Proc. of the 9th Int. Conf. on Intelligent Autonomous Systems (IAS-9) 1
Tokyo, Japan. March 7-9, 2006

Using JavaSpace for a PEIS Ecology

BeomSu Seo [a,1] Mathias Broxvall [b] Marco Gritti [b] Alessandro Saffiotti [b]
JungBae Kim [a]
{bsseo, jjkim}@etri.re.kr [a], {mbl, mgi, asaffio}@aass.oru.se [b]
[a] *ETRI(Electronics and Telecommunications Research Institute, Korea)*
[b] *AASS Mobile Robotics Laboratory, University of Orebro, Sweden*

Abstract.
The ecology of Physically Embedded Intelligent Systems (PEIS) is a new multi-robotic framework conceived by integrating insights from the fields of autonomous robotics and ambient intelligence. A PEIS-Ecology is a network of intelligent robotic devices that can provide the user with assistance, information, communication, and entertainment services. In this paper we introduce the concept of PEIS-Ecology, and we investigate about the use of JAVASPACE to build a *middleware* infrastructure that meets its special requirements. At the end, we illustrate a concrete realization of a PEIS-Ecology we implemented using JAVASPACE as a communication middleware.

Keywords. Multi-robot system, Ecological robotics, Ambient intelligence, Distributed architecture, Service robotics, JAVASPACE, JINI

1. Introduction

By virtue of the prodigious developments in embedded systems, ambient intelligence and autonomous robotics, in a decade or two we will live in the world of the Physically Embedded Intelligent Systems (PEIS), which will improve the quality of life for every citizen by providing physical and/or cognitive assistance [1].

PEIS do not exist and operate in isolation: they can cooperate and communicate in an **ecology**, both inside and outside our houses. The overall functionality of the full system emerges from the interaction between a number of simpler units. The functionality of each individual unit is improved by this interaction. As an example, an hypothetical table setting robot waiter will not need to use its vision sensors to detect the position, shape, and weight of the various objects in order to grasp them. But every object, enriched with an micro-PEIS, would hold this information and communicate it to the robot.

Most current approaches to building a ”robot companion” aim at building one isolated robotic device (often humanlike) empowered with extraordinary abilities for perception, action, and cognition (e.g., [2] [3]). By contrast, the PEIS-Ecology approach redefines the very notion of a robot to encompass the entire environment. Perception and manipulation of objects are thus replaced, or at least complemented, by direct communication between sub-systems in the environment. In the PEIS-Ecology vision, the robot will disappear in the environment quite in the same way as computers should disappear according to the well known vision of ubiquitous computing.

[1]Correspondence to: BeomSu Seo, 161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350, Korea Tel.: +82 42 860 6242 ; Fax: +82 42 860 6790; E-mail: bsseo@etri.re.kr

The PEIS-Ecology inherits from the idea of ubiquitous computing for distributing resources over adequate systems spread into the working environment. In every application field of computer science where distributed computation is needed, the presence of a *middleware* is a point of discussion. Middlewares built specifically for sensor networks [4] systems and for ambient intelligence [5] systems are currently a topic of research and development [6] [7]. In the robotics community, when service distribution is needed, the focus is instead on exploiting general purpose middlewares, as it happens in projects like OROCOS [8] and Miro [9], both based on the TAO/ACE [10]. The possibility to use UPnP [11] for such kind of ubiquitous robotics applications has been analyzed [12] too. On the other hand, recent works in cooperative robotics show that advances in multi-agent coordination techniques can still be made without exploiting any middleware, but just relying on simple message passing at application level.

The topic of deciding whether the PEIS-Ecology needs a dedicated middleware, or if it exists a general purpose middleware that can suit all its requirements is still open. In this paper we will concentrate on identifying the peculiarities of the PEIS-Ecology network, as a distributed computing environment, and on describing why JINI [13] enriched with the JAVASPACE [14] service can constitute an adequate middleware for it.

In the subsequent sections we introduce the basic concept of PEIS-Ecology, analyze the peculiarities of this networked *eco-system*, show how JINI and JAVASPACE can meet our requirements, and detail an implementation of the PEIS-Ecology using this platform. Before concluding, we describe the experiment we have run.

2. The concept of PEIS-Ecology

The concept of PEIS-Ecology builds upon the following ingredients.

First, any robot in the environment is abstracted by the *uniform notion* of PEIS: any device incorporating some computational and communication resource and possibly able to interact with the environment through sensors and/or actuators. A PEIS can be as simple as a toaster and as complex as a humanoid robot. In general, we define a PEIS to be a set of inter-connected software components residing in one physical entity. Each component may include links to sensors and actuators that connect it to the physical environment, as well as input and output ports that connect it to other components in the same PEIS or in other PEIS.

Second, all PEIS are connected by a *uniform communication model*, which allows the exchange of information among the individual PEIS, and can cope with their dynamic joining and leaving the ecology.

Third, all PEIS can cooperate by a *uniform cooperation model*, based on the notion of linking functional components: each participating PEIS can use functionalities from other PEIS in the ecology in order to compensate or to complement its own. The power of a PEIS-Ecology does not reside so much in the power of the individual PEIS in it, but rather it emerges from their ability to interact and cooperate.

We define a PEIS-*Ecology* to be a collection of inter-connected PEIS, all embedded in the same physical environment. We call *configuration* the set of connections between components within and across the PEIS in the ecology. Note that the same ecology can be configured in many different ways depending on the current context. Relevant contextual aspects here include the current goals, situation, and resources.

Stated in these terms, a PEIS-Ecology redefines the very notion of a *robot* to encompass the entire environment: a PEIS-Ecology may be seen as a “cognitive robotic

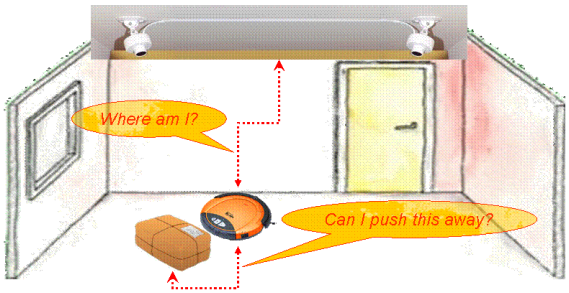


Figure 1. A simple example of cooperation in a PEIS-Ecology.

environment", in which perception, actuation, memory, and processing are pervasively distributed in the environment.

As an illustration of these concepts, consider a simple autonomous vacuum cleaning robot. This can be seen as a PEIS, which includes simple functionalities for reactive navigation and obstacle detection. By itself, this robot cannot devise and execute complex cleaning strategies because its perceptual abilities are too simple to reliably estimate its own position in the home.

Suppose, however, that the home is equipped with an ambient monitoring system using a set of cameras, which in addition to its other functions is able to estimate the position of the vacuum cleaner. Then, we can combine the monitoring system and the vacuum cleaner into a simple PEIS-Ecology, in which the former provides the latter with a global localization functionality. The vacuum cleaner can use this functionality to realize a smarter cleaning strategy(See Figure 1). Suppose next that the vacuum cleaner finds an unexpected parcel on the floor. The cleaner needs to know its properties (e.g., weight and fragility) in order to decide weather it can push it away or not. These properties can hardly be estimated using the robot's sensor. However, if the parcel is equipped with an IC tag holding information about its content, then it can be seen as a PEIS, which can join the ecology and deliver this information directly to the robot.

3. Requirements for a PEIS-middleware

The basic building blocks of a PEIS-Ecology are PEIS-components: the set of available software components implementing the robotic algorithms (perception, modeling, deliberation, control, etc.). These components can generally be seen as logical processes.

In order to allow the interaction between PEIS-components, we need a PEIS-*middleware*: a software layer between the network abstraction provided by the operating system and the PEIS-components themselves. This middleware should support flexible and dynamic communication among PEIS along with various services such as identification, authorization, directories, naming, security, etc. By exploiting the PEIS-middleware, PEIS-components can transparently communicate to each other without re-implementing the mechanisms needed by all these services.

In this section, we briefly describe the characteristics of a PEIS-Ecology which are critical for the choice of an adequate PEIS-middleware.

High adaptation to change. A PEIS-Ecology can be constituted by many components, and any PEIS-component can both enter and leave a PEIS-Ecology without notification; so the PEIS-middleware should allow communication among these components to dynamically change according to the running context. Therefore, the technology chosen to implement PEIS-Ecology should be based on scalable and flexible adaptive networks.

Loosely coupled communication. PEIS-Ecology is data centric and not communication centric: this means that there is no direct method call or message passing between source and target components. The vacuum cleaner can use both its local navigation component and the global monitoring component based on running environments. Or, as a new PEIS-component enters into this ecology, it may provide navigation data to the cleaner. Under these circumstances, it is not only difficult to fix every connection path in advance but this would also not follow the philosophy of the framework.

Distributed coordination-based model. In a PEIS-Ecology, what each component is interested in is *which data it receives, not* from whom it receives them. Since data sources could be changed according to the running context, we need a distributed coordination-based communication model [15]. This would provide not only loosely coupled communication but also anonymity between components. With this model, a component can publish its data without any information about the clients and a client can just receive a notification about presence of the data interesting for him.

Security support. Security is another very important technical issue if we need to protect and separate an ecology from unwanted -possibly dangerous- interactions with other sources. Security topics in PEIS-Ecology are data encryption/decryption, authentication, and authorization. Authentication consists in deciding whether or not a PEIS-component can access a PEIS-Ecology. Authorization consists in allowing or forbidding a PEIS-component to read and/or write data into different parts of the ecology.

4. Middleware for PEIS-Ecologies

One excellent PEIS-middleware candidate is JINI. It is the distributed dynamic networking architecture developed by Sun Microsystems. Within the middleware services provided by JINI, the most interesting one from the point of view of PEIS-Ecology is JAVASPACE. PEIS-component may exchange data by calling directly JINI and JAVASPACE APIs, but for the purpose of extensibility and flexibility we decided to abstract from the middleware specific API by means of an interface that we standardized for our purposes. We call this additional layer the PEIS-kernel, which provides a common abstraction to allow each component to exchange data hiding the existence of JINI and JAVASPACE. In this way, PEIS-Ecology is independent from the choice of the middleware. In this section, we briefly discuss what are JINI and JAVASPACE, and how they work; then we describe the architecture of the actual PEIS-middleware we implemented by means of these technologies.

4.1. JINI and JAVASPACE

JINI : The distributed, dynamic networking architecture. JINI provides a set of application programming interfaces as well as a network protocol. JINI is an open architecture that can be used to build adaptive networks in dynamic computing environments. The objectives of JINI are to provide an infrastructure to connect anything, at any time, anywhere, and enable network plug-and-work. This means that every service can join the network and become available to other applications without any manual installation or configuration. In order to use JINI, there should be at least one *lookup server* in the network. A lookup server is a service repository, where each service (such as JAVASPACE) is stored as a Java object and is made available to clients for downloading on demand. A service provider can register its services into a lookup server after having

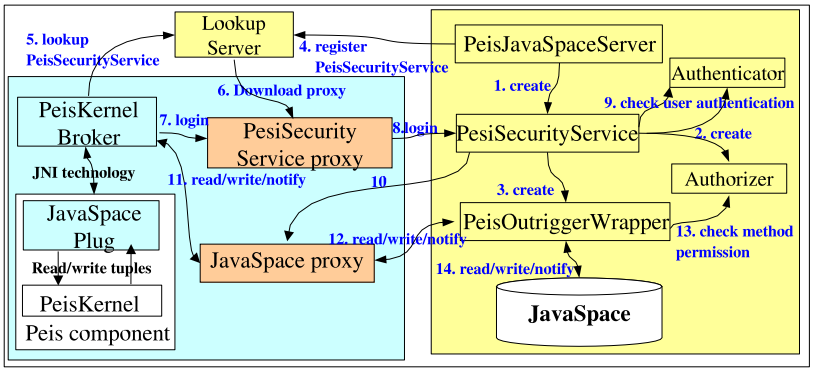


Figure 2. An overview of PEIS-middleware implementation.

found it through a multicast message on the network it joined. A client can download the service proxy using the same discovery protocol and then directly communicate to the service provider. Usually, the downloaded service object is a Java RMI stub which delegates method calls and forwards messages to the actual server. JINI features a distributed event mechanism, so clients can register to interesting events and then receive the corresponding notifications.

JAVASPACE **: Generative communication storage.** One of the interesting services of the current JINI distribution is JAVASPACE, which is an implementation of generative communication [16]: a coordination-based model. The main idea of generative communication is that distributed processes use a shared persistent database of tuples. A tuple is a data structure with a number of typed fields. A tuple is inserted into JAVASPACE by means of a write operation. JAVASPACE provides associative access to data, so a client can read/take a desired tuple issuing a template-based query on a JAVASPACE. The template tuple matches a tuple in a JAVASPACE if their non-null value fields have exactly the same values. To control the access to JAVASPACE, the JAAS (Java Authentication and Authorization Service) [17] mechanism could be used.

4.2. PEIS**-middleware implementation using** JAVASPACE

The basic role of PEIS-kernel and PEIS-middleware is to allow each PEIS-component to show its interest in special tuples with subscription mechanism and get tuple with notifications about the interesting tuple published by other PEIS-components. The PEIS-kernel provides several APIs for a PEIS-component to get/set a tuple and subscribe its interest. Each PEIS-component exchanges tuples which are restricted to be of the form : `<peisID, compID, key, val>`. For example, the monitoring system in Fig. 1 can send the position tuples to the cleaner which has subscribed its interest with `AbsolutePos` key through JAVASPACE. Because the PEIS-middleware with JAVASPACE makes it possible for a PEIS-component to get any data not taking care of who creates it, it provides not just only a shared storage for tuples but also loosely coupled communication as well as anonymity between components.

The implementation of the PEIS-middleware using JAVASPACE consists of three parts; JavaSpacePlug running under PEIS-kernel, PeisKernelBroker, and PeisJavaSpaceServer. The JavaSpacePlug is bridge which instantiates the Java VM and delegates requests/responses between the PEIS-components and the PeisKernelBroker. And it allows the PEIS-kernel to use JAVASPACE throughout the PeisKernelBroker, which acts as a

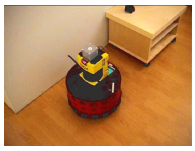
client to the PeisJavaSpaceServer. The PeisKernelBroker is a Java class which is responsible for reading and writing tuples using a JAVASPACE proxy. Finally, the PeisJavaSpaceServer is the PEIS-middleware server which makes the JAVASPACE and the PeisSecurityService available to the PEIS-Ecology. Fig. 2 gives the general overview of this implementation of the PEIS-middleware. The PeisJavaSpaceServer creates the PeisSecurityService to register its proxy into a lookup server. The PeisSecurityService creates the Authenticator and the Authorizer to checks component identification and method permission to write, read, and subscribe. The PeisSecurityService provides the PeisKernelBroker with a login method and returns a JAVASPACE proxy only if the login succeeds. The PeisOutriggerWrapper intercepts operations that access the JAVASPACE, and performs security check, logging, and multi-thread processing of tuples. In order to identify a PEIS-component during login, JAAS could be used. After the PeisKernelBroker has found the lookup server and downloaded the security service proxy, it logs in with a password or a special certificate. The security service asks the Authenticator for client identification at login time. Authentication can be performed with different security mechanisms depending on the actual implementation of the Authenticator interface. If the client PEIS-component is trusted, the security service returns to it a JAVASPACE proxy. Whenever the component will read/write tuples, its access permission will be checked by the PeisOutriggerWrapper through the Authorizer which in turn will verify such permissions by checking the security properties.

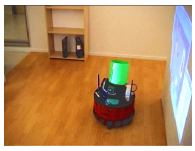
5. An Experimental Realization of PEIS-Ecology

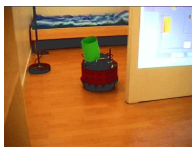
In order to verify the viability of the PEIS-Ecology framework, and to test our PEIS-middleware implementation, we have developed an experimental test-bed, into which run our experiments. This is the PEIS-Home: an home-like environment that incorporates a number of PEIS. Some of the PEIS included into the PEIS-Home are:

- The Thinking Cap PEIS-component [18] which is an architecture for autonomous robot control based on fuzzy logic. It provides a mobile robot with advanced navigation functionalities. It responds to tuples with key `ExecGoal`, which provide a navigation goal. It produces tuples with keys `Odometry`, `ExecStatus`, etc, which reflect the robot status.
- The PTLplan PEIS-component [19] which is a conditional planner used for high-level task planning. It can reason about uncertainty and about observation actions. It responds to goal tuples with key `Task` and information tuples with keys `WorldStatus` and `ExecStatus`. It produces tuples with keys `ExecGoal` to indicate new high-level actions to be performed (typically by the ThinkingCap).
- The Player PEIS-component [20] which is a PEIS-wrapped instance of the player program. It provides a low-level interface between the robot's sensors and actuators and the PEIS-Ecology's tuple-space. It reads robot command tuples (e.g., `VelCommand` and `TalkCommand`), and generates sensor reading tuples (e.g., `Odometry` and `SonarRange`).
- The localization PEIS-component which is a simple local positioning system which uses the web-cameras mounted on the ceiling to track a colored object using triangulation, and produces `AbsolutePos` tuples. In our experiments, we have equipped the tracked robot with a colored "hat".
- The PEIS-Ecology monitor which can be used by the experimenters to dynamically observe, and possibly influence, the internal state of the PEIS-Ecology.

And two real robots are;


1. Emil sends commands to Pippi


2. Pippi approaches the bedroom


3. Pippi exits the bedroom

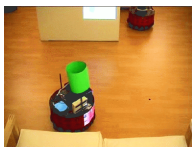

4. Task completed

Figure 3. Four snapshots taken during an actual run.

- The Pippi PEIS, a Magellan Pro robot with the ThinkingCap PEIS-component and the Player PEIS-component.
- The Emil PEIS, a Magellan Pro robot with the ThinkingCap, the Player, and the PTLplan PEIS-components.

We assume here that Emil is a master robot and Pippi is one of the robot workers within the PEIS-home environment.

One of the experiments we performed in the PEIS-Home considers the scenario in which a robot has to wake up Johanna, the hypothetical inhabitant of the PEIS-home. This experiment develops as follows.

1. Emil receives the high-level goal to wake up Johanna. She decides to delegate the execution of this task to Pippi and generates a plan for Pippi using PTLplan. The plan consists of three actions: GoTo(bed), Talk(wakeup), GoTo(sofa).
2. Emil sends the first action with `ExecGoal` tuple to Pippi, who execute it along with the information provided by the tracking PEIS for better localization.
3. If Pippi occasionally exits the field of view of the cameras, she reverts to using the (unreliable) self-location information from its internal odometry getting `AbsolutePos` tuples.
4. Pippi constantly publishes her execution status with `ExecStatus` tuples and receives through the JAVASPACE the next actions issued by Emil.
5. After Pippi has reached the bed and woken up Johanna getting `TalkCommand` tuples, she moves to the sofa.
6. After reaching the sofa, she notifies Emil about the completion of the last action.

All the above communications and synchronizations happen by the exchange of the relevant tuples through the JAVASPACE, and are mediated by the PEIS-kernel. Fig. 3 shows four snapshots taken during an actual execution of this experiment.

6. Conclusions

PEIS-Ecology offers a new paradigm to develop pervasive robotic applications that combines ideas and technologies from the fields of autonomous robotics and ambient intelligence. The major achievement of this approach will be the possibility to create real dynamic intelligent environments using the cognition, perception, and actuation capabilities of today's state-of-the-art robotic devices. In order to let intelligence *emerge* from cooperation, the PEIS-Ecology needs a PEIS-middleware which is compliant with the

frequent changes in the ecology configuration and that allows uncoupled and distributed communication along with security support. Probably, JINI is one of the best candidate architectures to fulfill these requirements.

To show the feasibility of PEIS-Ecology based on JINI and JAVASPACE we built the PEIS-Home, which is equipped with mobile robots, a web-cam localization system, and a PEIS monitoring system. In the future we will extend our experiments to include situations involving a large number of PEIS, higher dynamicity, and dynamic configuration and re-configuration.

Acknowledgments

This work has been supported partly by the Swedish KK Foundation, and partly by ETRI (Electronics and Telecommunications Research Institute, Korea) through the project "Embedded Component Technology and Standardization for URC (2004-2008)".

References

[1] A. Saffiotti and M. Broxvall. Peis Ecologies: Ambient Intelligence meets Autonomous Robotics. Proc. of the sOc-EUSAI conference on Smart Objects and Ambient Intelligence. Grenoble, FR, October 2005.
[2] The Cogniron Consortium. The Cognitive Robot Companion. www.cogniron.org
[3] The Honda Humanoid Robot ASIMO http://world.honda.com/ASIMO/
[4] D. Estrin and D. Culler and K. Pister, and G. Sukhatme. Connecting the Physical World with Pervasive Networks. *IEEE Pervasive Computing*, Vol 1, No. 1, 2002.
[5] IST Advisory Group. Ambient Intelligence: from vision to reality. In G. Riva and F. Vatalaro and F. Davide and M. Alcañiz, editors, *Ambient Intelligence*. IOS Press, 2005.
[6] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, and C. Ferraz. A message-oriented middleware for sensor networks, *In Proceedings of the* 2^{nd} *workshop on Middleware for pervasive and ad-hoc computing*, Toronto, Ontario, Canada, 2004.
[7] D. Arregui and C. Fernstrom and F. Pacull and J. Gilbert. Stitch: Middleware for ubiquitous applications, In *Proc. of the Smart Objects Conference*., Grenoble, France, 2003.
[8] Katholieke Universiteit Leuven. The Open Robot Control Software project. www.orocos.org.
[9] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar Miro- middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation*, Vol XX, No. Y, June 2002.
[10] Center for Distributed Object Computing. Washington University. TAO: A high-performance, real-time Object Request Broker (ORB). www.cs.wustl.edu/ schmidt/TAO.html
[11] The UPnP Forum. Universal Plug and Play. www.upnp.org
[12] S. Ahn, J. Kim, K. Lim, H. Ko, Y. Kwon, and H. Kim. UPnP Approach for Robot Middleware. *In Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, Barcelona, Spain, April 2005.
[13] Jini technology homepage. http://www.jini.org
[14] Eric Freeman, Susanne Hupfer, and Ken Arnold. JavaSpaces Principles, Patterns, and Practice. *Pearson Education*, 1999.
[15] A. Tanenbum, M Steen. Distributed Systems - Principal and Paradigms, *Prentice Hall*, 2002
[16] Gelernter, D. Generative communication in Linda. *ACM Transction on Programming Languages and Systems*. 7, 1 (1985) pp. 80-112.
[17] Java Authentication and Authorization Service. http://java.sun.com/products/jaas/index.jsp
[18] A. Saffiotti and K. Konolige and E. H. Ruspini. A multivalued-logic approach to integrating planning and control. *Artificial Intelligence*, 76(1-2): 481–526, 1995
[19] Karlsson, L. Conditional progressive planning under uncertainty. In *Proc. of the 17th IJCAI Conf.*, 2001.
[20] The Player/Stage Project. http://payerstage.sourceforge.net/.