

Dynamic Self-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—There is a tendency today toward the study of distributed systems consisting of many heterogeneous, networked, cooperating robotic devices. We refer to a system of this type as an *ecology* of robots. We call *functional configuration* of this ecology a way to allocate and connect functionalities among its robots. In general, the same ecology can perform different tasks by using different configuration. Moreover, the same task can often be solved using different configurations, and which is the best one depends on the available resources. This potential flexibility of a robot ecology is reduced by the fact that, in most current approaches, configurations are pre-programmed by hand. In this paper, we propose a plan-based approach to automatically generate a preferred configuration of a robot ecology given a task, environment, and set of resources. In contrast to previous approaches, the state of the ecology is automatically acquired at planning time, and it is monitored during execution in order to reconfigure if a functionality fails. We illustrate these ideas on a specific instance of an ecology of robots, called PEIS Ecology. We also show an experiment run on our PEIS Ecology testbed, in which a robot needs to reconfigure when the original configuration fails.

I. INTRODUCTION

The field of cooperative robotics is maturing, and there is now a tendency to consider complex systems which are fully distributed and highly heterogeneous. A particularly interesting case of this tendency is the emergence of a paradigm in which many robotic devices, pervasively distributed in everyday environments, cooperate in the performance of possibly complex tasks. This paradigm has been spelled out under different names, including network robot systems [13], intelligent spaces [9], sensor-actuator networks [4], ubiquitous robotics [8], and PEIS-Ecology [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. These heterogeneous 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”.

Robotic devices in a robot ecology can be organized to cooperate in many different ways. For instance, an autonomous vacuum cleaner can get location information from tracking cameras in the ceiling, and ask doors to open when moving from one room to the next. In this paper, we call *configuration* any way to instantiate and connect the different functionalities available in the robotic devices who participate in the ecology. Different configurations typically correspond to the performance of different tasks. Moreover, the same task can often be performed using different con-

figurations depending on the availability and cost of the functional resources. The ability to configure a robot ecology in different ways is the key to its flexibility and robustness. However, in most current approaches to robot ecologies, configurations are static and they are programmed by hand. In this paper, we propose a first step in the direction of automatic, dynamic self-configuration of an ecology of robot.

The approach presented here builds upon our previous work [11], in which we used techniques derived from the field of AI task planning to automatically synthesize and deploy a configuration (in fact, a sequence of configurations). In that work, the planner relied on static and hand-coded information about the robot ecology (which functional resources are available, and at what cost). As a consequence, the generated configurations were not sensitive to the current state of the robot ecology, and could not adapt to changes in this state. In this paper, we extend that work in two important ways. First, we let the system acquire in real-time the current state of the robot ecology, in terms of availability and cost of functional resources. Second, we monitor the execution of a configuration in order to detect failures (resources which become unavailable) and to automatically re-configure the system to account for it.

In order to make our discussion concrete, we apply our approach to a specific kind of robot ecology, called PEIS-Ecology [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 actuation functionalities from the doors. A *configuration* of a PEIS-Ecology is, roughly speaking, a way to connect some of the functionalities in the PEIS-Ecology to solve a task.

The rest of the paper is organized as follows. In section 2 we remind the notion of a configuration in the PEIS-Ecology framework. In section 3 we give an overview of the approach. Sections 4 to 7 detail the different steps of this approach: state acquisition, configuration generation, deployment, and monitoring. Section 8 presents experiments. Section 9 discusses some related work and Section 10 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



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.

generate a 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 or as complex as a humanoid robot.

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.

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. But, 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 configuration, so 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 that is able to open its own door, then the robot can simply ask 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 [3] 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.

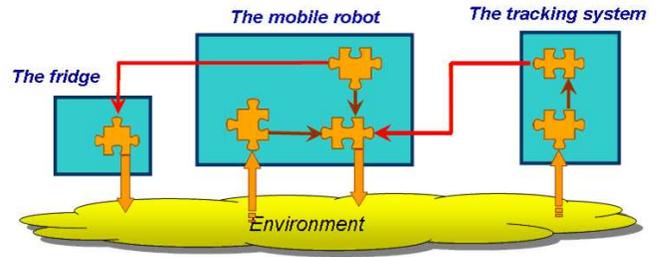


Fig. 2. Functional configuration of the above PEIS-Ecology.

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. This is typically a continuous loop, getting input, producing output.

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 latter functionality subscribe to the tuples produced by the former functionality.

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

A **PEIS-Ecology** is a collection of PEIS capable to communicate with each other, all embedded in the same physical environment.

A **configuration** in a PEIS-Ecology is a subset of PEIS-components within the ecology, together with a set of connections between them. Note that the elements in an 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.

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

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

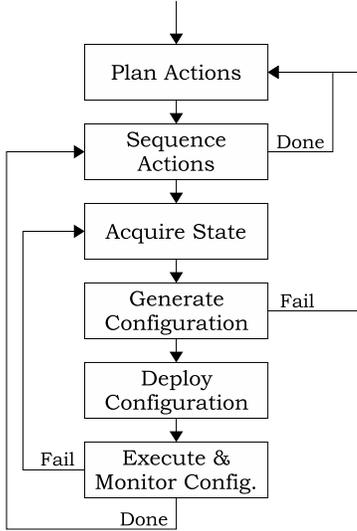


Fig. 3. The different steps of the approach

all functionalities hold in the current world state. A more formal definition of these properties can be found in [11].

Figure 2 shows a functional view of the PEIS-Ecology configuration from Figure 1. 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 together with a laser range finder to provide position information to its navigation component.

A **configuration problem** is the problem of finding an admissible configuration for a particular task given a domain describing all the functionalities in the ecology, a state of the available PEIS-components in the ecology, a state of the environment, and a given task. In the rest of this paper, we show our approach that automatically finds a solution to a configuration problem.

III. APPROACH OVERVIEW

As shown in Figure 3, the approach we use to generate and execute a configuration for a particular task is composed of several steps. Two of the steps, “plan actions” and “generate configuration”, consider a state. To plan actions, a *world state* is used. This is composed of conditions that hold in the world (e.g., robot *r* is in room *A*, door *d* is open). To generate configurations, an *ecology state* is used. This is composed of information specific for the ecology (e.g., available PEIS and PEIS-components).

The steps of the approach are as follows:

- 1) **Plan Actions:** Our present system works by first calling an action planner to find a conditional plan for solving a particular task. The actions in these plans are of the form “move robot to bedroom”, “give wake-up-signal”, and so on. In order to generate a plan, a domain and a world state is required. The domain describes all the actions and the state determines

which actions are available. Currently, the domain and the world state are hand-coded, and we assume that all actions are available. The action planner we use is a sensor-based probabilistic action planner, called PTLplan [6].

- 2) **Sequence Actions:** The sequence of actions from the action planner is executed one after the other. Usually each action requires a different functional configuration, thus the configuration generator (the configuration planner) is used for each action.
- 3) **Acquire State:** The ecology state is acquired to assure that only configurations that are currently admissible will be generated.
- 4) **Generate Configuration:** A configuration is automatically generated for the ecology based on the acquired state, the action from the action planner, and a domain description of all PEIS-components.
- 5) **Deploy Configuration:** Before a configuration can be executed it needs to be deployed (instantiated) on the ecology (i.e., inactive PEIS-components must be activated, channels must be created, etc).
- 6) **Execute and Monitor Configuration:** When the PEIS-components are started, the execution and monitoring of the configuration is started.

In this paper we mainly focus on state acquisition, configuration generation, configuration deployment, and execution and monitoring of configurations. A detailed description of the other steps can be found in [11], [10].

IV. STATE ACQUISITION

The state of the ecology is a representation of the facts that currently hold in the ecology, i.e., the state contains information about which PEIS are currently available, which PEIS-components are on/off, and their current cost. These facts are represented as literals, e.g., $peis(fridge1)$ – fridge1 is a PEIS; $pc(fridge1, door, d1)$ – fridge1 has a door *d1* that is a PEIS-component; $pc(fridge1, RFID-reader, rf1)$ – fridge1 has a RFID-reader *rf1* that is a PEIS-component; $cost(fridge1, d1, 100, on)$ – *d1* costs 100 to use, and it is currently active (*on*), i.e., it does not need to be initialized before it can be used; $cost(fridge1, rf1, 50, off)$ – *rf1* is inactive and costs 50 to start up; $status(fridge1, d1, closed)$ – the status of the PEIS-component *d1* on fridge1 is that it is currently closed.

The facts in the ecology state is used by the configuration planner to ensure that the resulting configuration is admissible and has the lowest cost. The ecology state is obtained from the ecology as follows.

To find out which PEIS-components are available in the PEIS-Ecology, subscriptions are made on the name tuples of all PEIS-components. For each component that is found, the following information is also retrieved: on which PEIS it is located, which cost is associated with that particular component, and its current status. On each PEIS there is a special PEIS-component called PEIS-init that offers information about which components are located on the PEIS, if they are on/off, and the cost of starting them. The information

```

(functionality
name: tracking-system(p, t, o)
input: images(p)
output: global-pos(o)
precond: peis(p), pc(p, tr-sys, t)
postcond: -
cost: 10
)

(config-method
name: get-location-info(r)
precond: peis(r), peis(p),
pc(p, tr-sys, t), pc(p, cams, c), r!=p
output: f2: global-pos(r)
channels: local(p, f1, f2, image(p))
body:
f1: cameras(p, c)
f2: tracking-system(p, t, r)
)

```

Fig. 4. A functionality operator schema, and a method schema for combining functionalities. (Syntax simplified for readability.)

from the PEIS-inits is used to also include inactive PEIS-components in the state. It is not possible to get the cost of inactive PEIS-components, and therefore for those the cost in the state is actually the cost of starting them. In the next section we describe how these facts are used in the planning process.

V. CONFIGURATION GENERATION

A. Domain Description

In addition to the state, the configuration planner requires a declarative description of the functionalities and methods for connecting them and the goal for what the configuration should produce. 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 functionalities is realized using operator schemas similar to those of AI action planners. One functionality operator schema that we have used in our experiments, *tracking-system*, is shown in the upper half of Figure 4.

As mentioned previously, a PEIS-component implements a functionality, and thus a PEIS-component is an instantiation of an operator schema. In order to include a functionality in a configuration, there must be at least one PEIS-component that implements that functionality. To verify that this is true, each operator schema has a precondition that there must exist a PEIS-component in the acquired state that implements the functionality. The *precond* and *postcond* fields encode the causal preconditions — in this case, that *p* is a PEIS with a PEIS-component tracking system (i.e., there is a tracking system in the state). The *name* field specifies the name and parameters of the functionality. The fields *input* and *output* specify the inputs (images) and the outputs (global-pos) of the functionality.

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 tracking system functionality.

The lower half of Figure 4 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* are 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 *tracking-system*.

The configuration planner takes as input a state *s* that is acquired from the ecology (see Sec IV), a stack of (unexpanded) method instances with initially one instance $l : m(c_1, c_2, \dots)$ 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:

- 1) Take the unexpanded method instance $l : m(c_1, c_2, \dots)$ at the top of the stack.
- 2) If $l : m(c_1, c_2, \dots)$ matches the name of a functionality operator *O* which has preconditions holding in the dynamically acquired state *s*, instantiate it and add the resulting functionality to the current configuration. We also calculate the cost of that functionality for that specific instantiation (the cost for the PEIS-component), first by querying the ecology state, and if that fails we use a default value from the operator. There is also an extra cost associated with having inactive PEIS-components in a configuration.
- 3) If $l : m(c_1, c_2, \dots)$ matches a method schema in *M* which has preconditions holding in the dynamically acquired state *s*, instantiate and expand that method schema. Add the channels 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 algorithm above the entire state is acquired before planning starts. For large PEIS-Ecologies, this could be a time consuming process. However, the algorithm is easy to modify such that the state is acquired on demand for those parts of the ecology that are currently of interest.

The output from the planner is a configuration description *C*, which essentially consists of a set of functionality names with labels, e.g., *18: tracking-system(hsm, ls1, astrid)*, and set of channels, e.g., *local(p, 12, 13, image(p))*, where labels 12 and 13 refer to functionalities.

Generally, there are several configurations that can solve a problem, but obviously, only one configuration per problem

can be performed at the time. As search strategy we use best-first search with branch-and-bound to find the admissible configuration with the lowest cost that is defined by the functionalities and methods in the domain.

VI. DEPLOYMENT OF A CONFIGURATION

In order to execute a configuration, we need to deploy it on the ecology. The deployment phase consists of three steps. The first step is to verify that the PEIS-components associated with the functionalities of the description are active. If they are not active we need to activate them through the PEIS-init on that PEIS. The second step is to set up the channels between the PEIS-components. Since communication in the PEIS-Ecology is done using a tuple space, the channels are setup by telling the different PEIS-components which information they should subscribe to and from whom. The third and last step of the deployment phase is to let the sequencer and the monitoring process subscribe to different status information of the PEIS-components. The sequencer subscribes to status information that tells if an action is accomplished or not. That is, the PEIS-components that have a termination condition (e.g., move to actions are terminated when their destinations are reached) will publish a tuple to notify the sequencer that they are done. The monitoring process subscribes to status information that describes if a PEIS-component functions correctly or not.

VII. CONFIGURATION EXECUTION AND MONITORING

When the configuration is executed we have an information flow that goes from the PEIS-components with sensing capabilities, through PEIS-components that process information to PEIS-components with terminating actions to perform.

The execution of the configuration continues until the completion of a step is reported by the PEIS-components that are the endpoints of the configuration. For instance, for a navigation task, the navigation module of the robot determines when it has reached the desired position. When the step is completed, the system proceeds to the next step and generates a new configuration, and so on. Of course, this is under the assumption that each PEIS-component in the configuration is able to perform its task without failing.

In case of failure, there are two levels of recovery in our approach: on the configuration level and on the action level (see Fig. 3). On the configuration level, the status of all PEIS-components are monitored by subscribing to their status tuples. If a PEIS-component fails it posts this in the status tuple which is received by the monitoring process that can start the reconfiguration. A reconfiguration consists of the same steps as the configuration, except that when the state is acquired, the failing PEIS-components are excluded. Since the monitoring of the configuration is done by subscribing to a status tuple of all PEIS-components, it is required that the individual PEIS-components are able to discover by themselves that they are not functioning correctly, i.e., the PEIS-components need to monitor their own performance. There are of course many different aspects that this individual monitoring could incorporate. What aspects to monitor and



Fig. 5. Snapshots of the PEIS-Home. (Left) Pippi in kitchen. (Right) Astrid with the newspaper in the gripper.

how to perform the monitoring of the individual PEIS-components is up to the developer of the PEIS-component.

The second level of recovery is on the action level. This level is activated if the configuration planner cannot find an admissible configuration for the current action. If this is case, the action is marked as failed and not available when the recovery process either replans the entire task or tries to repair the current plan [1].

VIII. EXPERIMENTS

We now describe an illustrative experiment that shows how the state acquisition, configuration generation, deployment, execution, and monitoring work in practice. The aim of this experiment is to show that it is possible to automatically find different configurations for the same task and to reconfigure if the original configuration fails.

A. Experimental Setup

For the experimental part, we have used a physical test-bed facility, called the PEIS-Home, which looks like a typical apartment of about $25m^2$. It consists of a living-room, a bedroom and a small kitchen. The PEIS-Home is equipped with a communication and computation infrastructure, and with a number of PEIS. The following PEIS are of particular importance for our experiments.

Astrid the mobile robot PEIS: A PeopleBot indoor robot from ActivMedia Robotics (see Fig. 5 right). On-board Astrid runs an instance of the Thinking Cap (TC), 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.

Astrid also has an action planner and a configuration planner, and the reconfigurations of the PEIS-Ecology in these experiments are done from here.

Pippi the mobile robot PEIS: An iRobot Magellan Pro indoor robot (see Fig. 5 left) equipped with a Pan-Tilt CCD color camera. Similar to Astrid, Pippi runs an instance of

the Thinking Cap, and an instance of the player program. In addition to this she has a PEIS-component for localizing and tracking objects (e.g., other robots). Pippi also has an action planner and a configuration planner, to use if she needs help.

The Home Security Monitor PEIS: A stationary computer which is connected to a set of web-cameras mounted in the ceiling. In addition to other monitoring tasks, not relevant here, the HSM provides a PEIS-component that is able to track a robot and localize it in the PEIS-home.

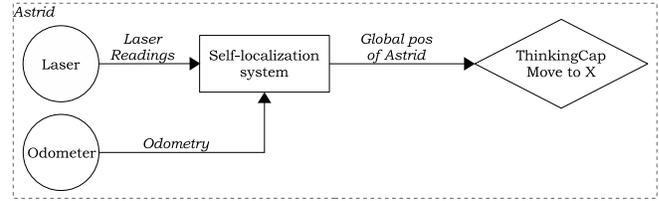
B. Experimental Execution

Each experimental run consisted of the following:

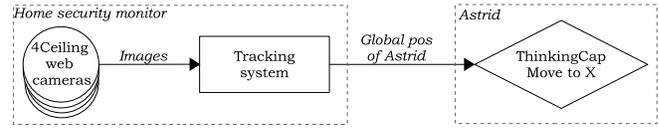
- 1) At start-up, Astrid and Pippi are in the living-room. When the morning paper arrives, Astrid is assigned the task to wake up and deliver the newspaper to Johanna, the resident of the apartment, who is sleeping in the bedroom.
- 2) With this high level goal, the action planner located at Astrid, generates a simple plan. The plan consists of “move to entrance”, “get newspaper”, “move to bedroom” and “give wake-up-signal”.
- 3) This plan is executed by first acquiring the state of the ecology, then generating and deploying a configuration for moving Astrid to the entrance. The configuration includes web cams in the ceiling, connected to a computer which estimates Astrid’s position and posts it as tuples. Astrid’s navigation system reads these position tuples when she moves to the entrance.
- 4) When Astrid arrives at the entrance, this is signaled to the sequencer that takes the next action “get newspaper”. Again, the state is acquired, and a configuration is generated and deployed. The configuration for this action is very simple and only contains an action for closing the gripper around the newspaper.
- 5) With the newspaper in the gripper, the next action “move to bedroom” is considered by the configuration process (acquire state – plan config. – deploy). The same configurations as for “move to entrance” is considered, and the one with ceiling cameras is chosen.
- 6) During the execution of “move to bedroom”, the tracking system on HSM fails since one of the cameras is not able to produce images anymore. This failure halts the execution of the configuration and a reconfiguration is started. When Astrid reconfigures, she again acquires the state, generates a configuration for the new state and deploys the new configuration. This time Astrid uses the on-board laser with scan matching to localize.
- 7) When the navigation system on Astrid realizes that she is in the bedroom, it signals this to the sequencer that sends the next action (“give wake-up-signal”) to the configuration planner. The generated configuration for this action contains only one PEIS-component: the speaker system on Astrid. When Johanna is awake, she can take the news paper from Astrid’s gripper.

The important steps of the experiment are the steps involving state acquisition and monitoring, that is, steps 3–7. In step 3, before Astrid can do her configuration planning,

Action: “Move to entrance/bedroom” with Laser



Action: “Move to entrance/bedroom” with HSM



Action: “Move to entrance/bedroom” with Pippi

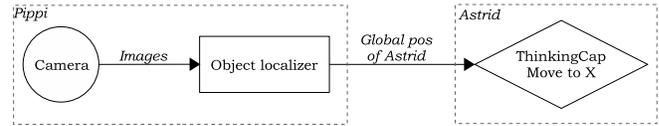


Fig. 6. The configurations for: “move to entrance/bedroom” with laser and scan matching, with ceiling cameras, and with another robot guiding. The configurations are simplified for readability (e.g., the player component is not shown since it is always connected to thinking cap).

she acquires the state of the ecology. The acquired state looks like this.

```

peis(Astrid), peis(HSM), peis(Pippi),
pc(Astrid, player, pl1), cost(Astrid, pl1, 20, on),
pc(Astrid, laser, l1), cost(Astrid, l1, 100, off),
pc(Astrid, odomet, od1), cost(Astrid, od1, 10, on),
pc(Astrid, TC, tc1), cost(Astrid, tc1, 20, on),
pc(Astrid, self-loc, s1), cost(Astrid, s1, 20, on),
pc(Astrid, gripp, gr1), cost(Astrid, gr1, 90, off),
pc(Astrid, speak, sp1), cost(Astrid, sp1, 10, off),
pc(HSM, cams, ca1), cost(HSM, ca1, 20, on),
pc(HSM, tr-sys, tr1), cost(HSM, tr1, 20, on),
pc(Pippi, cam, ca2), cost(Pippi, ca2, 20, on),
pc(Pippi, player, pl2), cost(Pippi, pl2, 20, on),
pc(Pippi, obj-loc, ol2), cost(Pippi, ol2, 130, on)
  
```

To execute the “move to entrance” action, the navigation system on Astrid needs to know her position. In this situation, there are three different ways that Astrid can retrieve this information (see Fig. 6): (1) She can update her initial position using odometry measurements and laser scan matching, (2) she can let the web cams in the ceiling track her position and send it to her, or (3) she can let Pippi track her position and send it to her. When generating the configuration, all alternatives are considered, and as a result the second configuration is generated since it only has a cost of 80. Configuration 1 would cost 170 (the laser costs 100 to activate and has 20 as default functionality cost), and configuration 3 would cost 190.

In step 4, Astrid closes the gripper around the newspaper. To overcome the difficult manipulation task, the newspaper was placed in the gripper by a human.

Step 5 is very similar to step 3. In step 6 however, the tracking system discovers that one of the cameras tracking the robot fails. This failure is received by the monitor as

the tracking system posts a status tuple. The configuration is then halted and the ecology state is again acquired, excluding the components suffering from failure. This time, only the second and third configuration in Fig. 6 are possible. The configuration using laser still has the lowest cost even though it needs to be activated before it can be used.

In step 7, when Astrid is inside the bedroom, she wakes up Johanna by giving a wake up signal.

IX. 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], dynamic software architectures [2], automatic web service composition [16], coalition formation [19], and single robot task performance [12] [7]. However, in the field of ecologies of robots and robots acting in intelligent environments, there are, to our knowledge, no works that address similar problems.

In the area of cooperative robotics, several works address the problem of task allocation/assignment [5]. However, the problem addressed here is essentially different from task allocation. Task allocation typically deals with the question: “Who should do which task?”. That is enough for tasks that only require loose coordination. For task that require tight cooperation (that is, they cannot be neatly divided among robots or modules), we must address the additional question: “How to execute a task in a cooperative manner?”. Configuration generation answers this question and can be seen as a succeeding step to task allocation, done after a task is assigned to a robot.

Parker and Tang [14] present an approach called ASyMTRe that also addresses this question. 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. Similar to our approach, ASyMTRe does not deal with detecting the actual component failures leaving this to the sensor itself or an external system. Our approach 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.

X. CONCLUSIONS

We have described an approach to the automatic on-line generation of a configuration of an ecology of robots using plan-based techniques. In contrast to previous work, we address the problems of how to acquire the state of the ecology automatically and how the system can reconfigure in response to failures. By acquiring the state at runtime we improve the generation of the “best” configuration since we assure that the PEIS-components are present in the ecology. The PEIS-components are also able to dynamically set their own costs, which opens up for a more market-based approach of deciding which PEIS-components to use. The reconfiguration is performed at two levels: first the configuration planner gets the chance to find a new configuration for the action for

the new ecology state; if this fails, the action planner gets the chance to repair the plan.

In this paper, we have illustrated our approach in our PEIS-Ecology testbed. 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. Examples of applications of our approach to other cooperative robotics tasks can be found in [10].

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] A. Bouguerra and L. Karlsson, “Symbolic probabilistic-conditional plans execution by a mobile robot,” in *IJCAI-05 Workshop: Reasoning with Uncertainty in Robotics (RUR-05)*, 2005.
- [2] J. Bradbury, J. Cordy, J. Dingel, and M. Wermelinger, “A survey of self-management in dynamic software architecture specifications,” in *Proc of the Int Workshop on Self-Managed Systems (WOSS’04)*, 2004.
- [3] 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.
- [4] F. Dressler, “Self-organization in autonomous sensor/actuator networks,” in *Proc. of the 19th IEEE Int Conf on Architecture of Computing Systems*, 2006.
- [5] B. Gerkey and M. J. Mataric, “A formal analysis and taxonomy of task allocation in multi-robot systems,” *Int Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.
- [6] L. Karlsson, “Conditional progressive planning under uncertainty,” in *Proc of the 17th Int Joint Conf on Artificial Intelligence (IJCAI)*, 2001, pp. 431–438.
- [7] D. Kim, S. Park, Y. Jin, H. Chang, Y.-S. Park, I.-Y. Ko, K. Lee, J. Lee, Y.-C. Park, and S. Lee, “SHAGE: a framework for self-managed robot software,” in *SEAMS’06*, 2006.
- [8] 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)*, 2004.
- [9] J. Lee and H. Hashimoto, “Intelligent space – concept and contents,” *Advanced Robotics*, vol. 16, no. 3, pp. 265–280, 2002.
- [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 an ecology of robots,” in *Proc of the Int Conf on Robotics and Automation (ICRA)*, 2007.
- [12] B. Morisset, G. Infante, M. Ghallab, and F. Ingrand, “Robel: Synthesizing and controlling complex robust robot behaviors,” in *Proc of the 4th Int Cognitive Robotics Workshop, (CogRob)*, 2004, pp. 18–23.
- [13] Network Robot Forum, www.scit.or.jp/nrf/English/.
- [14] L. E. Parker and F. Tang, “Building multi-robot coalitions through automated task solution synthesis,” *Proc of the IEEE*, 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 *Proc of the First Int Workshop on Semantic Web Services and Web Process Composition (SWSWPC)*, 2004.
- [17] A. Saffiotti and M. Broxvall, “PEIS ecologies: Ambient intelligence meets autonomous robotics,” in *Proc of the Int Conference on Smart Objects and Ambient Intelligence (sOc-EUSAI)*, 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.