The PEIS Kernel: a Middleware for Ubiquitous Robotics

Mathias Broxvall
AASS Mobile Robotics Lab
Örebro University, Örebro, Sweden
mbl@aass.oru.se

Beom-Su Seo, WooYoung Kwon
Intelligent Robot Research Division
Electronics and Telecommunications Research Institute, Korea
{bsseo,wykwon}@etri.re.kr

***Abstract*—The fields of autonomous robotics and ambient intelligence are converging toward the vision of smart robotic environments, or ubiquitous robotics, in which tasks are performed via the cooperation of many simple networked robotic devices. The concept of Ecology of Physically Embedded Intelligent Systems, or PEIS-Ecology, combines insights from these fields to provide a new solution to building intelligent robots in the service of people. To enable this vision, we need a common communication and cooperation model that allows dynamically assembled adhoc networks of robotic devices, a flexible introspection and configuration model allowing automatic (re)configuration and that can be shared between robotic devices at different scales, ranging from standard mobile robots to tiny networked embedded devices.**

In this paper we discuss the development of a middleware suitable for ubiquitous robotics in general and PEIS-Ecologies in specific. Our middleware is suitable for building truly ubiquitous robotics applications, in which devices of very different scales and capabilities can cooperate in a uniform way. We discuss the principles and implementation of our middleware, and also point to experimental results that show the viability of this concept.

I. INTRODUCTION

There is a marked tendency today toward the embedding of many intelligent, networked robotic devices in our homes and offices. A particularly interesting case is the recent emergence of a paradigm in which many robotic devices, pervasively embedded in everyday environments, cooperate in the performance of possibly complex tasks. Instances of this paradigm include the so called network robot systems [1], intelligent space [2], sensor-actuator networks [3], ubiquitous robotics [4], and PEIS-Ecologies [5]. Common to these systems is the fact that the term "robotic device" is taken in a wide sense, including both mobile robots, static sensors or actuators, and automated home appliances. In this paper, we generically refer to a system of this type as an "ecology of robots".

As part of a collaborative research project between the Electronics and Telecommunications Research Institute (ETRI), Korea, and the Centre for Applied Autonomous Sensor Systems, Sweden, we are developing ubiquitous robotic technologies to be used in domestic environments. In this project, we take an ecological view of the robot-environment relationship [6]. We see the robot and the environment as parts of the same system, which are engaged in a symbiotic relationship. We assume that robotic devices are pervasively distributed in the environment in the form of sensors, actuators, smart appliances, active tagged objects, or more traditional mobile robots. We further assume that these devices can communicate and collaborate with each-other by providing information or by performing actions. We call a system of this type an *Ecology of Physically Embedded Intelligent Systems*, or PEIS-Ecology.[1]

When realizing these PEIS-Ecologies, there is a need to focus as much on the communication and integration of the constituent components as on the individual functionalities of each participating robot. suitable for the application. In the literature, a large variety of middlewares [7], [8], [9], [10], [11] have been proposed for robotic applications. However, none of these existing middlewares satisfy the basic requirements necessary to realize PEIS-Ecologies and as such we require another solution for realizing this concept.

In this paper we describe the implementation of a middleware used in practice today for the realization of such ecologies of robots and discuss some of the current developments. This middleware is continuously updated and published under an OpenSource license at regular intervals. We also give an description of an extended experiment performed using the PEIS-Ecology concepts and this middleware, involving devices ranging from a simple sensor mote, actuated lamps to an autonomous mobile robot.

The goal of this paper is to give a general overview of the middleware used in the PEIS-Ecology project and on our progress towards its practical realization. Accordingly, we give only a brief overview of the other research issues posed by PEIS-Ecology and refer the reader to the relevant papers [5], [12], [13] in which more details and experimental results are reported. More information can also be found at the project web site [14].

The rest of this paper is organized as follows. In the next section, we briefly recall the concept of PEIS-Ecology and the middleware requirements for this concept. In Section III we discuss the basic concepts in the middleware and follow this with details on how the communication is implemented in Section IV, an implementation of it for tiny network devices in Section V and some of the software tools in Section VI. We take a look at an experiment run in Section VII and, finally, in section VIII we outline some future directions and new requirements posed on the middleware as more and more

[1]PEIS is pronounced /peɪs/ like in 'pace'.

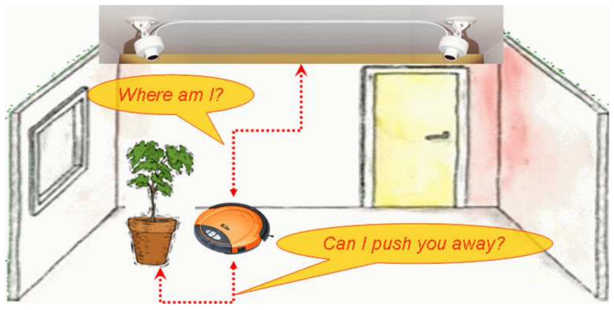

Fig. 1. A simple PEIS-Ecology consisting of a vacuum cleaner, an overhead tracking system, and a plant.

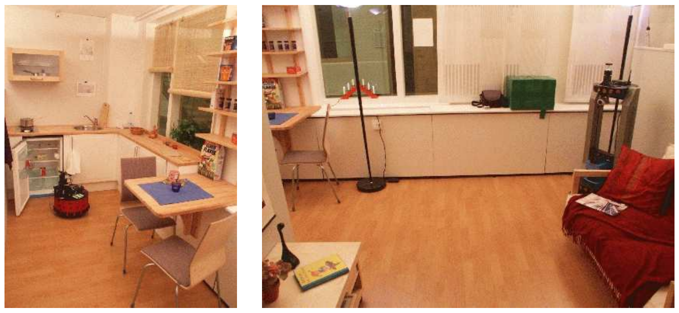

Fig. 2. Two views of the PEIS-Ecology testbed at AASS.

advanced ecologies of robots are created.

II. THE PEIS-ECOLOGY APPROACH

The concept of PEIS-Ecology, originally proposed by Saffiotti and Broxvall [5], combines insights from the fields of ambient intelligence and autonomous robotics, to generate a new approach to the inclusion of robotic technology into smart environments. In this approach, advanced robotic functionalities are not achieved through the development of extremely advanced robots, but rather through the cooperation of many simple robotic components.

A. General Approach

The concept of a PEIS-Ecology builds upon the following ingredients:

First, any robot in the environment is abstracted by the *uniform notion* of a PEIS (Physically Embedded Intelligent System), which is any device incorporating some computational and communication resources, and possibly able to interact with the environment via 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, called PEIS-components, residing in one physical entity. Each component may include links to sensors and actuators, as well as input and output ports that connect it to other components in the same or another PEIS.

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

Third, all PEIS can cooperate using 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. We define a PEIS-*Ecology* to be a collection of inter-connected PEIS, all embedded in the same physical environment.

As an illustration, consider the autonomous vacuum cleaner (PEIS) in Figure 1. By itself, the simple device can only use basic reactive cleaning strategies, because it does not have enough sensing and reasoning resources to assess its own position in the home. But suppose that the home is equipped with an overhead tracking system, itself another PEIS. Then, we can combine these two PEIS into a simple PEIS-Ecology, in which the tracking system provides a global localization functionality to the vacuum cleaner. Suppose then that the cleaner encounters a plant, and that the plant is equipped with a micro-PEIS (e.g., a mote) able to communicate its properties — e.g, size, humidity, temperature and type of support. Then, the vacuum cleaner can use these properties to decide whether it can push the plant away and clean under it.

B. Experiments

In order to be able to perform the large number of test runs and experiments required to validate the PEIS-Ecology concept and developed software we have constructed a testbed environment. To also validate the utility and acceptability of a PEIS-Ecology for humans in a realistic setting, this physical testbed facility, called the PEIS-Home, resembles a typical Swedish bachelor apartment (Figure 2 with familiar devices. The PEIS-Home is equipped with a communication infrastructure and with a number of PEIS, including: static cameras, mobile robots, sensor nodes, a refrigerator equipped with gas sensors and an RFID reader, actuated lamps, window blinds and many more devices.

This testbed have been used to implement a number of experiments validating both the PEIS-Ecology concept at large and the workings of individual PEIS and PEIS-components. For more details we refer the reader to [5], [12], [13].

Additionally, a previously existing testbed for intelligent home environments at ETRI, Korea, has also been retrofitted with PEIS-components and have been run as a PEIS-Ecology with various experiments to test reconfiguration, tiny devices and the incorporation of various automated home appliances into the PEIS-Ecology. We have also performed experiments in this testbed, primarily for testing the integration of the PEIS-Ecology with components and hardware not originally designed for the PEIS-Ecology. An example run of such a scenario will be described in Section VII.

C. A Middleware for the PEIS-Ecology

When realizing a system containing many heterogeneous components, such as an ecology of robots, communication and cooperation become an important part. Commonly this role is played by a middleware acting as a collaboration layer between all involved devices, lifting the complexity of low-level communication and hiding the heterogeneity of the underlying systems.

Many different middlewares have been proposed for robotic applications, often from a different point of view and with different applications. Examples include the Player framework [7] providing an infrastructure, drivers and basic algorithms for mobile robotic tasks; RT-middleware [8] or Miro [9] which are both based on CORBA [10] functioning as a classic middleware but for robotic applications; or more specialized libraries such as Lime [11] focusing more towards sensor networks and less on remote actuation.

For the purpose of the PEIS-Ecology project many requirements have been posed on the infrastructure to be used. The primary tenet for these requirements stems from the basic concepts behind the view of ecologies of robotic devices and a need for a simple interface tailored specifically to match the concepts of PEIS-Ecologies.

These requirements include: providing a shared memory model, a simple dynamic model for self-configuration and introspection, a small footprint suitable both for very small devices and larger embedded computers, and a minimalistic API easily usable by both expert roboticists as well as component programmers with little expertise in middleware and network design. Additionally, the middleware must support heterogeneous devices ranging from simple sensors and household appliances to complex robots with powerful embedded computers. These different devices must all be automatically detected and interconnected to realize an ecology and the middleware must work both for separate devices working in isolation as well as ad-hoc groups formed when devices come within communication range. Thus the infrastructure must smoothly scale as the number of devices increase and handle when individual devices as well as large groups of devices appear and/or disappear from the network.

Although not a formal requirement these constraints seem to impose also a decentralized middleware.

Since none of the classical middlewares, or eg. the robotics middlewares described above, satisfy these requirements, a new middleware called the PEIS-kernel was developed. Although developed independently this middleware has many concepts in common with Lime [11] such as using a distributed tuplespace but is tailored rather towards robotic applications and the concepts of PEIS-Ecologies instead of ambient intelligence type of applications. During the progress of the project this middleware is continuously updated and made available to the research community under an OpenSource license and is available from the PEIS project webpage [14].

III. CONCEPTS

At a first glance, the PEIS-middleware is a set of software libraries used while developing individual PEIS-components as well as a number of specific programs responsible for fixed functionalities in each PEIS and additional software to enable debugging and visualization of the current state of the ecology. The tools and libraries exist for many platforms and comprises a software stack used for implementing the PEIS-Ecology concept. See Figure 3 for a simple illustration of how these libraries and tools are related.

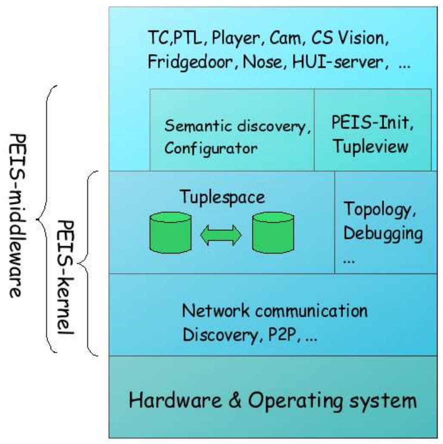

Fig. 3. The application stack used for individual PEIS-components

A. Distributed Tuplespace

From a conceptual point of view the PEIS-kernel enables each component in a PEIS to communicate and participate in the PEIS-Ecology by implementing a distributed tuplespace. This tuplespace is a decentralized version of the shared memory proposed by Linda [15], augmented with an event mechanism.

We recall that in a Linda-space a number of tuples containing keys and other data that can be stored and retrieved by any participating process using an abstract tuple where some of the fields have been initialized to wild-cards. In our version of this space a tuple has been specified to consist of a namespace, key, data as well as a number of meta attributes such as timestamps and expiration date. Separating the allowable keys into different namespaces is done not only for programming practices but is also used as an arbitration mechanism when storing and retrieving information. This is done by having the identifier for each PEIS-component as namespaces, and by using the corresponding PEIS-component as the arbiter for disambiguating all write operations to that tuple. Any PEIS-component can still store information in any namespace. We call the component corresponding to the namespace the owner of a tuple.

When components *write* to a tuple this is done, transparently by the PEIS-kernel, by sending a message to the owning PEIS-component which stores the master copy of that tuple. Depending on the order of arrival of these messages the owner commits the write operations as they come in, thus avoiding synchronization issues with simultaneous writes, and sends a notification of the modified value to all other PEIS-components which are subscribed to the specific tuples. A PEIS-component must always subscribe to a tuple before it can be accessed if it belongs to another PEIS-component. These subscriptions are created by giving an *abstract tuple* which corresponds to the tuples of interest. Furthermore, since read and write operations finishes atomically there is a latency caused by

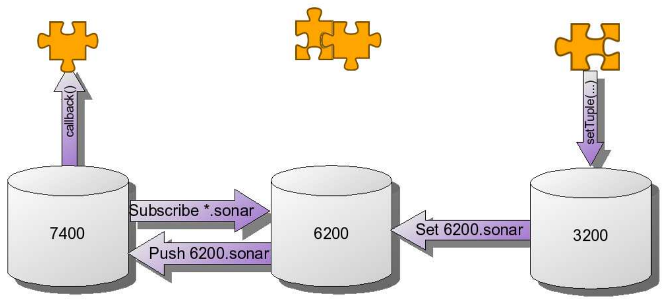

Fig. 4. Example of message passing when reading, writing and subscribing to tuples.

network delay between when a new tuple is written and the corresponding value can be read by other PEIS-components. If a PEIS-component reads a value before the latest value has arrived over the network, it receives the old value.

Upon receiving a notification that a tuple matching an abstract tuple has been modified, the receiving PEIS-component performs two things. First, the modified value of the tuple is stored in a local database containing the last observed value of that tuple, because of network latency this is sometimes not the real latest value of the tuple. This allows local *read* operations, returning all tuples matching a prototypical abstract tuple, to complete instantly. Second, if a callback function has been registered to an abstract tuple matching the modified tuple then this callback is invoked. This invocation is performed within the flow control of the PEIS-kernel and has some restrictions on callable kernel functionalities to ensure responsiveness of the kernel. Thus the kernel implements not only a pure distributed tuple space but also allows an event based flow control.

As an example, consider a situation where we have three components with the identfiers 3200, 6200 and 7400 as given in Figure 4. Assume that the later component, 7400, have registered a subscription and a callback to all tuples that have key ”sonar” in all namespaces. This is done by giving a wildcard in the first namespace field and the PEIS-kernel will send subscription messages to all relevant components. When the first component, 3200, writes to a tuple with the namespace 6200 and the key ”sonar”, a message is transparantly sent to the PEIS-kernel inside component 6200. The receiving PEIS-kernel will then update the tuplespace with this new value and alert the PEIS-kernel inside 7400 since it's has a registered subscription. When the later receives this updated tuplevalue, it in turn calls the registered callback function on the local processor before continuing the execution of any other PEIS-kernel related activities.

Other modifications to the Linda concept includes the use of timestamps for when tuples are created or given as user defined time stamp, as well as dates for expiring outdated tuples. To allow timestamps to be meaningful a distributed clock is implemented in the PEIS-kernel to ensure that all devices uses the same frame of reference.

B. *Configurations*

When creating an ecology of cooperating robotic components the issue of configuration is an important problem under much consideration. Solving this problem is akin to answering the questions of *which* components should collaborate, *what* data should be communicated and *how* these collaborations can be realized. The first two of these questions can be solved using planning techniques [16] while the last question translates into specific requirements on the used middleware to collect the information required to compute configurations as well as providing the mechanism to deploy configurations.

The first of these requirements is realized by using a fixed functionality component, PeisInit, which runs on all PEIS and provides a list of available PEIS-component which can be instantiated. This list is built up from the content of a configuration file on each PEIS listing the available resources and programs in an XML based format describing the semantics of the components. This information can easily be retrieved from the tuplespace by planners and once *which* components to be run has been determined these are started and monitored by the PeisInit. If a component is terminated abruptly this is detected by PeisInit, a failure is signaled in the tuple space and the component is restarted as needed.

For the question of *what* data to communicate and how to set up the collaborations between participating components this uses another concept of the PEIS-Ecology middleware, *meta tuples*. A meta tuple is a tuple whose data is a pointer to another tuple to be used. By implementing components to read inputs from named meta tuples in their own tuple space and produce outputs to concrete tuples we achieve an easy way of configuring components. Imagine that we have a vision component V which consumes the meta tuple V.IMAGE to produce a list of recognized objects V.OBJECTS. To allow this component to use the data produced by a camera component C we set the value of V.IMAGE to be C.IMAGE, where the later is the name of the camera images produced by C. This notion of meta tuples have been implemented in the PEIS-kernel to allow any components to be configured in a consistent way.

When a PEIS is powered on but before any communications with other PEIS have been established, it counts as an ecology consisting of only the components inside itself. As other PEIS are detected by the available communication devices and connections are established the ecologies are merged. From the component programmers point of view this is evident in the appearance of new tuples residing in the other PEIS. Similarly, when one or more PEIS are dropped from the ecology this is noted by the remaining PEIS-component which makes tuples belonging to these components lost. They are then dropped from the local cache of tuples and attempts to modify these tuples results in an error, possibly leading to a *reconfiguration* of the ecology. Since tuples are used for closed (control) loops used in robotic devices it is important to not queue tuple modifications, otherwise unwanted actions would be performed when the PEIS are reintroduced into the ecology.

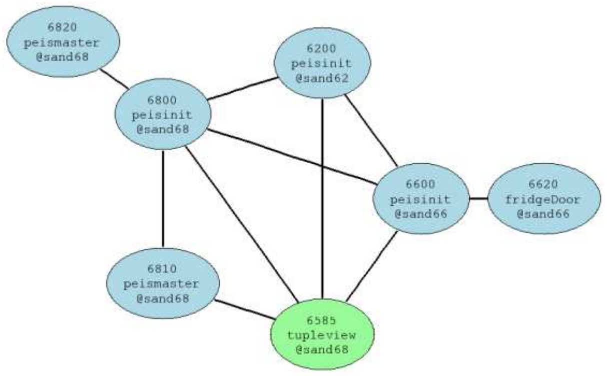

Fig. 5. Network topology for a simple PEIS-Ecology.

IV. MIDDLEWARE IMPLEMENTATION

In the implementation of the PEIS-kernel a layered structure has been used to build up the various services needed. See Figure 3 for an illustration of the layers of the PEIS-kernel.

A. Communication layer

At the lowermost level an abstraction of the operating system specific communication methods isi used to provide potential communication links and device detection for shared medias, eg. using TCP/IP broadcasts on wireless lan networks. In addition to the operating system functionalities this layer also provides general services for initializations, calling functionalities periodically etc.

On top of this layer the general peer-to-peer network is implemented using standardized algorithms for eg. optimizing connectivity and performing routing.

To realize the communications necessary to implement the distributed space, the PEIS-kernel detects and connect all PEIS devices in the local environment. This is done by abstracting the available communication devices on all participating PEIS and by establishing a peer-to-peer network between all detected devices. This allows PEIS without a direct means of communication to still share tuples. See Figure 5 for an example situation when some PEIS are used to route messages between other PEIS. Note that this routing even 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 PEIS as a bridge. An example of the later case is the embedding of Tiny sensor motes in the ecology with limited memory (10kb) and communication (802.15.4) capabilities which communicates as peers with the larger robots via a PEIS connected to both the 802.15.4 network and the wireless network. This later PEIS also serves as a bridge translating message to a more compact protocol suitable on the low-bandwidth network.

The services exported from this layer includes the transmission of routed point-to-point messages and of broadcasted messages. Message broadcasting uses a stochastic method only propagating fresh messages along a subset of the available connections, thus only consuming a global bandwidth consumption of $3N$ instead of VN where N is the number of nodes reached by a message and V is the average connectivity. Due to the use of broadcasting the total amount of bandwidth consumed per participating PEIS grows as the number of PEIS-components grows. Nonetheless, successfull tests have been performed with several hundred PEIS-components.

Since communications can go through multiple hops before reaching the destination the issue of congestion control becomes important. The method used here is based on the standard weighted random early detection (WRED) algorithm which works well to ensure a higher QoS for control messages and meta data.

B. Tuple layer

Finally, on top of the peer-to-peer network layer the actual functionalities for maintaining the distributed tuplespace and the API functionalities used by component programs are implemented. By implementing a small database for the storage of tuples each PEIS-component can be used to store all relevant tuples. To avoid some of the problems typically associated to distributed databases the namespaces of tuples are used to circumvent many of the synchronization issues.

In the implementation of these databases special attention has been given to abstract tuples. An abstract tuple is here a tuple in which one or more fields have been initialized to a wild-card value while the remaining fields have been given a concrete value. When comparing abstract tuples to concrete tuples an abstract tuple is said to match a tuple if all non wild-card fields are equal to the corresponding fields of the concrete tuple. When comparing two abstract tuples the first is said to be a generalization of the second if every non wild-card field of the second is matched by either an equal value or a wild-card field of the first, and if every wild-card field of the second is matched by a wild-card field of the first. As an example, consider the tuples $T_1 = (a, *, *)$, $T_2 = (a, b, *)$ and $T_3 = (a, *, b)$. We see here that T_1 is a generalization of T_2 and T_3 while the later two are unrelated to each other.

Abstract tuples are used for three purposes in the PEIS-Ecology middleware: Firstly, they are used whenever an application is accessing the database to query the current value of a tuple. In this case an abstract tuple is given as a prototype and a search is made to find all tuples matching that prototype. Secondly, before a PEIS can access tuples at a remote location, a subscription to the corresponding tuples must be made. Again, this takes the form of an abstract tuple given as a prototype and the PEIS-kernel sends messages to all PEIS-components which could possibly have a matching tuple, depending on if there is a wild-card on the owner field. If a tuple is later created, modified or a PEIS-component with a tuple matching the prototype is connected then a copy of this tuple is propagated to the original PEIS-component. Thirdly, abstract tuples are used by the event mechanism to setup callbacks when tuples changes value. In practice this happens by matching all received tuple change notifications against the

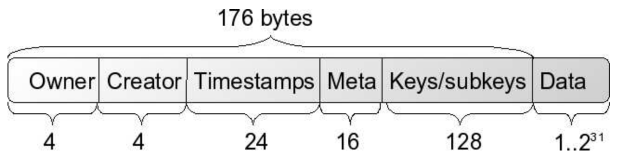

Fig. 6. Contents and sizes of an ordinary tuple in the PEIS-Ecology.

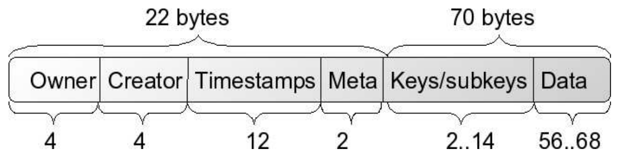

Fig. 7. Contents and sizes of a Tiny tuple.

prototypes of all registered callbacks. Those callbacks which match the concrete tuple is invoked with it as an argument.

By using the read operation to access tuples it is possible to poll for the current value of tuples, this is commonly used within control loops where only the latest value is of interest. For event driven applications or applications responding to all tuple changes the callback mechanism is typically used instead.

V. TINY PEIS KERNEL

By implementing the above software stack in first desktop machines and robots with powerful computers, this approach have been shown successful for scenarios including multiple robots, static sensors, and custom-made home appliances [5], [17], [18]. Some of these scenarios also included the automatic self-configuration of the ecology [16], [19]. These experiments provided us with a preliminary validation of the concept of PEIS-Ecology. In order to push the PEIS-Ecology vision further, however, we need to enable even smaller devices to be part of a PEIS-Ecology. This requires a middleware that needs to be tailored to meet the requirements of tiny networked embedded devices with as little as 4kb of memory.

For this purpose, we have extended the PEIS-middleware to allow for computationally simple devices to take part of a PEIS-Ecology as first class citizens. In doing so, we had two important requirements in mind. First, tiny embedded devices should be functionally equivalent to standard PEIS, that is, they should appear as standard PEIS to the rest of the PEIS-Ecology, using the same abstraction, communication and co-operation models. Said differently, we do not want to develop a separate, *ad-hoc* middleware model to suit the restrictions of tiny devices. Second, we want to be able to include in the PEIS-Ecology off-the-shelf tiny embedded devices, like micro controllers and WSN motes. Said differently, we do not want to restrict tiny PEIS to run on some specific *ad-hoc* hardware platform and we do not want to develop a separate *ad-hoc* middleware model to suit the restrictions of tiny devices.

This extension of the PEIS-middleware is implemented by providing a small version of the PEIS-kernel which can be run on top of TinyOS, a common operating system for motes and other tiny platforms. We refer to this slimmed down version of the PEIS-kernel as the Tiny PEIS-kernel and have performed a number of experiments with this kernel using small sensor motes and motes capable of actuation. This version of the PEIS-kernel requires at most 35 KB of programming memory, including the used TinyOS components, and about 2.5 KB of RAM. It does not assume that an external flash is available. Also, the actual footprint of the program may be smaller if an application only uses some functions in the PEIS-Kernel. For further details about the Tiny PEIS-kernel see Bordignon *et.al.*[20].

For the communication infrastructure, the Tiny PEIS-kernel uses 802.15.4 as the PHY/MAC layers. The main reasons for this choice are: the fact that it is meant for small, cheap devices; and its widespread availability and low price, given by the industrial interest behind it. Moreover, this choice enables us, if desired, to support ZigBee profiles, e.g., for lighting control, which would facilitate the interoperation of a PEIS-Ecology with standard home automation components.

One limitation with this communication choice is that we are not guaranteed the exchange radio packets of more than 100 bytes. As such, some internals of the communication and tuple layer of the PEIS-kernel have been modified to exchange compressed packages which are translated between the large and small protocol when entering or exiting the 802.15.4 or the TCP/IP part of a PEIS-Ecology. These transcodings are done automatically by one of the PEIS-software tools, the TinyGateway, which links different networks and are transparent to the component developers. See Figure 6 and 7 for an illustration of the sizes of these packages.

VI. SOFTWARE TOOLS

Apart from the PEIS-kernel library implementing the P2P network and the distributed tuplespace, we also have number of other tools necessary for reading meta information about the PEIS-Ecology and for launching, monitoring and stopping components residing at different PEIS. Two notable such tools are the PeisInit component which is a standard component which should run at all PEIS, and the TupleView component which can be used as a debugging tool and for bootstrapping goals into the PEIS-Ecology.

The first of these two tools was described briefly in Section III and is automatically started on all PEIS when they boot up. The later tool, Tupleview, consists of a graphical user interface which can display all the available PEIS, PEIS-components and can display and allow the user to modify all tuples. Tuples with a more complex encoding, such as images, are displayed by special handlers which allows the components to use this tool as a simple user interface (display and receiving feedback) during development. In addition to displaying the content of the distributed tuplespace this tool is also used to display the network topology of all participating components and to visualize all established collaboration patterns (subscriptions). See Figure 8 for a screenshot of this tool when displaying the current network topology, or Figure 10 for a view of a configuration graph.

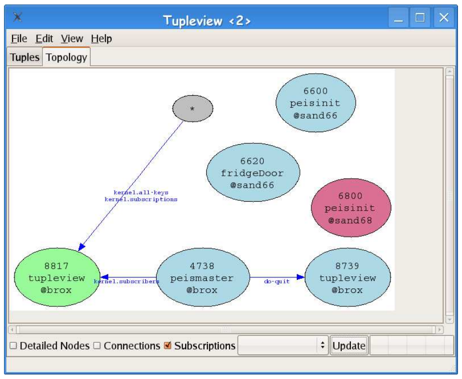

Fig. 8. Illustration of an ecology with seven components and four established subscriptions

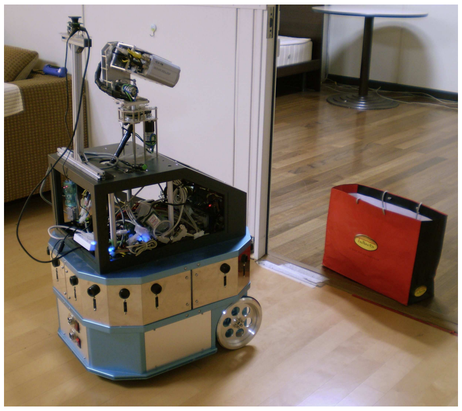

Fig. 9. The robot encounters an unexpected parcel when entering the bedroom

VII. Example run

As an example to illustrate the capabilities of the PEIS-Ecology concept in general and the middleware used we describe here one of the scenarios run in the ETRI testbed mentioned in Section II.

A. Scenario

The setting of this scenario is an intelligent home containing a few specialized devices for manipulating home appliances, as well as a full autonomous mobile robot. In this actual run, taking place early in the morning, the task of the robot is to navigate into the bedroom to wake up the inhabitant. During this task, the robot will interact and use information from sources such as; a localization system in the ceiling, controllable lamps, a door equipped with a magnetic sensor, a plant with a mote monitoring it's health, controllable window blinds, as well as an unknown parcel blocking the entrance to the bedroom. The late both constitutes a problem in the inital goal of reaching the bedroom and provide a solution by exporting it's weight as a readable propery in the tuplespace. Thus after replanning and establishing a one-to-one mapping between the digital representation of the box and the physical representation of it, the robot is able to determine that the object can be pushed and can proceed in waking up the inhabitant and returning to the resting state.

B. Execution

The execution of the experiment started with running a static configuration script consisting of a bash-script which requests all the necessary components to be started by issuing tuples belonging to the various PeisInit components that always run on all participating PEIS. Next the configuration script wrote the necessary meta tuples to setup the configuration. Note that both these two tasks could as well have been achieved using the dynamic configuration services [16] if deemed neccessary. See Figure 10 for an overview of this configuration involving 17 components and 77 subscriptions.

Next, we gave the high-level planner the goal that Johanna, the inhabitant of the apartment, should be woken up and that the robot should in the end be in the dedicated robot waiting area. This was expressed as a predicate logic goal:

```
(and (woken-up johanna) (at wever = garage))
```

In order to resolve this goal, the PTL planner component collected all world specification tuples (*.SPEC) in the tuple space and used this as the initial world situation to plan from. For example, the automated lamps in the environment provided this subset of the initial specification:

```
(and (lamp lamp-1) (in lamp-1 = bedroom)
     (on lamp-1 = f) (lamp lamp-2)
     (in lamp-2 = livingroom) ...)}
```

By using a domain description giving actions such as moving the robot with a precondition that the lights are on and actions for eg. turning on the lamps, the planner generated a task to turn on the lights necessary for safe robot navigation (the vision algorithms require this), moving into the bedroom and issuing a wake up message using the text-to-speech services. As the final step the plan involved the robot returning back to the waiting area.

This sequence of actions was used by the execution monitor to execute the actions, step by step, using the different components accessed via the tuplespace. As the robot was executing this sequence of actions, the home monitoring component in the ecology reacted to the changed light level reaching the mote supervising Johanna's favourite plant, and as such reacted by closing the curtains controlled by a home automation ZigBee network. During this part messages was passed between 802.15.4 devices and the wired/wireless 802.11 network via those PEIS-components which had capabilities to communicate on both networks.

Later in the execution the robot encountered an unknown package blocking the way into the bedroom, see Figure 9 and an automatic replanning to satisfy the goal was performed. Again receiving descriptions from all available PEIS containing their status, seen objects etc. The newly generated plan involved a query to test the weight of the box, and if possible to push it into the bedroom before resuming the original task. This plan was executed successfully thanks to a PEIS component publishing the weight and appearance of the box in the tuplespace, and thus Johanna was woken up and the robot continued by moving to it's resting place.

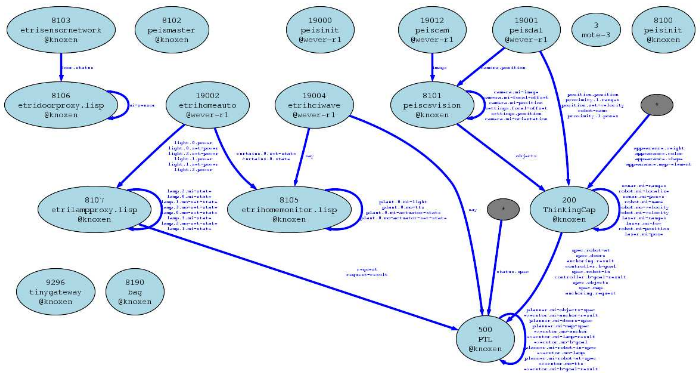

Fig. 10. Configuration snapshot of experiment run. Here components are represented by large ovals and subscriptions given by arrows annoted with the subscribed keys.

VIII. Conclusions

The integration of robots and smart environments is believed by many to be the key factor that will enable the massive inclusion of robotic technologies and products into our everyday life. A few projects were recently started with the aim to explore the scientific, technological and practical implications of this integration. Currently the largest efforts are probably the Network Robot Forum [1], the U-RT project at AIST [21], and the Korean Ubiquitous Robot Companion program [4]. The PEIS-Ecology project presented in this paper is part of the latter effort. The PEIS-Ecology project is distinct in its strong emphasis on the study of the fundamental scientific principles that underlie the design and operation of an ubiquitous robotic system.

In this paper, we have discussed the implementation of a middleware for the PEIS-Ecology approach. We believe that many of the requirements and techniques used for this robotic middleware are not limited to a PEIS-Ecology but also apply to any ubiquitous robotic system.

To the best of our knowledge, no comparable middleware was available until now. Today's robot middlewares are typically too heavy to run on tiny devices; and middleware for networked embedded devices are typically not powerful enough to support the complex cooperation models allowed by a PEIS-Ecology. An interesting exception is the RT-Middleware [8], for which a light-weight version has been defined [22]. However, this version runs on specialized devices. By contrast, our PEIS-Ecology middleware is intended to run on custom as well as off-the-shelf hardware, including standard WSN motes and hobbyist micro-controller boards.

We have run a large number of experiments on the PEIS-Ecology with many different ubiquitous robotic devices accomplishing various tasks. For details about some of these experiments see [5], [18], [19], [17], [20].

IX. Acknowledgments

The authors would like to express their gratitude to Prof. Alessandro Saffiotti for his work on the PEIS concept and to Jayedur Rashid and Mirko Bordignon for their implementation of the TinyOS version of the PEIS-middleware. This work has been supported by ETRI (Electronics and Telecommunications Research Institute, Korea) through the project "Embedded Component Technology and Standardization for URC (2004-2008)" and Vetenskapsrdet.

References

[1] "Network Robot Forum," www.scat.or.jp/nrf/English/.

[2] J. Lee and H. Hashimoto, "Intelligent space – concept and contents," *Advanced Robotics*, vol. 16, no. 3, pp. 265–280, 2002.

[3] F. Dressler, "Self-organization in autonomous sensor/actuator networks," in *Proc of the 19th IEEE Int Conf on Architecture of Computing Systems*, 2006.

[4] J. Kim, Y. Kim, and K. Lee, "The third generation of robotics: Ubiquitous robot," in *Proc of the 2nd Int Conf on Autonomous Robots and Agents*, Palmerston North, New Zealand, 2004.

[5] A. Saffiotti and M. Broxvall, "PEIS ecologies: Ambient intelligence meets autonomous robotics," in *Proc of the Int Conf on Smart Objects and Ambient Intelligence (sOc-EUSAI)*, Grenoble, France, 2005, pp. 275–280.

[6] J. Gibson, *An ecological approach to visual perception*. Boston, MA: Houghton Mifflin, 1979.

[7] T. H. Collett, B. A. MacDonald, and B. P. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Proc. of the Australasian Conference on Robotics and Automation (ACRA)*, Sydney, Australia, 2005.

[8] N. Ando, T. Suehiro, K. Kitagaki, and T. Kotoku, "RT-middleware: distributed component middleware for RT (robot technology)," in *Int Conf on Intelligent Robots and Systems*, 2005, pp. 3933–3938.

[9] S. Enderle, H. Utz, S. Sablatng, S. Simon, G. Kraetzschmar, and G. Palm, "Miro: Middleware for autonomous mobile robots." [Online]. Available: citeseer.ist.psu.edu/enderle01miro.html

[10] OMG, "Corba/iiop specification," Object Management Group, Inc., 2000.

[11] A. L. Murphy, G. P. Picco, and G.-C. Roman, "Lime: A middleware for physical and logical mobility," in *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, Phoenix, AZ, USA, 2001.

[12] A. Saffiotti, M. Broxvall, B. Seo, and Y. Cho, "The PEIS-ecology project: a progress report," in *Proc. of the ICRA-07 Workshop on Network Robot Systems*, Rome, Italy, 2007, pp. 16–22, online at http://www.aass.oru.se/˜asaffio/.

[13] A. Saffiotti, M. Broxvall, B. Seo, and Y. Cho, "Steps toward an ecology of physically embedded intelligent systems," in *Proc of the 3rd Int Conf on Ubiquitous Robots and Ambient Intelligence*, Seoul, Korea, 2006.

[14] "The PEIS ecology project," Official web site, www.aass.oru.se/˜peis/.

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

[16] R. Lundh, L. Karlsson, and A. Saffiotti, "Plan-based configuration of an ecology of robots," in *Proc of the IEEE Int Conf on Robotics and Automation*, Rome, Italy, 2007.

[17] A. Loutfi, M. Broxvall, S. Coradeschi, and A. Saffiotti, "An ecological approach to odour recognition in intelligent environments," in *Proc of the IEEE Int Conf on Robotics and Automation*, Orlando, FL, 2006.

[18] M. Broxvall, M. Gritti, A. Saffiotti, B. Seo, and Y. Cho, "PEIS ecology: Integrating robots into smart environments," in *Proc of the IEEE Int Conf on Robotics and Automation*, Orlando, FL, 2006.

[19] M. Gritti, M. Broxvall, and A. Saffiotti, "Reactive self-configuration of an ecology of robots," in *Proc of the ICRA-07 Workshop on Network Robot Systems*, Rome, Italy, 2007.

[20] M. Bordignon, J. Rashid, M. Broxvall, and A. Saffiotti, "Seamless integration of robots and tiny embedded devices in a peis-ecology," in *Proc of the IEEE/RSJ Int Conf on Intelligent Robots and Systems (IROS)*, San Diego, CA, 2007, online at http://www.aass.oru.se/˜asaffio/.

[21] O. Lemaire, K. Ohba, and S. Hirai, "Dynamic integration of ubiquitous robotic systems using ontologies and the rt middleware," in *Proc of the 3rd Int Conf on Ubiquitous Robots and Ambient Intelligence*, Seoul, Korea, 2006.

[22] Y. Tsuchiya, M. Mizukawa, T. Suehiro, N. Ando, H. Nakamoto, and A. Ikezoe, "Development of light-weight RT-component (LwRTC) on embedded processor," in *Proc of the SICE-ICASE Conf*, 2006, pp. 2618–2622.