Proceedings of the European Conference on Artificial Intelligence (ECAI)
Patras, Greece, July 2008.

Automatic Configuration of Multi-Robot Systems: Planning for Multiple Steps

Robert Lundh and **Lars Karlsson** and **Alessandro Saffiotti** [1]

Abstract. We consider multi-robot systems where robots need to cooperate tightly by sharing functionalities with each other. There are methods for automatically configuring a multi-robot system for tight cooperation, but they only produce a single configuration. In this paper, we show how methods for automatic configuration can be integrated with methods for task planning in order to produce a complete plan were each step is a configuration. We also consider the issues of monitoring and replanning in this context, and we demonstrate our approach on a real multi-robot system, the PEIS-Ecology.

1 Introduction

One essential property of planning is that it is possible to detect, before hand, if a task can be executed or not. It is common sense for traditional action planning that if there is a plan that achieves the goal we know that it is possible to accomplish the task. If we cannot find a plan, we cannot accomplish the task. For the planning problem we address in this paper, it is not enough that we find an action plan for a task. Why this is the case will be explained later in this section.

We assume that in a multi-robot system, each robot has a set of capabilities. If the robots are heterogeneous, they have different capabilities, and can thus provide a number a different functionalities which can be used to cooperate in different ways. For example, in a navigation task a robot must know its own position relative to the goal position, i.e., it needs to be localized. This position information can either be provided by using its own sensors, or another robot can provide this information by tracking the first robot and send the estimated position. We call *configuration* any way to instantiate and connect the different functionalities available in the multi-robot system. An action like *move the robot Pippi to the livingroom* can be implemented as a configuration. Different actions typically correspond to different configurations. Moreover, the same action can often be performed using different configurations depending on the availability and cost of the functional resources. The ability to automatically configure a multi-robot system in different ways for an action is the key to its flexibility and robustness.

Configurations typically implement one action at a time, and if a task requires several steps (actions) to be achieved there is also a need for several configurations to be executed one after the other. This can be considered as two different problems: (1) finding a sequence of steps (a plan) and (2) finding configurations for the individual steps. However, the two problems are not independent of each other. When problem 1 is considered, it is not possible to know, before hand, that the generated plan can achieve the goal of the task unless problem 2 is also considered. That is, there might be steps in the plan for which no configuration can be found, and that would make the plan non-executable. This could happen in our earlier approach [7], were these two problems were considered in sequence: an action plan was first generated, and then configurations were generated online as they were needed. This paper proposes a better integration of problems 1 and 2. The approach considers both problems at planning time and can tell before hand if a task is executable or not.

The single step configuration problem has been studied in several research areas, e.g., single robot task performance [8, 5], network robot systems [1, 3], and cooperative robotics [10]. However, none of these approaches consider sequences of configurations. There are also some works about integrating task planning with more detailed types of reasoning, such as the aSyMov planner [2] which combines symbolic and geometric reasoning.

The rest of the paper is organized as follows. In Section 2 we give a reminder about the notion of functional configurations. In Section 3 we describe different solutions to integrated action planning and configuration generation. Sections 4 details the different parts of the approach and Section 5 presents an illustrative experiment.

2 Functional Configurations

For the configuration part of our approach we use the approach proposed in [6, 7]. We here give a brief description of the concepts of functional configurations.

We assume that the world can be in a number of different states. The set of all states is denoted S. There is a number of robots $r_1, \ldots, r_n$. The properties of the robots, such as what sensors they are equipped with and their current positions, are considered to be part of the current state s_0. Robots are assumed to have modular functionalities that can be accessed and used independently, across and within the robots.

A *functionality* f is an operator that perform computations, sensing or actuation. It is characterized by the following elements:

- A specification of *inputs* I to be provided by other functionalities, including information about domain (e.g., video images), timing (e.g., 25 fps), etc.
- A specification of *outputs* O provided to other functionalities, also containing domain and timing information.
- A set of causal *preconditions* Pr: conditions in the environment that have to hold in order for the functionality to be operational.
- A set of causal *postconditions* Po: conditions in the environment which the functionality is expected to achieve.
- A specification of *costs* $Cost$, e.g., computation and energy.
- A *body* Φ, containing the code to be executed. This is typically a continuous loop, getting input, producing output.

[1] Center for Applied Autonomous Sensor Systems, Örebro University, Sweden. email: {robert.lundh, lars.karlsson, alessandro.saffiotti}@aass.oru.se

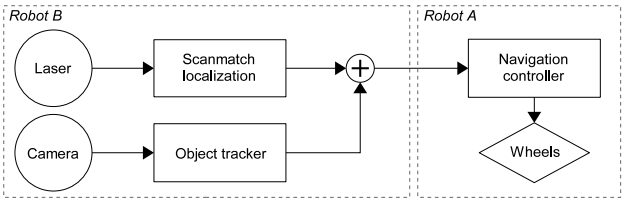

Figure 1. An example configuration

A *channel* ch transfers data from an output of a functionality to an input of another functionality.

A *configuration* C is a pair $\langle F, Ch \rangle$, where F is a set of functionalities and Ch is a set of channels.

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 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 precise definition of these properties can be found in [6].

As an example of a configuration (see Fig 1), consider a task in which robot B helps robot A to navigate. Robot A has two functionalities: a navigation controller and a wheel actuator. Robot B has four functionalities: a camera, an object tracker, a laser, and a scanmatch localization algorithm. The camera is connected to the tracker to obtain the position of A relative to B. The laser is connected to the localization to obtain the absolute position of B. These positions are combined to get the absolute position of A, which is sent to the controller on A that provides motion commands to the wheels.

Configuration problem Let Σ be a multi robot system, and let D be a domain describing, in some formalism, all the functionalities that exist in Σ. D implicitly defines the set $\mathcal{C}(D)$ of all the configurations that can be built in Σ (both admissible and not admissible). Let A denote an action (or task), and s denote the current state. A *configuration problem* $\langle A, D, s \rangle$ for Σ is the problem of finding a configuration $c \in \mathcal{C}(D)$, admissible in state s, to perform A.

Configuration planning To find a solution to a configuration problem we use a configuration planner [6]. The configuration planner uses techniques inspired by hierarchical planning, in particular the SHOP planner [9] in order to combine functionalities to form admissible configurations that solve specific tasks. This is done by searching the space of configurations to find one which is admissible in the current state and which has the lowest cost.

The configuration planner takes as input a domain that describes the existing functionalities, a state of the available functionalities, and a goal (action). The configuration planner returns a configuration description, which essentially consists of a set of functionality names, and set of channels describing how the functionalities can be connected. It also returns the pre- and post-conditions and the total cost of the configuration.

3 Integrated action and configuration planning

The configuration problem above is concerned only with finding a configuration for one action. However, in practice most tasks require more than one action to be completed. For instance, if the robot wants to wake up a person, the robot must first reach the bedroom, then move close to the bed, *before* it can wake the person up.

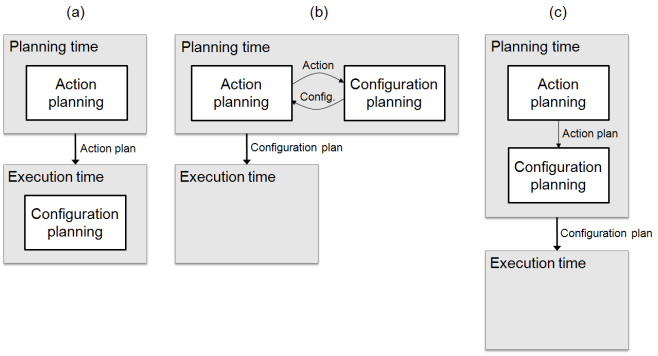

Figure 2. Different ways to combine action and configuration planning. **(a)** Independent. **(b)** Fully integrated. **(c)** Loosely coupled.

Different configurations may be required for each of these actions. We call such a plan, where each action is a configuration, a *configuration plan*. A configuration plan is a sequence of configurations $CP = \langle c_1, \ldots, c_k \rangle$, where $k \geq 0$. Note that from now on we reserve the term *task* to denote the top-level task, and use the term *action* to denote each individual sub-task achieved by each configuration. A configuration plan is admissible if and only if each c_i is admissible in the state s_{i-1} it will be executed in. Note that each configuration can also change the state according to its postconditions. Thus, a domain D can be considered to define a state-transition system $\langle S, C, \gamma \rangle$ with states S, configurations $C = \mathcal{C}(D)$, and a transition function $\gamma : S \times C \rightarrow S$ defined according to the pre- and postconditions of the configurations. The state-transition function defines the states $\langle s_1, \ldots, s_k \rangle$ in which configurations are executed. Thus, the domain D implicitly defines the set $\mathcal{CP}(D)$ of all the configuration plans that can be built in Σ (both admissible and not admissible). Let then T denote a task (or goal state), and s_0 denote the initial state. A *configuration plan problem* $\langle T, D, s_0 \rangle$ is the problem of finding a configuration plan $CP \in \mathcal{CP}(D)$ to perform T, which is admissible from starting state s_0. In the remaining part we detail and discuss solutions to the configuration plan problem.

The job of an action planner is typically to find a sequence of atomic actions $\langle a_1, \ldots, a_k \rangle$ that achieves a goal or task T. From a configuration perspective, each action a_i can then be seen as an abstraction of the set of configurations $\{c_{i1}, \ldots c_{in}\}$ that can implement it. Hence, combining an action planner with a configuration planner would let the robots deal with tasks that require more than one configuration/action to be performed.

There are several ways this combination could be done. These ways can be described with the following variables: (1) If the decisions about what actions to perform (i.e., the action planning) should be taken at planning time or execution time. (2) If the actions should be expanded into configurations (i.e., the configuration planning) at planning time or at execution time. (3) If the action and configuration planning should be done independently of each other or not. We here present three different settings for the variables above.

Independent action and configuration planning In [7], a simple approach to combine an action planner and a configuration planner is presented. It works by first calling the action planner to find an action plan $\langle a_1, \ldots, a_k \rangle$ for solving a particular task. That is, the decisions about which actions to perform (1) is done at planning time. This plan is then executed action by action. For each action a_i that is performed, a suitable configuration c_i is generated by the configuration planner at the time when the action must be ex-

ecuted. Thus, for (2) the decision about when to expand actions into configurations is taken at execution time. The action planning decisions and the configuration planning decisions are taken independently of each other.

Fully integrated action and configuration planning The second way is to have the planners fully integrated. Both the decisions about what actions to perform (1) and the expansion of the actions into configurations (2) are taken at planning time. The decisions for 1 and 2 are fully interdependent, i.e., the configuration planner is called immediately to generate configurations for each action that is considered during search, so the system is working directly with configuration plans $\langle c_1, \ldots, c_k \rangle$. In this way it is possible to cut parts of the search space based on the availability of configurations and to only create admissible configuration plans.

Loosely coupled action and configuration planning In this paper, we present an approach that is based on the idea to generate an action plan and configurations for this plan *before* we start to execute it. First a complete action plan $\langle a_1, \ldots, a_k \rangle$ is generated, and then for that plan, a configuration is generated for each action: $\langle c_1, \ldots, c_k \rangle$. That is, both the decision on actions to perform (1) and the expansion of actions into configurations (2) are done at planning time as in the fully integrated approach above. However, configuration generation is only done when a complete action plan has been found, in order to validate that plan. If the action plan is not valid (i.e., there are not configurations for all actions), the control returns to action planning to generate an alternative action plan, taking into account information about the failed action and its state, and so on. In this way, it is possible know if there is an admissible configuration plan for the generated action plan.

In Fig. 2, the three different cases are shown side by side for comparison. The independent approach (Fig. 2a) assumes that the two planning problems can be addressed independently of each other. This approach has problems when an action cannot be expanded into a configuration at execution time. If this happens, a new action plan must be generated that fulfills the goal. Since some actions may be irreversible, there may be situations in which this solution would not be able to complete the task. Even if a new plan can be found, the fact that the actions in the failed plan were executed leads to a suboptimal performance. The fully integrated approach (Fig. 2b) considers both planning problems simultaneously. It is possible to guarantee that the generated configuration plans are admissible and optimal. However, since configurations are generated for all actions in the search space (even the actions that do not lead to the goal), the complexity of the problem makes it unusable in most practical cases. The loosely coupled approach (Fig. 2c) can, like the fully integrated approach, guarantee that the generated configuration plan is admissible. It avoids the complexity problems of the integrated approach by only trying to generate configurations for actions that are on a path to the goal. Compared to the independent approach, the loosely coupled approach can reject bad action plans before they are actually executed, and find better alternatives. The price to pay is that global optimality of the configuration plan cannot be guaranteed in general.

4 Implementation

The loosely coupled action planning and configuration generation approach has been implemented and tested on a special case of a multi-robot system, called the PEIS-Ecology.

4.1 The PEIS-Ecology testbed

The concept of PEIS-Ecology was originally proposed by Saffiotti and Broxvall [13]. 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 generalizes the notion of robot, and it can be as simple as a toaster or as complex as a humanoid robot. A PEIS-Ecology consists of a number of PEIS embedded in the same physical environment, and endowed with a common communication and cooperation model. Communication relies on a shared tuple-space: PEIS exchange information by publishing tuples and subscribing to tuples. Cooperation relies on the notion of linking functional components: each PEIS can use functionalities from other PEIS in the ecology to complement its own. The PEIS-Ecology model has been implemented in an open-source middleware, called the PEIS-kernel [11].

4.2 Top-level process

To be used in a practical multi-robot system, such as the PEIS-Ecology, action planning and configuration planning must be embedded in a larger process. This process must implement the integration of the two planners, and it must also consider the following aspects. First, both action planning and configuration planning depends on the current state of the environment and the system. Hence, this state should be dynamically acquired before the planning is started. Second, when the action plan is executed, each generated configuration should be instantiated in the actual PEIS-Ecology, and the configuration execution should be monitored in order to decide when to switch to the next action, and to detect possible failures.

Fig. 3 gives an overall view of the top-level process. In this process, there are several "paths" for different situations. The solid arrows constitute the normal path. That is, all the different steps ($1-8$) are completed without any discrepancies. The dotted arrows represent different recovery paths. The rest of this section details the different steps and paths of the top-level process. The top-level process is run by one single robot that configures the PEIS-Ecology to help it solving the top-level task. The steps that consider state acquisition, configuration deployment, and configuration execution and monitoring are also reported in [7].

4.3 Planning

As noted above, both action and configuration planning use state information to ensure that both action plans and configurations are admissible. This state consists of two parts: system state and world state. The *system state* contains information relative to the system itself, e.g., which functionalities are currently available, and what is their current cost. The *world state* is a representation of the facts that currently hold in the environment, e.g., information about rooms and places, how they are connected, etc. To acquire the current (system and world) state from the PEIS-Ecology, we use the mechanisms provided by the PEIS-kernel.

In order to generate action plans, we employ a state of the art action planner called PTLplan [4]. It requires as input a domain and a world state. The domain describes all the actions potentially available, and it is hand-coded. The state, acquired right before the planning is done, determines which actions are actually available in the current situation.

An action plan consists of actions like "move(Pippi, bedroom)", "dock(Pippi, bed)", and "wakeup(Pippi, Johanna)". This plan is

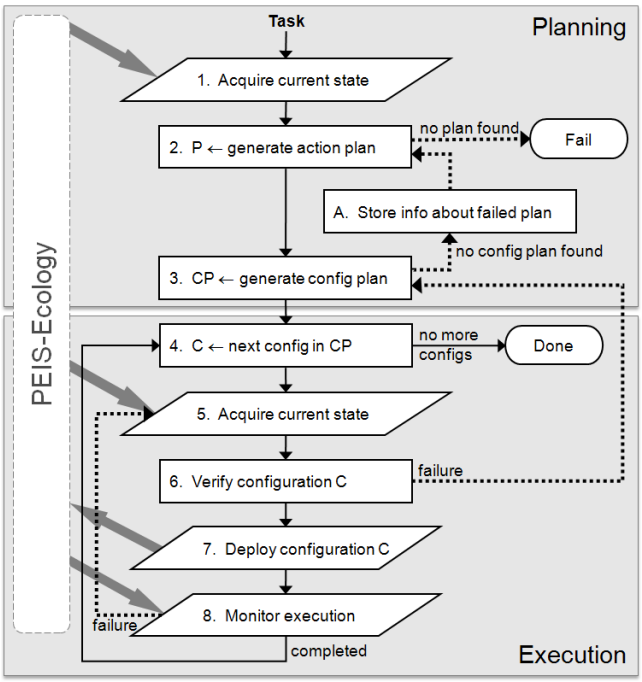

Figure 3. Flow chart of the top-level process.

given to the configuration planner (step 3 in Fig. 3). In this step, a configuration is generated for each action in the action plan, thus creating a configuration plan. If there is a problem of finding a configuration for an action, information about that action and its state is stored (step A in Fig. 3). The action planner is then called again, and it removes that particular action in that particular state and then tries to find an alternative sequence of actions that can achieve the task. If an alternative action plan is found, it is given to the configuration generator which again tries to turn it into a configuration plan.

4.4 Execution

When a configuration plan is found, it is given to a sequencer (step 4 in Fig. 3) that is responsible for taking the next configuration in the configuration plan. When an action/configuration is reported to be completed (step 8), the sequencer takes the next configuration in the plan to deploy.

Since all configurations are generated before the execution of the action plan, it is very important to verify that they are still admissible when it is time to execute them. To guarantee this, the state is dynamically acquired before the execution of each action (step 5). The preconditions of the configuration are then checked in the state (step 6). If they still hold, the configuration can be deployed (step 7). If they do not hold, an alternative configuration must be generated (step 3). If there is an alternative configuration, the postconditions of the alternative configuration must be compared with the postconditions of the initial configuration. If they are equal, the configuration can safely be added to the configuration plan and deployed. If they differ, the remaining part of the configuration plan must be regenerated to comply with the new configuration. In this case, the sequencer does not take the next action in step 4, but retries the same action. If in step 3 no alternative configuration was found, the information about

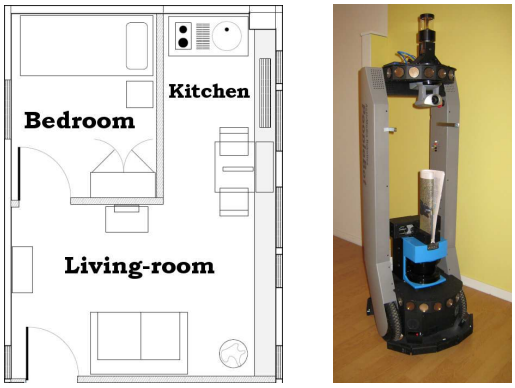

Figure 4. **Left:** A sketch of the PEIS-Home. **Right:** Astrid with the newspaper in the gripper.

this action is stored in step A and the action planner tries to find a new action plan (step 2) as described in the previous section.

Once a configuration description is generated, it must be deployed on the PEIS-Ecology. This involves activating functionalities, setting up the channels between the functionalities, and subscribing to the appropriate signals from the functionalities to know when a configuration is completed or if it has failed.

After a configuration has been deployed, execution (step 8 Fig. 3) continues until the action is completed or it fails. When a configuration is completed, the next one is selected (step 4). If a configuration fails during execution, the top process tries to generate an alternative configuration.

5 An illustrative experiment

We have performed an experiment to show that (and how) the combined planner can handle situations when there are actions for which there is no configuration (this may occur both during planning and execution time). To facilitate comparisons with our previous approach [7], we repeat the scenario presented in that paper where a robot wakes up a person.

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.

Pippi and Astrid: Two PeopleBot indoor robots from ActivMedia Robotics (see Fig. 4 right). Each one runs an instance of the Thinking Cap (TC), an architecture for autonomous robot control based on fuzzy logic [14], and an instance of the Player program [12], which provides a low-level interface to the robot's sensors and actuators. The two robots are identical except that Astrid is equipped with a laser range finder and Pippi is not.

The Home Security Monitor(HSM): 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. HSM also has an action planner and a configuration planner, and the reconfigurations of the PEIS-Ecology in these experiments are done from here. Note however that these could as well be done elsewhere, e.g., in Pippi.

The experiment unfolds as follows:

a. At start up, Pippi is located in the living-room and Astrid in the kitchen. When the morning paper arrives, the HSM wants to wake up Johanna, who is sleeping in the bedroom, and give it to her.
b. With this task, the configuration process acquires the current state (step 1 Fig 3). For this state and task, an action plan is generated (step 2). This plan has the actions: dock-to(Pippi, entrance), take(Pippi, newspaper), move-to(Pippi, bedroom), dock-to(Pippi, bed), wake-up(Pippi, Johanna).
c. In step 3, the search for a configuration for each action is started. For the first three actions, configurations are found. The first and third action (dock-to(entrance), move-to(bedroom)) uses a camera mounted in the ceiling for localization. For the fourth action (dock-to(bed)), the search fails since no configuration can be found. The ceiling camera used in the other actions can only track robots in the living-room and kitchen, and Pippi has no other means of localization. The information about the failed action is stored (step A Fig 3), and the action planer is again called.
d. The action planner finds an alternative plan with the following actions: move-to(Astrid, living-room), dock-to(Astrid, entrance), take(Astrid, newspaper), move-to(Astrid, bedroom), dock-to(Astrid, bed), wake-up(Astrid, Johanna). When revisiting step 3 with the new action plan, it is possible to find a configuration for each action. Unlike Pippi, Astrid is able to localize on its own using a laser range finder and scan matching techniques.
e. The first action/configuration in which Astrid is using the ceiling camera to localize is taken by the sequencer (step 4). It has a lower cost then using the laser. This configuration is then verified (step 6), deployed (step 7), and executed (step 8). When arriving to the living-room, the navigation module signals completion of the action and the next action is prepared for execution.
f. The state is dynamically acquired (step 5). To demonstrate the behavior of the system under dynamically changing conditions, we manually made the ceiling camera unavailable. Thus, in the verification step, the configuration preconditions for dock to entrance using the ceiling camera does not hold. An alternative configuration is generated (step 3) in which the laser is used for localization instead. Since this new configuration has the same postconditions as the original configuration, it can be deployed and executed without regenerating the configuration plan.
g. The remaining actions for take newspaper, dock to bed and waking up Johanna and delivering the newspaper proceeds without any complications and the task is completed.

The critical point of this experiment is step c. where a configuration cannot be found for action dock-to(Pippi, bed) and the action planner is again called to find a new action plan. In the approach with independent action planning and configuration [7], Pippi would have started to execute the action plan and would not discover that it cannot achieve the goal until it reached the bedroom. The HSM would try to find a new plan to reach the goal. The HSM cannot simply generate the same plan as above, where Astrid gets the newspaper at the entrance and delivers it to Johanna, since Pippi is now holding the newspaper in the bedroom. If there is an action for giving a newspaper between robots, the HSM may find an alternative plan, otherwise it will fail.

6 Conclusions

We have presented an approach that, by combining different planning techniques, is able to find a solution for tasks that require sequences of configurations to be completed. For this purpose we employ two different planners: one for action planning [4] and another for configuration planning [6]. The planners are loosely integrated, i.e., configuration planning is used to validate and correct action plans. With this integration, it is possible to guarantee that the execution of a task is not started if there is no admissible configuration plan. In other words, it is possible to know *before hand* if a plan is executable or not. To use a loose integration also makes it easy to replace the current planners with other generation techniques if this is desirable. We have demonstrated the approach in the PEIS-Ecology framework, but it applies to generic multi-robot systems as long as the robots are able to share their functionalities with each other.

An important limitation of the current implementation is that we only consider the execution of a single top-level task. In general, several tasks might be performed concurrently, and new tasks might dynamically appear. A natural extension of the current framework would be to use task allocation techniques to assign different tasks to different configuration processes. With such an extension, issues such as resource handling, conflict resolution and deadlocks must also be considered. Our next step is to consider multiple top-level tasks.

ACKNOWLEDGEMENTS

This work was supported by the Swedish National Graduate School in Computer Science (CUGS).

REFERENCES

[1] D. Baker, G. McKee, and P. Schenker, 'Network robotics, a framework for dynamic distributed architechtures', in *Proc of the IEEE/RSJ Int Conf on Intelligent Robots and Systems*, pp. 1768–1773, (2004).
[2] S. Cambon, F. Gravot, and R. Alami, 'A robot task planner that merges symbolic and geometric reasoning', in *Proc of the European Conf on AI*, pp. 895–899, (2004).
[3] M. Gritti, M. Broxvall, and A. Saffiotti. Reactive self-configuration of an ecology of robots. In: ICRA workshop on Network Robot Systems, 2007.
[4] L. Karlsson, 'Conditional progressive planning under uncertainty', in *Proc of the Int Joint Conf on Artificial Intelligence (IJCAI)*, pp. 431–438, (2001).
[5] 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 *Proc of the Int Workshop on self-adaptation and self-managing systems*, (2006).
[6] R. Lundh, L. Karlsson, and A. Saffiotti, 'Plan-based configuration of a group of robots', in *Proc of the 17th European Conf on Artificial Intelligence (ECAI)*, pp. 683–687, (2006).
[7] R. Lundh, L. Karlsson, and A. Saffiotti, 'Dynamic self-configuration of an ecology of robots', in *Proc of the IEEE/RSJ Int Conf on Intelligent Robots and Systems*, pp. 3403–3409, (2007).
[8] B. Morisset and M. Ghallab, 'Learning how to combine sensory-motor functions into a robust behavior', *Artificial Intelligence*, **172**(4-5), 392–412, (2008).
[9] D. Nau, Y. Cao, A. Lothem, and H. Munoz-Avila, 'SHOP: simple hierarchical ordered planner', in *Proc of the Int Joint Conf on Artificial Intelligence (IJCAI)*, pp. 968–973, (1999).
[10] L. E. Parker and F. Tang, 'Building multi-robot coalitions through automated task solution synthesis', *Proc of the IEEE, special issue on Multi-Robot Systems*, **94**(7), 1289–1305, (2006).
[11] The PEIS ecology project. Official web site. www.aass.oru.se/~peis/.
[12] Player/Stage Project. playerstage.sourceforge.net/.
[13] 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)*, pp. 275–280, (2005).
[14] A. Saffiotti, K. Konolige, and E. H. Ruspini, 'A multivalued-logic approach to integrating planning and control', *Artificial Intelligence*, **76**(1-2), 481–526, (1995).