, the abstract connection layer (ACL), is responsible for the peer-to-peer connectivity between instances of the middleware. Different discovery and message transfer protocols, such as UPnP, R-OSGi or Bluetooth, have been used to provide competing solutions that realize an exported interface called P2PConnector. Listeners can register to such connectors for discov-

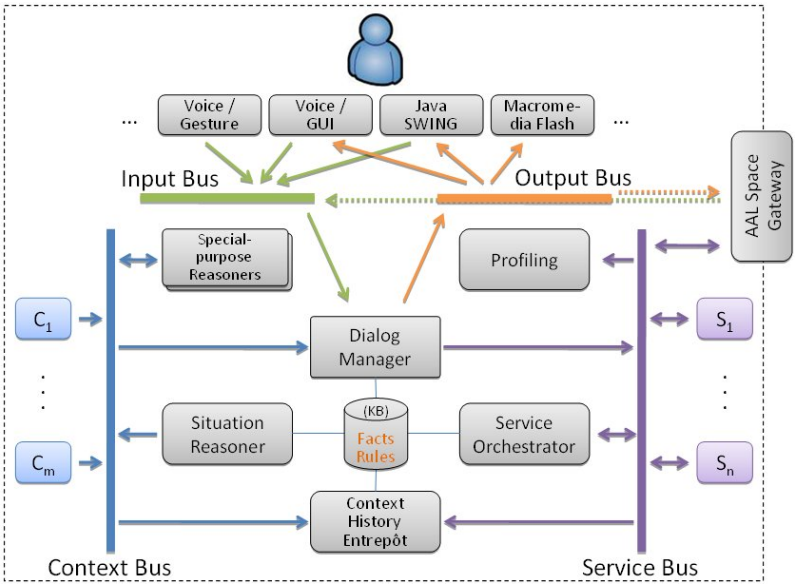

Fig. 7 The PERSONA reference-architecture for AAL spaces

ering *peer*s with a predefined interface. A bridging solution over all available protocols guarantees the overall coherence of nodes within an ensemble.

- The Sodapop Layer implements the peer and listener interfaces from ACL and registers as the local peer to all connectors found. It introduces the concepts of *bus* (either event-based or call-based), *bus strategy* and *message* along with an interface for message serialization.
- The PERSONA-specific layer implements the input, output, context, and service buses with their distributed strategies according to the Sodapop protocol, using an RDF serializer for the exchange of messages among peers.

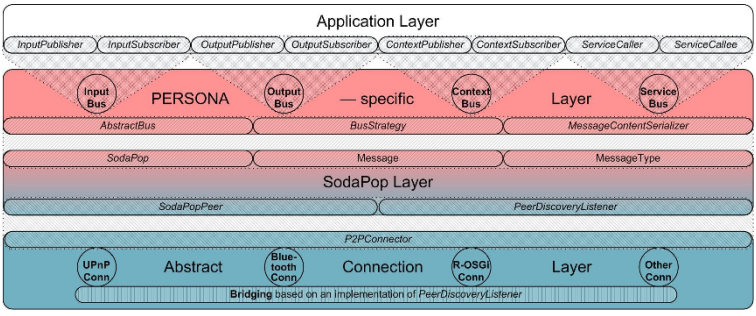

Fig. 8 The internal architecture of the PERSONA middleware

The context bus provides a lightweight communication channel which can be used for publishing context events to local and remote end-points. Context publishers must find the OSGi service realizing the context bus and register to it (this is readily done within a corresponding abstract class), in order to be able to publish context events. Context subscribers must additionally specify a filter for context events they are interested in as their registration parameters. The context bus strategy is as simple as broadcasting a received event to all peers, then each peer does a local match-making with limited ontological inference capabilities for finding interested subscribers. Therefore, the registration parameters of the subscribers are stored only locally by each instance of the context bus. Alternatively, one single node could play the role of a permanent coordinator that gathers all registration parameters from all subscribers to all peers, where a peer that has received an event from a locally registered context publisher, would forward the event only to the coordinator which in turn would forward the message only to those peers that had at least one local subscriber interested in that event. Of course, other strategies are also possible, but in the first version we chose the broadcasting strategy to avoid a single point of failure, accepting the relatively increased messaging traffic.

On the service bus, callees provide service profiles in OWL-S[12] at the registration time. The callers request services using "service queries". The service bus finds the appropriate callee(s) by examining its repository of service profiles using the same limited ontological inference as in case of the context bus, calls it (them) by providing the required input extracted from the original query, gets the output and prepares it as the response to be returned to the caller. The details of the service bus strategy, however, go beyond the scope of this paper. Interested readers are invited to read the project deliverable D3.1.1 at www.aal-persona.org.

In order to have a clear separation between application logic and presentation layers of the system, PERSONA distinguishes between handling dialogs and handling I/O:

- A Dialog handler is a part of applications that expresses the application output using a device-, modality- and layout-independent language (e.g., XForms that has been found as a promising means in our experiments so far) and publishes it to the output bus together with adaptation parameters fetched from the profiling component. It then waits for the user input on the input bus expecting it at the same abstraction level and in relation to the output previously published to the output bus. The Dialog Manager is a special dialog handler provided by the PERSONA platform.
- 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 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 ap-

[12] An OWL-based ontology for describing services. See www.daml.org/services/owl-s/.

plication 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
  - A loose connection using contextual events on the context bus
- Context-free user input is handled by the Dialog Manager 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

4.5 The Ontological Modeling

Apart from the abstract classes InputPublisher, InputSubscriber, ..., and ServiceCallee shown in figure 8, the interfaces facilitating interoperability in PERSONA are specified in terms of ontological concepts and their counterparts in the Java world. This has been possible, because the APIs are defined in a way that the actual information is exchanged as the content of SODAPOP messages.

In order to overcome the complexity of an open distributed system and still enable extensible interoperability, PERSONA has adopted / provided three elementary tools: *a*) the knowledge representation technologies of the Semantic Web consisting of RDF and OWL, *b*) an upper ontology with appropriate programming support consisting of those concepts that all users of the middleware must know, and *c*) a general conceptual solution described in section 5 with certain shared tools for integrating thin devices and embedded sensors and transforming the tapped data into an appropriate ontological representation. Using this framework, two end points that share the same ontological concepts can achieve the needed level of interoperability without the need for the middleware to know those concrete concepts. Still, the middleware is able to adopt ontological reasoning, to some extent, in its brokerage function.

Figure 9 summarizes the Java class hierarchy used by the middleware for handling ontological resources. The class ManagedIndividual is the root of all ontology classes that register to the middleware. This way, each instance of the middleware will have a repository of ontological classes that are relevant for the local members of its buses. The repository provides a mapping between class URIs and their Java representation and enables the middleware to infer hierarchical relationships between the registered classes and check class memberships at Java level. This reveals

the previously mentioned limitations of the ontological reasoning within the middleware: the limited knowledge stored about the underlying ontologies and match-making to the extent supported by Java. The reason for this way of realizing the first version of the middleware is to be spare of the resources needed by the middleware, in order to reach the widest portability by keeping it as small as possible, even at the level of Java 1.3. As a trade-off for these limitations, the middleware supports not only the whole capacity of the OWL class expressions but also enhances it by supporting more specific restrictions that can be posed on the properties of classes. The OWL class expressions are used by subscribers to event-based buses as registration parameters for specifying their interests in certain classes of individuals. Havng received a concrete event, an event-based bus checks if it matches[13] any of the registered class expressions; if yes, the corresponding subscriber will be notified. Also service callers specify their requests in terms of such class expressions, which is used by the service bus in the course of match-making against registered service profiles.

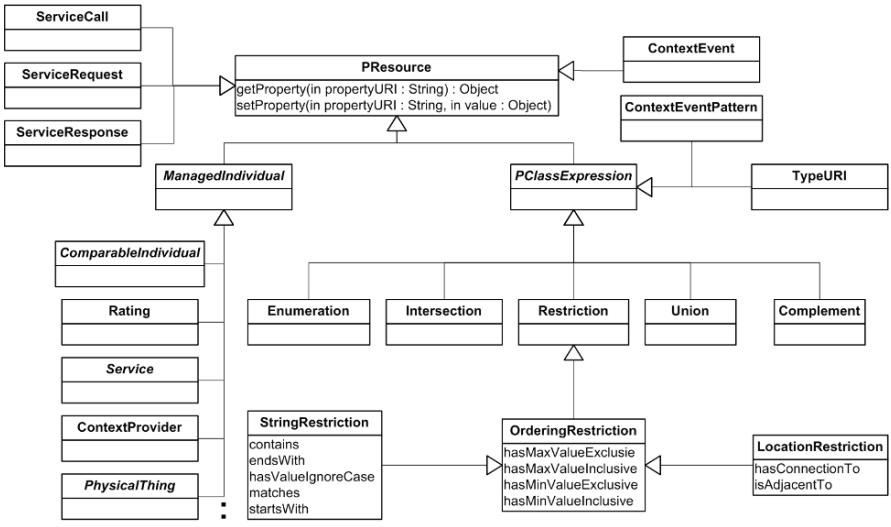

Fig. 9 The Java class hierarchy used and exported by the PERSONA middleware for handling ontological resources

5 Sensing the environment

The artifacts composing AAL-spaces can be classified in *stationary*, *portable* and *wearable components*. The stationary artifacts run on a desktop PC, Set Top Box,

[13] Here comes the ontological inferencing into play.

Residential Gateway or they belong to the environmental infrastructures like Home Automation sub-systems; *portable components* are, for instance, medical devices or mobile/smart phones. Garments equipped with sensors are examples of *wearable components*. The PERSONA middleware targets small but reasonably powerful devices, therefore not all the components can be integrated by using an instance of the PERSONA middleware. Typically, the integration of wearable components as well as nodes of wireless sensor networks (WSN) or Home Automation Systems (HAS) requires a different approach because of their limited computation resources; in such cases a gateway solution can be adopted, which allows the PERSONA application layer to share information by communicating with other application layers resident on different network infrastructures (e.g. ZigBee [4] or Bluetooth applications). Considering the OSGi platform[14], it is sufficient to write a bridging component that interacts with the PERSONA middleware, on one side, and with the software drivers that access the networked devices, on the other side. However, it is worth to notice that many Home Automation Systems have developed their proprietary solution to provide external access to the networked devices. Usually it is possible to find cut-of-the-shelf device servers that are themselves gateways, thus exposing the network through standard protocols like TCP/IP or HTTP (e.g. Siemens EIB-TCP gateway - http://www.eib-home.de/siemens-eib-ip-schnittstelle.htm). In this case an easier integration is achieved through an application proxy that communicates with the remote server rather than with network drivers (see node #3 in figure 3).

In the next section, we focus on the integration of wireless sensor networks, generally adopted for sensing the environment, and we introduce the Sensors Abstraction and Integration Layer (SAIL) developed within PERSONA, by highlighting different aspects and requirements of sensor network applications. A specific section related to the integration of ZigBee devices concludes the chapter.

5.1 Wireless Sensor Networks

Wireless Sensor Networks [4] (WSN) are an important technological support for smart environments and ambient assisted applications. Up to now, most applications are based on ad hoc solutions for WSNs, and efforts to provide uniform and reusable applications are still in their youth. As general requirements for the use of WSN, we can consider the following list:

- R1: *Integration of various sensor network technologies*. AAL-spaces like home environments may be populated by different network technologies typically used in different application domains like Home Automation, Digital Entertainment or Personal Healthcare Systems. As a concrete example we can consider ZigBee or IEEE 802.15.4 standard, and Bluetooth.

[14] initially OSGi was designed as a framework to develop Open Service Gateways

- R2: *Sharing of communication medium by concurrent sensor applications*. Sensor applications involving the same network protocol may imply conflicts at the communication level and waste of resources like energy, bandwidth, and so on.
- R3: *Management of different applications on the same WSN*. The WSN may support different applications at the same time, for example localization of a number of users, posture detection of users, monitoring of energy consumption of appliances, etc.
- R4: *Management of multiple instances of the same application*. Further complications occur when the sensor application is associated with users so that there are as many many instances of the same application as the number of users; in this case, the different instances must be discovered and safely integrated into the system.
- R5: *Dynamic discovery of sensor applications*. Because of the innate dynamism of users' activities, such applications may join and leave the AAL-spaces at any time, according to the user behavior.
- R6: *Management of logical sensors*. The physical sensors deployed either in the environment or worn by the users could be represented (virtualized) by a dissimilar number of logical sensors. For example, let's consider an application that detects the user's posture by means of a number of accelerometers placed on the body: in this case, the posture can be represented by a single logical sensor, which aggregates information produced by all the accelerometers for producing the state of the user (sitting, walking, etc.). In this example, the accelerometers may not even be visible to the application layer.
- R7: *Configuration and calibration of sensor applications*. Many sensor applications need to be configured during the deployment phase, for example to bind the user's metadata to the sensor used to locate him/her.

The requirement R1 is usually satisfied by enabling the dynamic deployment of ad hoc network drivers in the system, while we can assume that requirements from R2 to R5 largely depend on the operating systems and middleware used for programming the sensor nodes (e.g. ZigBee [26], TinyOS [24], TinyDB [16], TeenyLIME [6] and others [18]). Unlike the requirements R1 to R5, addressing the requirement R6 is still an open issue, because, on one side, the sensor nodes should locally pre-process and transmit aggregated data as far as possible in order to reduce the power consumption[15]. On the other side, arranging such processing on the network is not always possible, due to the limited computational power. In [5] a declarative language called WADL (Wireless Application Description Language) is used to describe sensor-based applications which combine the sensed data on the Host PC side. WADL is based on a producer-consumer pattern constructed according to the OSGi WireAdmin specification: both the producers (sensors) and the consumers are modeled as OSGi services and connected at runtime by "wire" objects. WADL is an XML-based language, which defines a dynamic set of wires that connects different

[15] In this respect, it is worth to mention the SPINE (Signal Processing in Node Environment [10]) open source framework that provides simple feature extractors directly on the sensor nodes with TinyOS.

producers and consumers. This approach is strongly characterized by the assumption that the sensors behave according to a producer-consumer pattern. Although this assumption leads to a simple and elegant solution, it may become too limiting when used to abstract WSNs with more complex behaviors.

In a first attempt to provide a uniform model for sensor applications, we designed the SAIL architecture [9] based on three layers, namely the Access, Abstraction, and Integration layers, constructed over the OSGi framework. Network drivers reside on the access layer for handling communication protocols specific to each WSN type. The abstraction layer of SAIL defines a shared sensor model over all WSN types integrated through the access layer. The realization of the abstraction layer was based on the Service Provider Interfaces (SPI) pattern [21] used in different application domains, like database oriented applications (e.g. ODBC, JDBC drivers). Each network driver is hereby considered as a service provider hidden by a manager that abstracts the different network models by providing a common API for the upper layer (the integration layer). The sensor model provided in [9] also took into account data-centric paradigms adopted in many sensor network applications (e.g. query oriented TinyDB [16]).

However Smart Environments are generally indoor and limited; thus also the size of the WSN hardly scales up to hundreds of sensors, and the network often has a small diameter, in some cases it may even be a star. On the contrary, data centric approaches have been designed while keeping in mind very general monitoring applications where the number of sensors can (almost) arbitrarily scale and the sensors are homogeneous in terms of features and monitoring capabilities. For this reason and because of the many hardware developers in PERSONA consortium converging to the ZigBee industry standard, we decided to design an optimized version of SAIL tailored to ZigBee.

5.2 ZigBee Networks Integration

Maintaining the same three-layered architecture introduced in [9], we developed a ZigBee Base Driver for the SAIL access layer that uses native libraries implementing the ZigBee Application Layer. In fact, while Bluetooth drivers are usually integrated with different operating systems and platforms, the standardization process of the network interfaces for ZigBee is still in progress. Therefore, we chose to adopt USB ZigBee Dongles available on the market which come either with a simple AT command-like interface or a more elaborated API based on a native driver.

The basic operations carried out by the base driver include the following set of functions: *a*) *discovery* functions that notify the connection and disconnection of sensors to/from the WSN, *b*) *description* functions that retrieve the properties associated with the sensor nodes and the relevant service interfaces, *c*) *control* functions for setting the state of the actuators embedded in the sensor nodes, *d*) *access* functions that provide read access to the sensors transducers, and *e*) *eventing* functions that enable the subscription to data updates and that notify changes. The implemen-

tation of these functionalities by using the ZigBee Application Layer is straightforward; the ZigBee standard allows both device and service discovery, as well as description request and reporting commands (eventing). However, the implementation of these functionalities is not always mandatory, which often makes the full integration of the network somewhat complex. For example, in the discovery phase, the implementation of a broadcast announcement of the devices that join or leave the network is optional.

For describing devices, ZigBee does not employ the descriptive approach of protocols like UPnP; rather, a device responds to a description request by replying with a reference (ClusterID) to the implemented interfaces. It implies that the consumer of a service must know to which message structure (cluster format) the clusterID refers. The possibility to answer with additional description in CompactXML does exist (e.g. <deviceUrl>), but even this feature is optional.

However, the ZigBee Alliance, the standardization body defining ZigBee, publishes application profiles that allow multiple OEM[16] vendors to create interoperable products. The current list of application profiles already published or still in progress, consists of Home Automation, ZigBee Smart Energy, Telecommunication Applications, and Personal Home and Hospital Care. In general, these profiles use a combination of clusters defined in the ZigBee Cluster Library. Vendors can define their custom library for devices not defined by the ZigBee Alliance. According to this, the PERSONA project is now working on the definition of a PERSONA Cluster Library and Profile, in order to integrate the personal healthcare system and special devices like the smart glove or the smart carpet that will be developed by the consortium partners.

A diagram of the current SAIL architecture is depicted in Figure 10. The ZigBee Base Driver is a network driver in charge of scanning the network, getting the description of the various nodes, and registering a proxy service for accessing the discovered remote services. This proxy is a generic ZigBee service, registered with the OSGi Platform, which exposes the properties retrieved during the network inquiry. It allows to access the remote service by means of simple primitives that have to be filled by appropriate clusters. Thus, in contrast with the more generic model used by the previous version of SAIL, we have defined a specialized model tailored on an extension of ZigBee profiles, namely the PERSONA profiles. The components on the upper layers may act as Refinement Drivers (in OSGi terms). By using the OSGi Framework facilities, as soon as a service implementing a known cluster is registered, these drivers refine the service by registering another service proxy with a more detailed interface (e.g. action/command based). Thus the second layer is specialized to represent the service according to a specific profile, for instance the Home Automation profile. The upper layer is the final step for integrating the ZigBee services within PERSONA. It is composed of Sensor Technology Exporters (STE), which discover the services by implementing standard or extended profiles

[16] From Wikipedia: An original equipment manufacturer, or OEM is typically a company that uses a component made by a second company in its own product, or sells the product of the second company under its own brand.

and register proxies that are PERSONA-aware components [17]. The mapping between services compliant to the ZigBee model and PERSONA OSGi services is realized at this level; these proxies send events or register service callees according to the PERSONA ontologies.

In conclusion, the abstract layer is populated by custom drivers which may combine and process the sensed data to instantiate logical sensor services (see R6; e.g. a sensor providing the user's position by elaborating RSSI measurements - Received Signal Strength Indication - coming from different stationary sensors), as well as refine the cluster-based services. Of course, such an elaboration could be realized also at PERSONA application level, but, from a practical point of view, the realization at lower layers is expected to be more efficient. In fact, we even plan to implement the user interfaces addressing the last requirement introduced in the previous section (R7) at this layer, in order to provide a uniform way for configuring sensor applications.

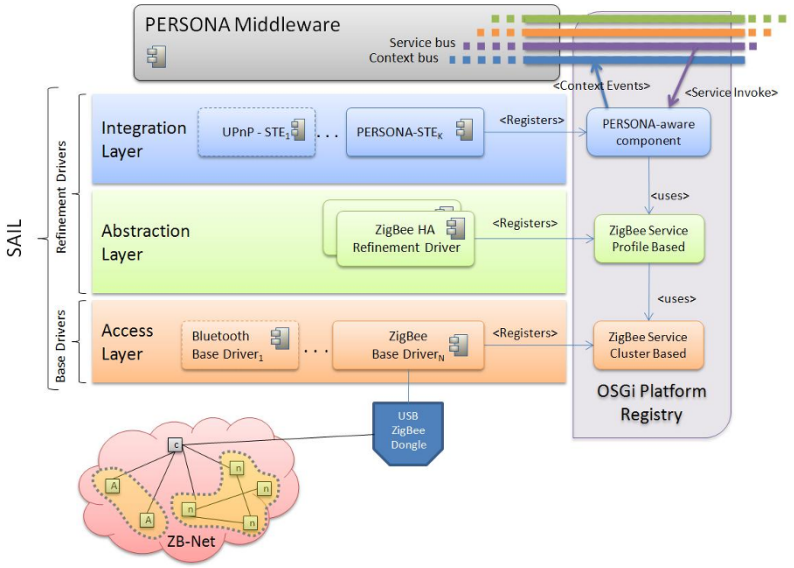

Fig. 10 The SAIL layered architecture

[17] The same approach can be used to expose the services according other access technologies (e.g. UPnP)

6 Conclusions

The AAL service platform developed within the PERSONA project and introduced in this chapter is expected to facilitate the gradual evolvement of AAL spaces based on a very compact core. To reach this goal, AAL spaces are modeled in PERSONA as Open Distributed Systems (see [8] for our undersanding of ODS). The resulted platform allows for self-organization of both physical nodes and logical components by providing a middleware that supports seamless connectivity and semantic interoperability. For providing aggregated added value, the solution relies on administrative re-configurability of platform components, such as the Situation Reasoner, the Dialog Manager, and the Service Orchestrator. On the basis of the achieved re-configurability, future research can work in parallel on both simplifying the configuration task towards end user programming and developing intelligent algorithms towards automatic re-configuration.

The next steps in the project plan foresee a thorough evaluation of the technological results during and after the deployment of the system to the PERSONA three pilot sites in Denmark, Italy, and Spain. We have already started to define the evaluation criteria, methodology, procedures, and tools. The rising awareness of the importance and difficulties of evaluating AmI systems in the research community (see, for example, the summary provided in [20]) shows that the evaluation phase will be one of the most challenging tasks of the project in the near future.

Further research planned by the authors of this chapter relates to breaking out of the boundaries of the home environment towards societal systems, as suggested in [19]. The more suspenseful topics with relation to the PERSONA approach are the ad-hoc formation of temporary spaces and related security concerns, rapid configuration changes in public spaces, and interoperability among different spaces.

References

[1] AAL: The Ambient Assisted Living Joint Programme – www.aal-europe.eu (2007)

[2] ARTEMIS SRA Working Group: The ARTEMIS Strategic Research Agenda. Research Priorities, IST Advanced Research and Technology for Embedded Intelligence in Systems, www.artemis-office.org/DotNetNuke/SRA/tabid/60/Default.aspx (2006)

[3] Avatangelou, E., Dommarco, R.F., Klein, M., Müller, S., Nielsen, C.F., Soriano, M.P.S., Schmidt, A., Tazari, M.R., Wichert, R.: Conjoint PERSONA – SOPRANO Workshop. In: Constructing Ambient Intelligence – AmI 2007 Workshops, pp. 448–464. Springer CCIS Series, Darmstadt, Germany (2007)

[4] Baronti, P., Pillai, P., Chook, V.W.C., Chessa, S., Gotta, A., Hu, Y.F.: Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards. Computer Communications **30**(7), 1655–1695 (2007)

[5] Cervantes, H., Donsez, D., Touseau, L.: An Architecture Description Language for Dynamic Sensor-Based Applications. In: Proceedings of Consumer Communications and Networking Conference (CCNC 2008), pp. 147–151. IEEE, Las Vegas (Nevada USA) (2008)

[6] Costa, P., Mottola, L., Murphy, A.L., Picco, G.P.: Programming Wireless Sensor Networks with the TeenyLIME Middleware. In: Proceedings of the 8th ACM/IFIP/USENIX International Middleware Conference (Middleware 2007). Newport Beach, (CA, USA) (2007)

[7] Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., Burgelman, J.C.: Scenarios for Ambient Intelligence in 2010. ISTAG Report, European Commission – IST Advisory Group, ftp.cordis.europa.eu/pub/ist/docs/istagscenarios2010.pdf (2001)

[8] Fides-Valero, A., Freddi, M., Furfari, F., Tazari, M.R.: The PERSONA Framework for Supporting Context-Awareness in Open Distributed Systems. In: To appear in the proceedings of the European conference on Ambient Intelligence – AmI-08. Nuremberg, Germany (2008)

[9] Girolami, M., Lenzi, S., Furfari, F., Chessa, S.: SAIL: a Sensor Abstraction and Integration Layer for Context Aware Architectures. In: Proceedings of the 34th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2008), pp. 374–381. IEEE, Parma, Italy (2008)

[10] Gravina, R., Guerrieri, A., et Al., S.I.: SPINE (Signal Processing in Node Environment) framework for healthcare monitoring applications in Body Sensor Networks. White paper, TILAB, spine.tilab.com/papers/2008/DemoSPINE.pdf (2008)

[11] Heider, T., Kirste, T.: Architecture considerations for interoperable multimodal assistant systems. In: Proceedings of the 9th International Workshop on Design, Specification, and Verification of Interactive Systems, pp. 403–417. Rostock, Germany (2002)

[12] Hellenschmidt, M., Kirste, T.: SODAPOP: A Software Infrastructure Supporting Self-Organization in Intelligent Environments. In: Proceedings of INDIN '04: the 2nd IEEE International Conference on Industrial Informatics, pp. 479–486. IEEE, Berlin, Germany (2004)

[13] Ambient Intelligence: from vision to reality. ISTAG Report, European Commission – IST Advisory Group, cordis.europa.eu/ist/istag-reports.htm (2003)

[14] Janse, M., Vink, P., Georgantas, N.: Amigo Architecture: Service Oriented Architecture for Intelligent Future In-Home Networks. In: Constructing Ambient Intelligence – AmI 2007 Workshops, pp. 371–378. Springer CCIS Series, Darmstadt, Germany (2007)

[15] Krakowiak, S.: Middleware Architecture with Patterns and Frameworks. INRIA Rhône-Alpes, France, sardes.inrialpes.fr/~krakowia/MW-Book/ (2007)

[16] Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: TinyDB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst. **30**(1), 122–173 (2005). DOI http://dx.doi.org/10.1145/1061318.1061322

[17] Mayrhofer, R.: Context Prediction based on Context Histories: Expected Benefits, Issues and Current State-of-the-Art. In: Proceedings of ECHISE '05: 1st International Workshop on Exploiting Context Histories in Smart Environments. Munich, Germany (2005)
[18] Molla M.M.; Ahamed, S.: A survey of middleware for sensor network and challenges. In: Proceedings of nternational Conference on Parallel Processing Workshop (ICPP 2006) (2006)
[19] Nakashima, H.: Cyber Assist Project for Ambient Intelligence. In: J.C. Augusto, D. Shapiro (eds.) Advances in Ambient Intelligence, pp. 1–20. IOS Press (2007)
[20] Neely, S., Stevenson, G., Kray, C., Mulder, I., Connelly, K., Siek, K.A.: Evaluating pervasive and ubiquitous systems. In: J. Hong (ed) The "Conferences" column of PERVASIVE computing **7**(3), 85–89. IEEE CS (2008)
[21] Seacord, R.C.: Replaceable Components and the Service Provider Interface. In: ICCBSS '02: Proceedings of the First International Conference on COTS-Based Software Systems, pp. 222–233. Springer-Verlag, London, UK (2002)
[22] Tazari, M.R., Grimm, M.: D11 – Report on Context-Awareness and Knowledge Representation. Public deliverable, MUMMY (IST-2001-37365), www.mummy-project.org/downloads.html (2004)
[23] Thomson, G., Sacchetti, D., Bromberg, Y.D., Parra, J., Georgantas, N., Issarny, V.: Amigo Interoperability Framework: Dynamically Integrating Heterogeneous Devices and Services. In: Constructing Ambient Intelligence – AmI 2007 Workshops, pp. 421–425. Springer CCIS Series, Darmstadt, Germany (2007)
[24] TinyOS Project: `www.tinyos.net` (2008)
[25] Turunen, M.: Jaspis – A Spoken Dialogue Architecture and its Applications. Ph.D. thesis, University of Tampere, Department of Computer Sciences, Finnland (2004)
[26] ZigBee Alliance: ZigBee specifications – `www.zigbee.org` (2006)