Can Emil Help Pippi?

Robert Lundh, Lars Karlsson, and Alessandro Saffiotti
Center for Applied Autonomous Sensor Systems
Örebro University, SE-70182 Örebro, Sweden
{*robert.lundh, lars.karlsson, alessandro.saffiotti*}*@aass.oru.se*

***Abstract*—This work is about the use of artificial intelligence (AI) planning techniques to automatically configure a group of cooperating robots. In particular, we study societies of autonomous robotic systems in which robots can help each other by offering information-producing resources and functionalities. A *configuration* in our societies is a way to allocate and connect functionalities among robots. In general, different configurations can be used to solve the same task, depending on the current situation. In this paper, we show a general approach to define, generate, and execute configurations. We use knowledge-based planning to automatically generate a configuration for a given task, environment, and set of resources. We describe an experimental system where these ideas are implemented, and show an example of it in which two robots mutually help each other to cross a door.**

***Index Terms*—Multi-Robot Systems, Cooperating Systems, Planning, Automatic Configuration**

I. Introduction

Consider the situation shown in Figure 1, in which a mobile robot, named Pippi, has the task to push a box across a door. In order to perform this task, Pippi needs to know the position and orientation of the door relative to itself at every time during execution. It can do so by using its sensors, e.g., a camera, to detect the edges of the door and measure their distance and bearing. While pushing the box, however, the camera may be covered by the box. Pippi can still rely on the previously observed position, and update this position using odometry while it moves. Unfortunately, odometry will be especially unreliable during the push operation due to slippage of the wheels. There is, however, another solution: a second robot, called Emil, could observe the scene from an external point of view in order to compute the position of the door relative to Pippi, and communicate this information to Pippi.

The above scenario illustrates a simple instance of the general approach that we suggest in this paper: to let robots help each other by borrowing functionalities one another. In the above example, Pippi needs a functionality to measure the relative position and orientation of the door in order to perform its task: it has the options to either compute this information using its own sensors, or to borrow this functionality from Emil. On first sight this type of cooperation may look like a task allocation problem, however, as we shall explain in the next section, it is a different type of problem.

More generally, we consider a society of autonomous robotic systems embedded in a common environment. Each robot in the society includes a number of *functionalities* organized in some way, for instance, in a generic two-layer hybrid architecture as shown in Figure 2. In these architectures, the top layer implements higher cognitive processes for world modeling (M) and for planning and deliberation (D). The bottom layer implements sensori-motor processes for sensing and perception (P) and for motion control (C), which are connected to a set of sensors (S) and actuators (A).

We do not assume that the robots are homogeneous: they may have different sensing, acting, and reasoning capacities, and some of them may be as simple as a fixed camera monitoring the environment. Thus, each robot may include several, or none, functionalities in each one of the {P, M, D, C, S, A} classes, which it can use to perform the tasks assigned to it. The key point here is that each robot may also use functionalities from other robots in order to compensate for the ones that it is lacking, or to improve its own. In the situation shown in Figure 2, Pippi borrows from Emil a P functionality for measuring the relative position between the

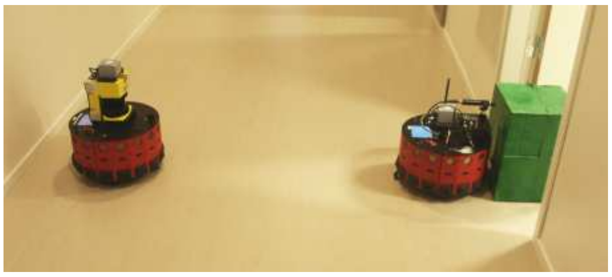


Fig. 1. Can Emil help Pippi to push the box through the door?

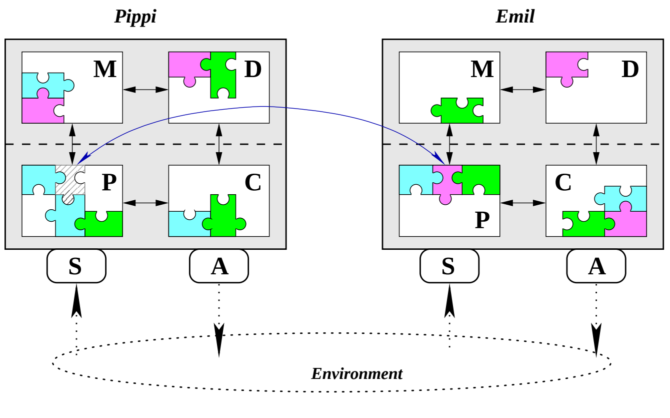


Fig. 2. A simple configuration consisting of two-robots: Emil is providing a missing perceptual functionality to Pippi.

door and itself.

We informally call *configuration* any way to allocate and connect the functionalities of a distributed multi-robot system. Note that we are interested in functional software configurations, as opposed to the hardware configurations usually considered in the field of reconfigurable robotics (e.g., [7], [13]).

Often, the same task can be performed by using different configurations. For example, in our scenario, Pippi can perform its door-crossing task by connecting its own door-crossing functionality to either (1) its own perception functionality, (2) a perception functionality borrowed from Emil, or (3) a perception functionality borrowed from a camera placed over the door. Having the possibility to use different configurations to perform the same task opens the way to improve the flexibility, reliability, and adaptivity of a society of robotic agents. Ideally, we would like to automatically select, at any given moment, the best available configuration, and to change it when the situation has changed.

Our overall research objective is threefold:

1) To formally define the concept of functional *configuration* of a robot society.
2) To study how to *automatically generate* a configuration of a robot society for a given task, environment, and set of resources.
3) To study when and how to *change* this configuration in response to changes in the environment, in the tasks, or in the available resources.

In this paper, we focus on the first two objectives above: the third objective will be the subject of future work. More specifically, we define a concept of configuration which is adequate from the purpose of automatically reasoning about configurations, and show how to use AI planning techniques to generate a configuration that solves a given task. We also describe an experimental system where these ideas are implemented, and show an example of it in which two iRobot Magellan Pro robots mutually help each other to cross a door.

II. Related fields

Although the field of cooperative robotics has received much attention recently, only few works focus on problems which are similar to the one addressed here.

Parker *et al* [16] presents an application for a team of heterogeneous robots that help each other by sharing capabilities. The application involves many different tasks, among which the task of having sensor-rich leader robots guiding simple robots is particularly interesting.

Simmons *et al* [22] consider a task involving a heterogeneous team of robots — a crane, a robot with a manipulator, and a robot with stereo cameras — solving a construction task where a beam is placed on top of a stanchion. This task requires tight cooperation between the robots involved. For specifying tasks, they use TDL (task description language) [21], an imperative language which is a superset of C++.

Our work and the works by Parker *et al* and by Simmons *et al* share similar objectives, that is, to solve tightly-coupled tasks with a team of heterogeneous robots. They both intend to develop general techniques for such applications, but until now it appears that they use configure their teams manually. In contrast to these works, this paper presents a general way to automatically generate configurations of heterogeneous robots for tightly-coupled tasks involving sharing of capabilities.

Benoit *et al* [1] presents a supervision system for a single robot that is able to learn how to perform high level tasks. The system generates modalities, using a hierarchical planner, that consists of a combination of sensory-motor functions. By combining these in to modalities, the robustness of the system improves. An MDP is used to know which modality is appropriate for a particular situation. This approach is similar to the approach presented here in many aspects. The main differences are due to the fact that they consider a single robot, as opposed to our multi-robot system.

In addition to the above works, there are a few related areas from which one might take inspiration to address the objectives of our work.

In the area of multi-robot systems, much work has been done on the problem of multi-robot task allocation, that is, how to allocate a number of tasks to a number of robots taking into account that different robots may be differently adequate for different tasks (see, e.g., [9] for an overview and analysis). Some examples are the ALLIANCE architecture [15] and Local Eligibility approach [26] based on local utility estimates, and the M+ [3] and MURDOCH [8] approaches. Closely related to task allocation are the issues of robotic team configuration and of dynamic role assignment [25], [23], [11]. Chaimowicz *et al* [4] consider roles as the part of an individual agent in a cooperative task. They define a role as a control mode in a hybrid automaton, and a role assignment is a transition in that automaton. The common aspect of all these multi-robot tasks is that the members of the team needs to decide which robot should perform which task. If the team consist of homogeneous members the decision is based upon robot location, work load, among others. If the team is heterogeneous the decision is also based upon the capabilities of the individual team members. However, even when considering teams of heterogeneous robots these approaches do not typically address help in terms of helping each other to achieve individual goals.

The problem of distributing the performance of a task across a number of agents according to their respective capabilities has been widely addressed in the Distributed AI (DAI) and in the Multi-Agent Systems (MAS) communities. Early work in DAI considered distributed problem solving settings with a precedence order among sub-tasks [6]. Later work has included the notion of coalitions between sub-groups of more closely interacting agents [19]. The notions of team-work [17], capability management [24] and norms [2] have also been used in the MAS community to account for the different forms of interactions between the sub-tasks performed by the agents in a team. These works, however, typically assume software agents, and are not concerned with issues of physical action, mobility, and perception, which play a central role in our work.

Another area of interest is program supervision, where program modules are combined, tuned and evaluated in order to solve specific computational tasks such as image processing, often using planning techniques [10], [5], [20]. Our work adds several dimensions to program supervision since we deal with multiple physical agents with both sensing and acting capabilities.

III. Functional configurations

The first goal in our research program is to develop a definition of configuration that is adequate for the three objectives presented in the introduction. In general, a configuration of a team of robots may include interconnected functionalities of two types: functionalities that change the internal state by providing or processing information, and functionalities that change the state of the environment. (Some functionalities can have both aspects.) In the work presented here we focus on the former functionalities.

To define our notion of configurations, a clarification of the three concepts of functionality, resource and channel is in order.

A. Functionality

A *functionality* is an operator that uses information provided by other functionalities to produce additional information. Each instance of a functionality is located in a specific robot (or other agent). The functionality consists of:

- a specification of inputs, to be provided by other functionalities. For each input, it contains information about domain (e.g. video images) as well as timing information (e.g. every 100 ms).
- a specification of outputs, to be provided for other functionalities. They also contain domain and timing information.
- a specification of relations between inputs to outputs.
- a set of causal preconditions, that is conditions in the environment that have to hold in order for the functionality to be operational.
- a set of causal postconditions, that is conditions in the environment which the functionality is expected to achieve.
- possibly also a specification of costs in terms of e.g. computation and energy.

A typical functionality could be the measure door operation mentioned in the introductory example. This functionality takes an image from a camera as an input and measures the position and orientation of a door in the image. To produce the output, the position and the orientation of the door, this functionality has a precondition that needs to be satisfied. The precondition is that the door must be fully visible in the (input) image.

B. Resource

A *resource* is a special case of a functionality. There are two different types of resources: sensing resources and action resources. Only sensing resources will be considered in this paper. A *sensing resource* has no input from other functionalities, and is typically a sensor that gives information about the current state of the surrounding environment or perhaps information about the internal state of the robot. An example of a resources could be a camera. This sensing resource produces images as output as long as the preconditions (e.g. camera is on) are fulfilled.

C. Channel

A *channel* transfers data from one functionality to another. A channel can be in terms of either inter-robot or intra-robot communication, and be on different mediums (radio, network, internal connections). A channel may have requirements of band width, speed and reliability.

D. Configuration

A *configuration* is the set of functionalities and the set of channels that connects functionalities to each other. Each channels connects the output of one functionality to the input of another functionality.

In the context of a specific world state, a configuration is *admissible* if the following conditions are satisfied:

- each input of each functionality is connected via an adequate channel to an output of another functionality with a compatible specification (information admissibility).
- all preconditions of all functionalities hold in the current world state (causal admissibility).
- the combined requirements of the channels can be satisfied (communication admissibility).

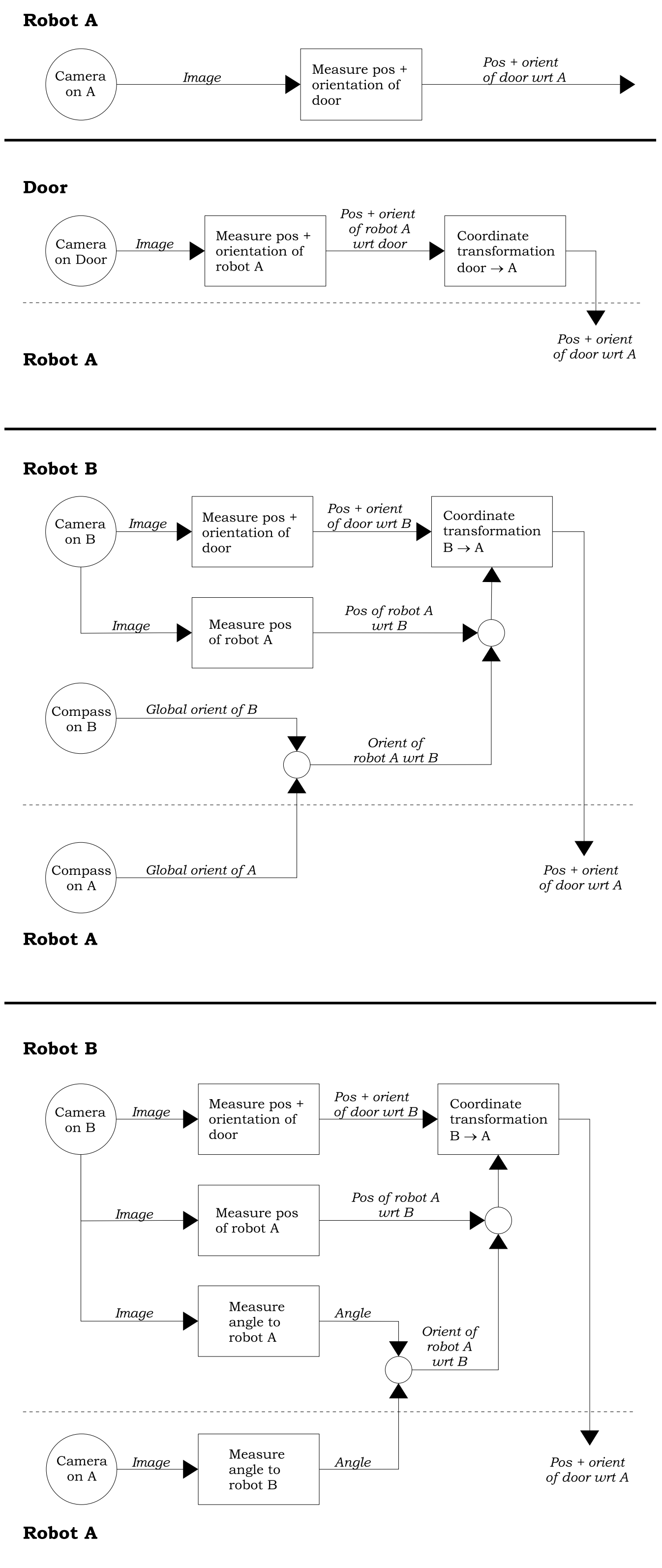
E. Examples

In order to illustrate the above concepts, we consider a concrete example inspired by the scenario described in the introduction. A robot is assigned the task of pushing a box from one room to another one by crossing a door between the two rooms. The “cross-door” action requires information about position and orientation of the door with respect to the robot performing the action. The resources available are two indoor robots (including the one crossing the door) each one equipped with a camera and a compass. The door to cross is equipped with a wide-angle camera.

Figure 3 illustrates four different (admissible) configurations that provide the information required by the action “cross-door”, which include the functionalities above.

The first configuration involves only the robot performing the action. The robot is equipped with a panoramic camera sitting on a post that makes it possible to view the door even when pushing the box. The camera produces information to a functionality that measures the position and orientation of the door relative to the robot.

The second configuration in Figure 3 shows the other extreme, when all information is provided by the door that the robot is crossing and the robot is not contributing with any information. The door is equipped with a camera and functionalities that can measure the position and orientation of



Robot A
Camera on A
Image
Measure pos + orientation of door
Pos + orient of door wrt A
Door
Camera on Door
Image
Measure pos + orientation of robot A
Pos + orient of robot A wrt door
Coordinate transformation door → A
Pos + orient of door wrt A
Robot A
Robot B
Camera on B
Image
Measure pos + orientation of door
Pos + orient of door wrt B
Coordinate transformation B → A
Image
Measure pos of robot A
Pos of robot A wrt B
Compass on B
Global orient of B
Orient of robot A wrt B
Compass on A
Global orient of A
Pos + orient of door wrt A
Robot A
Robot B
Camera on B
Image
Measure pos + orientation of door
Pos + orient of door wrt B
Coordinate transformation B → A
Image
Measure pos of robot A
Pos of robot A wrt B
Image
Measure angle to robot A
Angle
Orient of robot A wrt B
Camera on A
Image
Measure angle to robot B
Angle
Pos + orient of door wrt A
Robot A

Fig. 3. Four different configurations that provide the position and orientation of a given door with respect to robot A. See explanation in the text.

the robot relative to the door. This information is transformed into position and orientation of door with respect to the robot before it is delivered to robot A.

The third and fourth configurations in Figure 3 consist of two robots (A and B), each with its own set of resources and functionalities.

In the third configuration, robot A (the robot performing the "cross-door" action) only contributes with one resource, a compass. Robot B's resources are a compass and a camera. The camera provides information to two functionalities: one that measures the distance and orientation to the door, and another one that measures the distance to robot A. All these measurements are computed relative to robot B. In order to compute the position and orientation of the door relative to robot A, we need to use a coordinate transformation. This in turn requires that we know the relative position and orientation of robot A relative to B. The relative position is obtained from the camera information. The relative orientation can be obtained by comparing the absolute orientations of the two robots, measured by their two on-board compasses.

The fourth configuration in Figure 3 is similar to the third one, except that the orientation of robot A relative to B is obtained in another way, i.e., no compasses are used. Both robots are equipped with cameras and have a functionality that can measure the bearing to an object. When the robots are looking to each other, each robot can measure the bearing to the other one. By comparing these two measurements, we obtain the orientation of robot A relative to robot B.

IV. Configuration generation

Our second objective is to generate configurations automatically. This process requires a declarative description of:

- the configuration domain,
- the state in which the configuration should be applicable, and
- the goal for what the configuration should produce.

The domain of our configuration gives a specification of the available functionalities and their properties. The state declares the available robots and their capabilities, as well as the state of the surrounding environment. The goal for the configuration generation process is to produce a desired information input. For example, in the cross-door example, information goal is the position and orientation of the door that is required as input by the cross-door behavior.

The description of available functionalities is realized using operators similar to those of AI action planners. Two functionality operators from the scenario in the previous section, respectively called `measure-door` and `camera`, are shown in Figure 4.

The input and output of a functionality represent the data flow associated with the functionality. In the `measure-door` example we have an image taken by camera `r` as input and from that we are able to compute the position and orientation of the door `d` relative to `r` as output. The second example is an operator for a camera. Output from `camera` is an image taken by camera located on robot `r`.

```
(functionality
    name:     measure-door(r, d)
    input:    image(r, d)
    output:   position(r, d),
              orientation(r, d)
    precond:  visible(r, d)
    postcond:
    )

(functionality
    name:     camera(r, o)
    input:
    output:   image(r, o)
    precond:  camera-on
    postcond:
    )
```

Fig. 4. Two functionality operators.

Since `camera` is a sensing resource no input is specified. There are also certain conditions that need to be satisfied in order for the functionality to operate, and conditions that will be satisfied if the functionality is executed. This causal flow is represented as preconditions and post-conditions in the operator. For instance the precondition for `measure-door` is that the door `d` is fully visible in the input image and the precondition for `camera` is that the camera is switched on. Notice that the output of `camera` matches the input of `measure-door`. Intuitively, this means that a channel between these two functionalities can legally be created.

In order to combine functionalities to form admissible configurations that solve specific tasks, we have chosen to use techniques inspired by hierarchical planning, in particular the SHOP planner [14]. Configuration planning differs from action planning in that the functionalities, unlike the actions in a plan, are not temporally and causally related to each other, but related by the information flow as defined by the channels. Functionalities are executed in parallel, and a functionality becomes active when the input data is present. In order to achieve the data flow between functionalities, a mechanism that creates channels between functionalities is required. Such connections are not required for action planning, and obviously, no existing planning technique facilitate such a mechanism. Therefore we outline how the planner works.

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 camera functionality and the measure-door functionality above with a channel on the same robot in order to obtain door measurements of a specific door.

Figure 5 shows an example of a method that does exactly that. There is a local channel inside the method connecting the two functionalities (labeled `f1` and `f2`). In addition, the position and orientation output of `f2` is declared in the `out` field to be the output of the entire method. Thereby, any channel that in a method higher up in the hierarchy is connected to `get-door-info` will effectively be connected to `measure-door`.

```
(config-method
  name: get-door-info(r, d)
  precond: ( camera(r), in(r, room),
             in(d, room), robot(r), door(d))
  in:  -
  out: f2: pos(r, d)
       f2: orient(r, d)
  channels: (local(r), f1, f2, image(r, d))
  body:
    f1: camera(r, d))
    f2: measure-door(r, d)
)
```

Fig. 5. A method used by our planner.

The configuration planner takes as input a current causal state s, a method ”stack” with initially one unexpanded method 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 also maintains an initially empty configuration C and an initially empty sets of causal postconditions P. It works as follows:

1) Take the method instance $l : m(c_1, c_2, ...)$ at the top of the stack.
2) For this method instance:
    a) If the method instance is a functionality, check if it already exist in the current configuration C. If it does not, add it to the current configuration C. Go back to 1.
    b) If it is a method, select an instantiated version of the method schema m from M with preconditions which are applicable in s (this is a backtrack point). If there is none, report failure and backtrack.
3) Expand the method as follows:
    a) Instantiate the remaining fields of the selected method, and generate new unique labels instead of the general labels `f1`, `f2` etc.
    b) Add the channels of the method body (with new labels) to the current configuration C.
    c) Add the method instances (with new labels) of the method body to the top of the stack.
    d) Use the in and out fields of the method to reconnect any channels in C from the method instance being expanded (i.e. with label l) to the new method instances as labeled in the method body.
4) If the stack is empty, return C. Otherwise go back to step 1.

By trying all applicable method versions, a set of admissible configurations can be obtained. The set of postconditions for the chosen configuration can be used to update the current state, which then can be used as input for generating the configuration following the current one (if any).

An example of a configuration description generated by the planner is shown in Figure 6. This description is equivalent to the second configuration in Figure 3.

```
(configuration
  :functionalities
    f-31 cross-door(robota, door1)
    f-42 transform-info(door1, robota)
    f-47 measure-robot(door1, robota)
    f-46 camera(door1, robota)
  :channels
    local(door1, f-46, f-47,
          image(door1, robota))
    local(door1, f-47, f-42,
          pos(robota, door1),
          orient(robota, door1))
    global(door1, robota, f-42, f-31,
           pos(robota, door1),
           orient(robota, door1))
  )
```

Fig. 6. A configuration description generated by our planner.

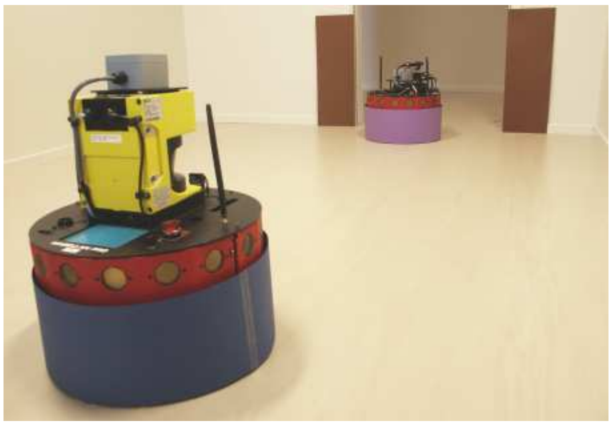


Fig. 7. Robot B is guiding robot A through the door.

Generally, there are several configurations that can address a problem. Our planner generate descriptions for all admissible configurations. Obviously, only one configuration per problem can be performed at the time. In order to choose the ”best” configuration to execute, the costs for configurations need to be considered. Currently, we compute the cost as a weighted sum of the number of components used (robots involved, functionalities, global and local channels).

The planning takes place on the robot that has the information goal. The selected configuration description is then implemented in two steps. First, the different functionalities and channels are distributed according to their location parameter (e.g. in the configuration description above, cross-door should be executed by `robota`). Then each robot launches a process for executing its assigned functionalities and sets up its channels.

Even thought the example we use to describe our approach is simple, our system is able to generate configurations for more complex problems. In the area of object transportation we have addressed two tasks with our configuration planner. The first task is considering two robots carrying a bar. The second tasks involves three robots building a wall with several wall blocks. In this task, two robots are pushing a wall block together and a third robot is guiding them in order to get the block aligned with the rest of the wall. Both tasks require tight coordination between the involved robots.

V. Experiments

We have implemented the approach described above, and we have conducted a series of experiments using a pair of real robots equipped with different sensors. These experiments were also aimed at assessing the mechanisms for the switching between configurations. In these experiments, however, we do not automatically monitor the quality of configurations during execution: the decision about when to switch configuration is done manually. The state given to the planner is not updated during execution.

We report here a representative experiment based on the third and fourth configurations in Figure 3. The platforms used were two Magellan Pro robots from iRobot, shown in Figure 7. Each robot runs an instance of the layered hybrid architecture Thinking Cap, based on [18].

Both robots are equipped with compasses and fixed color cameras. They have additional sensors (e.g., sonars, laser, and an electronic nose) not used in this experiment. The environment consists of two rooms (R1 and R2) with a door connecting them. The door and the robots have been marked with uniform colors in order to simplify the vision task (see Figure 7).

The following scenario describes how the two configurations were used, and demonstrates the importance of being able to reconfigure dynamically. Robot A and robot B are in room R1. Robot A wants to go from room R1 to room R2. Since the camera on A is fixed and it has a narrow field of view, the robot cannot see the edges of the door when it is close to it. Therefore, robot A is not able to perform the action on its own. Robot B is equipped with the same sensors as robot A, but since robot B is not crossing the door it is able to observe both the door and robot A from a distance during the whole procedure. We therefore configure our team according to the third configuration in Figure 3, and execute the task. Robot A continuously receives information about the position and orientation during the execution of “cross-door”.

When robot A enters room R1 it signals that the task is accomplished. This signal is received by robot B and the current configuration is played out. Next, robot B is assigned the task of going from room R1 to room R2. The same configuration as before is used to solve this task, but with the roles exchanged — i.e., robot A is now guiding robot B. This time, however, during the execution of the “cross-door” behavior a compass fails due to a disturbance in the magnetic field. This makes the current configuration not admissible, and a reconfiguration is necessary to proceed. The fourth configuration in Figure 3 is still admissible even with no compass, and we therefore use this one to carry out the remaining part of the task. Figure 8 shows the trajectories performed by the robots in a sample run of this experiment. In the picture, robot A is standing still at the observing position

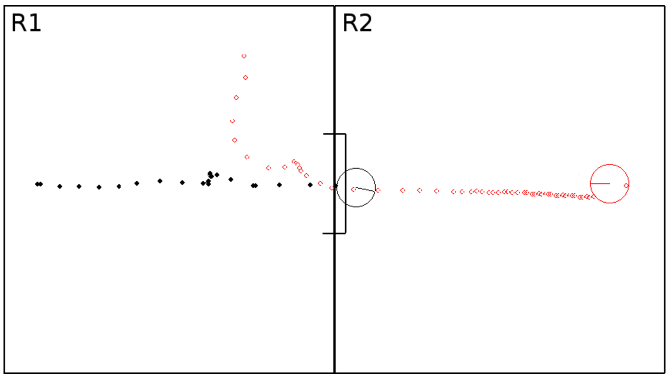


Fig. 8. Robot A and B have both reached room R2. Circles show robot A's trajectory and dots show robot B's trajectory.

and robot B has just accomplished its task.

VI. Discussion

We have presented a general approach to automatically synthesize a team configuration using knowledge-based techniques. Our approach combines resources and functionalities, residing in different robots, into a functional configuration, in which the robots cooperate to generate the information required to perform a certain task. Configurations are generated automatically using a hierarchical planner. Other techniques could be used, e.g., a constraint satisfaction system [12]. However, we expect that the use of a hierarchical planner will make it easier to scale up our approach to more complex scenarios, and to deal with the problem of when and how to change (replan) a configuration.

An interesting point to note is the relation between our functional configurations and task assignment. As it can be seen in the above experiment, configuration generation is used after a task is assigned to a robot, and it concerns the problem of deciding if and how that task can be executed cooperatively with the help from other robots. A related problem is how to motivate the robots involved in a configuration to commit to it, and to stay committed during the entire execution. For example, in the experiment above, we must guarantee that the observing robot does not run away from the scene. For most tasks, robots are motivated by the individual or the team goals. However, if the task is beneficial only to another individual, as in the case of robots that help each other, it may be harder to motivate a commitment. Task assignment, commitment and functional configurations all address different aspects of the multi-robot cooperation problem. In our work, we concentrate on the last one.

With respect to the three objectives stated in the introduction, we note that we not yet addressed the third one: how to monitor the performance of a configuration while executing it. Such a monitoring process is important since it enables replanning and configuration switching. Our current work aims at extending the configuration evaluation process using costs to include run-time monitoring and dynamic reconfiguration.

The next important step will be to consider sequences, or plans, of configurations, in order to address more complex tasks. Our current system only considers the generation of configurations for performing one step of a particular task, and cannot deal with situations requiring several steps. In our box pushing example, if a second box is blocking the door, a configuration for removing that box would have to precede the configuration for getting the first box through the door. We are investigating the extension of our planning techniques to generate plans involving sequences of configurations.

Acknowledgments

This work was supported by the Swedish National Graduate School in Computer Science (CUGS), the Swedish Research Council (Vetenskapsrådet), and the Swedish KK Foundation.

References

[1] M. Benoit, I. Guillaume, G. Malik, and I. Felix. Robel: Synthesizing and controlling complex robust robot behaviors. In *Proceedings of the Fourth International Cognitive Robotics Workshop, (CogRob 2004)*, pages 18–23, August 2004.

[2] G. Boella. Norms and cooperation: Two sides of social rationality. In H. Hexmoor, C. Castelfranchi, and R. Falcone, editors, *Agent Autonomy*. Kluwer, Boston/Dordrecht/London, 2003.

[3] S. Botelho and R. Alami. M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1234–1239, 1999.

[4] L. Chaimowicz, M. Campos, and V. Kumar. Dynamic role assignment for cooperative robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 293–298, 2002.

[5] S. A. Chien and H. B. Mortensen. Automating image processing for scientific data analysis of a large image database. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):854–859, 1996.

[6] E.H. Durfee, V.R. Lesser, and D.D. Corkill. Coherent cooperation among communicating problem solvers. In A.H. Bond and L. Gasser, editors, *Readings in Distributed AI*, pages 268–284. Morgan Kaufmann, San Mateo, CA, 1988.

[7] T. Fukuda and S. Nakagawa. Approach to the dynamically reconfigurable robot systems. *Intelligent Robtics Systems*, 1:55–72, 1988.

[8] B.P. Gerkey and M.J. Matarić. Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768, October 2002.

[9] B.P. Gerkey and M.J. Matarić. Multi-Robot Task Allocation: Analyzing the Complexity and Optimality of Key Architectures. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Taipei, Taiwan, May 2003.

[10] L. Gong and A.C. Kulikowski. Composition of image analysis processes through objectcentered hierarchical planning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(10), 1995.

[11] J. S. Jennings and C. Kirkwood-Watts. Distributed mobile robotics by the method of dynamic teams. In *Proc. of the Intl. Symp. on Distributed Autonomous Robotic Systems (DARS)*, Karlsruhe, Germany, 1998.

[12] V. Kumar. Algorithms for constraints satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992.

[13] F. Mondada, M. Bonani, S. Magnenat, A. Guignard, and D. Floreano. Physical connections and cooperation in swarm robotics. In *Proc. of the 8th Int. Conf. on Intelligent Autonomous Systems (IAS8)*, pages 53–60. IOS Press, 2004.

[14] D. Nau, Y. Caoand, A. Lothem, and H. Munoz-Avila. SHOP: simple hierarchical ordered planner. In *Proc. of the Int. Joint Conf. on Artificial Intalligence IJCAI*, pages 968–973, 1999.

[15] L. Parker. ALLIANCE: An architecture for fault tolerant multi-robot cooperation. *IEEE Trans. on Robotics and Automation*, 14(2), 1998.

[16] L. Parker, B. Kannan, F. Tang, and M. Bailey. Tightly-coupled navigation assistance in heterogeneous multi-robot teams. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[17] D.V. Pynadath and M. Tambe. Automated teamwork among heterogeneous software agents and humans. *Journal of Autonomous Agents and Multi-Agent Systems*, 7:71–100, 2003.

[18] 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.

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

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

[21] R. Simmons and D. Apfelbaum. A task description language for robot control. In *Proceedings of the Conference on Intelligent Robotics and Systems*, Vancouver Canada, October 1998.

[22] R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith. First results in the coordination of heterogeneous robots for large-scale assembly. In *Proceedings of the International Symposium on Experimental Robotics (ISER)*, Honolulu Hawaii, December 2000.

[23] P. Stone and M. Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, 1999.

[24] I. J. Timm and P-O Woelk. Ontology-based capability management for distributed problem solving in the manufacturing domain. In M. Schillo and et al., editors, *Multiagent System Technologies – Proceedings of the First German Conference, (MATES 2003)*, pages 168–179. Springer Verlag, September 2003.

[25] D. Vail and M. Veloso. Multi-robot dynamic role assignment and coordination through shared potential fields. In A. Schultz, L. Parker, and F. Schneider, editors, *Multi-Robot Systems*. Kluwer, 2003.

[26] B. B. Werger and M. J Matarić. Broadcast of local eligibility for multi-target observation. In L. E. Parker, G. Bekey, and J. Barhen, editors, *Distributed Autonomous Robotic Systems*, pages 347–356. Springer Verlag, 2000.