

Seamless Integration of Robots and Tiny Embedded Devices in a PEIS-Ecology

M. Bordignon¹

Intelligent Autonomous Systems Laboratory
Dept. of Information Eng., University of Padova, Italy
mirko.bordignon@ieee.org

J. Rashid, M. Broxvall, A. Saffiotti

AASS Mobile Robotics Laboratory
Dept. of Technology, University of Örebro, Sweden
{jayedur.rashid,mathias.broxvall,alessandro.saffiotti}@aass.oru.se

Abstract—The fields of autonomous robotics and ambient intelligence are converging toward the vision of smart robotic environments, in which tasks are performed via the cooperation of many networked robotic devices. To enable this vision, we need a common communication and cooperation model that can be shared between robotic devices at different scales, ranging from standard mobile robots to tiny embedded devices. Unfortunately, today’s robot middlewares are too heavy to run on tiny devices, and middlewares for embedded devices are too simple to support the cooperation models needed by an autonomous smart environment. In this paper, we propose a middleware model which allows the seamless integration of standard robots and simple off-the-shelf embedded devices. 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 show an experiment in which a mobile robot, a commercial mote, and a custom-built mote cooperate in a home service scenario.

Index Terms—Ubiquitous robotics, network robot systems, sensor-actuator networks, middleware, open-source, TinyOS.

I. INTRODUCTION

There is a marked tendency today toward the embedding of intelligent, networked robotic devices in our homes and offices. An interesting case is the emergence of a paradigm in which many robotic devices, pervasively embedded in the environment, cooperate to perform possibly complex tasks. Instances of this paradigm include network robot systems [1], intelligent spaces [2], sensor-actuator networks [3], ubiquitous robotics [4], and PEIS-Ecology [5]. In this paper, we generically refer to a system of this type as an “ecology of robots”.

Common to these systems is the idea that tasks are not performed by a single, very capable robot (e.g., a humanoid robot butler), but they are performed by the entire ecology via the cooperation of many simpler networked robotic devices. The term “robotic device” is meant here in a wide sense: any embedded device with computing, communication, and sensing or actuation capabilities. In order to fully exploit the potential of this vision, then, we need to consider not only classical (stationary or mobile) robots, but also small and

inexpensive networked devices which can be pervasively distributed in the environment. These devices could, for instance, be embedded in home appliances, furniture, everyday objects like cups or books, or be worn by people.

Embedded devices of this kind are today available in the form of wireless sensor network (WSN) motes, smart objects, and hobbyist micro-controller boards. These devices have strong inherent limitations in memory, processing power and communication bandwidth, and specialized operating systems and middlewares have been developed which are able to cope with those limitations [6], [7], [8]. Unfortunately, these operating systems and middlewares are very different from the ones used to connect standard robotic devices (e.g., [9], [10]). In fact, to the best of our knowledge, there is no uniform middleware that allows the seamless integration of standard robots and (off-the-shelf or custom) tiny, resource-constrained embedded devices. Thus, in order to integrate standard robots, embedded devices, and possibly other entities (e.g., home appliances), one would have to use different middleware models and implementations, and then combine those middlewares in some way. In our opinion, this is a major obstacle to the building of a truly integrated, distributed, heterogeneous robot ecology.

In this paper, we describe a model for a robot ecology, called PEIS-Ecology, in which robots and small embedded devices can coexist and cooperate. This model has been implemented in an open-source middleware, called PEIS-Kernel, which allows us to build an ecology of robots that can be configured to achieve different tasks in a distributed, cooperative manner. Moreover, we describe a “tiny” version of the PEIS-kernel, which implements the PEIS-Ecology model with only minimal processing, memory and communication requirements. The Tiny PEIS-Kernel runs on TinyOS, one of the most popular open-source operating systems for WSN and embedded devices. Virtually any embedded device capable to run TinyOS can also run the Tiny PEIS-Kernel, and thus become a first-class citizen of a PEIS-Ecology.

In the rest of this paper, we introduce the concept of PEIS-Ecology and the PEIS-Ecology middleware, and describe the Tiny PEIS-Kernel. We then show an experiment in which devices as different as a mobile robot, a stationary PC, two commercial motes and a microcontroller board cooperate inside a PEIS-Ecology via the PEIS-Ecology middleware.

Work supported by the Swedish Research Council, and by the Electronics and Telecommunications Research Institute (Korea) through project “Embedded Component Technology and Standardization for URC (2004-2008)”.

¹Currently with the Maersk Mc-Kinney Møller Institute, University of Southern Denmark, Denmark. This work was performed while this author was a visiting student at AASS, Örebro University, Sweden.

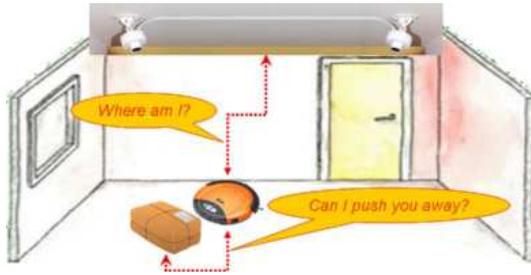


Fig. 1. A simple example of cooperation in a PEIS-Ecology.

II. THE PEIS-ECOLOGY APPROACH

The concept of PEIS-Ecology [5] provides a radically new approach toward the inclusion of robotic technologies in everyday environments. In this approach, advanced robotic functionalities are not achieved through the development of extremely advanced robots, but through the cooperation of many simple robotic components. The concept of PEIS-Ecology builds upon the following ingredients. (See [11], [12], [13] for more details.)

First, any robot in the environment is abstracted by the *uniform notion* of PEIS (Physically Embedded Intelligent System): 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 PEIS-component may include links to sensors and actuators.

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. This model has been implemented in a specific middleware, detailed below, in which PEIS communicate via a distributed tuple space.

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. In our middleware this cooperation comes from subscriptions to tuples, where a consuming PEIS automatically receives the tuples created by a producing PEIS.

Figure 1 illustrates these concepts. The robot vacuum cleaner (a PEIS) 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 (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 cleaning robot, which can thus realize smarter cleaning strategies. Suppose further that the cleaner encounters an unexpected parcel on the floor. It could push it away and clean under it, but it needs to know its weight to decide so. If the parcel is itself a PEIS, e.g., it is equipped with a WSN mote or a smart RFID tag, then it can communicate this information directly to the cleaner.

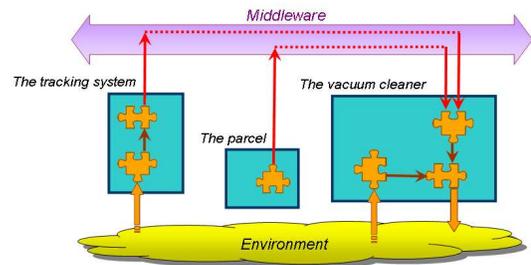


Fig. 2. Functional view of the same PEIS-Ecology.

We define a *PEIS-Ecology* to be a collection of inter-connected PEIS, all embedded in the same physical environment. A *configuration* of a PEIS-Ecology is a set of connections between components within and across the PEIS in the ecology. Figure 2 shows the configuration of the above ecology, in which all connections are mediated by a shared middleware. Note that the same ecology can be configured in different ways depending on the goal, situation, and resources.

III. PEIS-ECOLOGY MIDDLEWARE

An extensive stack of software programs has been developed for realizing PEIS-ecologies, as shown in Figure 3. The middleware in this stack, as well as several other components, have been released under an open source license [13].

At the top of the stack are a number of specific application components, which deliver the actual robotic functionalities performed in a PEIS-Ecology, like navigation, planning, image processing, and so on.

Below this layer is the PEIS-Ecology middleware. This consists of two parts: a software library used by all participating PEIS, called the PEIS-kernel; and a few special PEIS-components capable of performing advanced meta-level actions on the ecology at whole. The meta-level components are in charge of introspection, planning, self-configuration, monitoring, and debugging. In particular, the configuration components [14], [15] use an introspection mechanism to reason about the available components and create a PEIS-Ecology configuration suitable for performing the needed tasks. In our middleware, a configuration corresponds to a *set of subscriptions* between components.

The PEIS-kernel is the most important part of the middleware for the scope of this paper. This is a cross platform library which implements the PEIS abstraction discussed above, and which can be run on heterogeneous hardware and software environments. Simply put, any networked device which runs the PEIS-kernel can be seen as a PEIS, and can communicate and cooperate with other PEIS using the PEIS-Ecology communication and cooperation models.

As a communication model, we have adopted a Linda-like tuple based approach [16] augmented with an event mechanism. This approach has been selected since tuple space approaches have proven successful in robotics, ambient intelligence and sensor network applications (e.g., [17]). In our middleware, different PEIS communicate by publishing information as tuples consisting of a $\langle \text{key}, \text{data} \rangle$ pair, together

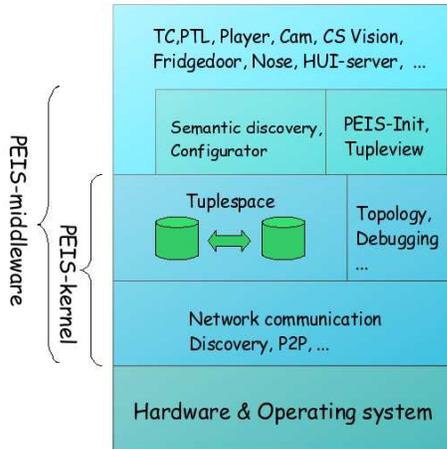


Fig. 3. Overview of the PEIS-Ecology software stack

with various pieces of meta information such as time-stamps, creator etc. The structure and sizes (in bytes) of a PEIS-tuple is shown in Figure 4. Through a search mechanism, any PEIS can find and consume the relevant tuples produced by any other PEIS, which allows for an expressive and flexible communication model. This *tuple space* thus constitutes a distributed database into which any PEIS-component can read and write information regarding any PEIS-component.

The implementation of the distributed tuple space relies on a peer-to-peer (P2P) network maintained among all detected PEIS in the ecology. This P2P network allows each PEIS to access all other PEIS and their tuples regardless of the presence of a direct communication link. The network is dynamically updated as PEIS enter or leave the ecology.

In order to simplify the management of the distributed database, all tuple keys belong to a name space, and each name space is assigned to one specific PEIS which handles all event arbitrations. This PEIS is referred to as the *owner* of a key name, and it is implicitly given in the key name by way of the name space. Each PEIS is thus the owner of exactly one name space. When a component modifies a tuple, this is first propagated to the owning PEIS-component, which performs all temporal arbitration of events and propagates copies of the value to all other interested PEIS-components. Propagation of tuples is handled by the PEIS-kernel in a way which is transparent to the individual PEIS-components.

In order to improve network efficiency, tuples are only propagated to those components which have registered an interest in that type of tuples. For this purpose, the middleware provides a subscription mechanism, and *abstract tuples* are used to register interest in different kinds of data. After having subscribed, a component can read tuples either through a direct read operation, from the local cache, or through a callback mechanism. The former allows for a value-based semantics, while the later allows for an efficient event mechanism.

Subscription to tuples is the main means to implement the cooperation model described in the previous section. A specific set of subscriptions (configuration) corresponds to a

specific data-flow in the PEIS-Ecology, which in turns leads to a specific cooperation schema. In this way the capabilities of a PEIS-Ecology is dependent not on any one specific PEIS but rather on the set of all functionalities provided by the constituent components.

A configuration can itself be represented and instantiated using tuples. This is done by creating *meta tuples* which describe, for each component, which resources it should consume (subscribe to). In the example shown in Figure 2, in order to let the vacuum cleaner (VAC) use position information produced by the overhead tracking system (TRACK), one simply creates a tuple with key `VAC.use-position` and value `TRACK.position`. This will tell VAC to create a subscription to the `position` tuples owned by TRACK. This mechanism allows the configuration components in the meta-level to automatically create and deploy configurations.

IV. TINY PEIS-KERNEL

The PEIS-Ecology software stack above has been implemented on desktop machines, robots and home appliances equipped with PC-like computers, and tested in a number of scenarios [5], [11], [12]. In order to enable the pervasive inclusion of robotic technologies in everyday environments, however, we need to allow even simpler, smaller and cheaper devices in a PEIS-Ecology. We now describe an extension of the PEIS-middleware that allows these devices to participate in a PEIS-Ecology as first class citizens.

This extension is based on two important requirements. 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 cooperation models. Said differently, we do not want to develop a separate, *ad-hoc* middleware model to suit the restrictions of simple devices. Second, it should be possible to include in the PEIS-Ecology off-the-shelf embedded devices, like microcontrollers and WSN motes. Said differently, we do not want to restrict tiny PEIS to run on some specific, *ad-hoc* hardware platform.

To meet these goals, we have designed a tiny replica of the PEIS-Kernel, called Tiny PEIS-Kernel, which provides the same functionalities of the standard PEIS-Kernel but only makes minimalist assumptions on the underlying hardware and operating system. In this section, we first describe the assumptions made at the hardware and OS level, and then detail our implementation of the Tiny PEIS-Kernel.

A. Platform assumptions

Small-sized and low-cost embedded devices, such as standard Motes or custom-built sensor-actuator platforms, are characterized by limited memory (typically tens to hundreds KB of programming flash and a few KB of RAM), low computational power, small radio packet size, and low data transfer rate. Correspondingly, we set as design requirements for our Tiny PEIS-Kernel to use at most 40 KB of programming memory, 4 KB of RAM, and exchange radio packets of

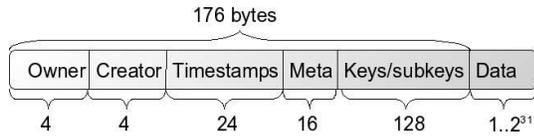


Fig. 4. Contents and sizes of an ordinary tuple in the PEIS-Ecology.

at most 100 bytes. We do not assume that an external flash is available.

At the OS level, we have decided to write the Tiny PEIS-kernel on top of TinyOS, a small, energy-efficient, open-source operating system developed within the WSN community [6]. Its design criteria make it an ideal choice to meet the tight constraints outlined above. Moreover, the strong academic and industrial following of TinyOS, coupled with its open-source policy, make it a good choice in term of long term support and community involvement. Given its availability on a wide selection of platforms and portability to nearly every architecture for which a C compiler is available, little restrictions on the hardware design for future PEIS are imposed by this choice.

For the communication infrastructure, we opted for the IEEE 802.15.4 standard as the PHY/MAC layers. The main reasons for this choice are: the fact that it is meant for small, cheap devices; its widespread availability and low price, given the industrial support behind it; and the potential to support ZigBee profiles, which would facilitate the interoperation of a PEIS-Ecology with standard home automation components.

B. The Tiny PEIS-Kernel

The Tiny PEIS-Kernel implements the same mechanisms described in the previous section: shared memory scheme for *ad-hoc* components through Linda-like tuple space, and the publish/subscribe cooperation model. The underlying P2P network is implemented using the Active Message (AM) communication stack provided by TinyOS 2.0 [18]. The Tiny PEIS-Kernel includes additional provisions to ensure reliability of communication and to deal with the dynamic nature of a PEIS-Ecology. For instance, it broadcasts a periodic beacon message to detect all existing PEIS in the ecology, and re-sends subscription messages repeatedly at a specific interval.

The assumed memory and communication restrictions do not allow us to use the ordinary tuple structure described in Figure 4. Accordingly, we have redesigned the tuple structure so that tuples can be transmitted in a single tiny network package (less than hundred bytes payload for safe transmission over the radio) and that several tuples can be stored in RAM. The new tuple format, called tiny-tuple, is shown in Figure 5. It has a fixed 22 bytes header and a 70 bytes storage field (tuple key plus data). The tuple key consists of maximum 7 sub-keys, each represented as a short integer of two bytes. The data field has variable length, depending on how many sub-keys are used.

The tiny tuple format has several practical limitations compared to the standard one, although it provides the same basic functionalities. The most notable restrictions are: all

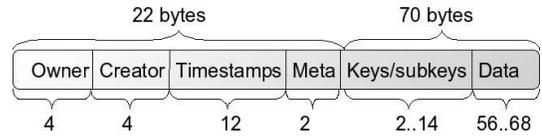


Fig. 5. Contents and sizes of a Tiny tuple.

time-stamps are 32 bits (rather than 64) thus reducing the resolution and maximum lifetime; keys are limited to 14 bytes (rather than 256) thus reducing the maximum number of possible keys and sub-keys; and data size is limited to 68 bytes (rather than 2^{31}). The last limitation is usually not a problem for tiny devices, which typically only need to exchange a few bytes of sensor data or control values.

Another important restriction in the tiny-tuple format is that keys and sub-keys use integer values, instead of strings as in the standard tuples. This means that we have to rely on a central naming registry to associate key names to key indexes. In order to perform the conversion between standard tuples and tiny tuples, we use a dedicated PEIS component called ‘tinybridge’. This component also performs the translation of tuple keys from strings to integers and back. The translation is transparent to the applications thus allowing seamless integration of normal and tiny PEIS components, so that every one of them can communicate and cooperate with the others independently of their specific types.

C. The payoff

The Tiny PEIS-Kernel requires at most 35 KB of programming memory, including the linked TinyOS components, and about 2.5 KB of RAM. The actual footprint of the program may be smaller if an application uses only some of the functions provided by the PEIS-Kernel. As for RAM, actual usage depends on the number and size of the stored tuples: the above estimate is based on 10 tuples each of 92 bytes, which in our experience is more than what is typically needed.

By running the PEIS-Kernel, any embedded device satisfying the above minimalist requirements can be turned into a PEIS, and thus it can acquire the capability to cooperate in a seamless way with all the other PEIS in the ecology, using the same mechanisms, independently on their being tiny devices or full devices. To the best of our knowledge, the PEIS-Ecology middleware is the first middleware specifically tailored to ubiquitous robotics that allows this integration.

V. AN ILLUSTRATIVE EXPERIMENT

We now show a simple experiment that illustrates the use of the PEIS-Ecology Middleware and tests its functionality.

A. Scenario

A home mobile robot, called Astrid, has received the task to take a photo of the sofa in the living room and to send it to the owner’s mobile phone. While Astrid is moving to the living room, a Mote installed on a plant near the window detects that the plant is receiving too much direct sunlight. Consequently, this Mote sends a shut request to the window blinds, which are

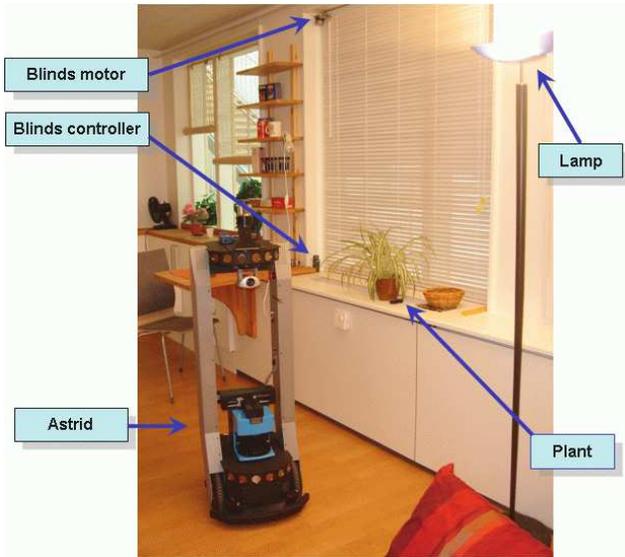


Fig. 6. A snapshot taken during the execution of the experiment. The labels indicate the main PEIS involved in this experiment.

equipped with a motor and a microcontroller. When Astrid is about to take the photo, it realizes that the light is insufficient. Since the window cannot be open due to the conflicting request from the plant, Astrid sends a turn-on request to the floor lamp in the living room, itself controlled by a Mote. Astrid can now take the photo and complete the task.

For the purposes of this paper, the interesting aspect of this scenario is that it involves several PEIS at different levels of complexity: a mobile robot, two Motes, and a microcontroller. The goal is to show that these very different platforms can smoothly interoperate and cooperate as part of one and the same PEIS-Ecology via the PEIS-Ecology middleware. Note that the logic behind the scenario, e.g., how the plant decides to send a shut request precisely to that specific window, is not what is tested in this experiment. We have shown elsewhere [14], [15] how to include mechanisms to allow a PEIS-Ecology to self-configure, that is, to decide and deploy a set of subscriptions for a given task. In this experiment, the ecology configuration has been hand-coded for simplicity.

B. Experimental setup

We run our experiment on a physical test-bed facility called the PEIS-home: a small apartment consisting of a living room, a bedroom and a small kitchen. The PEIS-Home is equipped with communication and computation infrastructure, together with a number of mobile robots, automatic appliances, and embedded sensors. For the experiment reported here, the following PEIS have been deployed — see Figure 6:

Astrid: a PeopleBot mobile robot by Activmedia, equipped with an embedded PC 104 computer and 802.11g wireless connectivity. The PC runs the full version of the PEIS-kernel over Linux, as well as a number of PEIS-components for vision, navigation, task planning, and so on.

HomePC: a 64 bit multi-core workstation, mainly used to ease the control and monitoring of the experiment. The

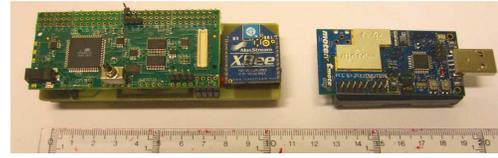


Fig. 7. A PEIS-Mote (left) and a MoteIV TMote (right).

HomePC is connected both to the 802.11g network (via an access point) and to the 802.15.4 network (via a TMote on a USB port). It runs the full PEIS-kernel over Linux, a script component to set-up our test configuration, the ‘tinybridge’ translation component, and some monitoring components.

Plant: an apartment plant with a MoteIV TMote attached to it. This mote provides 802.15.4 connectivity, and includes sensors for humidity, temperature and light. It runs the Tiny PEIS-kernel over TinyOS, and a simple PEIS-component which polls the sensors every second to detect the amount of sunlight, and sends ‘blind-open’ and ‘blind-close’ tuples if some pre-defined thresholds are exceeded.

Lamp: a floor lamp controlled by a second MoteIV TMote, connected to a light dimmer. It runs the Tiny PEIS-kernel over TinyOS, and a simple PEIS-component that changes the light intensity in response to ‘light-value’ tuples.

Blind: an electrically actuated blind, controlled by a PEIS-Mote (PMote) connected to a stepper motor driver circuit. It runs the Tiny PEIS-kernel over TinyOS, as well as a simple actuation PEIS-component that opens and closes the blinds in response to ‘blind-open’ and ‘blind-close’ tuples.

The PMote [19] is a custom-built mote platform consisting of a Robostix hobbyist microcontroller and a XBee 802.15.4 wireless module — see Figure 7. We ported TinyOS 2.0 to this platform [20], thus making it functionally equivalent to a standard mote, but with the advantages of lower cost (under 70 USD) and higher flexibility in input/output connections. By running the tiny PEIS-kernel on it, the PMote can serve as a small, inexpensive PEIS that can be easily embedded in devices, appliances, and furniture.

C. Execution

Before running our experiment, the above PEIS are configured as shown in Figure 8. The Plant is sending tuples to the Blind, and Astrid is sending tuples to the Lamp. Other PEIS-components (e.g., those used for robot navigation) are not shown since they are not the focus of this experiment.

When the experiment begins, Astrid starts to execute its plan to move close to the sofa and take a photo. At the same time, the PEIS Plant detects too much direct sunlight, so it sends a request to the PEIS Blind to shut. This is done by the Lamp posting the tiny tuple (simplified syntax)

`<3, 43, 0>`

where 3 is the ID of the Plant, 43 is the index of the key `TiltValue`, and 0 indicates fully closed blinds. Since Blind is subscribed to tuples with key 43 from PEIS 3, the PEIS-kernel routes this tuple to Blind and notifies it. Blind then closes the blinds.

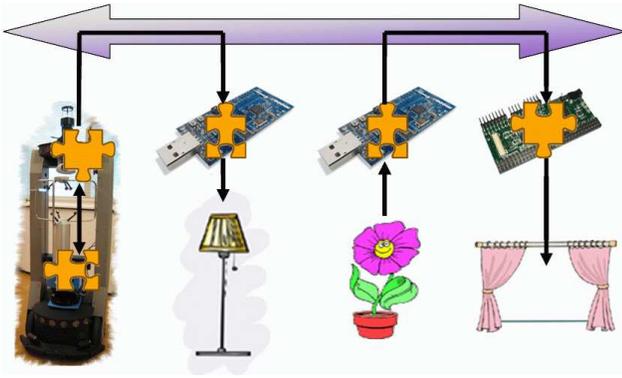


Fig. 8. The PEIS-Ecology configuration used in our example scenario.

When Astrid finds that the camera images are too dark, it sends a request to turn on the floor lamp by posting the tuple `<6890, "LightValue", 100>`

where 6890 is the ID of the plan executor component inside Astrid, and `LightValue` is the key, given as a string because Astrid is running the full PEIS-kernel. Since Lamp is subscribed to tuples with key `LightValue` from PEIS 6890, the PEIS-kernel routes this tuple to Lamp and notifies it. Lamp then turns the light on. Notice that when this tuple is routed to Lamp through the tinybridge component, it is converted to the tiny tuple

`<6890, 42, 100>`

where 42 is the index of the key `LightValue`.

Finally, Astrid takes a snapshot photo with sufficient brightness and the experiment concludes.

VI. CONCLUSIONS

The main contribution of this paper is to propose PEIS-Ecology as a common middleware model, which allows the seamless integration of standard robots and off-the-shelf tiny embedded devices. This model is suitable for building truly ubiquitous robotic applications, in which devices of very different scales and capabilities can cooperate in a uniform way. Our implementation of this model is available as open-source [13], and has been tested on hardware platforms ranging from multi-core 64 bit processors down to 8-bit microcontrollers, and on software environments such as Linux, MacOS, Windows, TinyOS and embedded Linux.

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 [10], for which a light-weight version has been defined [21]. 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 microcontroller boards.

While the experiment described here is very simple, it shows that normal and tiny PEIS can coexist and be treated

in the same way. In particular, this will allow us to use our existing mechanisms for self-configuration [14], [15] to automatically configure and reconfigure a mixed PEIS-Ecology including both robots and tiny devices. The performance of more involved hybrid scenarios is our next research priority.

Our Tiny PEIS-Kernel only requires very few computational resources. It might be claimed that future motes are likely to have less restricted resources. Nonetheless, we do not plan to relax our minimalist requirements: this will allow us to take advantage of the dropping cost and size of hardware, and eventually create tiny-PEIS that can be embedded in any object in the environment with minimal cost, size and energy overhead.

REFERENCES

- [1] "Network Robot Forum," www.scats.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 Int Conf on Architecture of Computing Sys*, 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 (ICARA)*, 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)*, 2005, pp. 275–280.
- [6] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An operating system for sensor networks," *Ambient Intelligence*, 2005.
- [7] J. Branch, J. II, D. Sow, and C. Bisdikian, "Sentire: A framework for building middleware for sensor and actuator networks," in *Proc of the IEEE Int Conf on Pervasive Computing*, 2005, pp. 396–400.
- [8] H. Noguchi, T. Mori, and T. Sato, "Network middleware for flexible integration of sensor processing in home environment," in *Proc of IEEE Int Conf on System Man and Cybernetics*, 2004, pp. 3845–3851.
- [9] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, "Miro – middleware for mobile robot applications," *IEEE Tran on Robotics and Automation*, vol. 18, no. 4, pp. 493–497, 2002.
- [10] 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.
- [11] 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 (ICRA)*, 2006, pp. 212–218.
- [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*, 2007, pp. 16–22.
- [13] "PEIS ecology homepage," www.aass.oru.se/~peis.
- [14] 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 (ICRA)*, Rome, Italy, 2007.
- [15] 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.
- [16] D. Gelernter, "Generative communication in Linda," *ACM Tran on Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, 1985.
- [17] A. Murphy, G. Picco, and G. Roman, "Lime: A coordination middleware supporting mobility of hosts and agents," *ACM Tran on Software Engineering and Methodology*, vol. 15, no. 3, pp. 279–328, 2006.
- [18] P. Buonadonna, J. Hill, and D. Culler, "Active message communication for tiny networked sensors," in *Proc of the 20th Joint Conf of the IEEE Computer and Communications Soc*, 2001.
- [19] M. Bordignon, E. Pagello, and A. Saffiotti, "An inexpensive, off-the-shelf platform for networked embedded robotics," in *Proc of the Int Conf on Robot Communication and Coordination*, Athens, GR, 2007.
- [20] M. Bordignon, "An open framework for networked embedded robotics," Master's thesis, University of Padova, 2007.
- [21] 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.