Digital Representation of Everyday Objects in a Robot Ecology via Proxies

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*— Robotic middlewares increasingly allow the seamless integration of multiple heterogeneous robots into one distributed system. Unfortunately, very simple devices like tagged everyday objects and smart objects are left orphan in this otherwise pervasive trend. We claim that the inclusion of simple everyday objects as part of distributed robot systems would have many advantages, and propose a design pattern to allow this inclusion. We make this pattern concrete by describing an implementation of it using a specific multi-robot middleware, called PEIS-Ecology Middleware. We also show an illustrative experiment which integrates everyday objects in a smart home equipped with mobile robots as well as more advanced distributed sensor nodes.**

I. Introduction

Distributed multi-robot systems are becoming increasingly common in the robotic field [1], [2], [3]. These systems often rely on some middleware to provide a common abstraction of the robotic devices in the system, as well as communication mechanisms that allow these devices to seamlessly exchange data and commands. Many of these middlewares allow heterogeneous robots and devices to be part of one and the same system, provided that they have the required computation and communication resources.

Robots in a distributed robot system can access properties and send commands among each-other via this middleware. However, if they need to access the properties of simpler objects and devices that lack the required resources, they have to resort to some other means. For instance, a robot equipped with an RFID tag reader can access the information stored in the RFID tag attached to a parcel; and a robot equipped with a ZigBee interface can turn on a lamp controlled by a simple ZigBee based module. The parcel and the lamp, however, remain external to the distributed robotic system.

The starting point of this paper is the realization that it would be extremely useful to make everyday objects and simple devices part of a distributed robot system, by making them accessible throughout the system via the same middleware [4]. Consider for instance a transport robot T perceiving a smaller robot S, which is part of the same distributed system. By using techniques such as anchoring [5], T can realize that its perception of S and the digital representation of S in the distributed system refer to the same physical object. Thus T can acquire needed physical properties of S by querying its digital representation through the middleware — e.g., its grasping points, in order to be grasped. Suppose now that C is a coffee cup with an RFID tag that stores, among other things, its grasping points. If C could be accessed by T through the middleware in the same way as S was, then T would have a simple uniform way of addressing the grasping task.

Today's robotic middlewares are usually limited to be run on relatively powerful processors [6], [7] and cannot incorporate very small devices or everyday objects. Some previous works have shown robotic middlewares that with help of a light-weight/tiny version can also incorporate simple devices, like microcontrollers or wireless sensor network motes [7], [8], [9], [10]. Although simple, these devices are assumed to be able to run software capable of communicating and interfacing with the general middleware. In this paper, we go one step further, and we propose a technique to include in a distributed system even objects and devices which are unable to run this software. These may be, for instance, everyday objects attached to RFID tags, or augmented with simple sensors and small communication modules (e.g., based on infrared communication or ZigBee) [11]. Our technique is based on the notion of *proxy*: a process hosted by a component of the distributed robot system, which acts as a representative of the simple object inside the middleware. The proxy maintains an image of the external object which is made accessible to the middleware, and it uses a dedicated communication channel to synchronize this information with the actual object (e.g., an RFID reader, or a ZigBee radio module).

In the next section, we elaborate the above idea into a general design pattern. We then show an implementation of this pattern using a specific multi-robot framework, called PEIS-Ecology. Finally we describe an experiement, which incorporates everyday objects in the PEIS-Ecology using the proposed technique, as a proof of concept, and conclude.

II. Concept of Proxied Objects

The basic idea of proxied objects is to give a representation of everyday objects which is consistent with how other objects such as other robots are represented. By giving a uniform mechanism for interfacing to any kind of object, we can easily query it for its capabilities and properties or ask it to perform tasks regardless of if it is a fancy robot or a simple coffee cup. As lofty as this goal may seem, and in spite of the limited capabilities of simple devices (regardless of the representation, a coffee cup just cannot brew itself),

Proceedings of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS-08) pp. 1908-1914
Nice, France, September 22--26, 2008

we will see that this slight shift in viewpoint gives a number of advantages.

In this section we describe a general design pattern for implementing this viewpoint and outline the various components and the needed information flow for this to happen.

From a hardware point of view we require, obviously, the objects to be proxied. These objects can be any kind of object to which we have any form of communication channel. Examples include, for instance, everyday objects equipped with RFID tags or simple sensors and actuators connected with ZigBee modules or Radio modems to transmit/receive analog and digital signals.

In addition to these objects, we also require components that can perform the communication with the proxied objects, eg. an RFID reader or a ZigBee radio link, and that can relay these communications to the general middleware and connected robotic devices. We call these devices *interface* components. These devices need not perform any major operations or interpretations on the communicated data, but only transform them into a form usable by other components. This is necessary since the same proxied object may be communicating with different interface components at different timepoints or even simultaneously.

Last, but not least, we need to run two more types of components on devices with sufficient computational power to let multiple components participate in the general middleware. The first type of component that we need to run on these machines are the *proxy* components themselves. For the proxy components to be able to give meaningful information about the proxied objects, they obviously need knowledge about their existence and state. Some of this information is given implicitly by the kind of object being proxied, some can exist a-priori from knowledge databases and some are dynamic and must be derived from the object at run-time. Thus, these components must receive an information flow from all interface components which are currently capable of communicating with the proxied objects. The task of these proxy components is to fuse the information from different interfaces and to interpret and present the data into a format usable by the general middleware. Each proxy component is responsible for presenting information from one specific instance of a proxied object determined at its instantiation time and presents the fused information as if it were the proxied object.

For example, a proxy component for simple RFID tagged groceries would receive a serial number when instantiated. During run time, it would look for information from all RFID readers in the environment and when it finds a reader which can perceive the tag with that serial number, it uses the reader to read further tag data to compute properties of this specific item, eg. translating the hexcode of the first few bytes into a representation of what type of food, expiry date, shape of packaging etc. that it has. Additionally, it also computes its own position[1] from the position of the RFID reader.

When the interface component communicating with this

[1]Or literary, the position of the proxied object

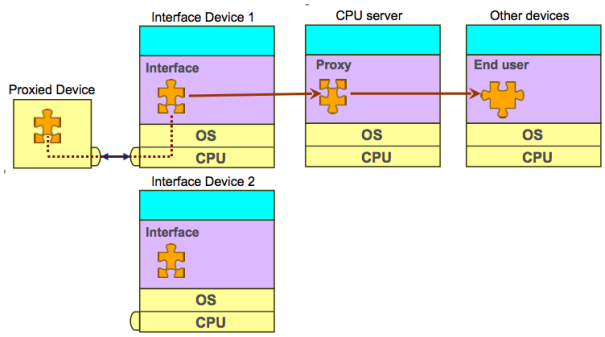

Fig. 1. Dataflow between a proxied box equipped with an RFID tag, an interface component with an RFID reader, a proxy for the box and the end-user application. Dashed lines show communication outside the standard middleware.

proxied object is changed, for instance by moving the object or the case when the interface moves, the proxy should automatically be configured to use any new interface(s). In the implementation described in the next section this is done automatically by associative data retrieval, but in the case of general middlewares this can rather be done explicitly in the proxy.

Consider the case in Figure 1 when we have one proxied component, a parcel, connected to an interface component through RFID electromagnetic coupling. The interface component produces data which are consumed by the proxy component and which in the end are consumed by other devices in the environment. When the EM coupling is broken, the proxy component will receive no data and use default values for any updates of the object which is transparent to the end user application. When the EM coupling is reestablished, perhaps to a second interface component instead, the data flow is resumed and the proxy can present a new, accurate state of the object.

Apart from the components described above, the execution of proxied objects requires one problem to be handled when dealing with real life scenarios. This deals with the dynamic nature of everyday environments in which the available components as well as proxied objects can change dynamically when objects are introduced or removed from the environment. To deal with this we are using a so called *proxy manager* which is responsible for the instantiation of proxy components as interfaces detect new objects. To do this, the proxy manager relies on *signatures* which must be provided by all interface components for each object with which they communicate. In the case of the RFID reader above, the signature corresponds to the ID part of the transmitted RFID signal. These signatures are compared with a database of available latent proxy components. When encountering a new signature, the proxy manager launches a proxy component with capability of handling those kind of objects. The proxy manager also takes care of providing the initial configuration of the proxy component by giving it the interface component with which it should communicate and the specific object which it should be representing. Incidentally, by allowing

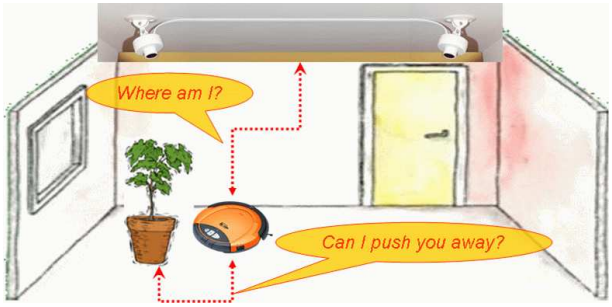

Fig. 2. A simple PEIS-Ecology consisting of a vacuum cleaner, an overhead tracking system, and a plant.

for the dynamic creation and removal of not only proxied objects but also interface components, we also increase the robustness of the whole system.

The steps necessary for incorporating proxied objects in robot environments can easily be implemented in different types of middlewares. Using these steps we can not only use proxies to integrate devices too simple to be integrated with other mechanisms, but we can also incorporate advanced devices that cannot be retrofitted with the necessary middleware to be used with the rest of the robotic ecology. One example of such a case is to integrate into our robotic ecology proprietary technology such as a robotic vacuum cleaner which cannot be reprogrammed to use a previously existing middleware. The role of the proxy component would here be to translate state information and commands to and from the middleware, but also to use other sources of information and act as a container for all the information related to this device.

III. PEIS-ECOLOGY IMPLEMENTATION

As an illustration on how to apply the design pattern of proxied objects we describe an implementation that has been performed as part of the PEIS-Ecology project [3], [12]. In this project advanced robotic environments have been developed, not by the inclusion of extremely advanced robots, but rather through the cooperation of many simple robotic components. These components range from very heterogeneous mobile robots to automated household appliances and sensors consisting of simple motes. A middleware, called the PEIS-middleware [9], is used for tying together all these heterogeneous devices. With the notion of proxied objects, we have been able to extend this ecology of intelligent systems to include also everyday objects such as coffee cups, food packages or very simple devices such as simple distributed sensors.

A. PEIS-*Ecology*

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. We define a PEIS to be a set of software components, called PEIS-components, which may include links to sensors and actuators as well as input and output ports that connect it to other components. By using the PEIS-kernel which implements a distributed and decentralized peer-to-peer network these connections transparently traverse any hosts that can communicate directly or indirectly. This allows for a dynamic environment in which PEIS and individual components may appear and disappear at any time.

Second, all PEIS are connected by a *uniform communication model*, which allows the exchange of information among PEIS, and can cope with their joining and leaving the ecology dynamically. The communication model chosen here is a distributed tuplespace allowing associative access to data by key, content or context. Components use subscriptions and registered callbacks to efficiently access data and minimize network traffic.

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. These links between components are themselves expressed as data accessible in the tuplespace, allowing for both introspection and dynamic reconfiguration of collaboration patterns using so called configurators [13].

As an illustration, consider the autonomous vacuum cleaner (PEIS) in Figure 2. 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, or a proxied object) 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.

To validate the PEIS-Ecology concepts and it's utility and acceptability for humans in a realisitc settings, we have developed a physical testbed called the PEIS-home (figure 3). This testbed resembles a typical Swedish bachelor apartment 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 Saffiotti *et.al.*[3], [12].

Before looking at how proxied objects can be implemented in a PEIS-Ecology we first take a look at one of the foundational components of a PEIS-Ecology, PeisInit. This

(a) (b)

Fig. 3. Two views of the PEIS-Home experimental testbed. (a) The kitchen, with the nose-equipped robot Pippi inspecting a smart fridge. (b) The living room, with the human-interface robot Astrid at its parking position.

```
1  RfidProxy(id: P, signature: X)
2  P.STATUS ←"active" ; P.POSITION ←"unknown"
3  P.SIGNATURE ←X
4  subscribe *.TAGS.X.DATA
5  loop
6     On callback I.TAGS.X.DATA=D  then
7        P.POSITION ←I.position
8        P.CONTENT, WEIGHT, ... ←decode-state(D)
9     end
10    On expire I.TAGS.X.DATA  then
11       P.POSITION ←"unknown"
12    end
13 end loop
```

Fig. 4. Simplified steps of each proxy component. Tuples keys are written as $C.name$ where C is a component or a wildcard $*$

is a standardized component in the PEIS-framework which is present on all machines capable of participating in the ecology and which is started on boot time and always present. The main responsibility of this component is to launch all other components that can be run on the various machines. This is done by reading a list of configuration files containing descriptions of all available components. PeisInit publishes semantical descriptions of each such found component, and when requested via the tuplespace it launches the Unix processes necessary for running the components. In addition to describing and launching components the tasks of PeisInit also include monitoring and restarting components in case of failures.

B. Interface components

The simplest example of an interface component is that of RFID reader components. These components are connected via a serial port to a Texas instruments RFID reader and they continuously publish three types of tuples:

- **position**: and other context variables. These give the location and state of the reader.
- **tags**: a list of all tags currently within range of the reader.
- **tags.X.data**: for each tag X it publishes a tuple mirroring the data written on X.

Note that the position of the interface component can be predefined during the deployment or it can obtain the position from any other references eg, the reader is connected to a mobile platform, which can provide and update the position of the interface when necessary. However, in our discussion the position of the interface component is an instance of the indirect information that can be used by the proxy to provide some on-line/real-time information of the proxied object such as its position.

C. Proxy components

In the PEIS-Ecology, proxies have been implemented as PEIS-components running onboard one of the home monitoring computers in the testbed. Each proxy consists of a separate unix process linked against the PEIS-kernel. The proxies are started and monitored, like any other PEIS-component, by the PeisInit component residing on each PEIS. The decision to start a new proxy is taken by the proxy manager which sends the startup request as well as initialization parameters through the tuplespace.

In general, the proxy components should have three different types of information:

- *Prior information*, built into the proxy or being given as argument to the proxy when instantiated.
- Information provided by the proxied communication channel, for instance, information written in RFID tags. We call this *direct information*.
- Information from the interface or any other components to whom it is subscribed to, giving us an *indirect information* flow.

Consider for instance the proxy for RFID tagged groceries in Figure 4. The initial parameters given here are the serial number X of the tag attached to the proxied, and the corresponding proxy component P. By using the signature X, the proxy P can obtain some general information about the proxied either by accessing the database or from the tag provided by the manufacturer. This is an example of prior information being used by the proxy. Using this serial number, the main loop is using associative search in the tuplespace to return the data for any interface I that publishes a tag I.TAGS.X.DATA matching this serial number. By interpreting this data written in the tag the proxy can publish properties about the object. This is an example of direct information. Furthermore, the proxy uses the position given in the interface I to compute the position of the proxied object. This is an example use of indirect information.

Note that in the actual implementation, a few additional steps are performed to select which interface device to use when multiple devices are available, but such arbitration mechanism are omitted here for conciseness.

D. The Proxy manager

The proxy manager is a PEIS-component that uses all existing interface components to listen for the appearance of

```
1  ProxyManager(id: M)
2  subscribe *.TAGS, *.COMPONENT.*.SEMANTICS
3  loop
4    On callback I.TAGS or
       In.COMPONENT.*.SEMANTICS  then
5      for each signature S in I.TAGS  do
6        if ¬∃ P : P.SIGNATURE = S  then
7          find In, P such that:
8            In.COMPONENT.P.SEMANTICS covers S
9          In.COMPONENT.P.ARGUMENT ←S
10         In.COMPONENT.P.REQ-STATE ←"on"
11       end if
12     end for
13   end
14 end loop
```

Fig. 5. Simplified steps of the proxy manager

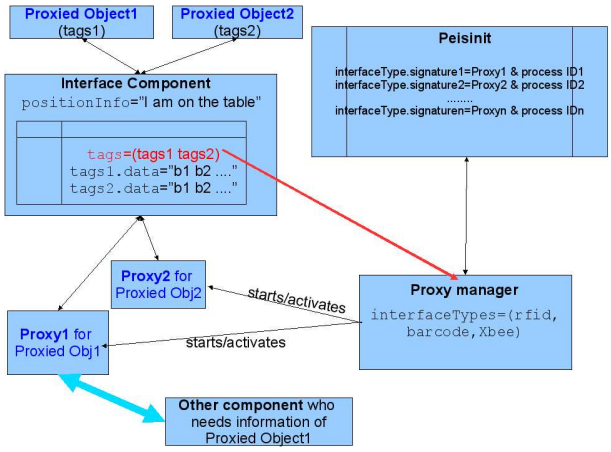

Fig. 6. Example of how the proxy manager, proxy components, PeisInit and other components communicate in a sample scenario.

new proxied objects by subscribing to the I.TAGS tuple from all interfaces I. When this tuple is changed the meta proxy receives a new list of signatures corresponding to the tags within range of the interface. These are then compared to all running proxies, and unless a proxy is already running for each signature a new proxy is started via PeisInit. To know which proxy component to start the proxy manager uses the semantical information published by the various PeisInit components currently running in the ecology by subscribing to the tuple In.COMPONENT.P.SEMANTICS. If it finds a PEIS with a component description whose semantics include the capability of working as a proxy for this signature it starts it. If there exists no proxy component for this signature at that time instance but available later, the proxy manager starts the proxy as soon as it receives the callback response from any of the PeisInit that matches the semantics for this signature. See Figure 5 for a simplified algorithm of this. Figure 6 shows, on a simple scenario, how the proxy manager, proxy components, PeisInit and other components communicate together in the ecology. Note that in the actual implementation, a number of additional steps have to be performed to also handle other types of interface components (eg. the XBee interfaces) left out here for brevity.

IV. ILLUSTRATIVE EXPERIMENT

To illustrate the workings of the concepts presented in the previous sections we describe an illustrative experiment performed in the testbed demonstrator. In this experiment we have two types of proxied objects, simple groceries equipped with RFID tags since manufacture and some simple Figaro gas sensors using ZigBee for relaying sensor data. Although the notions of proxied objects allows for actuation and sending commands back to the proxied objects no such objects where used in this experiment.

The experiment begins with an RFID tagged milkbox placed in the refrigerator in the apartment, and one of the gas sensors placed in a garbage can. At the start of the experiment Johanna, the owner of the apartment, enters the kitchen and grabs the milkbox from the refrigerator and places it on the table next to a box of cookies, also equipped with an RFID tag. After pouring herself some milk and starting on the cookies, Johanna decides to watch a movie and takes her food with her to the livingroom. Since she doesn't want to get up afterwards she also brings the garbage can so she can discard the leftovers of the food.

As we will see, during this seemingly innocent interaction with the ecology a number of operations are required to keep all information up to date. In order for her cleaning robot to be triggered by the food leftovers smelling in the garbage can, and in order to know where the garbage can is, the gas sensor in the can needs to publish its location as well as trigger an alarm when the odor raises above a threshold. Similarly, the state of the various groceries is also updated when the milk is moved from the refrigerator to the table, information which can later be used to (gently) remind Johanna that the milk will go bad if it is left on the kitchen table.

A. Experiment Setup

The experiment outlined above is performed in the PEIS-Home, our ubiquitous multi-robots testbed (figure 3). The milk package and the cookie package are ordinary objects equipped with Texas Instruments "Tag-IT" RFID tags [14]. There are short range RFID readers mounted in the refrigerator and the kitchen table (see Figure 7) connected to two RFID reader interface components running onboard the refrigerator control PC. These readers function as interface PEIS providing information about all tags detected inside the refrigerator or on top of the table. For this purpose, the POSITION tuples of the two interfaces have been given the initial values of "refrigerator" and "kitchen", respectively. In this experiment we assume that the table and the refrigerator are not moveable, though the interface can be connected to a mobile-platform.

In the environment a number of simple Figaro TGS-2600 gas sensors [15] are used to detect air contaminants, and a more advanced electronic nose is mounted on a mobile robot

Fig. 7. Kitchen of PEIS-home. Left figure shows RFID-reader equipped kitchen fridge with milk box (starting moment of the experiment). Right figure denotes RFID reader attached table with milk and cookies proxied.

(a) (b)

Fig. 8. Proxied objects: (a) A XBee mounted figaro proxied in garbage can. b) RFID tags under cookies and milk proxied objects.

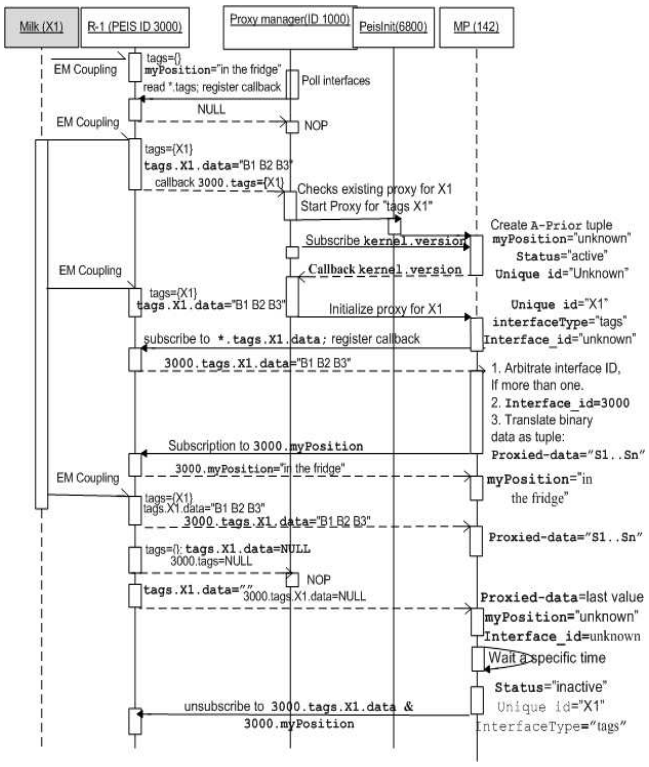

Fig. 9. Sequence diagram of the first steps of the experiment. R-1 is the RFID reader in the refrigerator and MP is the created proxy object for the milkbox. PeisInit with id 6800 runs on a machine with available proxy programs for a number of different objects.

which can approach any detected contaminants to make a more detailed assessment of the situation [16]. These simple gas sensors are mounted in the refrigerator, in the kitchen are as well as inside one of the loose garbage cans. For the case of the refrigerator, the sensors are connected to the I/O boards of the PC. The sensors in the garbage can and kitchen are connected to the analog inputs of a MaxStream XBee [17] IEEE 802.15.4 communication module, see Figure 8.

Since the XBee modules have a limited indoor range, we use two additional XBee devices connected to two other PCs in the livingroom and the kitchen, respectively, with serial cables. These connections allows two XBee interface components to run on these PCs and publish all detected XBee objects and data streams from them as well as send commands to them (used only for XBee modules connected to actuators). The XBee interface component in the kitchen has been given the initial value ”kitchen” for the POSITION tuple and the one in the livingroom have been given initial value ”livingroom”. This allows the sensed gas values to be received regardless of where the garbage can is placed, and also serves to give a coarse location of it.

In the experiment, all RFID tagged objects or XBee mounted objects (see Figure 8) are proxied objects. Initially, a proxy manager component is running in the refrigerator and another manager component is running on the home monitoring PC’s. When the objects are first introduced in the ecology, a proxy processes is instantiated on some of the available PC’s. Furthermore, a monitoring component called *peisalarm* is run on one of the PC’s, the purpose of this component is to monitor the sensed gas levels published by all gas sensors in the environment and to trigger an alarm (send a mobile robot to the source of any discrepancies) when the air contaminants exceed a threshold value given by the THRESHOLD tuple.

B. Execution

Consider the first few steps in the experiment as illustrated in Figure 9. The execution of the experiment starts with the introduction of the milkbox in the refrigerator. This is detected by the RFID reader in the refrigerator (R-1) through the electromagnetic (EM) coupling, and the interface component updates the list of available tags to include the signature (X1) of this tag and forwards it to all subscribed components, ie. the proxy managers. This triggers the proxy manager which performs a search to see if any proxy component is already running for the signature X1. Since no such component is found, it continues by searching for any *potential* component which can be started and whose semantics describe it as being a proxy for objects matching X1. In this implementation, the semantic information is represented in a very simplistic manner as lists of serial numbers. In the future, more complete semantic descriptions should be used and incorporated with the semantical descriptions already in place in the PEIS-middleware. After finding a matching component which can be launched by the PeisInit component on one of the available PC’s the proxy manager starts it.

This new proxy (MP-142) is started through the standard

PeisInit mechanisms of the PEIS-middleware. This proxy receives as argument the specific tag to proxy and subscribes to information about the relevant tag from any and all interface components. In order to get RFID-tag data from the milk proxied object, MP does a wild-card subscription to tuple `*.tags.X1`. When the interface components communicate with the hardware, they create/update this tuple and push it to subscribers as a callback response. From the received tuple the proxy extracts the interface components id and saves this as the current interface. Additionally, it uses the actual tuple data to read the properties of the milkbox and creates shape and content tuples accordingly. By subscribing to the current interface component's position it can also update its own position.

In this manner the proxy interacts with the tag and the interface components to compute the properties of the milkbox as well as its position. See Figure 9 for an illustration of these first steps.

When the milk is later moved to the table, the initial EM coupling will break and a new one will be established by the second RFID reader. Since the proxy is using associative search (wildcards) it receives the update from the second RFID reader and modifies the tuple, including the position, accordingly.

The remainder of the execution, with the XBee interface objects, works in a similar manner. The XBee module connected to the Figaro sensor samples it once a second and broadcasts the sensed readings once every other second. The garbage can proxy initially receives binary data from the XBee interface component deployed in the kitchen, translates it into decimal format and publishes the latest twenty samples as proxied data to the ecology. It deals with the position and any other properties similarly to how the RFID tagged objects are treated. The only difference here being the alternative type of interface components and that the increased range of this communication gives a coarser localization.

The tuples produced by these two proxies are working like any other data used by the remainder of the robotic components. As such it is completely transparent for the peisalarm component if it receives sensor data from the mobile garbage can or from the refrigerator. In both cases it receives a set of twenty samples for which it can compute the mean value and compare to a given threshold function in order to trigger an alarm. Also, when a robot is asked to approach a source of the alarm, such as a sensor in the refrigerator or the trashcan, it can transparently fetch the position without any considerations of the type of device (fully fledged control PC or a simple XBee module) with which it is interacting.

V. CONCLUSIONS

The main contribution of this paper is to propose a new design pattern for how very simple devices and everyday objects can be integrated with robots in smart environments. By using the notion of proxies it is possible to deal with very limited and heterogeneous devices in a uniform manner compatible with other, previously existing, robotic middlewares for these environments. To the best of our knowledge, this is the first approach that allows the inclusion of general objects in a robot middleware. We have illustrated this concept with a reference implementation and some illustrative experiments which have been performed using an OpenSource middleware, the PEIS-middleware, for ecologies of robotic devices. All of the developed notions and implementations can be found at the webpage [18] for the PEIS-middleware.

VI. ACKNOWLEDGMENTS

This work was supported by ETRI (Electronics and Telecommunications Research Institute, Korea) through the project "Embedded Component Technology and Standardization for URC (2004-2008)", CUGS (the Swedish National Graduate School in Computer Science), and Vetenskapsrådet (the Swedish Research Council).

REFERENCES

[1] J. Lee and H. Hashimoto, "Intelligent space – concept and contents," *Advanced Robotics*, vol. 16, no. 3, pp. 265–280, 2002.
[2] 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.
[3] 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.
[4] International Telecommuncation Union, "The Internet of things," ITU, Tech. Rep., 2005.
[5] K. LeBlanc and A. Saffiotti, "Issues of perceptual anchoring in ubiquitous robotic systems," in *Proc. of the ICRA-07 Workshop on Omniscient Space*, Rome, Italy, 2007, online at http://www.aass.oru.se/˜asaffio/.
[6] 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.
[7] 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.
[8] 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.
[9] M. Broxvall, B. S. Seo, and W. Y. Kwon., "The peis kernel: A middleware for ubiquitous robotics." in *In Proc. of the IROS-07 Workshop on Ubiquitous Robotic Space Design and Applications.*, 2007.
[10] 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/.
[11] M. Beigl, H. Gellersen, and A. Schmidt, "MediaCups: experience with design and use of computer-augmented everyday objects," *Computer Networks*, vol. 25, no. 4, pp. 401–409, 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*, 2007, pp. 16–22.
[13] 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.
[14] "Tag-IT RFID tags home page," http://www.ti.com/.
[15] "FIGARO TG-2600 home page," http://www.figarosensor.com/.
[16] A. Loutfi, M. Broxvall, S. Coradeschi, and A. Saffiotti, "An ecologcial approach to odour recognition in intelligent environments," in *Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Orlando, USA, 2006.
[17] "MaxStream home page," http://www.maxstream.net/.
[18] "PEIS ecology homepage," www.aass.oru.se/˜peis.