2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Plan-Based Configuration of an Ecology of Robots

Robert Lundh, Lars Karlsson, Alessandro Saffiotti
AASS Mobile Robotics Lab
Örebro University, 70182 Örebro, Sweden
{robert.lundh,lars.karlsson,alessandro.saffiotti}@aass.oru.se

***Abstract*— We consider an *ecology* of robots in which robots can help each other by offering information-producing functionalities. A functional *configuration* of this ecology is a way to allocate and connect functionalities among the participating robots. In general, different configurations can be used to solve the same task, depending on the current situation, and some tasks require sequences of different configurations to be solved. In this paper, we propose a plan-based approach to automatically generate a preferred configuration for a given task, environment, and set of resources. We also describe how our configuration planner can be integrated with an action planner to deal with tasks that require sequences of configurations. We illustrate these ideas on a specific instance of an ecology of robots, called a PEIS Ecology. We also show an experiment run on our PEIS Ecology testbed, in which a sequence of configurations for an olfactory task is automatically generated and executed.**

I. Introduction

The field of cooperative robotics is maturing, and there is now a tendency to consider more complex systems, including systems which are fully distributed and highly heterogeneous. A particularly interesting case of this tendency 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 ubiquitous robotic systems [5], [8], network robot systems [13], and PEIS-Ecologies [17]. 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. The devices rely on a distributed middleware to communicate and cooperate. In this paper, we generically refer to a system of this type as an "ecology of robots".

To exploit the power of this paradigm, robots in an ecology need to dynamically connect in a given *configuration* in order to pull their functionalities together. For instance, an autonomous vacuum cleaner can get information from tracking cameras in the ceiling to navigate through a home, and ask doors to open when moving from one room to the next. Note that the configuration to use depends on the specific task, and often the same task can be performed using different configurations.

In this paper, we deal with the problem of how the robots in an ecology can autonomously decide and realize a sequence of configurations needed to cooperatively perform a given task. To make our investigation concrete, we consider a specific approach, the PEIS-Ecology approach [17]. In this approach, each robotic device (called PEIS) contains a number of functional modules, and robots can help each-other by borrowing functionalities from one another. In the above example, the cleaner PEIS would borrow a functionality to self-localize from the cameras, and a actuation functionality from the doors. A *configuration* of a PEIS-Ecology is, roughly speaking, a way to connect the functionalities in the PEIS-Ecology.

In most existing work on ecologies of robots and on cooperative robotics, the way to connect robots functionalities was hand-coded [3], [9]. Similarly, in our previous work on PEIS-Ecology we had hand-coded the configuration of the PEIS-Ecology [1], [2]. In contrast to this, in this paper we propose to use techniques derived from the field of AI planning to: (1) automatically generate a configuration that allows a set of PEIS to perform a given task in a (tightly) cooperative way; and (2) automatically generate a sequence of such configurations that will make the PEIS-Ecology perform a sequence of tasks that achieves a given goal. To do so, we start from our previous work on plan-based configuration of a group of robots, originally proposed in [11]. We extend that work in two important ways. First, by applying that approach to the more complex case of an ecology of robots; second, by considering the on-line generation of sequences of configurations as opposed to single configurations.

The rest of the paper is organized as follows. In section 2 we give reminder of the notion of a configuration in the PEIS-Ecology framework. In section 3 we present the configuration planner, and in section 4 we discuss how sequences of configurations can be planned. Section 5 presents our experiments. Section 6 discusses some related work and section 7 concludes.

II. Configurations

A. The PEIS-Ecology Approach

The concept of PEIS-Ecology, originally proposed by Saffiotti and Broxvall [17], puts together insights from the fields of autonomous robotics and ambient intelligence to generate a radically new approach to building assistive, personal, and service robots. The main constituent of a PEIS-Ecology is a *physically embedded intelligent system*, or PEIS. This is any computerized system interacting with the environment through sensors and/or actuators and including some degree of "intelligence". A PEIS can be as simple as a smart toaster and as complex as a humanoid robot.

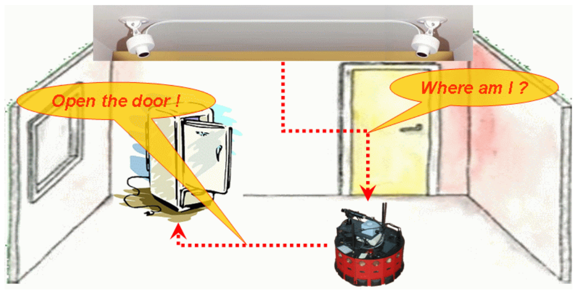

Fig. 1. A simple PEIS-Ecology. The ceiling cameras provide global positioning to the robot. The robot performs the door opening action by asking the refrigerator to do it.

Individual PEIS in a PEIS-Ecology can co-operate based on the notion of linking functional components: each PEIS can use functionalities from other PEIS in the ecology in order to compensate or to complement its own. The power of the PEIS-Ecology does not come from the individual power of its constituent PEIS, but it emerges from their ability to interact and cooperate. For example, a robot which needs to grasp a bottle would not use its sensors to detect its position, shape, and weight and thus compute the parameters of the grasp — a task which proved to be surprisingly difficult in years of robotic research. Instead, the bottle itself, enriched with a micro-PEIS, would hold this information and communicate it to the robot.

Figure 1 shows an example of a simple PEIS-Ecology. A mobile robot is equipped with an artificial nose. This robot can be seen as a PEIS, which includes functionalities for reactive navigation, obstacle detection, and odor classification. It may not have enough perceptual abilities to reliably estimate its own position in the home. Suppose, however, that the home is equipped with an ambient monitoring system using a set of cameras, which is able to track the position of the robot. Then, we can combine the monitoring system and the robot into a simple PEIS-Ecology, in which the former provides the latter with a global localization functionality, thus enabling the robot to perform more complex tasks. Suppose next that the robot needs to approach the refrigerator to inspect the quality of its content using its artificial nose. If the refrigerator is a PEIS and it is able, among other things, to open its door, then the robot can open the fridge door by simply asking the fridge to do so.

The PEIS-Ecology approach has been implemented in an experimental platform that includes a distributed middleware, called the PEIS-middleware, a number of PEIS, and a physical testbed. The PEIS-middleware implements a distributed tuple-space on a P2P network: PEIS exchange information by publishing tuples and subscribing to tuples, which are transparently distributed by the middleware. (See [2] for more details.)

B. *Configurations of a* PEIS*-Ecology*

Central to a PEIS-Ecology is the notion of a *functional configuration*, or simply configuration. To define this notion, we first define the following ingredients.

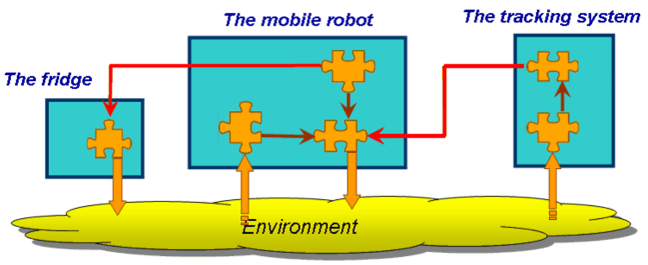

Fig. 2. Functional configuration of the above PEIS-Ecology.

A PEIS**-component** is a software module that implements a functionality. A *functionality* is an operator that uses information to produce additional information. It is characterized by the following elements:

- A specification of *inputs* to be provided by other functionalities, including information about domain (e.g., video images), timing (e.g., 25 fps), etc.
- A specification of *outputs* provided to other functionalities, also containing domain and timing information.
- A set of causal *preconditions*: conditions in the environment that have to hold in order for the functionality to be operational.
- A set of causal *postconditions*: conditions in the environment which the functionality is expected to achieve.
- A specification of *costs*, e.g., computation and energy.
- A *body*, containing the code to be executed.

A *channel* transfers data from an output of a functionality to an input of another functionality. In the PEIS-Ecology-context, this is realized by letting the later functionality subscribe to the output of the former functionality.

A PEIS is a set of PEIS-components, located in the same physical device.

A PEIS**-Ecology** is a collection of inter-connected PEIS, all embedded in the same physical environment.

Generally, a *configuration* is set of functionalities and a set of channels that connect functionalities to each other. More specifically, a **configuration** of a PEIS-Ecology is a subset of PEIS-components within the ecology, together with the set of connections between them within and across the PEIS in the ecology. It is important to note that the same ecology can usually be configured in many different ways depending on the current context, where relevant contextual aspects include the current task, situation, and resources. Moreover, different configurations can often be used to perform the same task. This redundancy can be exploited to improve the flexibility, reliability, and adaptivity of a PEIS-Ecology.

A configuration also has a **cost**. This can be based on functionality costs, communication cost, etc, but also on performance accuracy and reliability of the configuration. Currently, we compute this cost as a weighted sum of the costs of the individual components.

An important property of a configuration is that all the components in it are connected “in the right way”. We call this property **admissibility**, and we distinguish two brands: a configuration is *information admissible* if each input of each functionality is connected to a compatible output of another functionality; it is *causally admissible* if all preconditions of all functionalities hold in the current world state. A formal definition of these properties can be found below.

Information admissibility:

$\forall f \in F\, \forall i \in I_f \exists ch \in Ch = (f_{send}, o, f_{rec}, i)$ such that
$description(o) = description(i)$,
$domain(o) = domain(i)$, and
$Freq(f_{send}(ch)) \geq Freq(f_{rec}(ch))$

where F is the set of all functionalities, and *Ch* is the set of all channels. The channel ch is represented as a tuple with a sending functionality f_{send} with output o, and a receiving functionality f_{rec} with input i.

Causal admissibility:

$$\forall f \in F : Pr_f(s) = True$$

where $Pr_f(s)$ is a truth value of the preconditions of functionality f in state s.

Figure 2 shows a functional view of the PEIS-Ecology configuration shown in Figure 1 above. The robot's navigation component receives position information from the tracking component of the localization system, and the robot's deliberation component sends the door-opening action to the refrigerator. As an alternative configuration, the robot could use its own odometric estimator to provide (less reliable) position information to its navigation component.

In the rest of this paper, we show our approach to automatically generate sequences of admissible configurations which are causally related.

III. CONFIGURATION GENERATION

A. Domain Description

In order to generate configurations for a particular PEIS-Ecology, the configuration planner requires a declarative description of:

- the functionalities and methods for connecting them,
- the current world state,
- the goal for what the configuration should produce.

The world state, declares the available PEIS and their physical capabilities, as well as the state of the surrounding environment. It is encoded as a set of clauses, such as `robot(r1)`, `door(d1)`, `robot--at(r1,kitchen)`. In a PEIS ecology, the world state is compiled from information received from the different components via the PEIS-middleware [2]. The goal for the configuration generation process is to produce a desired information output. For example, in the smell-fridge scenario in Figure 1, the information goal is to produce the food status.

The description of available functionalities is realized using operator schemas similar to those of AI action planners. One functionality operator schema that we have used in our experiments, `localization-system`, is shown in the upper half of Figure 3.

The `name` field specifies the name and parameters of the functionality. The fields `input` and `output` specify

```
(functionality
 name: localization-system(p, o)
 input: images(p)
 output: global-pos(o)
 precond: o visible in at least 1 of images(p)
 postcond: -
)

(config-method
 name: get-location-info(r)
 precond: robot(r), peis(r), peis(p),
  localization-system(p), 4-cameras(p)
 output: f2: global-pos(r)
 channels: local(p, f1, f2, image(p))
 body:
    f1: cameras(p)
    f2: localization-system(p, r)
)
```

Fig. 3. A functionality operator schema, and a method schema for combining functionalities. (Syntax simplified for readability reasons.)

the inputs (images) and the outputs (global-pos) of the functionality. The `precond` and `postcond` fields encode the causal preconditions — in this case, the object o must be visible in at least one of the input images.

B. The Configuration Planner

Our configuration planner allows us to define methods that describe alternative ways to combine functionalities (or other methods) for specific purposes, e.g. combining the ceiling cameras functionalities with a localization system functionality.

The lower half of Figure 3 shows an example of a method schema that does exactly that. There is a channel inside the method connecting two functionalities (labeled `f1` and `f2`). The descriptor of the channel (`image(p)`) tells which specific input and output of the functionalities should be connected. In PEIS terms, the channel indicates that the receiving PEIS component should read from a specific tuple (in this case for images) owned by the sending PEIS component. In addition, the outputs (`global-pos(r)`) of `f2` is declared in the `output` field to be the output of the entire method. Thereby, any channel that in a method higher up in the hierarchy is connected to the output of `get-location-info` will be connected to the output of `localization-system`.

The configuration planner takes as input a current causal state s, a stack of (unexpanded) method instances with initially one instance $l : m(c_1, c_2, ...)$ representing the goal of the robot (l is a label), and a set of methods M and a set of functionality operators O. It basically works as follows (a more technical description is found in [10]):

1) Take the unexpanded method instance $l : m(c_1, c_2, ...)$ at the top of the stack.
2) If $l : m(c_1, c_2, ...)$ matches the name of a functionality operator O, instantiate and add that operator to the current configuration.
3) If $l : m(c_1, c_2, ...)$ matches a method schema in M which has preconditions holding in s, instantiate and expand that method schema. Add the channels (with new labels) to the current configuration. Add

the method instances (with new labels) of the method body to the top of the stack. Redirect channels in the current configuration that are connected to the input or output slots of the method to the corresponding method instances in the method body.

4) If the stack is empty, return the current configuration. Otherwise go back to 1.

In the following example, the robot called Pippi and the home monitoring system (hms) are PEIS. More details about different PEIS and PEIS-components can be found in the experiments section. To illustrate how the planner functions, let us assume we are expanding `l5: get--location-info(pippi)` (step 1). We choose to use method in Figure 3 (step 3). First, we need to replace `r` with `pippi` everywhere in the schema. We need to look in s for a PEIS `p` with a localization-system, e.g. the `hms`, and replace `p` with `hms` everywhere in the schema. New labels, say `l7` and `l8`, replace `f1` and `f2`. The channel is added to the current configuration and the two functionalities `l7: cameras(hms)` and `l8: localization--system(hms, pippi)` are added to the top of the stack. Finally, we go through the channels already in the configuration, and any channel we find with label `l5` (i.e. `get-location-info(pippi)`) for its out connection is reconnected to `l8` (i.e. `localization-system(hms, pippi)`). We proceed to step 4, and then return to step 1. There we find `l7: cameras(hms)` at the top of the stack, which matches a functionality operator and is added to the current configuration (step 2). And so on.

The first output from the planner is a configuration description C, which essentially consists of a set of functionality names with labels, e.g. `l8: localization--system(hms, pippi)`, and set of channels, e.g. `local(p, f1, f2, image(p))`.

The second output from the planner is the set of postconditions P specified in the functionalities in C, which can be used to update the current state, which then in turn can be used as input for generating the configuration following the current one (if any).

It is possible to accidentally specify methods that can result in configurations with cycles. However, these cycles are easily detected and the faulty configurations are excluded.

Generally, there are several configurations that can solve a problem, but obviously, only one configuration per problem can be performed at the time. By trying different applicable method versions, guided by the cost of configurations, our planner generates the admissible configuration with the lowest cost first.

C. Deployment of a Configuration

In order to execute a configuration, we need to instantiate it on the ecology. We call this phase the deployment of the configuration. The deployment phase consist of three steps. The first step is to verify that the functionalities of the description are up and running. This is achieved by searching through the distributed tuple space provided by the PEIS-middleware for tuples matching the names of the functionalities. The second step is to setup the channels between the functionalities. Since communication in the PEIS-Ecology is done using a tuple space, the channels are setup by telling the different functionalities which information they should subscribe to and from whom. That is, (a) for each channel, the destination key tells which functionality that should be the owner of the subscription, (b) the source key, tells from which functionality it should be subscribed, and (c) the information field gives the information to subscribe. The third and last step of the deployment phase is to activate the actuator functionality(s) of the configuration and to let the plan executor to subscribe to the status information of this/these actuator functionality(s), in order to tell if an action is accomplished or not.

IV. TASK PLANNING

The configuration planner is capable of generating a single configuration for one particular task. However, often one needs to perform several steps in order to solve a task. For instance, to check the fridge, the robot must first move into the kitchen, then the fridge door needs to be opened, and the robot must move near the fridge and then smell its contents. Different configurations are needed at each of these steps.

We employ a sensor-based probabilistic action planner, called PTL Planner [7], in order to generate conditional plans that specify the different steps needed to complete particular tasks. The steps in these plans are actions of the form ”move robot to kitchen”, ”open fridge door”, and ”smell fridge”. For each type of action, one can specify an information goal. For instance, for an action of the type ”move robot to X”, one may specify an information goal to estimate the position of the robot. When a configuration has been generated, this information can be fed to the robot’s movement behavior.

Our present system works by first calling the action planner to find a conditional action plan for solving a particular task. This plan is then executed step by step. At each step, the configuration planner generates a configuration. The configuration is then deployed as described in Section III-C. When the step is completed, the system proceeds to the next step and generates a new configuration, and so on. The completion of a step is reported by the PEIS component that is the endpoint of the configuration. For instance, for a navigation task, the navigation module of the robot determines when it has reached the desired position.

V. EXPERIMENTS

For the experimental part we have used a scenario first presented by Loutfi et al[1]. In that paper, the purpose of the experiments was to show that by using a PEIS-Ecology approach, it is possible to by-pass some of the difficult problems of mobile olfaction (e.g. odor localization, and domain information for odor classification). The subscriptions where handled using a hand written script.

In this paper, we use the same experiment to show that it is possible to automatically generate a sequence of configurations that that enables the robot to smell inside the fridge. We do this by using the configuration framework and the action

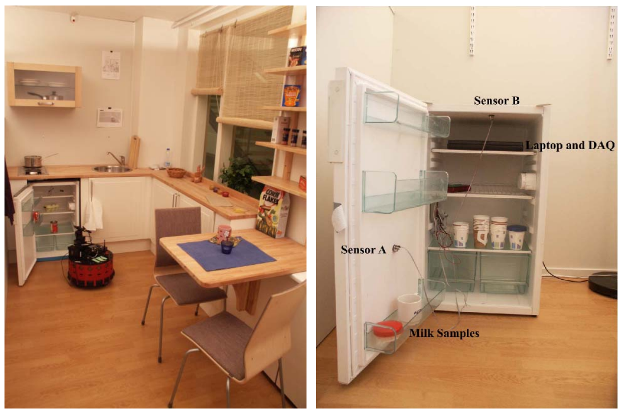

Fig. 4. *(Left)* Pippi is about to smell inside the fridge. *(Right)* The PEIS fridge showing the placement of the two gas sensors and the products to be sampled.

planner presented above to setup the subscriptions and to activate the functionalities in the configuration description.

A. Experimental Setup

For the experimental part, we have used a physical test-bed facility, called the PEIS-Home, which looks like a typical bachelor apartment of about $25m^2$. It consists of a living room, a bedroom and a small kitchen. The PEIS-Home is equipped with communication and computation infrastructure, and with a number of PEIS. The picture to the left in Fig. 4 shows a snapshot of the kitchen.

The following PEIS are of particular importance for our experiments.

Pippi the mobile robot PEIS*:* An iRobot's Magellan Pro indoor robot (see Fig. 4) that in addition to the usual sensors, it is equipped with a CCD color camera and with a Cyranose 320^{TM} electronic nose used to identify and discriminate between odors. On-board Pippi also runs an instance of the Thinking Cap, an architecture for autonomous robot control based on fuzzy logic [18], and an instance of the player program [15], which provides a low-level interface between the robot's sensors and actuators and the PEIS-Ecology's tuple-space.

This is also where the action planner and the configuration planner are located, and the reconfigurations of the PEIS-Ecology are done from here.

The refrigerator PEIS*:* The refrigerator PEIS consists of an apartment sized refrigerator with two simple gas sensors, a motorized door, an RFID-reader, and a laptop computer with data acquisition technology (see Fig. 4 right). The RFID-reader can read the RFID tags of the objects in the fridge.

The Home Security Monitor PEIS*:* This is a PEIS which consists of a stationery computer which is connected to a set of web-cameras mounted in the ceiling. In addition to other monitoring tasks, not relevant here, this PEIS reacts to alarms published in the tuple-space by the simple gas sensing component in the fridge. If such an alarm is signaled, it sends a task to an available robot to go to investigate the food status on the fridge. It contains a high-level component responsible for detecting possible problems using the available olfactory resources: the simple gas sensors which are present in the refrigerator-PEIS and possibly in other devices, and the electronic nose on Pippi.

B. Experimental Execution

All the experiments reported here have been performed using the PEIS-Ecology described in the previous sections, and placing milk cartons in the PEIS-fridge. Each experimental run consisted of the following:

1) At start-up, Pippi started at a resting position in the living room.
2) After some time, the gas sensors in the refrigerator triggered an alarm, and a tuple of type `fridgeAlarm` was placed in the tuple-space. This was notified to the deliberator component inside the Home Security Monitor PEIS, thus triggering the need for Pippi to investigate the fridge. Pippi was notified by a tuple, describing the a high level goal (smell-fridge), that was published by the deliberator.
3) With this high level goal, the action planner located at Pippi, generated a plan. The plan consisted of ”move to kitchen”, ”open fridge door”, ”move near fridge”, ”smell fridge”.
4) This plan was executed by first generating and deploying a configuration for moving Pippi to the kitchen. The configuration involved web cams in the ceiling, connected to a computer which estimated Pippi's position and posted this information as tuples. Pippi's navigation system read these tuples and used the information when moving into the kitchen.
5) When Pippi's navigation system had established that Pippi had reached the kitchen, it published a tuple that declared that this was the case.
6) The next action was to open the fridge door, which requires only a very simple configuration (the fridge door opening component). This was done and completion was reported.
7) Then the robot moved near the fridge. This was done in a similar way as moving to the kitchen.
8) Pippi then activated the electronic nose by publishing another tuple, and waited for the classification results. When those results were published, that step was considered completed.
9) If the results signified no problem (i.e. “no bad smell”), the deliberator asked Pippi to return to its starting position and resumed its monitoring activity.
10) In all other cases, the deliberator would ask Pippi to move to the bedroom to alert the occupants of the PEIS-Home. (This step was not consider in the conducted experiment)

The important steps of the experiment are the steps involving the configuration planner, that is, step 4, 6, 7, and 8. In step 4, the navigation system on Pippi needs to know Pippi's position in order to move to the kitchen. In this situation, there are two different ways that Pippi can retrieve this information: (1) She can update her initial position using

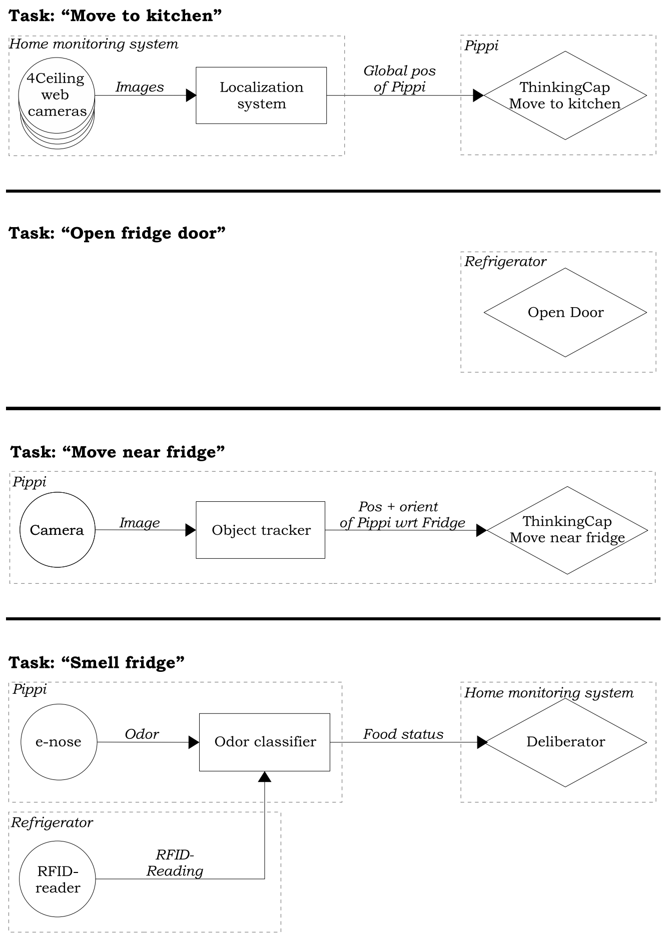

Fig. 5. The configurations for: "move to kitchen", "open fridge door", "go near fridge", "smell fridge". See explanation in the text.

odometry measurements, or (2) she can let the web cams in the ceiling track her position and send it to her. When generating the configuration, all alternatives are considered, and the second one is selected since it has a lower cost. (See first config. in Fig 5)

In step 6, the action is to open the door of the refrigerator. In the current implementation, we assume that this action does not need any information, i.e., the configuration planner will generate a configuration only consisting of one functionality, namely open-fridge-door. (See second config. in Fig 5) However, it might be a good idea for this functionality to have some information about its closest surroundings to avoid hitting objects with the door. This information could of course be obtained in different way, e.g. using the ceiling web camera system, a camera on a robot in the kitchen, or perhaps from a tactile sensor in front of the fridge.

Step 7 is very similar to step 4 where Pippi moved in to the kitchen, but here the task is for Pippi to move close to the fridge. In order to move near the fridge, Pippi needs the information about where the fridge is with respect to herself. This information can be retrieved either by using the different localization methods described for step 4, or by using a camera and vision system on-board Pippi that is able to track the fridge. By tracking the fridge, Pippi can get continuous information about the position of the fridge in her local coordinate system. When generating the configuration for moving close to the fridge, these three alternatives are considered, and the alternative using on-board camera and vision system (object tracker) is selected since it has a lower cost than the other alternatives. (See third config. in Fig 5)

Step 8 considers the action of smelling inside the fridge. When smelling the fridge, the odor classifier gets the context information by reading the RFID tags of the objects in the fridge. To read these tags, it is possible to either use the RFID reader inside the fridge, or the one on Pippi. The configuration planner generates a configuration where the RFID reader inside the fridge is used. (See last config. in Fig 5)

VI. Related Work

Problems similar to the work on automatic generation of configurations have been studied in several different research areas, e.g. in program supervision [20], automatic web service composition [16], coalition formation [19], and single robot task performance [12]. However, in the field of ecologies of robots and robots acting in intelligent environments, there are only few works that address similar problems. Intelligent Spaces [9] deal with the problem of how an intelligent environment can actively provide humans and robots with information. In these, distributed intelligent networked devices (DINDs) that are composed of a sensor, processor, and network, act as providers of information. DINDs can share information and cooperate with each other. In contrast to the work in this paper, Intelligent Spaces are homogeneous, the cooperation is hard coded, and it can only handle cooperation that does not require simultaneous operations. Ha *et al.* [5] present an approach for automated integration of networked robots into intelligent environments. They use a hierarchical planner to generate sequences of services for a given task. As in traditional web service composition, there is a data relation between services rather than a causal relation. A part from that relation, the problem is very much like traditional action planning. The services are executed in a sequence like actions of a plan. In contrast, functionalities in our approach are executed in parallel with continuous streams, which allows us to address tasks that require tight coordination.

In the area of ambient intelligence (not including robots) Heider and Kirste[6] propose an approach that uses plan-based techniques to control an intelligent environment. The aim is to make interaction between the user and the environment more goal oriented, i.e. the user should not have to learn how to operate all functions on all devices, but rather just give the goal of the interaction (e.g. Find the media source containing media event "Terminator"). Planning is used to develop strategies on how different functions can be executed to reach the goal given by the user. Similar to the approach by Ha *et al.*[5] mentioned above, this approach has more in common with traditional action planning than it has with configuration planning.

In the area of cooperative robotics, very few works address problems similar to configuration generation. Parker and Tang [14] present an approach called ASyMTRe. The principle of ASyMTRe is to connect different schemas (similar to instantiated functionalities) in such a way that a robot team is able to solve tightly-coupled tasks by information sharing. The approach presented in this paper goes one step further, by addressing the automatic generation of *sequences* of configurations, using a combination of hierarchical planning for individual configurations and probabilistic action planning for the sequences.

VII. Conclusions

We have described a novel approach to the automatic on-line generation of a configuration of an ecology of robots using plan-based techniques. We have validated our approach in our PEIS-Ecology testbed, by showing that a scenario that was previously hand-coded can now be run fully autonomously. It is important however to note that our approach is not restricted to the case of a PEIS-Ecology, but it applies to generic groups of robots. As a consequence, the approach can also be applied beyond the PEIS-Ecology framework considered here. Examples of applications of our approach to other cooperative robotics tasks can be found in [10].

An especially interesting feature of our approach is that the configuration generation is done in two dimensions: both a causal and an information dimension. For the causal dimension we employ a sensor-based probabilistic action planner in order to generate conditional plans that specify the different steps needed to complete particular tasks. For the information dimension we employ a configuration planner that generates descriptions of configurations that specify the functionalities and the information flow between functionalities that are required to execute a particular task.

While our plan-based approach has interesting properties, like the guaranteed correctness and optimality of the generated configurations, it has the usual problems of plan-based approaches: it requires full knowledge about the available PEIS and about the state of the world, and the generated configuration might not be correct any more if these data change after it was generated. In a related paper [4], we explore a reactive approach to the self-configuration of a PEIS-Ecology, which might be more suitable for highly dynamic environments. Our next step will be to systematically compare these approaches, and to explore their integration.

Acknowledgments

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

References

[1] M. Broxvall, S. Coradeschi, A. Loutfi, and A. Saffiotti, "An ecological approach to odour recognition in intelligent environments," in *Proceedings of the IEEE International Conference on Robotics and Automation ICRA*, Orlando, FL, 2006.

[2] M. Broxvall, M. Gritti, A. Saffiotti, B. Seo, and Y. Cho, "PEIS ecology: Integrating robots into smart environments," in *Proceedings of the IEEE International Conference on Robotics and Automation ICRA*, Orlando, FL, 2006.

[3] L. Chaimowicz, A. Cowley, V. Sabella, and C. J. Taylor, "ROCI: a distributed framework for multi-robot perception and control," in *Proceedings of the 2003 IEEE/RJS International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, USA, October 2003, pp. 266–271.

[4] M. Gritti, M. Broxvall, and A. Saffiotti, "Reactive self-configuration of an ecology of robots," In: ICRA workshop on Network Robot Systems, Rome, Italy, 2007.

[5] Y. Ha, J. Sohn, and Y. Cho, "Automated integration of service robots into ubiquitous environments," in *Proceedings of the 3rd International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2006, pp. 177 – 182.

[6] T. Heider and T. Kirste, "Smart environments and self-organizing appliance ensembles," in *Mobile Computing and Ambient Intelligence*, 2005.

[7] L. Karlsson, "Conditional progressive planning under uncertainty," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, Seattle, USA, 2001, pp. 431–438.

[8] J. Kim, Y. Kim, and K. Lee, "The third generation of robotics: Ubiquitous robot," in *Proceedings of the 2nd International Conference on Autonomous Robots and Agents (ICARA)*, Palmerston North, New Zealand, 2004.

[9] J. Lee, K. Morioka, N. Ando, and H. Hashimoto, "Cooperation of distributed intelligent sensors in intelligent environment," *IEEE/ASME Transactions on Mechatronics*, vol. 9, no. 3, pp. 535–543, 2004.

[10] R. Lundh, "Plan-based configuration of a group of robots," Licentiate Thesis. University of Örebro, Sweden, September 2006.

[11] R. Lundh, L. Karlsson, and A. Saffiotti, "Plan-based configuration of a group of robots," in *Proceedings of the 17th European Conference on Artificial Intelligence ECAI*, Riva del Garda, Italy, 2006, pp. 683–687.

[12] B. Morisset, G. Infante, M. Ghallab, and F. Ingrand, "Robel: Synthesizing and controlling complex robust robot behaviors," in *Proceedings of the Fourth International Cognitive Robotics Workshop, (CogRob 2004)*, August 2004, pp. 18–23.

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

[14] L. E. Parker and F. Tang, "Building multi-robot coalitions through automated task solution synthesis," *Proceedings of the IEEE, special issue on Multi-Robot Systems*, vol. 94, no. 7, pp. 1289–1305, 2006.

[15] Player/Stage Project, playerstage.sourceforge.net/.

[16] J. Rao and X. Su, "A survey of automated web service composition methods," in *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC)*, San Diego, California, USA, July 2004.

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

[18] A. Saffiotti, K. Konolige, and E. H. Ruspini, "A multivalued-logic approach to integrating planning and control," *Artificial Intelligence*, vol. 76, no. 1-2, pp. 481–526, 1995.

[19] O. Shehory and S. Kraus, "Methods for task allocation via agent coalition formation," *Artificial Intelligence*, vol. 101, pp. 165–200, 1998.

[20] C. Shekhar, S. Moisan, R. Vincent, P. Burlina, and R. Chellappa, "Knowledge-based control of vision systems," *Image and Vision Computing*, vol. 17, pp. 667–683, 1998.