Grant Agreement number: 288899

Project acronym: Robot-Era

Project title: Implementation and integration of advanced Robotic systems and intelligent Environments in real scenarios for ageing population

Funding scheme: Large-scale integrating project (IP)

Call identifier: FP7-ICT-2011.7

Challenge: 5 – ICT for Health, Ageing Well, Inclusion and Governance

Objective: ICT-2011.5.4 ICT for Ageing and Wellbeing

Project website address: www.robot-era.eu

D7.1
Report on the interoperability aspects of Robot-Era services

Due date of deliverable: 31/05/2012
Actual submission date: 27/06/2012

Start date of project: 01/01/2012 Duration: 48 months

Organisation name of lead contractor for this deliverable: SSSA
Deliverable author: Massimo Filippi, Alessandro Saffiotti, Filippo Cavallo

Version: 2.0

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Service)	
RE	Restricted to a group specified by the consortium (including the Commission Service)	
CO	Confidential, only for members of the consortium (including the Commission Service)	**X**

Document History

Version	Date	Author	Summary of Main Changes
1.0	23-02-2012	Filippo Cavallo (SSSA)	First version of the template for Robot-Era Deliverables
1.1	22-05-2012	Massimo Filippi (SSSA)	Contribution on middleware and standards
1.2	24-05-2012	Filippo Cavallo (SSSA)	Structure of the document
1.3	03-06-2012	Alessandro Saffiotti (ORU)	Created overall system architecture and abstract robot architecture; wrote Sec 4, 5, 6; comments to Sec 3.
1.4	22-06-2012	Massimo Filippi (SSSA)	Integrated contributions in Sec 3; corrections in the document.
1.5	27-06-2012	Alessandro Saffiotti (ORU)	Final check on Sec 3,4,5,6; minor corrections.
1.6	**27-06-2012**	**Massimo Filippi (SSSA)**	**Final submitted version**
2.0	26-11-2012	Filippo Cavallo (SSSA)	Project Manager minor refinements

Table of Contents

Executive summary

This work introduces the interoperability aspects of Robot-Era services. The term "interoperability" indicates the ability of independent systems, such as robotic systems, to exchange meaningful information and initiate actions from each other, in order to operate together to mutual benefit.
Since in the Robot-Era project different independently developed devices coming from multiple suppliers and providers are installed in different environments (public and private spaces, living labs and residential sites), it's crucial to analyse and define all interoperability aspects that will favour an easy, scalable, and flexible integration of Robot-Era platforms and services in all experimental settings.

Many different middlewares have been proposed for robotic networks applications, often from a different point of view and with different features and applications. The whole Robot-Era system has demanding requirements in terms of overall software organization and interoperability. The system must be able to accommodate devices which are highly heterogeneous.

The Robot-Era consortium has decided to rely on a two-layer reference architecture: at the upper layer ("*Ecology Layer*") a common network-oriented middleware provides seamless integration among heterogeneous devices, including the Robot-Era robots, while addressing all the above requirements; at the lower layer ("*Device Layer*") each robot and device may use a specific architecture and middleware internally, while externally it presents a uniform interface toward the ecology layer.

The Robot-Era partners considered that the PEIS-Middleware satisfies all requirements, it was therefore chosen by partners to be adopted as the Robotic Ecology middleware in the Robot-Era project, to provide interoperability among all the robots and devices participating in the system. The PEIS-Middleware is available open-source, implements a fully decentralized communication model based on a distributed tuple-space, runs on a wide range of platforms, and its main developer is a partner of the Robot-Era consortium.

The requirements of the Device Layer depend on each specific robot and device, and they are therefore discussed in other Deliverables. Robot-Era includes three types of robotic platforms: domestic robots, condominium robots, and outdoor robots. Although these platforms will differ both in hardware and software, the Robot-Era consortium agreed that they should share a similar abstract architecture.

Each robot platform comprises a number of robot components for perception, reasoning, navigation and manipulation. These components are integrated with the help of a robot integration framework, or "robot middleware". In Robot-Era, it has been decided that two such frameworks will be used: the ROS (Robot Operating System), and MIRA (Middleware for Robotic Applications). MIRA is the native middleware implemented in the MetraLab Scitos G5 and G6 robotic platforms, which will be used as basis for the Robot-Era domestic and condominium robots. ROS is assessing itself as the *de-facto* standard in robot middleware, and it will be used for manipulation functionalities.

Not all Robot-Era platform will need to run both the MIRA and ROS robot middleware. If both MIRA and ROS are needed on the same platform, however, an adaptor needs to be implemented to ensure the proper exchange of information and coordination between robot algorithms implemented in the two sides.

The architecture discussed is intentionally abstract. Each concrete Robot-Era robot platform is discussed in Deliverables D4.1, D5.1 and D6.1, respectively.

1. Introduction

Since in the Robot-Era project different independently developed devices coming from multiple suppliers and providers are installed in different environments (public and private spaces, living labs and residential sites), it's crucial to analyse and define all interoperability aspects that will favour an easy, scalable, and flexible integration of Robot-Era platforms and services in all experimental settings.

The term "interoperability" indicates the ability of independent systems, such as robotic systems, to exchange meaningful information and initiate actions from each other, in order to operate together to mutual benefit. In particular, it envisages the ability for loosely-coupled independent systems to be able to collaborate and communicate.

Networked robotic systems consist of a collection of interconnected components, and in the Robot-Era project different robotic platforms work together and with external systems such as Wireless Sensor Networks (WSN), wearable devices, actuators or servers. These components require cooperation and collaboration to achieve a common goal.

However application development for such collaborative distributed systems composed of many robots with sensors, embedded computers, and human users is very difficult.

Therefore, middleware services can provide advanced approaches offering many possibilities and drastically enhancing the development process and the overall functionalities needed for networked robotic systems [1].

Robotics middleware is an abstraction layer that resides between the operating system and software applications (Figure 1). It is designed to manage the heterogeneity of the hardware, improve software application quality, simplify software design, and reduce development costs [2]. In particular, a middleware could be based on a two-layer reference architecture: the upper layer for seamless integration among heterogeneous devices, and the lower layer in which each device may use a specific architecture and middleware internally, while externally it presents a uniform interface toward the upper layer.

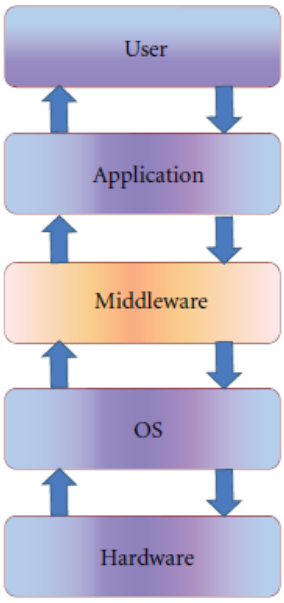

Figure 1. Middleware Layers

Due to their high components heterogeneity and unique characteristics, networked robotic systems in general pose considerable impediment and make the development of networked robot applications non-trivial. Therefore, there must be new software services, middlewares, that act as the glue to link everything together in an efficient manner, supporting concurrency-intensive operations, enhancing collaboration, and insuring efficiency and robustness.

Robotics middleware should be instrumental to make the robot system customizable to different scenarios, applications and environments, and to enable advanced properties like self-configuration, self-adaptation, and self-optimization.

2. Middleware for Robotic Systems

Middleware can play an important role in developing and operating networked robots. As with many other distributed systems, middleware can provide abstractions, hide heterogeneity, facilitate applications development, and provide several value-added functionalities [1][3].

While the older generations of robots were designed to achieve specific tasks and manufactured as one unit, the new generations of robots are diverse and ubiquitous. New robotic applications, such as the Robot-Era system, are composed of multiple robots and other devices (such as ZigBee sensors and actuators) that are connected through wireless networks. These robots and devices are usually controlled by software modules developed by different manufactures using different programming languages. Robots and other devices may use different communication mechanisms. Software modules are also needed to process sensor information and control actuators for performing computational and cognitive tasks like navigation, localization, planning and interaction.

Although utilizing networked robots for some applications have many efficiency and accuracy advantages, it raises some integration issues, first of all the interoperability. These issues could be solved by including a middle layer, that is middleware.

2.1. Roles of Middleware

Generally, middlewares are used in distributed systems to reduce development time and cost. This is achieved by providing well-structured and well-tested services for often-needed functionalities. In addition, it provides some functions that cannot be added to the operating system such as reliability, security, and abstraction.

The design and development of middleware for Robot-Era system needs to deal with many challenges dictated by the robotics platforms and other devices characteristics on one hand, and the scenarios needs on the other.

Middleware for the Robot-Era system, but the same goes generally for robotics middleware, have several roles:

- *Support communication and interoperability*: Robots and other devices are designed and implemented by different manufactures. In Robot-Era scenarios, among robots and other devices (such as WSN, wearable device,…), efficient communication and simple interoperability mechanisms are needed. Therefore, middleware should provide these functions. In addition, multiple robots may be arranged in ad hoc manner in which they cannot directly communicate with each other. In this case, communication support for ad hoc networks is needed. This type of support can be provided by middleware.

- *Provide automatic discovery and self-configuration of new devices*: Networked robots are considered dynamic systems due to the mobility of robots, the different operating environments and their changes. In Robot-Era system, external devices can be dynamically available/unavailable for a robot's use. These external resources can be utilized by robotic platforms to enhance processing power, accuracy or some functionalities, when they are available. Hence, automatic and dynamic resource discovery and configuration is needed.

- *Simplify the development process*: Application development is not easy for networked robotic platforms. The middleware should simplify the development process by providing higher-level abstractions with simplified interfaces (APIs) that can be used by all partner researchers. In addition, the middleware should promote for software integration and reuse.

- *Provide collaborative operations among robotics platforms*: Robot-Era system relies on different robotic platforms to achieve specific tasks, so these robots must be able to work together efficiently. Robotics middleware must provide some functionalities and high-level abstractions to facilitate the development of the collaboration mechanisms. Tools and APIs are necessary to support the development and utilization of specialized collaboration functionalities.

- *Provide heterogeneity abstractions*: Robot-Era system contains heterogeneous hardware and software components, communication and cooperation among these components is an important aspect. The abstraction role is played by middleware which acts as a collaboration software layer among all involved components, hiding the complexity of the low-level communication and the heterogeneity of the components.

- *Provide integration mechanisms with other systems*: Robot-Era robotic platforms need to interact with other systems such as WSN. Most of these interactions should be done in an abstract way and in real-time. Middleware should provide real time interaction services with other systems and devices in different environments.

- *Offer often-needed robot services*: A great deal of effort is spent writing new implementations of existing algorithms and control services for networked robotic applications multiple times. The same algorithms/services may be rewritten several times due to changes in the robotic platform's hardware, the development of new services and applications, changes in the operating systems, changes of technical staff, or just for adding new functionalities. These often-needed robot services should be provided by robotics middleware which allows for reuse of the modules offering these functionalities.

All these fundamental roles of middleware are needed in the Robot-Era system, and they were analysed by partners in order to define the middleware.

2.2. Existing Middlewares for Robotics

Many different middlewares have been proposed for robotic networks applications, often from a different point of view and with different features and applications.

Robotics middleware can be based on standard or nonstandard communication models. Some middleware are based on a standard distributed object model, CORBA (Common Object Request Broker Architecture), allowing interprocess and cross-platform interpretability for distributed robot control. The main motivation of using the distributed object model is to improve the software development process for robotic systems and to enable the interaction among robots and other systems.

Miro [4] is an object-oriented middleware for robots. The main features of this middleware are improving the software development process for mobile robots and enabling the interaction between the robots and enterprise information systems. It is open source, and it has no explicit fault handling capabilities. Miro is designed and implemented by applying object-oriented design and implementation approaches using the common object CORBA standard. This allows inter-process and cross-platform interoperability for distributed robot controls. Miro was implemented using multiplatform libraries for easy portability, both for Windows and Linux. It has high flexibility but no automatic discovery and configuration.

Another middleware based on standard communication model is *RT* (Robot-Technology)-Middleware [5]. The main goal of this middleware is to build robots and their functional parts in a modular structure at the software level and to simplify the process of building robots by simply combining selected modules. Another important goal is to make robots more intelligent by distributing their necessary resources over a network. RT-Middleware provides the necessary services to enable implementing robotic applications that need these types of distributed systems. It6 provides automatic discovery and configuration, and high flexibility in adding new services.

Universal Plug and Play (*UPnP*) was developed to offer peer-to-peer network connectivity among PCs, wireless pervasive devices, and intelligent appliances [6]. The UPnP has automatic discovery and configuration mechanisms. UPnP Robot middleware was developed to configure robot components and to allow ubiquitous robots to discover and interact with other devices like sensor networks, cameras and electromechanical devices. Using UPnP mechanics robots are able to configure their internal components to interact with external devices based on the specific goals or services they should provide. This is an essential feature for the implementation of intelligent robotics. The intelligence component can be internal or external since software components can cooperate with each other regardless of their location. This approach provides a simple scheme for building intelligent robots with a lot of hardware and software components, but it has low flexibility in adding new services.

The *Player/Stage system* provides infrastructure, drivers and some algorithms for mobile robotic applications [7]. This middleware has two major components: Player and Stage. Player is a device repository server for actuators, sensors, and robots. Stage is a graphical simulator that models devices in a user defined environment. The Player/Stage system is implemented as a three-tier architecture in which the first-tier is the clients which are software components developed for specific robot applications, the middle-tier is the Player which provides common interfaces for different robot devices and services, and the third-tier is the actual robots, sensors, and actuators. Player's modular architecture makes it flexible to support new hardware, it provides high flexibility, but it is not designed for automatic discovery and configuration.

MARIE (Mobile and Autonomous Robotics Integration Environment) is a middleware created for developing and integrating new and existing software components for robotic systems [8]. MARIE provides high flexibility in adding new services and devices, it aims to create a flexible distributed components system that allows robotic systems developers to share, reuse, and integrate robotic software programs for rapid robots application development. MARIE middleware provides some services that allow the adaptation of different communication protocols and applications which make it very flexible, but it doesn’t provide automatic discovery and self-configuration.

The middleware of *AWARE* is a data centric middleware for the integration of WSN and mobile robots [9]. The main aim of this middleware is to provide simplified mechanisms for integrating information gathered by various types of sensors including WSN and mobile robots. This type of integration is needed for applications where robots are used to obtain and process data from their environment through a WSN. It provides high flexibility in adding new functionalities, but no automatic discovery and configuration of new devices.

RSCA (Robot Software Communication Architecture) is a middleware for networked service robots [10]. Its main feature is the real-time support. It provides a standard operating environment and development framework for robot applications. The operating environment consists of a Real-Time Operating System, communication middleware, and deployment middleware. The operating system provides an abstraction layer that makes robot applications both portable and reusable on different hardware. The communication middleware is compliant to minimum CORBA and RT-CORBA and provides mechanisms for distributed heterogeneous components to communicate in real-time.

The *PEIS* Middleware is an open source middleware that provides a nonstandard communication model [11]. The PEIS Middleware provides a shared memory model and supports heterogeneous devices. This middleware is designed to implement the concept of Ecology of Physically Embedded Intelligent Systems (PEIS-Ecology), in which many robotic devices, pervasively embedded in real environments, cooperate in performing some tasks in the service of people. In this approach, complex robotic functionalities are not achieved via the implementation of extremely advanced robots, but rather through the cooperation of many simple robotic components. The main aim of the PEIS Middleware is to provide a common communication and cooperation model that can be shared among heterogeneous robotic devices such as mobile complex robots, sensors or actuators, and automated home appliances. With this middleware, any robot device with software controls in the environment is defined as PEIS; each PEIS is a set of inter-connected software components developed to control sensors or actuators. All PEIS can communicate and cooperate each other using a uniform communication and cooperation model. In this model, each participating PEIS can use functionalities from other PEIS in the ecology in order to complement its own. PEIS-Middleware provides automatic discovery of new devices, introspection, and run-time self-configuration capabilities.

Sensory Data Processing Middleware [12] is developed to provide abstracted services for accessing sensor information to support service mobile robots. Two types of services were implemented to provide obstacle information and to localize the robot position using landmark observations from multiple external sensors. This middleware provides a unified model for different configurations of external sensors on a service mobile robot. The unified model abstracted from sensors can be used in any service mobile robot application independent of the sensors configuration. The developed services can be reused in multiple applications without dealing with individual sensors.

A *Layer for Incorporations* among Ubiquitous robots [13] is developed to enable communication among ubiquitous robots which are usually of different types. These types can be software robots, mobile robots, and embedded robots. Software robots are similar to mobile agents while mobile robots are usually hardware robots controlled by software. This middle layer is mainly designed to allow software robots and mobile robots to communicate even when they use different communication mechanisms. The middle layer consists of two mappers: sensor mapper and behavior mapper. The sensor mapper helps software robots get physical sensor information from mobile robots; while the behavior mapper helps software robots make physical behavior using the actuators of the mobile robots.

Orca [14] is an open source middleware framework for developing component-based robotics. It is designed to target applications from single vehicles to distributed sensor networks, and his main aim is to enable software reuse in robotics. Orca enables implementing a distributed component-based robotic system by allowing the user to define

interfaces and communication mechanisms. It was implemented using CORBA. It doesn't provides real-time capabilities, and it has no explicit fault handling capabilities.

The *MIRA* framework, for instance, provides a middleware that allows to compose different modules dynamically at runtime to form the final complex application [15]. This middleware handles the communication between the modules efficiently and transparently. It allows the modules to be freely distributed - no matter if they are located within a single process, in different processes or even on different machines. MIRA is designed to allow fast and easy creation and testing of new distributed software modules. The interface is very lightweight and fully transparent. Mira middleware allows a fast and efficient development of (robotic) applications, and it provides stability and reliability, and the usage of C++ language-constructs only, without the need of a meta-language or meta-compilers. For communication the MIRA framework offers message passing by implementing the publisher/subscriber pattern as well as Remote Procedure Calls (RPC).

ROS provides the operating system's services such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management [16]. ROS is an open source metaoperating system for robot software consisting of many small tools designed to work together. It supports distributed environment. The ROS system is a computation graph consisting of a set of nodes communicating with one another over edges It consists of nodes (software modules), messages (passed peer to peer), topics, and services (analogous to web services). Nodes communicate together by passing messages through publish/subscribe model. Messages are not based on a specific programming language. A node sends a message by publishing it to a given topic, which is simply a string. A node that is interested in a certain kind of data will subscribe to the appropriate topic.

LIME (Linda in a Mobile Environment) is an open source model and middleware supporting the development of applications that exhibit physical mobility of hosts, logical mobility of agents, or both [17]. Lime adopts a coordination perspective inspired by work on the Linda model [18], such as the PEIS middleware. The context for computation, represented in Linda by a globally accessible, persistent tuple space, is refined in Lime to transient sharing of identically-named tuple spaces carried by individual mobile units. Tuple spaces are also extended with a notion of location and programs are given the ability to react to specified states. The resulting model provides a minimalist set of abstractions that facilitate rapid and dependable development of mobile applications.

OROCOS (Open Robot Control Software) is an open source middleware based on a standard communication model (CORBA model). The main goal is to develop a general purpose modular framework for robot and machine control [19]. It is a Real-Time Toolkit that provides the components to be able to run on a real-time operating system. It consists of the following libraries: the OROCOS Components Library (OCL) that provides some ready-to-use control components, the OROCOS Kinematics and Dynamics Library (KDL) that provides the real-time calculation of kinematic chains, and the Orocos Bayesian Filtering Library. OROCOS does not support distributed environment. OROCOS was implemented using multiplatform libraries (C++ libraries) for easy portability, both for Windows and Linux. It has high modularity and flexibility but no automatic discovery and configuration.

URBI is a software platform used to develop portable applications for robotics and artificial intelligence. It is based on a parallel and event-driven script language, and on a distributed component architecture [20]. Urbi is based on a standard communication model, and it consists in a non-modular architecture. It consists essentially of software libraries, but not components, that wrap accesses to physical devices; it defines a scripting language to access sensors and actuators. The Urbi platform sits on top of the large variety of software and/or hardware components for robotics, and provides the user with a unified, standardized, interface with which complex and portable applications can be developed.

2.3. Summary table

Table 1 shows a list of main networked robotic middleware platforms for networked robots, with a brief list of their characteristics and technologies/standards used.

Platform	Main Objectives	Comm. Model	Standards/ Technology followed	Service provided	Automatic discovery & configuration	Flexibility of adding new services
Miro	To improve the software development process for mobile robots and enable interaction between robots and enterprise systems using the distributed object paradigm	Standard	CORBA, ACE	Generic	No	High
RT-Middleware	To make robots and their functional parts in a modular structure at the software level and to simplify the process of building robots by simply combining selected modules	Standard	CORBA	Generic	Yes	High
UPnP Robot	To enable automatic discovery, configuration, and integration for robot components in both modular and ubiquitous robots	Non Standard	UPnP	For automatic integration	Yes	Low
Player / Stage System	To provides a development platform that supports different robotic hardware and provides common services needed by different robotic applications	Non Standard	Three-tier architecture, Proxy objects	Generic	No	High
PEIS Kernel	To provide a common communication and cooperation model that can be shared among multiple robots and devices	Non Standard	Uniform communication & cooperation models, Distributed tuple-space, fully decentralized	Generic	Yes	High
MARIE	To create flexible distributed components that allows developers to share, reuse,	Non Standard	Mediator interoperability technology, ACE	Generic	No	High

	and integrate new or existing software programs for rapid robotic application development					
RSCA	To provide real-time support for robotic applications and to provide abstractions that makes robotic applications both portable and reusable on different hardware platforms	Standard	RT-CORBA	Generic and for QoS support	Yes	High
AWARE	To provide data-centric capabilities for the integration of wireless sensor networks and mobile robots	Standard	TinyOS, TinySchema, Publish/ subscribe	For sensory service	Yes	Low
Sensory Data Processing	To provide abstracted services for accessing external sensor networks information to support service mobile robots	Non Standard	N/A	For sensory service	No	Low
Layer for Incorporation	To enable communication among ubiquitous robots which are usually of different types	Non Standard	Sensor and behavior mappings	incorporation among different robot types	No	Medium
Orca	To enable software reuse in robotics using component-based development.	Standard	CORBA	developing component-based robotics	Yes	Medium
MIRA	To compose different modules dynamically at runtime to form the final complex application	Standard	Publish/ subscribe	Generic	No	Yes
ROS	To provide the OS services (e.g. hardware abstraction, low level device control, message-passing between processes, package management)	Standard	Message oriented RPC services	Generic	No	Yes

LIME	To develop applications that exhibit physical mobility of hosts, logical mobility of agents, or both	Non Standard	Distributed tuple-space, fully decentralized	Generic	No	Yes
OROCOS	To develop a general purpose modular framework for robot and machine control	Standard	CORBA ACE/TAO	Generic	No	Yes
Urbi	To develop portable applications for robotics and artificial intelligence	Standard	Non-modular, distributed component architecture	Generic	No	No

Table 1. Summary of list of main robotics middleware

3. Robot-Era System Overview

The whole Robot-Era system has demanding requirements in terms of overall software organization and interoperability. The system must be able to accommodate devices which are *highly heterogeneous*, ranging from the computational powerful domestic, condominium and outdoor robots; to relatively simpler devices like hand-held tablet computers or robotic vacuum cleaners; all the way down to the very simple (wired or wireless) sensors and actuators that form the fabric of the Ambient Intelligence infrastructure. The system must afford seamless *distribution of information and cooperation* among devices, irrespective of the differences in hardware and software across devices. It must allow the *dynamic insertion and removal* of devices into and from the overall Robot-Era robotic ecology with no need for human intervention. Finally, the system must be open to the introduction of new devices in the future, again with no or minimal human intervention.

3.1. Reference Architecture

To accommodate the above requirements, the Robot-Era consortium has decided to rely on a two-layer reference architecture, shown in Figure 2. In this architecture:

- at the upper layer, called "*Ecology Layer*", a common network-oriented middleware provides seamless integration among heterogeneous devices, including the Robot-Era robots, while addressing all the above requirements;
- at the lower layer, called "*Device Layer*", each robot and device may use a specific architecture and middleware internally, while externally it presents a uniform interface toward the ecology layer, possibly through a gateway.

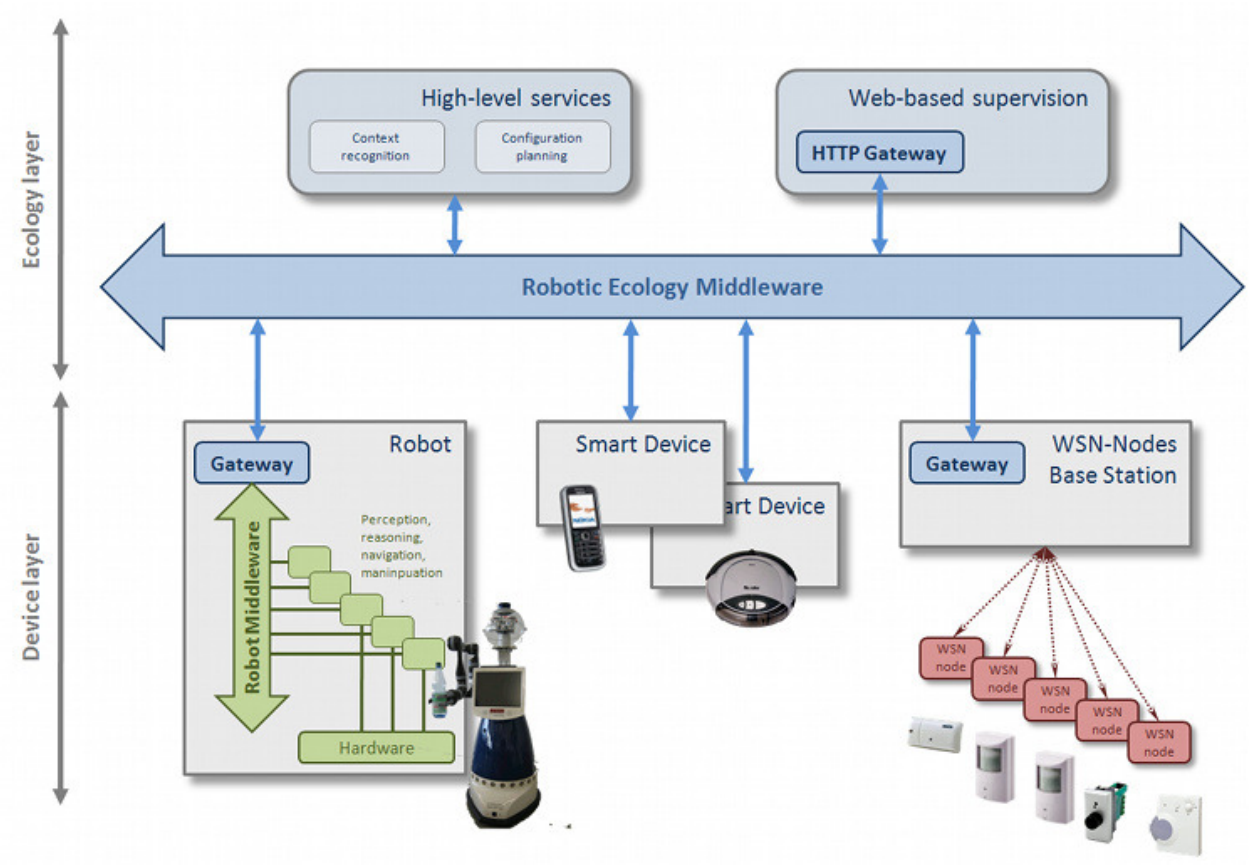

Figure 2: Reference architecture of the whole Robot-Era system.

Sections 4 and 5 below will discuss the Ecology and Device layers in more detail.

3.2. Requirements for the Ecology Layer

For the purposes of the Robot-Era system many requirements have been posed by partners on the infrastructure to be used. The requirements are different for the Ecology Layer and for the Device Layer.

For the *Ecology Layer*, the middleware that implements it should:

- provide a *shared memory model*, that allows seamless sharing of information and content-based access to information;
- be *fully decentralized*, to minimize the infrastructure requirements and hence reduce cost to the user while improving robustness;
- provide simple mechanisms for *introspection* and *dynamic configuration*;
- have a *small footprint*, suitable both for smaller devices and for larger embedded computers;
- expose a *minimalistic API* easily usable by both expert as well as component programmers with little expertise in middleware and network design;
- support *heterogeneous devices* ranging from simple sensors, tiny embedded devices, and household appliances to complex robots (the three robotic platforms);
- provide a simple way to *interface with the device-specific middleware* which will be used inside each single robotic platform, e.g., ROS, Player or MIRA;

- afford *automatic discovery, configuration and interconnection* of different devices to realize dynamic, task-based ecologies, with no need of manual intervention;
- handle the *dynamic appearance and disappearance* of devices, or group of devices, from the network;
- allow devices to operate *both in isolation as well as in ad-hoc groups* formed when devices come within communication range;
- smoothly *scale* as the number of devices increase;
- be easily *available* to the Consortium, ideally through an open-source licence.

From the study of many classical middleware described in Section 2.2 above, the Robot-Era partners considered that the PEIS-Middleware satisfies the above requirements. The PEIS-Middleware was therefore chosen by partners to be adopted as the Robotic Ecology middleware in the Robot-Era project. The PEIS-Middleware is available open-source, and its main developer is a partner of the Robot-Era consortium.

The PEIS-Middleware is implemented without following a standard communication model. This allows for providing some advanced functions that are specifically needed by the Robot-Era scenarios and applications. Other middlewares, for example RT or RSCA middleware, also satisfy many of the above requirements, but they are implemented following a standard communication model, which makes them less suitable than PEIS-Middleware in order to perform some specific tasks.

In particular, automatic discovery, configuration and integration of new services and devices are important features that the Robot-Era middleware has to satisfy. The automatic discovery and configuration mechanisms are appropriate for dynamic computing environment such as ubiquitous robots in Robot-Era system. Mobile robots can discover the existence of external devices and can configure themselves to interact with them. These devices can be WSN, wearable devices, and controllable electromechanical devices. The PEIS-Middleware provides a simple dynamic model for self-configuration and introspection. All devices (PEIS, in the PEIS-Middleware terminology) are connected by a uniform communication model that allows dynamic joining and leaving of PEIS.

3.3. Requirements for the Device Layer

The requirements of this layer depend on each specific robot and device, and they are therefore discussed in the Deliverables dealing the AmI devices and infrastructure (D3.1), and with the domestic / condominium / outdoor robots (D4.1 / D5.1 / D6.1). From the point of view of interoperability, which is the central concern of the present document, the following general observations can be made:

- Some devices will be capable of running the PEIS-kernel, the core part of the PEIS-Middleware, and therefore they can be *directly integrated* in the Robot-Era system through the PEIS-Middleware functionalities. Examples of these devices depicted in Figure 2 above include a tablet PC for user interface and a robotic vacuum cleaner.
- Other devices may not be capable of running the PEIS-kernel, because of severe computational limitations. These include WSN (wireless sensor network) nodes, which incorporate the sensors and actuators embedded in the smart environment. These devices will be connected to one or more base stations, which will provide a *gateway toward the PEIS-Middleware* to allow interoperation with any other device in the Robot-Era system via the PEIS-Middleware mechanisms.
- Robotic platforms use a specific *robot middleware* internally, and this may be different from the PEIS-Middleware. The purpose of the robot middleware is to integrate robotic components within the platform itself, e.g., components for perception, reasoning, navigation and manipulation, as well as components that provide access to the robot's hardware – see Figure 2. The robot middleware concerns the implementation of a given

robot, and it should not be confused with the Ecology Layer middleware which concerns the orchestration of the whole Robot-Era system. In particular, the robot middleware should address typical robotic issues, e.g., availability of reusable robotic components, real- or near-real-time performance, and smooth integration with robotic hardware. These issues are different from the requirements for the Ecology Layer middleware listed above.

- The Robot-Era consortium has decided to adopt ROS and MIRA as robot middleware inside the robotic platforms. A *gateway between ROS/MIRA and the PEIS-Middleware* must therefore be created. This will be further discussed in Section 5 below.

4. Interoperability in the Ecology Layer

At the Ecology Layer, a Robot-Era system will rely on the PEIS-Middleware to provide interoperability among all the robots and devices participating in the system. The PEIS-Middleware implements a fully decentralized communication model based on a distributed tuple-space, and runs on a wide range of platforms. Below, we recall the main features of the PEIS-Middleware. A more comprehensive description can be found in the literature [11,21,22].

4.1. The PEIS-Kernel

The PEIS-Middleware is a set of software libraries and tools that allow a developer to implement a robot ecology, to visualize its state, and to debug it. The tools and libraries are available for many platforms and programming languages (C/C++, Java, LISP). They are organized in the PEIS-Kernel software stack, which is visualized in Figure 3.

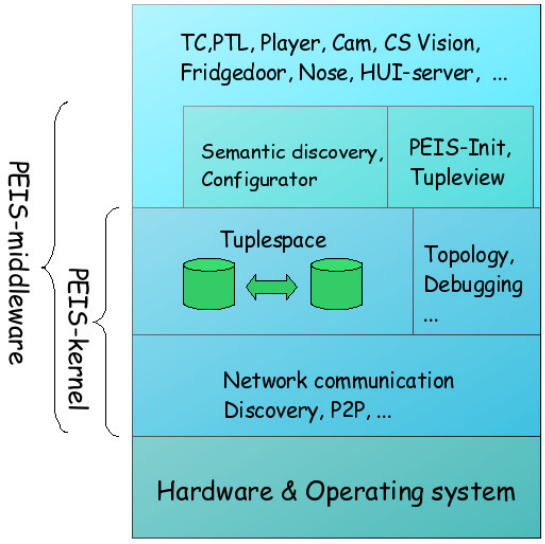

Figure 3: The software stack of the PEIS-Middleware

At the bottom of the stack, we rely on a standard POSIX compliant operating system and hardware. Above this lays the PEIS-Kernel library. This library utilizes the available communication devices, such as network cards or bluetooth interfaces, to establish a peer-to-peer (P2P) network between all components running on the different machines in the

neighbourhood. This allows even components without a direct means of communication to communicate.

The decentralized nature of this P2P network allows for dynamically changing topology and devices appearing/disappearing from the network. The primary interface to this network is the capability of detecting devices, routing unicast and broadcast messages reliably at different virtual ports. In this multi-hops routing we use the standard weighted random early detection (WRED) congestion control algorithm to ensure a higher QoS for control messages and meta data. Interestingly, this routing allows devices which have mutually exclusive communication methods such as wireless LAN (802.11) and ZigBee (802.15.4) to still communicate using any set of intermediate nodes as a bridge. Additionally, this layer provides a mechanism for registering callable functions as hooks to be invoked periodically, or upon different events such as receiving specific messages.

At the next layer, we utilize this P2P network to implement a number of different services. The most important of these is a distributed tuple-space, which is described in detail in the next subsection. Other services include synchronization of a decentralized network clock, and various debugging functionalities. The P2P layer together with this service layer constitute the PEIS-Kernel library. Any component that should participate in a robot ecology is run as a normal user-space process linked to this library.

In addition to the basic functionalities implemented in the PEIS-Kernel, the PEIS-Middleware includes a number of standardized components such as a visualization and debugging components, automatic configurators, and the PeisInit component. The latter component plays a critical role in the introspection and dynamic configuration capability of the PEIS-Middleware. It is started on boot-time on every platform; it provides information about all the components which may run on that platform; and it dynamically starts, stops and monitors the execution of those components.

4.2. The Tuple-Space

As a shared memory model for communication and coordination, the PEIS-Middleware implements a Linda-type distributed tuple-space, augmented with an event mechanism. In a Linda-space, tuples containing keys and other data can be stored and retrieved by any participating process. In our version of this space, a PEIS-tuple consists of a name-space, key, data as well as several meta-attributes such as timestamps and expiration date – see Figure 4. A component corresponding to the name-space is called the owner of the tuple. The key is a string consisting of dot-separated fields (currently up to seven), e.g., "`camera.position`". Wild-cards are allowed in fields, e.g., "`*.position`" to allow associative searches.

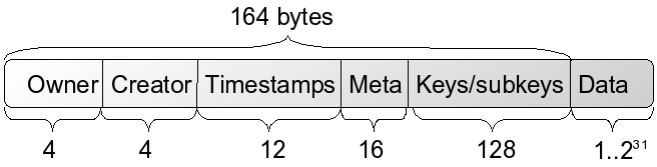

Figure 4: Format of a PEIS-Tuple

We also allow the use of abstract tuples, that is, tuples in which one or more fields contain a wild-card value. An abstract tuple is said to match a concrete (fully instantiated) tuple if all the non wild-card fields are equal to the corresponding fields of the concrete tuple. Abstract tuples allow us to make associative searches in the distributed tuplespace. These searches result in the subset of all tuples that is generalized by the given abstract tuple.

In addition to reading, the PEIS-Middleware allows remote components to update tuples in the tuplespace. This capability is pivotal to enable actuation over the network. When a component writes a tuple with a specific name-space and key, the kernel sends a message with a write request to the owner of that tuple. The owner is responsible for synchronization, serving requests in FIFO order: this circumvents many problems with decentralized databases and provides some load balancing. The tuple owner commits the write in its local memory and sends a notification to all components which have registered an interest in this tuple. Components can register an interest in specific tuples by means of a subscription mechanism, making sure that they will continuously receive updates whenever that tuple is written. Subscriptions are typically made using abstract tuples containing wild-cards that match zero or more specific tuples. Subscriptions and notifications are transmitted over the P2P network and are guaranteed to be delivered in order. Whenever a component receives a tuple change notification, it invokes any user specified hook associated with a tuple generalizing the updated tuple, and stores the tuple in a local cache. Retrievals from the local cache of tuples are instantaneous, and do not incur any network overhead. This allows the use of both asynchronous and instant read by value (polling) access to the tuples, as well as callback access, again using abstract tuples matching zero or more specific tuples.

To establish a given pattern of communication and collaboration between a given set of components, a robot ecology designer typically creates the right set of tuple subscriptions and callbacks among those components. This set is called a configuration of the robot ecology. The PEIS-Middleware allows a dynamic configuration mechanism, in which a designer as well as any software component can create and remove subscriptions at run-time. This mechanism is based on the notion of meta tuples, which contain a reference by name to other tuples.

5. Interoperability in the Device Layer

To support interoperability at the Ecology Layer, each robot and device in a Robot-Era system must appear as a PEIS device to the PEIS-Middleware. At the Device Layer, however, some robots and device may not be natively implemented in such a way to support the PEIS-Middleware. From this point of view, we distinguish three types of devices participating in a Robot-Era system (see Figure 2 in Section 4 above):

1. *Devices which can run the PEIS-Kernel*; these devices can be integrated into the overall system through the PEIS-Middleware.
2. *Devices which are too limited to run the PEIS-Kernel*, e.g., wireless sensor networks; these devices are integrated into the overall system through one or multiple dedicated base stations, which include a gateway toward the PEIS-Middleware.
3. *Devices which internally use a different middleware*, e.g., robotic platforms.

Integration of devices of the first type is obviously straightforward. The integration of devices of the second type will be discussed in Deliverable D3.1 "Report on the implementation of the AmI infrastructure modules". In the next section, we focus on the integration of devices of the third type.

5.1. Robot platform architecture

Robot-Era includes three types of robotic platforms: domestic robots, condominium robots, and outdoor robots. Although these platforms will differ both in hardware and software, the Robot-Era consortium agreed that they should share a similar abstract architecture. This architecture is shown in Figure 5 below.

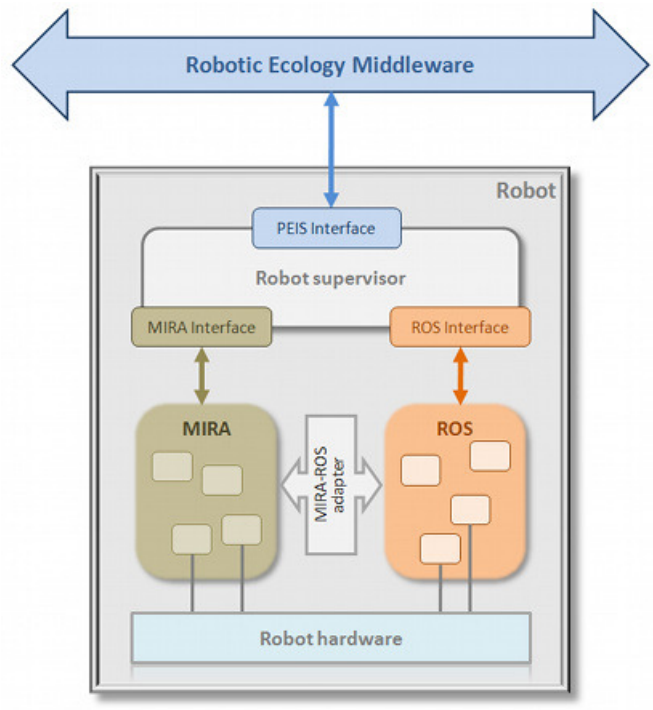

Figure 5: Abstract architecture of a Robot-Era robotic platform.

Each robot platform comprises a number of robot components for perception, reasoning, navigation and manipulation. These components are integrated with the help of a robot integration framework, or "robot middleware". In Robot-Era, it has been decided that two such frameworks will be used: the ROS (Robot Operating System) created by Willow Garage, and MIRA (Middleware for Robotic Applications) developed by MetraLab in cooperation with the Ilmenau University of Technology. The rationale for using both frameworks is as follows.

MIRA is the native middleware implemented in the MetraLab Scitos G5 and G6 robotic platforms, which will be used as basis for the Robot-Era domestic and condominium robots. MIRA includes stable drivers for the sensors and actuators in those robots, as well as a mature navigation software, called Cognidrive, geared toward operation of these robots. Using MIRA will therefore allow us an optimal reuse of existing results when developing the needed navigation functionalities for the Robot-Era platforms.

ROS is assessing itself as the *de-facto* standard in robot middleware. It has an impressive and rapidly growing user base, and a correspondingly very large repository of robotic algorithms which have been packaged for easy deployment using ROS. These include algorithms needed for navigation, like mapping, localization and path planning; but also algorithms needed for object manipulation, like 3D scene interpretation, manipulation planning, and visual servoing. In particular, ROS packages are available for processing 3D point clouds acquired by the Kinect sensor, and for controlling the Kinova Jaco robot arm: both of these devices will be part on the Robot-Era domestic robots. Using ROS will therefore allow us an optimal reuse of existing results when developing the needed manipulation functionalities for the Robot-Era platform.

Not all Robot-Era platform will need to run both the MIRA and ROS robot middleware. For example, the condominium robot, which does not have a manipulator, may only need MIRA. If both MIRA and ROS are needed on the same platform, however, an adaptor needs to be

implemented to ensure the proper exchange of information and coordination between robot algorithms implemented in the two sides.

Irrespective on how the individual robotic algorithms are implemented, the whole robot platform should appear to the overall Robot-Era system as one PEIS device able to provide a certain number of services and functionalities. To achieve this abstraction, each Robot-Era robot system will include a Robot Supervisor module. The role of the Robot Supervisor is to implement the abstract services and functionalities that the whole robot is meant to provide, by coordinating the activation of the needed robotic components in the MIRA side, in the ROS side, or in both. For instance, a "navigate to (x,y)" service can be implemented by first using the arm controller in the ROS side to fold the arm in a safe position, and then invoking the relevant localization, path planning and path following modules in the MIRA side. In general, the Robot Supervisor must be able to interact both with MIRA and with ROS. It will be implemented as a PEIS software component, in order to interact with the other robots and devices in the Robot-Era system using the PEIS Middleware. In this sense, the Robot Supervisor will cover the role of "PEIS Gateway" for the robot, as indicated in the whole Robot-Era system architecture depicted in Figure 2 above. The Robot Supervisor will also advertise to the rest of the Robot-Era system the services and functionalities that its robot can provide.

The architecture discussed above is intentionally abstract. Each concrete Robot-Era robot platform will implement a concrete version of this abstract architecture. The concrete architectures for the domestic, condominium and outdoor robots are discussed in Deliverables D4.1, D5.1 and D6.1, respectively ("Report on specifications and middleware architecture of the domestic / condominium / outdoor robotic platform").

6. Conclusions

This work has introduced the interoperability aspects of Robot-Era services. Since in the Robot-Era project different independently developed devices coming from multiple suppliers and providers will be installed in different environments (public and private spaces, living labs and residential sites), it was crucial to analyse and define all interoperability aspects that will favour an easy, scalable, and flexible integration of Robot-Era platforms and services in all experimental settings.

All participating partners in the project have discussed the requirements of the whole Robot-Era system in terms of overall software organization and interoperability, to accommodate devices which are highly heterogeneous.

Many different middlewares have been proposed for robotic networks applications, often from a different point of view and with different features and applications. These middlewares were studied by partners in order to evaluate their main features and possible applications, related to the Robot-Era project.

The Robot-Era consortium has discussed the whole architecture of the system, and decided to rely on a two-layer reference architecture: at the upper layer ("*Ecology Layer*") a common network-oriented middleware provides seamless integration among heterogeneous devices, including the Robot-Era robots, while addressing all the above requirements; at the lower layer ("*Device Layer*") each robot and device may use a specific architecture and middleware internally, while externally it presents a uniform interface toward the ecology layer.

The Robot-Era partners considered that the PEIS-Middleware satisfied all requirements, it was therefore chosen by partners to be adopted as the Robotic Ecology middleware in the

Robot-Era project, to provide interoperability among all the robots and devices participating in the system.

The requirements of the Device Layer depend on each specific robot and device, and they are therefore discussed in other Deliverables. Robot-Era includes three types of robotic platforms: domestic robots, condominium robots, and outdoor robots. Although these platforms will differ both in hardware and software, the Robot-Era consortium agreed that they should share a similar abstract architecture.

Each robot platform comprises a number of robot components for perception, reasoning, navigation and manipulation. These components are integrated with the help of a robot integration framework, or "robot middleware". In Robot-Era, it has been decided that two such frameworks will be used: the ROS (Robot Operating System), and MIRA (Middleware for Robotic Applications). MIRA is the native middleware implemented in the MetraLab Scitos G5 and G6 robotic platforms, which will be used as basis for the Robot-Era domestic and condominium robots. ROS is assessing itself as the *de-facto* standard in robot middleware, and it will be used for manipulation functionalities.

Not all Robot-Era platform will need to run both the MIRA and ROS robot middleware. If both MIRA and ROS are needed on the same platform, however, an adaptor needs to be implemented to ensure the proper exchange of information and coordination between robot algorithms implemented in the two sides.

The architecture discussed is intentionally abstract. Each concrete Robot-Era robot platform, and their architecture, is discussed in Deliverables D4.1, D5.1 and D6.1, respectively.

References

[1] N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "A review of middleware for networked robots," *International Journal of Computer Science and Network Security*, vol. 9, no. 5, pp. 139-148, 2009.

[2] A. Elkady and T. Sobh, "Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography,"

[3] N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "Middleware for robotics: a survey," in *Proceedings of the IEEE International Conference on Robotics, Automation and Mechatronics (RAM '08)*, pp. 736–742, September 2008.

[4] H. Utz, S. Sablatnög, S. Enderle, G. Kraetzschmar, "Miro - Middleware for Mobile Robot Applications," IEEE Transactions On Robotics And Automation, vol. 18, no. 4, August 2002.

[5] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, W. Yoon, "RT-Middleware: Distributed Component Middleware for RT (Robot Technology)", *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3555-3560, Aug. 2006.

[6] S. Ahn, K. Lim, J. Lee, H. Ko, Y. Kwon and H. Kim, "UPnP Robot Middleware for Ubiquitous Robot Control," *The 3rd International Conference on Ubiquitous Robots and Ambient Intelligence (URAI 2006)*, Oct. 2006.

[7] M. Kranz, R. Rusu, A. Maldonado, M. Beetz, A. Schmidth, "A Player/Stage System for Context-Aware Intelligent Environments," in *Proc. of the System Support for Ubiquitous Computing Workshop (UbiSys)*, Sep. 2006.

[8] C. Côté, Y. Brosseau; D. Létourneau; C. Raïevsky, F. Michaud, "Robotic Software Integration Using MARIE," *International Journal of Advanced Robotic Systems*, Vol. 3, No. 1, pp. 55-60, March 2006.

[9] Gil, P., I. Maza, A. Ollero, P. Marrón, "Data centric middleware for the integration of wireless sensor networks and mobile robots," in *proc. 7th Conference on Mobile Robots and Competitions, ROBOTICA 2007*. April 2007.

[10] J. Yoo, S. Kim, and S. Hong, "The Robot Software Communications Architecture (RSCA) QoS-Aware Middleware for Networked Service Robots," in *Proc. International Join Conference SICE-ICASE*, pp. 330- 335, Oct. 2006.

[11] Broxvall, M., B.S. Seo, W.Y. Kwon, "The PEIS Kernel: A Middleware for Ubiquitous Robotics," in *proc. IROS-07 Workshop on Ubiquitous Robotic Space Design and Applications*, Oct. 2007.

[12] E. Takcuchi and T. Tsubouchi, "Sensory Data Processing Middlewares for Service Mobile Robot Applications", in *Proc. International Join Conference SICE-ICASE*, Oct. 2006.

[13] T. Kim, S. Choi, and J. Lim, "Incorporation of a Software Robot and a Mobile Robot Using a Middle Layer," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Vol. 37, No. 6, pp. 1342-1348, November 2007.

[14] M. Henning, "A New Approach to Object-Oriented Middleware," *IEEE Internet Computing*, pp. 66-75, Jan.-Feb. 2004.

[15] MIddleware for Robotic Applications (MIRA) , http://www.mira-project.org/joomla-mira.

[16] Robot Operating System (ROS), 2011, http://www.ros.org.

[17] A.L. Murphy, G.P. Picco and G.C. Roman, "Lime: A Coordination Model and Middleware Supporting Mobility of Hosts and Agents," ACM Transactions on Software Engineering, Vol. X, No. X, X 2006, Pages 1–49.

[18] D. Gelernter, "Generative communication in linda," ACM Trans. Program. Lang. Syst., vol. 7, no. 1, pp. 80–112, 1985.

[19] P. Soetens, RTT: Real-Time Toolkit, 2010, http://www.Orocos.org/rtt.

[20] J.C. Baillie, A. Demaille, Q. Hocquet, M. Nottale and S. Tardieu, "The Urbi Universal Platform for Robotics", in Workshop Proceedings of SIMPAR 2008, ISBN 978-88-95872-01-8, pp. 580-591.

[21] A. Saffiotti and M. Broxvall. PEIS Ecologies: Ambient Intelligence meets Autonomous Robotics. Proc. of the Int. Conf. on Smart Objects and Ambient Intelligence (sOc-EUSAI) pp. 275-280. Grenoble, France, 2005.

[22] M. Bordignon, J. Rashid, M. Broxvall and A. Saffiotti. Seamless Integration of Robots and Tiny Embedded Devices in a PEIS-Ecology. Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS). San Diego, CA, 2007.

Appendix A - Electrical Standards

Many different issues were studied in order to favour an easy, scalable, and flexible integration of Robot-Era platforms and services in all experimental settings. Therefore many aspects, such as electrical issues, were studied and faced to avoid error-prone and time-consuming situations during the set-up of the experimental sites, and to favour an easy upgradeability and extendibility of the Robot-Era services along the entire project and beyond.

An important studied aspect was the electrical standards used in Italy and Sweden, the two countries in which the experimental sites of Robot-Era system will be set up.

The plugs, voltages and frequencies they use for providing electrical power to electrical appliances were considered. Every country has differing rules regarding distribution of electricity for appliances. Voltage, frequency, and wall socket type vary widely, but large regions may use common standards. In some areas, older standards may still exist, and physical compatibility of receptacles may not ensure compatibility of voltage and frequency.

In both countries, Italy and Sweden, the electrical voltage and frequency are the same: 230 V (formerly 220 V) @ 50 Hz; this is the most important aspect for the power supply of robotic platforms and the other devices in the Robot-Era services during the experimental tests.

About the wall sockets, the lettering system used here is from a U.S. government document [1], which defines the letter names and gives a list of what plug types are used where.

In Italy the C, F, L types are used. In Italy common sockets have 8-shaped holes to accept both 16A and 10A version of the L plug, but in hotels 10A sockets are still common. Schuko sockets are unusual, but adaptors rated up to 1500 Watt are widespread. C sockets are not used in modern installations. Italian wall-boxes are similar to American ones, but are usually horizontally mounted.

In Sweden the C and F socket types are used; non-grounded sockets are prohibited in new installations. 400 V for some washing machines and other fixed installations. In bathroom etc. 110-115 socket can be found and used for shavers and other “bathroom tools”.

[1] U.S. Department of Commerce, International Trade Administration, Electric Current Abroad, 2002.